Decentralized Infrastructure for File Storage

by

Nicola Greco

BSc. University College London (2014)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

Author .... Signature redacted .....
Department of Electrical Engineering and Computer Science
August 31, 2017

Signature redacted

Certified by ....
Professor Tim Berners-Lee
Thesis Supervisor

Signature redacted

Accepted by ....
Professor Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Decentralized Infrastructure for File Storage

by

Nicola Greco

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

How might we incentivize a peer-to-peer network to store users' files? The purpose of this

Acknowledgments

A mia nonna Santa Paccone

This research was performed under the supervision of Prof. Tim Berners-Lee, in collaboration with Juan Benet and the team at Protocol Labs and based off the Filecoin Whitepaper [12]. The beautiful drawings are by Virginia Alonso and Figure 7-1 and 6-1 by Dr. Evan Miyazono.

I would like to thank Tim for being an incredible inspiration. In particular, for motivating an fantastic community to work on democratizing technology and decentralizing the web. Working with Tim has been a dream of mine for many years, I am tremendously grateful to have him as my advisor and I look forward to continue learning from him.

I am deeply grateful to have met Juan in my first year. I consider him my second advisor. He introduced me to a lot of great problems and the history of Bell Labs. I am honored to have collaborated with him on the new Filecoin and other projects.

My special thanks go to: David Weinberger, for our insightful and inspiring recurring meetings on decentralizing the web and its importance (when I was 16, I decided to learn English to learn from him one day!); and Srini Devadas, who provided timely and insightful feedback on this thesis and to other ideas throughout my second year.

I am proud to be part of three incredible groups at MIT: Decentralized Information Group (DIG), Internet Policy Research Initiative (IPRI) and the Digital Currency Initiative (DCI). Thank you to Gerry Sussman, Lalana Kagal, Ilaria Liccardi and Amy van der Hiel for being an incredible (and often life-saving!) resource to graduate students. Thank you to Neha Narula, Mark Weber, Christian Catalini and Madars Virza for insightful conversations on cryptocurrencies and to Chelsea and Sunoo for giving me a FAIL in all the Cryptography classes I took. Thank you the real spirit of the IPRI group: Andrei Sambra, Dan Friedman, Mike Specter, Jonathan Frankle, Leilani Gilpin, Ben Yuan, Marc Aidinoff, Cecilia Pacheco!

I am proud to have been a fellow at the Berkman-Klein Center at Harvard during my graduate studies. During my time there, I participated to the Blockchain Working Group with Samer Hassan, Primavera De Filippi, Patrick Murck. I am so grateful to have learned and brainstormed with this marvelous group.

I would never be at MIT without the life-long support of my family in my education: Vittoria, Enrico, Lucia. A *super special* mention goes to my girlfriend Virginia which has been a constant source of inspiration with her art, gave me strength in difficult moments and finally annotated in red every page of this thesis.

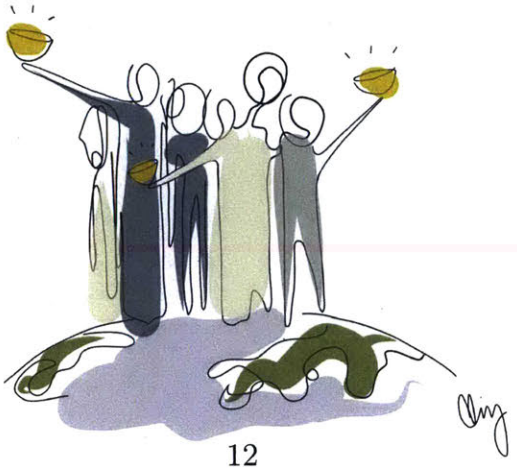
Last but not least, my experience would not be the same without my friends and flat-mates Niccolo Pignatelli and Matthieu de Vergnes - which unfortunately hardly believed that cryptocurrencies would ever be a thing and won't get rich this round. I thank Adam Yala for being my point of reference and real friend in the peaks and lows of my time at MIT. The list of friends would be infinite, but I will save this very last line for Dhivya Ravikumar; and this one for Carmelo Presicce - that printed the final final final version of this thesis.

Contents

List of Figures

Chapter 1

Introduction



One Policy, One System, Universal Service.

— Theodore Vail, Bell Systems, 1907

Critical infrastructures such as postal services, financial services and telecommunication services have long been provided in a centralized fashion: governments or a small number of companies build the infrastructure and operate the service. The World Wide Web [16], for instance, originally intended to be a way for everyone to host, share and link web pages from their own computers, has gradually turned into a few critical online services, such as search, email, cloud computing, e-commerce, being offered by a few technology giants. Beyond the ethical or political concerns that emerged [2, 31], the centralization of the Web also created technical issues: users are now locked in silo-ed platforms - the so-called 'walled gardens', and the Web relies on few points of control and single points of failure. In light of these issues, a question emerges: Can we design critical infrastructures in a decentralized fashion, where anyone can become a provider, anyone can access? Despite the myriad of possible new decentralized infrastructure, this thesis studies the case of online cloud storage: How can we incentivize a peer-to-peer network of computers to coordinate and provide cloud storage services, in an accountable and interoperable manner?

More specifically, in this thesis we explore the potential of using *blockchain technologies* and recent advances in the *Proof-of-Storage* to build a decentralized infrastructure for cloud storage. This consists in creating a distributed protocol that coordinates a network of independent storage providers to offer cloud storage services in exchange of payments, without relying on a single point of control or trusted party. In practice, the purpose of this thesis is to create a decentralized market for storage services, any node in the network can rent their disk space as long as they can prove that they are storing or serving data.

Novel breakthroughs in distributed ledgers made it possible to create decentralized payments systems [50]. Ever since, distributed ledgers have been explored as replacement for the "trusted third party": instead of trusting an intermediary, one can trust a network of users mantaining the ledger. In this thesis, we use a blockchain-based ledger to validate that storage providers have correctly stored files and to perform payments between clients and providers.

In summary, this thesis presents elementary building blocks for creating decentralized

infrastructures, in particular a decentralized file storage. The thesis culminates in describing

1.2 Contributions

In contrast with past work published on the topic, this thesis introduces novel work on the *fair-exchange* problem, *proofs-of-storage* and usage of *blockchain-based protocols*. Additional topics that are explored in this thesis are presented below:

1. **Decentralized Storage Network** (DSN): We provide an abstraction for a network of independent storage providers to offer storage and retrieval services (in Section 5).

2. **Verifiable Markets**: We present a specific class of market protocols and a novel take on the fair-exchange problem, where an exchange between two parties is guaranteed to happen if a seller can generate convincing cryptographic evidence of having provided the requested service. We use Verifiable Markets to create a Storage Market and a Retrieval Market where storage providers and clients can respectively submit storage and retrieval orders.

3. **Filecoin Protocol**: We combine our study of Verifiable Markets and novel *Proofs-of-Storage* (in Section 6) to create an incentivized DSN. Consequently, we show how to operate the DSN on a blockchain-based ledger. The two novel *Proofs-of-Storage* that we present are *proof-of-Replication* and *Proof-of-Spacetime*. Firstly, *Proof-of-Replication* allows storage providers to prove that data has been *replicated* to its own uniquely dedicated physical storage. Enforcing unique physical copies enables a verifier to check that a prover is not deduplicating multiple copies of the data into the same storage space. Secondly, *Proof-of-Spacetime* allows storage providers to prove they have stored any given data during a specified amount of time.

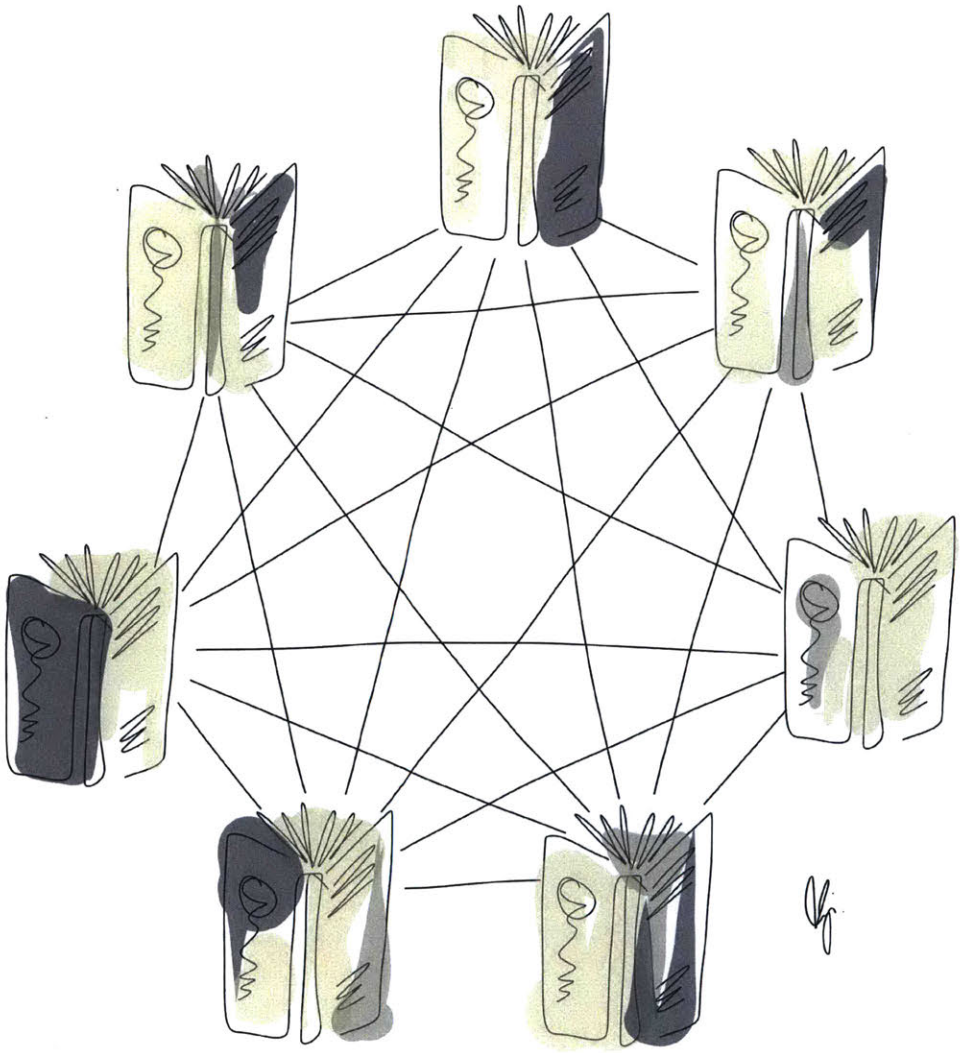
4. **Intuitions for Useful *Proof-of-Work***: We show how to construct a useful *Proof-of-Work* based on *Proof-of-Spacetime*, which can be used in consensus protocols. In this scenario, miners do not need to spend wasteful computation to mine blocks, but instead must store data in the network.

1.3 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, we conduct a literature review of decentralized systems and related work to incentivized file storage. In Chapter 3, we review the state of the art of the token economy. Chapter 4 introduces the definition of Verifiable Markets, introducing a novel variant to the fair-exchange problem and a basic protocol on how to construct decentralized markets on a blockchain-based ledger. Chapter 5 presents our definition of and requirements for a theoretical *Decentralized Storage Network* (DSN) scheme. In Chapter 6, we introduce *Proof-of-Replication* and *Proof-of-Spacetime* protocols, used within Filecoin to cryptographically verify that data is continuously stored. In Chapter 7, we present the setting for an incentivized DSN and we give an overview of the Filecoin protocol. Consequently, in Chapter 8 we describes the specific design of the Filecoin DSN, where we define and detail data structures, protocols, and the interactions between participants. Lastly, Chapter 9 presents future work, in particular the intuition for a *Useful Proof-of-Work* based on file storage and *Proof-of-Spacetime*, as well as ways to integrate a smart contracts within Filecoin. Finally, we present conclusions on the study conducted in Chapter 10.

Chapter 2

Related Work



In this work, we follow the definition of decentralized systems presented in [64]. Distributed systems are system with multiple components that co-ordinate by exchanging messages via a network and are managed by a single party. Decentralized systems are a subset of distributed systems, where, unlike the former, multiple authorities control different components and no authority is fully trusted, implying that any component could be an adversary.

In this chapter, we give a background and review related work in cryptocurrencies, fair-exchange, proofs of storage and peer-to-peer file sharing.

2.1 Fair-exchange

2.1.1 The Fair-Exchange Problem

The problem of fair-exchange studies how two mutually distrusting parties want to swap digital goods such that neither can cheat the other has been studied extensively in the cryptographic literature [5, 37, 52]. We know, due to a classic result [24, 51], that in the presence of malicious parties a fair-exchange is impossible: one party will always have an advantage over the other. The traditional way to perform a fair-exchange is by relying on a Mediator, a trusted third party, which is assumed to be honest and will help the involved parties to perform the exchange fairly. The study of fair exchange leads to the design of

contracts, one can create an escrow mechanism for the exchange of digital assets. A line of work has monetary penalties, where parties are incentivized to complete a protocol fairly, if one party receives its input but aborts before the other party does, the cheating party is automatically subject to pay a penalty [4, 15, 40]. Other research has shown how to build fair decentralized prediction markets [21] and fair-exchange of physical goods [33].

2.1.3 Zero-Knowledge Contingent Payments

Zero Knowledge Continegent Payments (ZKCP) is an elegant exchange protocol where two parties atomically exchange the solution of an NP problem in for a payment. In the ZKCP protocol, the seller knows a solution to an NP problem that a buyer is interested in buying. The problem was originally introduced in 2011 [44] and later revisited [20].

Informally, the seller sends to the buyer (i) the encrypted solution, (ii) the hash of the encryption key, and (iii) a zero-knowledge proof of having done so correctly. The buyer receives it and verifies the proof. Then, it deposits to the blockchain some funds that can

[53] proposes the use of PoSpace to replace expensive Proof-of-Work for achieving consensus in permissionless blockchains.

2.2.2 Provable Data Possession

Provable Data Possession (PDP) schemes [7] allow a user (a verifier $\mathcal{V}$) who outsources data $\mathcal{D}$ to a server (a prover $\mathcal{P}$) to repeatedly check if the server is still storing $\mathcal{D}$. The user can verify the integrity of its data outsourced to a server in a very efficient way (i.e. more efficiently than downloading the data). The server generates probabilistic proofs of possession by sampling a random set of blocks, and transmits a small constant amount of data in a challenge/response protocol with the user. The original scheme [7] has been improved in [8] to achieve more scalability: the idea is to come up with all the future challenges during the setup and store pre-computed answers as meta-data. This protocol has been later improved to support dynamic updates of data [29].

2.2.3 Proof-of-Retrievability

Proof-of-Retrievability (PoR) schemes [38, 61] are similar to PDPs, but offer an extra property: the client can actually "recover" the data outsourced. In PDPs, the server may store the data $\mathcal{D}$ and provide valid PDP proofs, yet hold the data hostage and never release it. PoRs solves this problem by making the proofs leak pieces of data. Users can reconstruct $\mathcal{D}$ by collecting proofs from multiple challenge/response interactions. The first scheme [38] has been followed by many variants, with improved efficiency [18, 61], public verifiability [61] and the ability to support file systems [62].

2.3 Decentralized Storage

In early 2000s, there emerged an unprecedented surge of public interest in peer-to-peer systems for persistent storage of data. The first mainstream P2P system was Napster, which attracted millions of users. Following Napster, more distributed solutions appeared, such as Bittorent [25], Freenet [23], Kazaa [42] and Gnutella [57], based on gossip protocols and

distributed hash tables [45, 59].

Initially, these services relied upon altruistic behavior by their users. The assumption was that users in the network were willing to provide storage and bandwith to the network. However, this resulted in selfish individuals opting out of voluntary contribution.

Most incentive schemes are based on reciprocation to prevent the *free-riding* problem - users have no incentive to store someone else's files - and thereby adding value to the network. A study in 2001[34] constructed a game theoretical model to analyze the incentives in these systems.

2.3.1 Peer-to-Peer File Sharing

Peer-to-Peer (P2P) file-sharing systems allow users to download files directly from one another. We refer the reader to [3] for a comprehensive survey on P2P systems for content distribution.

P2P systems have been successfully used to distribute popular files: the more popular the file, the more users are storing it and willing to serve it to the rest of the network. For example, Bittorrent [25] is a peer-to-peer file sharing platform, where peers find each other via online trackers or a DHT. Users that contribute with upstream bandwidth are rewarded with bandwidth from other users, and consequently can download files faster. When a file is popular, many users announce it to the trackers and users can download it faster by requesting it to multiple sources. A study in 2013 [68] shows that systems like Bittorrent can support over 20 millions of users daily. In contrast, IPFS [11] is a distributed file system with a self-certifying name space: differently from the previous protocol, it does not rely on a specific content routing mechanism. Another example is Freenet [23], an early decentralized content-distribution system, where a peer-to-peer network self-organizes to create a collaborative virtual file system, which focuses on security, publisher anonymity and deniability.

Other systems tried to tackle storage of less popular files (such as personal backups), by allowing users to share part of their disks in exchange of some else's storage [26].

2.3.2 Blockchain-based File Storage

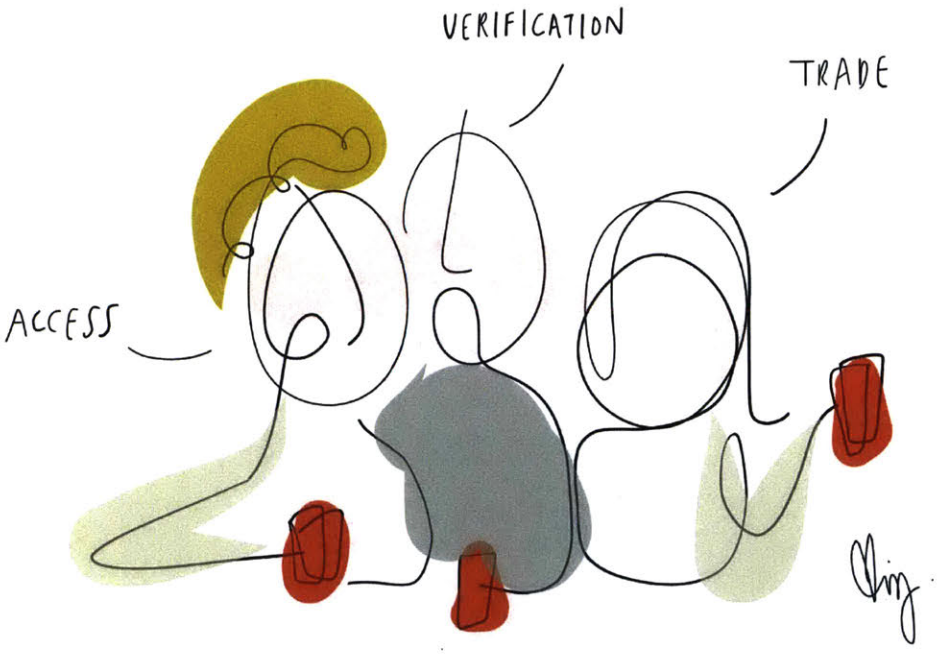
After Bitcoin [50] presented a decentralized solution for digital payments, industry and academia started exploring incentive schemes for creating file storage networks. Instead of using reputation systems or tit-for-tat incentives, storage providers are directly paid via a cryptocurrency for the storage they are providing.

An early application of blockchain technology for file storage is Permacoin [48]. Permacoin is a blockchain with a novel Proof-of-Work consensus protocol based on storing useful data

Clients make storage deals with a set of miners of their choice and anyone in the network can participate in serving the data, regardless of whether they have been assigned to store a file or not.

Chapter 3

Review of Protocol Tokens



In this chapter, we provide some background to *protocol tokens* and present a review of the different type of tokens, their properties and their infrastructure.

3.1 Background on Tokens

3.1.1 Cryptoeconomics

Cryptoeconomics is a new field, originated from the new wave of cryptocurrencies, that combines ideas from cryptography, computer networks and game theory to design distributed protocols with intrinsic economic incentives. *Cryptoeconomic protocols* create financial incentives to drive a network of rational economic agents to coordinate their behaviour towards desired objectives. Originally, economic incentives have been used to create decentralized currencies [19, 50], where "miners" coordinate to maintain a payment system and obtaining coin rewards for doing so. In the past few years, the attention from designing new currencies moved into designing new decentralized services that can be accessed by spending the token of the protocol.

3.1.2 Protocol Tokens

Protocol Tokens are the native currencies of cryptoeconomic protocols. In this new paradigm, protocol designers can incentivize nodes in a network to coordinate to provide a service by rewarding them in tokens; in return users spend tokens to access the service. Following the principles of decentralization, anyone can participate in the protocol to earn the tokens and anyone can use the service by spending tokens. Correct delivery of a service and exchange of tokens must be designed to be publicly verifiable (see Section 4). The *Token Model* can be considered as a novel business model. Companies are creating new decentralized protocols with native tokens and allocating to themselves a percentage of the tokens - often selling part of it (for other tokens or fiat currency) as a way to crowdfund the work required to create the protocol. Depending from the economics of the protocol, the higher the usage of the tokens, the higher their value will be in the market. A simple way to see tokens is as unique API tokens that can be spent with the network to accessing a service; however, these

tokens can be traded and anyone can become a reseller of API tokens and anyone can earn by serving API requests.

What are applications of tokens? At the time of this writing there are over 900 cryptocurrencies and tokens[1] (launched since Bitcoin [50]). There are services powered by tokens, where spending of tokens allows to delegate some computation, to make bets in prediction markets, to send messages, to issue stocks, to vote, to acquire decentralized domain names and to delegate storage (see Section 8).

3.2 Properties of Tokens

Protocol Tokens generally share the following three properties:

- **Access**: *Tokens give access to a special service (and reward).*

  User can spend tokens to get access to a specific service the network provides. Augur Tokens allow to participate in a prediction market [54], Truebit Tokens allow to delegate computation [63], Ethereum Tokens can be spent to do function call in the Ethereum Virtual Machine [19]; finally in our case, tokens can be spent rent storage.

  Users can participate in providing the infrastructure for the service and are (generally) rewarded for doing so. For example, in our case, storage providers are rewarded in currency for offering storage.

- **Verification**: *Transactions and execution of services are public and verifiable*

  Given the nature of decentralized services, participants in the protocol are independent parties that do not trust each other. Transactions must be logged on a distributed ledger. In order to appear on the ledger, transactions must be validated via a network of verifiers. In addition, the system must provide security/rational guarantees that if a token is spent to access a service, the service must be correctly provided.

- **Trade**: *Tokens can be exchanged amongst users.*

[1]Source: CoinMarketCap on 08/06/17 at `https://coinmarketcap.com/`

Users have wallets, generally in the form of a cryptographic public/private key pair, which they can use to send and receive tokens to other users. These transactions are secured by an underlining blockchain protocol.

3.3 Different Types of Tokens

There are different categorizations of tokens. Tokens can serve multiple purposes and these categories are not mutually exclusive.

- **Equity Tokens**: These tokens resemble the concept of shares in company. Protocols (or companies) issue a tokenized equity, ownership of these token gives access to payments of dividends or other revenue streams of the protocol (or the company). For example, every time a service is used, the owner of these tokens receive a proportion based on the amount they own.

- **Voting Tokens**: Users can vote to influence the behaviour of a protocol. In systems with decentralized governance, users can vote on improvements on the protocol proportionally to their currency. For example, in Aragon [27], token holders can use them to vote new services to be implemented and to vote on how to resolve conflicts. In Proof-of-Stake blockchain protocols systems, tokens can be used by users to vote on the execution of the protocol. For example, in Tezos [35], users vote on the next block, proportionally to the tokens they hold.

- **Staking Tokens**: Users can put some token "at stake" as a collateral to access a service and guarantee good behaviour. In the messaging platform Status, public chats can prevent spam by requiring users to put some money at stake when entering the chat (and will lose the collateral if violating the terms) [36]. In the delegated computation platform Truebit [63], users can put a collateral to guarantee that their computation was correct; if found to be incorrect by other users, this collateral is lost.

- **Market Tokens**: Users use these tokens to participate in a specific marketplace where there is supply and demand for a specific service/product. Buyers and sellers post

their requests/offerings and perform the exchange with the market's token. Depending on the system, the underlining protocol facilitates the exchange (by verifying proofs, providing a market mechanism). There exists already different marketplaces (and a protocol for creating new marketplaces [43]): a market for buying and selling decentralized domain names, and a market for talent, among others. In this work, we present a generalization for Verifiable Markets (see Section 4), where the exchange of a service must be proven in order for the exchange to happen. Consequently, we present a decentralized market for storage (see Section 8).

- **Service Tokens**: Some of the above can also be considered services. However, for the purpose of this classification, service token allow to get a service from the entire network and not from very specific users, as is in the case of market tokens.

- **Stable Tokens**: All of the above can also be stable tokens. A stable token is a token whose value is pegged to another currency. Some protocols prefer stable tokens as it can avoid price volatility.

3.4 Infrastructure for Tokens

3.4.1 How do we power tokens?

Tokens can be powered by their dedicated ledger or reusing existing infrastructure. In this work, we will only consider tokens that are based on blockchain ledgers.

- **Reusing infrastructure**: Often tokens are implemented on top of existing blockchains. A strategy is to use an existing blockchain as an append-only log and an off-chain client verification validates the correctness of the transactions and keeps a state, as in Colored Coins [58] and the Blockstack token [49]. Another strategy is to use existing

Ethereum community has proposed a standard interface for tokens [66] and several open source implementations are available [65].

- **New blockchains**: Tokens can be powered by their own dedicated blockchain. Often the decision is based to technical reasons: the token could require a special consensus protocol that other blockchains do not support or require for transactions to be verified in a special way that the smart contract language of other blockchains do not support, or that would be too expensive to run. Often the decision is based on incentives: conflict of interests between the blockchain mining and the token-based service or different monetary policies needed to be implemented.

3.4.2 Do we always need new tokens?

Not every (incentivized) decentralized service requires its own specific token. Some of the systems mentioned above can be implemented as a smart contract with the native blockchain token, as long as the smart contract language of the hosting blockchain is expressive enough. For example, the Ethereum Name System (ENS) protocol, the domain name system allows to register, transfer and renew Ethereum domain names by using the Ethereum currency only.

However, there are situations in which the service requires separation from the hosting blockchain currency. The specific service could require a different consensus mechanism, a special smart contract operation or special monetary policies. In fact, protocol designers can define the allocation of token at the beginning of the protocol, the strategy to mint new tokens and more importantly, the ability to sell a portion of the token can be used as a way to fundraise or incentivize the early users in participating in the network.

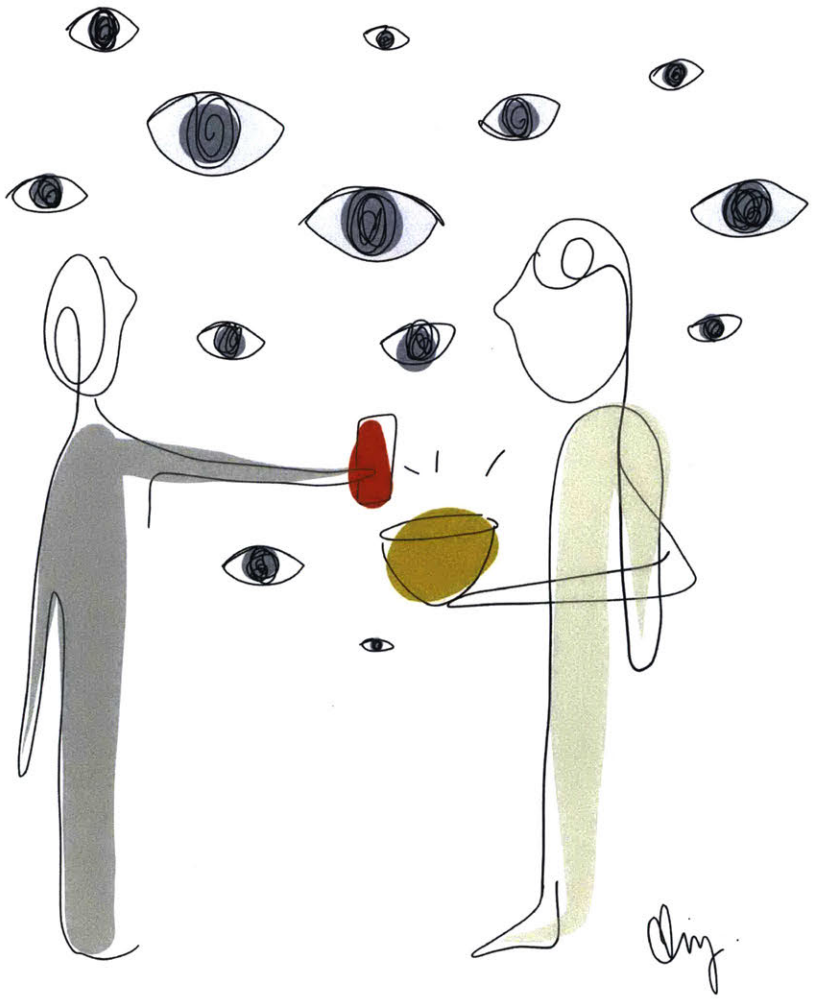
3.4.3 How do users buy tokens?

Depending on the service, users can earn tokens by providing their service to the network. For example, in the case of delegated file storage, users can earn tokens by renting out their storage. Users can buy tokens during a possible fund-raising stage before the token is

launched in the network or directly form other users, alternatively. Popular tokens are often sold both via centralized and decentralized exchanges.

Chapter 4

Verifiable Markets



In this chapter, we present *Verifiable Markets*, a class of market protocols where an exchange is guaranteed to happen if a seller can generate a convincing proof that their good/service has been successfully provided to their buyer. These proofs must be verifiable by a trusted market operator.

To our knowledge, we are the first to introduce the problem of Verifiable Markets and formally study fair-exchange for provable services. In order to achieve this, we introduce the notion of *Provable Services*, a generalization for services whose execution can be proven, and *Verifiable Exchange of Services*, a novel variant of the fair-exchange problem. Finally, we show how Verifiable Markets can be operated in a decentralized setting, where the market operator is replaced by a blockchain network, where no single entity controls exchanges, transactions are transparent, and anybody can participate pseudonymously. Similarly, the consistency of the Order Books, orders settlements and correct execution of services are independently verified via the network.

Note on concurrent work: The ZKCSP protocol presented in [20] is concurrent work to *Verifiable Exchanges of Services* presented in this thesis. The authors use the idea behind ZKCP to sell services instead of goods. In contrast, in this work we present an abstraction for verifiable exchange of services that does not necessarily require use of SNARK (Succinct Non-Interactive ARguments of Knowledge) [10].

4.1 Problem Definition

Markets are protocols that facilitate the exchange of a specific good or service between buyers and sellers. In a decentralized setting, where any node in a network can be a buyer or a seller, how can we guarantee that exchanges are happening correctly and fairly?

The difference between an electronic exchange of services and conventional commerce and barter essentially lies in (i) enforceable laws, (ii) service providers being chosen based on their reputation, and (iii) industries guaranteeing external certified quality control. Below, we present the two problems we are trying to tackle:

- **Problem one (fairness)**: A buyer wants to pay to obtain a service that a seller claims

to provide. If the buyer pays immediately, they risk not receiving the service; it the buyer pays after receiving the service, the seller runs the risk of being defrauded and never being paid. When should the buyer pay the seller?

- **Problem two (correctness)**: The buyer wants to make sure that the seller is correctly providing the service without performing the service herself, since a dishonest seller could perform partial or no work. How can the buyer make sure she is getting the service she requested?

In light of this, Verifiable Markets aim at providing a way for demand and supply of a specific service to meet and to provide fair exchange of services, whose execution must be correct.

4.2 Provable Services

Services provided in a Verifiable Market must be *provable.* A *Provable Service* is an operation performed by a service provider which can also be proven via an interactive proof between a service provider and a client. The service provider must be able to generate a proof that can convince a client of the correct execution of the requested service, without requiring the client to perform the service themselves. The formal requirements of completeness and soundness of a valid proof are described in the *Verifiable Exchange of Services* definition in the following section. In a decentralized setting, we require *Provable Services* to also be publicly verifiable, so that any node in a network can be a verifier that can both challenge the prover (in interactive protocols) and validate their proofs.

4.2.1 Provable Service for File Storage

This thesis focuses on designing a decentralized system that can provide file storage services. In order to make a file storage provable, we must find a valid proof scheme that would give a guarantee to the user that the seller is storing the file for the amount of time required. *Proof-of-Retrievability* is an interactive proof scheme which, on a client's challenge, the prover can generate a proof showing that they are storing the outsourced files. A simple approach

(detailed in Section 7) is to require the seller to submit a *Proof-of-Retrievability* to the Mediator every ten minutes. This would guarantee that a file has been stored throughout time.

4.2.2 Other Provable Services

If a service can be phrased as a statement that can be verified efficiently (e.g. in probabilistic polynomial time or constant time), then we could create Verifiable Markets for these services. Other provable services could be the delegation of computation, where nodes in the network must prove to have correctly executed some computation. If one could delivery energy from one node to another and generate a proof that some energy has been correctly delivered (for example, via trusted hardware), then we could create a decentralized verifiable market for energy.

4.3 Verifiable Exchange of Services

In contrast to the problems of fair-exchange of goods and secrets previously presented in Section 2.1.1, in our setting we are interested in defining a fair-exchange of provable services. In the literature, there is no consensus on what fair-exchange protocols (or its variants) have to provide. Nevertheless, most authors seem to include formulations of fairness and timeliness similar to the ones proposed by [5]. We extend these definitions to provide our own formulation.

Definition 4.3.1. An exchange between two parties, a buyer and a seller, is a *Verifiable Exchange of Services* if it can guarantee the following four properties:

- **Completeness**: If both parties are honest, at the end of the execution the seller always generates a valid proof of their service and receives a payment from the buyer and the buyer receives the outcome of the service.
- **Soundness**: The probability that a malicious seller generates a valid proof and receives the payment, without having correctly provided the service, is negligible.

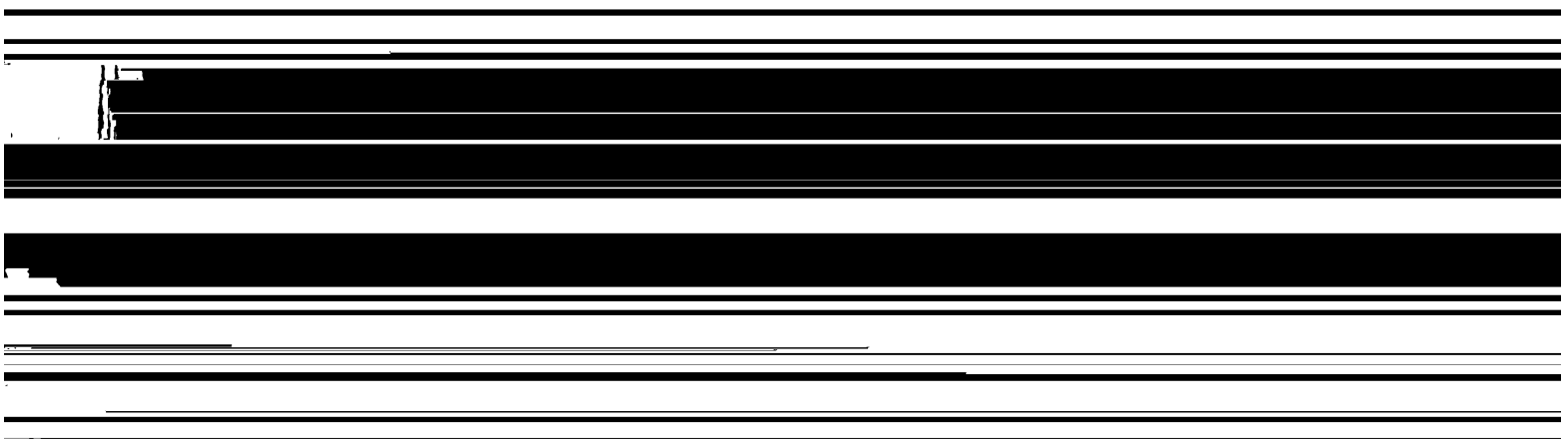
- **Fairness**: There are only two valid outcomes for the protocol:
  1. Seller gets a payment from the buyer and the buyer gets a service/product from seller (when both parties want so)
  2. Seller gets nothing, buyer gets nothing (if at least one party wants so)
- **Timeliness**: If both participants are honest, the exchange can terminate without any help from the other. None of the participants can arbitrarily force the other to wait for the termination of the exchange.

4.4 Verifiable Markets in a Centralized Settings

In this section, we use the intuition of *Provable Services* and *Verifiable Exchange of Services* to design a simple protocol for Verifiable Markets that relies on a trusted third party acting as a mediator. We present the different participants, define Verifiable Markets, and provide a basic protocol and its variations.

4.4.1 Participants

- **Seller**: A seller is interested in providing a specific service. Sellers are paid only if they correctly perform their service and generate a valid proof of having done so.



Verifiable Market Protocol Sketch

Order matching

1. *Orders Submission*: Participants submit their *buy* orders and *sell* orders to the order book. Buyers deposit their payment (and sellers deposit their collateral in case they don't fulfill their order in some cases, so the seller can commit their service) with the Mediator.
2. *Deal*: As new orders come, when two orders matches, the Mediator requires both parties to jointly create a *deal*, or automatically generates *deal* (depending the service). A *deal* commits the two parties to the exchange.

Settlement

3. *Service Execution*: The seller performs the service and generates a proof that the service was provided correctly and sends it to the buyer and the Mediator. Generation of the proof might require interaction between the Mediator and the seller.
4. *Exchange*: The Mediator validates the proof. On success, it processes the payment and clears the order from the order book.

Figure 4-1: Abstract *Verifiable Market* protocol executed by a trusted Mediator

- **Order Book**: The Order Book is collection of *buy* and *sell* orders. Anyone can add/remove orders, until two orders are matched (according to an arbitrary matching algorithm). Orders are removed when settled.

4.4.2 Definition

A Verifiable Market is a protocol between sellers and buyers and it is operated via a Mediator. The service offered in the market must be a *provable service* and individual exchanges must be *verifiable exchanges of service* (see Definition 4.3.1).

The protocol is divided in two phases: *Order Matching* and *Settlement*. During the Order Matching phases, buyers and sellers orders are submitted. When orders match, a *deal* is created. During Settlement, the seller must generate a proof of the service requested in their *deal* and the Mediator performs the exchange.

A basic protocol is described in Figure 4-1.

4.4.3 Note on optimistic exchange

The presented protocol requires the Mediator to witness every proof. However, one could make adjustments to the protocol to only have Mediator interaction in case of conflict. Informally, the buyer could put a deposit with the Mediator, request the service to the seller directly and, upon receiving a proof, can directly send a payment to the seller. In case a malicious buyer doesn't pay the seller, the seller can present a proof to the Mediator, who would perform the exchange.

4.5 Markets on the Blockchain

A Decentralized Market should not be operated by a single party, but by a network of users. In order to remove the single trusted third party, we replace the Mediator with a blockchain network. The intuition is to introduce the following two changes to a standard blockchain ledger:

- **Special transactions**: Two special transactions must be introduced: one for sellers to submit their proofs and one for buyers to deposit their funds. Funds are released if a valid proof is presented.

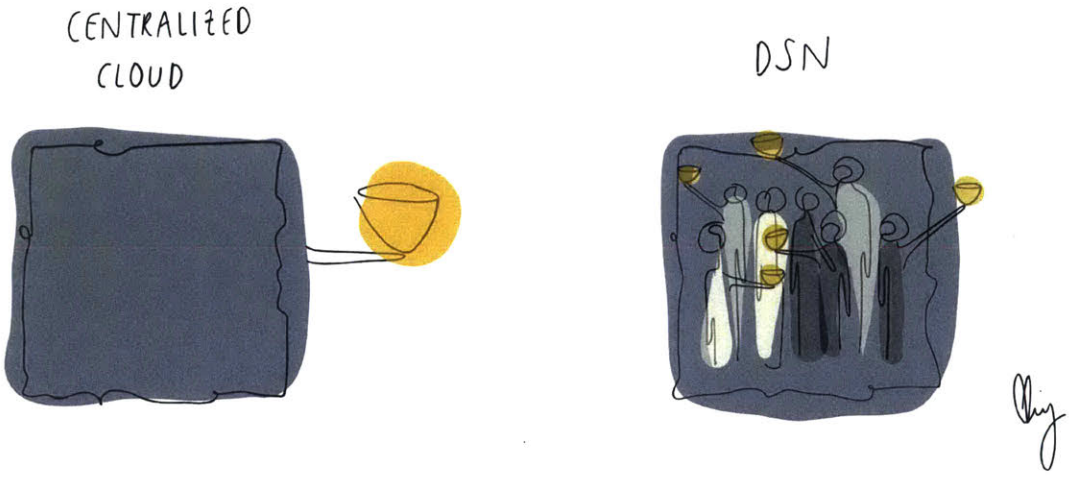
- **Proof verification**: The Auditor algorithm must be added in the transaction verification, such that when a seller submits a transaction with the proof, they trigger the verification. In this way, every node in the network can verify the proofs

Verifiable Markets can be implemented on new blockchains by adding these changes at the consensus/transaction verification level, or on existing blockchains in the form of smart contracts. For example, in a platform such as Ethereum, one could create a smart contract that with a deposit method, and a verification method, as long as the verification can be done with the current cryptographic primitives implemented in the Ethereum Virtual Machine. For simplicity, we refer to the Verifiable Market as a special contract on a blockchain.

In the next two sections, we present how to run the *Order Book* both in chain and off chain. The two strategies resembles two different type of markets: an *exchange* market and

Chapter 5

Decentralized Storage Network



A *Decentralized Storage Network* (DSN) is a network of independent storage providers that self-coordinates to provide storage and data retrieval as one service to their users. There is no central point of coordination and no trusted party: a decentralized protocol helps the participant to coordinate and verify each other's operations. Coordination can be achieved in different ways, according to the requirement of the system (e.g. Byzantine Agreement). Later, in Chapter 8, we provide a construction for the Filecoin DSN.

5.1 DSN Definition

We model a Decentralized Storage Network as a single storage system that aggregates storage from multiple providers and offers Get and Put requests to its users. The network of users and storage providers execute the Manage protocol to coordinate requests and audit the service. Our definition captures systems described in our related work (see Section 2), both altruistic and incentivized: Bittorrent, IPFS, Freenet, StorJ, Sia and Filecoin.

Definition 5.1.1. A DSN scheme Π is a tuple of protocols run by storage providers and clients:

$$(\mathsf{Put}, \mathsf{Get}, \mathsf{Manage})$$

- Put(data) $\rightarrow$ key: Clients execute the Put protocol to *store* data under a unique identifier key.
- Get(key) $\rightarrow$ data: Clients execute the Get protocol to *retrieve* data that is currently stored using key.
- Manage(): The network of participants coordinates via the Manage protocol to: control the available storage, audit the service offered by providers and repair possible faults. The Manage protocol is run by storage providers often in conjunction with clients or a network of auditors[1].

[1]In the case where the Manage protocol relies on a blockchain, we consider the miners as auditors, since they verify and coordinate storage providers

A DSN scheme Π must guarantee *data integrity* and *retrievability* as well as tolerate *management* and *storage faults* as defined in the following sections.

5.2 Modeling Faults

5.2.1 Management faults

Faults during coordination in the Manage protocol are defined as *management faults*. Management fault can compromise the liveness and safety of Get and Put requests, but they do not model storage losses.

Example: Consider a DSN protocol that relies on Byzantine Agreement (BA) to coordinate and audit storage providers. Proofs-of-Storage are submitted to storage providers to the network and the network runs BA to agree on the validity of these proofs. If the BA tolerates up to f faults out of n total nodes, then if more than f faulty nodes are present, the outcome of audits can be compromised.

5.2.2 Storage faults

Faults that result in preventing retrieval of data are defined as *storage faults*. A node is considered faulty if it has lost data or if it stops serving data. The tolerance of a protocol to storage faults is defined as follows:

Definition 5.2.1. Given n independent storage providers, a DSN scheme Π is (f, m)-*tolerant* to storage faults if: (1) a Put execution results in m independent storage providers storing the input data (until expiration) even in presence of up to f faulty providers, (2) a Get execution succeeds even in presence of up to f faulty providers.

Example: Consider a simple scheme, where a successful Put execution results in every participant in the network to store the input data. In this scheme, $m = n$ and $f = m - 1$. Is it always $f = m - 1$? No, some schemes can be designed using erasure coding, where each storage provider stores a special portion of the data, such that x out of m storage providers are required to retrieve the data. In this, case $f = m - x$.

5.3 Properties

Decentralized Storage Networks must provide *data integrity* and *retrievability*. We describe these two properties and we present additional properties required by our incentivized DSN, presented in Section 7.

5.3.1 Data Integrity

Data Integrity guarantees that given a key, clients are guaranteed to retrieve the data originally stored under that key. There is no adversary that can convince clients to accept altered or falsified data at the end of a Get execution.

Definition 5.3.1. A DSN scheme Π provides *data integrity* if: for any successful Put execution for some data d under key k, there is no probabilistic polynomial time adversary $\mathcal{A}$ that can convince a client to accept d', for $\mathsf{d}' \neq \mathsf{d}$ at the end of a Get execution for identifier k.

5.3.2 Retrievability

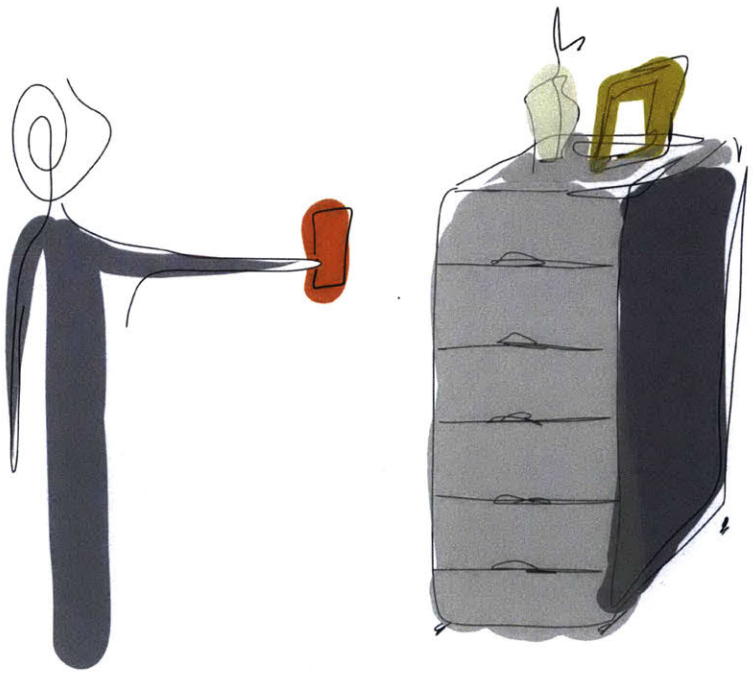
The *Proof-of-Storage* must convince any efficient verifier, who only knows key and does not have access to data.

Definition 5.3.4. A DSN scheme Π is *auditable*, if it generates a *verifiable* trace of operations that can be checked in the future to confirm that storage was indeed stored for the right duration of time.

Definition 5.3.5. A DSN scheme Π is *incentive-compatible*, if storage providers are rewarded for successfully offering storage and retrieval service, or penalized for misbehaving, such that the storage providers' dominant strategy is to store data.

Chapter 6

Novel Proofs of Storage



In a incentivized DSN protocol, storage providers must convince their clients that they stored the data they were paid to store. We require storage providers to generate *Proofs-of-Storage* (PoS) that the nodes in the blockchain network (or the clients themselves) can then verify. The author originally presented this work in [14]. In this chapter we present and outline implementations for the *Proof-of-Replication* (PoRep) and *Proof-of-Spacetime* (PoSt) schemes.

6.1 Motivation

PDP and PoR schemes, previously reviewed in Section 2, only guarantee that a prover had possession of some data at the time of the challenge/response interaction. In our incentivized DSN, we require stronger guarantees to prevent three types of attacks that malicious storage providers could exploit to get rewarded for storage they do not have: *Sybil attack*, *outsourcing attacks*, *generation attacks*.

- *Sybil Attacks*: Malicious storage providers could pretend to store (and get paid for) more copies than the ones physically stored by creating multiple Sybil identities, but storing the data only once.

- *Outsourcing Attacks*: Malicious storage providers could commit to store more data than the amount they can physically store, relying on quickly fetching data from other storage providers.

- *Generation Attacks*: Malicious storage providers could claim to be storing a large amount of data which they are instead efficiently generating on-demand using a small program. If the program is smaller than the purportedly stored data, this inflates the malicious storage provider's likelihood of winning a block reward in Filecoin, which is proportional to the storage provider's storage currently in use.

6.2 Proof-of-Replication

Proof-of-Replication (PoRep) is a novel *Proof-of-Storage* which allows a server (i.e. the prover $\mathcal{P}$) to convince a user (i.e. the verifier $\mathcal{V}$) that some data $\mathcal{D}$ has been *replicated* to its own uniquely dedicated physical storage. Our scheme is an interactive protocol, where the prover $\mathcal{P}$: (a) commits to store n distinct *replicas* (physically independent copies) of some data $\mathcal{D}$, and then (b) convinces the verifier $\mathcal{V}$, that $\mathcal{P}$ is indeed storing each of the replicas via a challenge/response protocol. To the best of our knowledge, PoRep improves on PoR and PDP schemes, preventing *Sybil Attacks*, *Outsourcing Attacks*, and *Generation Attacks*.

Note. For a formal definition, a description of its properties, and an in-depth study of *Proof-of-Replication*, we refer the reader to [14].

Definition 6.2.1. (Proof-of-Replication) A PoRep scheme enables an efficient prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $\mathcal{P}$ is storing a *replica* $\mathcal{R}$, a physical independent copy of some data $\mathcal{D}$, unique to $\mathcal{P}$. A PoRep protocol is characterized by a tuple of polynomial-time algorithms:

$$(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$$

- $\mathsf{PoRep.Setup}(1^\lambda, \mathcal{D}) \rightarrow \mathcal{R}, \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{V}$, where $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{V}$ are scheme-specific setup variables for $\mathcal{P}$ and $\mathcal{V}$, λ is a security parameter. PoRep.Setup is used to generate a replica $\mathcal{R}$, and give $\mathcal{P}$ and $\mathcal{V}$ the necessary information to run PoRep.Prove and PoRep.Verify. Some schemes may require the prover or interaction with a third party to compute PoRep.Setup.

- $\mathsf{PoRep.Prove}(\mathcal{S}_\mathcal{P}, \mathcal{R}, c) \rightarrow \pi^c$, where c is a random challenge issued by a verifier $\mathcal{V}$, and π^c is a proof that a prover has access to $\mathcal{R}$ a specific *replica* of $\mathcal{D}$. PoRep.Prove is run by $\mathcal{P}$ to produce a π^c for $\mathcal{V}$.

- $\mathsf{PoRep.Verify}(\mathcal{S}_\mathcal{V}, c, \pi^c) \rightarrow \{0, 1\}$, which checks whether a proof is correct. PoRep.Verify is run by $\mathcal{V}$ and convinces $\mathcal{V}$ whether $\mathcal{P}$ has been storing $\mathcal{R}$.

Completeness. For every security parameter λ and any data $\mathcal{D}$, an honest prover can convince the verifier with probability $1 - negl(\lambda)$, in the following experiment: $\mathcal{R}, \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{V} \leftarrow$

$\mathsf{PoRep.Setup}(1^\lambda, \mathcal{D})$; $\pi^c \leftarrow \mathsf{PoRep.Prove}(\mathcal{S}_\mathcal{P}, \mathcal{R}, c)$, then $1 \leftarrow \mathsf{PoRep.Verify}(\mathcal{S}_\mathcal{V}, c, \pi^c)$.

Soundness The scheme is sound if no probabilistic polynomial time adversary $\mathcal{A}$ can pass the RepGame with more than $negl(\lambda)$ probability.

Definition 6.2.2. (RepGame) Let $\mathcal{A}$ be an adversary with fixed amount of storage l and access to a party $\mathcal{P}$ with infinite space which can store and retrieve data for $\mathcal{A}$ with latency Δ. $\mathcal{A}$ wins the RepGame if for any data $\mathcal{D}$, $\mathcal{A}$ can convince a verifier $\mathcal{V}$ that $\mathcal{A}$ is storing n replicas, such that $n|\mathcal{D}| > l$. $\mathcal{A}$ runs PoS.Setup for each different replica $i \in \{0...n\}$. $\mathcal{A}$ wins the game if she can produce proofs π^{c_i} that for all i, $\mathcal{V}$ outputs $1 = \mathsf{PoS.Verify}(\mathcal{S}_\mathcal{V}, c_i, \pi^{c_i})$.

6.3 Proof-of-Spacetime

Proof-of-Storage schemes allow a user to check if a storage provider is storing the outsourced data at the time of the challenge. *How can we use* PoS *schemes to prove that some data was being stored throughout a period of time?* A natural answer to this question is to require the user to repeatedly (e.g. every minute) send challenges to the storage provider. However, the communication complexity required in each interaction can be the bottleneck in systems such as Filecoin, where storage providers are required to submit their proofs to the blockchain network.

To address this question, we introduce a new proof, *Proof-of-Spacetime*, where a verifier can check if a prover is storing her/his outsourced data for a range of time. The intuition is to require the prover to (1) generate sequential *Proofs-of-Storage* (in our case *Proof-of-Replication*), as a way to determine time (2) recursively compose the executions to generate a short proof.

Definition 6.3.1 (Proof of Spacetime) A PoSt scheme enables an efficient prover $\mathcal{P}$ to

- PoSt.Setup$(1^\lambda, \mathcal{D}) \rightarrow \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{V}$, where $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{V}$ are scheme-specific setup variables for $\mathcal{P}$ and $\mathcal{V}$, λ is a security parameter. PoSt.Setup is used to give $\mathcal{P}$ and $\mathcal{V}$ the necessary information to run PoSt.Prove and PoSt.Verify. Some schemes may require the prover or interaction with a third party to compute PoSt.Setup.

- PoSt.Prove$(\mathcal{S}_\mathcal{P}, \mathcal{D}, c, t) \rightarrow \pi^c$, where c is a random challenge issued by a verifier $\mathcal{V}$, and π^c is a proof that a prover has access to $\mathcal{D}$ for some time t. PoSt.Prove is run by $\mathcal{P}$ to produce a π^c for $\mathcal{V}$.

- PoSt.Verify$(\mathcal{S}_\mathcal{V}, c, t, \pi^c) \rightarrow \{0, 1\}$, which checks whether a proof is correct. PoSt.Verify is run by $\mathcal{V}$ and convinces $\mathcal{V}$ whether $\mathcal{P}$ has been storing $\mathcal{D}$ for some time t.

6.4 Practical PoRep and PoSt

In this thesis, we are interested in practical PoRep and PoSt constructions that can be deployed in existing systems and do not rely on trusted parties or hardware. We give a construction for PoRep (see *Seal-based Proof-of-Replication* in [14]) that requires a very slow sequential computation Seal to be performed during Setup to generate a replica. The protocol sketches for PoRep and PoSt are presented in Figure 6-2 and the underlying mechanism of the proving step in PoSt is illustrated in Figure 6-1, presented in the following pages.

6.4.1 Cryptographic building blocks

Collision-resistant hashing. We use a collision resistant hash function CRH : $\{0,1\}^* \rightarrow \{0,1\}^{O(\lambda)}$. We also use a collision resistant hash function MerkleCRH, which divides a string in multiple parts, construct a binary tree and recursively apply CRH and outputs the root.

zk-SNARKs. Our practical implementations of PoRep and PoSt rely on zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) [10, 17, 32]. Because zk-SNARKs are *succinct*, proofs are very short and easy to verify. More formally, let L be an NP language and C be a decision circuit for L. A trusted party conducts a one-time setup phase that results in two public keys: a proving key pk and a verification key vk. The proving key

pk enables any (untrusted) prover to generate a proof π attesting that $x \in L$ for an instance x of her choice. The non-interactive proof π is both *zero-knowledge* and *proof-of-knowledge*. Anyone can use the verification key vk to verify the proof π; in particular zk-SNARK proofs are publicly verifiable: anyone can verify π, without interacting with the prover that generated π. The proof π has constant size and can be verified in time that is linear in $|x|$.

A zk-SNARK for circuit satisfiability is a triple of polynomial-time algorithms

$$(\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$$

- $\mathsf{KeyGen}(1^\lambda, C) \rightarrow (\mathsf{pk}, \mathsf{vk})$. On input security parameter λ and a circuit C, KeyGen probabilistically samples pk and vk. Both keys are published as public parameters and can be used to prove/verify membership in L_C.

- $\mathsf{Prove}(\mathsf{pk}, x, w) \rightarrow \pi$. On input pk and input x and witness for the NP-statement w, the *prover* Prove outputs a non-interactive proof π for the statement $x \in L_C$.

- $\mathsf{Verify}(\mathsf{vk}, x, \pi) \rightarrow \{0, 1\}$. On input vk, an input x, and a proof π, the *verifier* Verify outputs 1 if $x \in L_C$.

We refer the interested reader to [10, 17, 32] for formal presentation and implementation of zk-SNARK systems. Generally these systems require the KeyGen operation to be run by a trusted party; novel work on Scalable Computational Integrity and Privacy (SCIP) systems [9] shows a promising direction to avoid this initial step, hence the above trust assumption.

6.4.2 Seal operation

The generation of a replica guarantees that the prover is dedicating some space to store a unique encoding of the original data. During the Setup, the prover runs a Seal operation. The role of the Seal operation is to (1) prevent sybil attacks by forcing replicas to be physically independent copies by requiring provers to store a pseudo-random permutation of $\mathcal{D}$ unique to their public key and (2) prevent outsourcing and generation attacks by forcing the generation

of the replica during PoRep.Setup to take substantially longer than the time expected for responding to a challenge.

For a more formal definition of the Seal operation see [14].

Intuition for preventing the *Sybil Attack*. We define a *replica* of $\mathcal{D}$ to be an encoding using a per-replica encoding key ek: $\mathcal{R}_{ek}^{\mathcal{D}} \leftarrow \mathsf{Seal}_{\mathsf{ek}}(\mathcal{D})$. Replicas are different from each other and indistinguishable from randomness. Provers must be storing the replica of the data in order to generate valid proofs of storage. A malicious attacker cannot pretend to store more data than what their capacity allows, mitigating in this way Sybil attacks.

Intuition for preventing the *Outsourcing* and *Generation Attacks*. We ensure that provers cannot get the replica just-in-time by retrieving data from outsourced storage. To achieve this, we force malicious provers to be distinguishably slower than honest provers when responding to a challenge. Computing $\mathsf{Seal}_{\mathsf{ek}}(\mathcal{D})$ must take a distinguishable amount of time, such that a verifier, $\mathcal{V}$, can distinguish between:

(a) $\mathsf{Time}(\mathsf{PoRep.Prove}(\mathcal{S}_{\mathcal{P}}, \mathcal{R}_{ek}^{\mathcal{D}}, c))$

(b) $\mathsf{Time}(\mathsf{PoRep.Prove}(\mathcal{S}_{\mathcal{P}}, \mathsf{Seal}_{\mathsf{ek}}(\mathcal{D}), c))$

We require that the wall clock running time of $\mathcal{P}$ computing $\mathsf{Seal}_{\mathsf{ek}}(\mathcal{D})$ must be noticeable. Statistical estimation of the round-trip time between $\mathcal{P}$ and $\mathcal{V}$, and PoRep.Prove to deter

The above operation can be realized with $\mathsf{Seal}^{\tau}_{\mathsf{AES-256}}$, and τ such that $\mathsf{Seal}^{\tau}_{\mathsf{AES-256}}$ takes 10-100x longer than the honest challenge-prove-verify sequence. Note that it is important to choose τ such that running $\mathsf{Seal}^{\tau}_{\mathsf{BC}}$ is distinguishably more expensive than running Prove with random access to $\mathcal{R}$.

6.4.3 Practical PoRep construction

This section describes the construction of the PoRep protocol and includes a simplified protocol sketch in Figure 6-2; implementation and optimization details are omitted.

1. **Creating a Replica**: The Setup algorithm generates a replica via the Seal operation and a proof that it was correctly generated. The prover generates the replica and sends the outputs (excluding $\mathcal{R}$) to the verifier.

2. **Proving Storage**: The Prove algorithm generates a proof of storage for the replica. The prover receives a random challenge, c, from the verifier, which determines a specific leaf $\mathcal{R}_c$ in the Merkle tree of $\mathcal{R}$ with root rt; the prover generates a proof of knowledge about $\mathcal{R}_c$ and its Merkle path leading up to rt.

3. **Verifying the Proofs**: The Verify algorithm checks the validity of the proofs of storage given the Merkle root of the replica and the hash of the original data. Proofs are publicly verifiable: nodes in the distributed system maintaining the ledger and clients interested in particular data can verify these proofs.

6.4.4 Practical PoSt construction

This section describes the construction of the PoSt protocol and includes a simplified protocol sketch in Figure 6-2; implementation and optimization details are omitted. The Setup and Verify algorithm are equivalent to the PoRep construction, hence we describe here only Prove.

Proving space and time. The Prove algorithm generates a *Proof-of-Spacetime* for the replica. The prover receives a random challenge from the verifier and generate *Proofs-of-*

Replication in sequence, using the output of a proof as an input of the other for a specified amount of iterations t (see Figure 6-1).

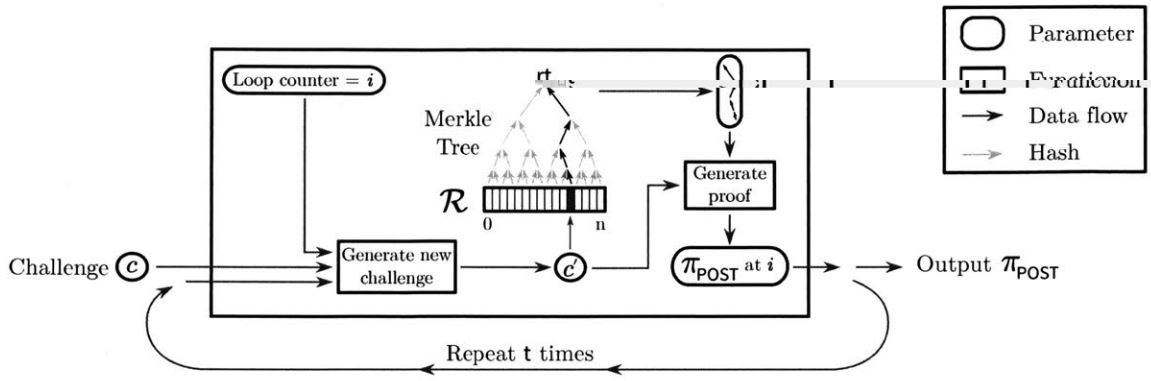


Figure 6-1: Illustration of the underlying mechanism of PoSt.Prove showing the iterative proof to demonstrate storage over time.

6.5 Usage in Filecoin

Filecoin is a protocol for incentivizing file storage. Storage providers must able to prove they are storing their clients' data in order to receive a payment. The Filecoin protocol employs *Proof-of-Spacetime* to audit the storage offered by storage providers, in order to overcome the previous attacks. To use PoSt in Filecoin, we modify our scheme to be non-interactive since there is no designated verifier, and we want any member of the network to be able to verify. Since our verifier runs in the public-coin model, we can extract randomness from the blockchain to issue challenges.

Filecoin PoRep protocol

Setup

- INPUTS:
  - prover key pair $(\mathsf{pk}_{\mathcal{P}}, \mathsf{sk}_{\mathcal{P}})$
  - prover SEAL key $\mathsf{pk}_{\mathsf{SEAL}}$
  - data $\mathcal{D}$
- OUTPUTS: replica $\mathcal{R}$, Merkle root rt of $\mathcal{R}$, proof π_{SEAL}

1) Compute $h_{\mathcal{D}} := \mathsf{CRH}(\mathcal{D})$
2) Compute $\mathcal{R} := \mathsf{Seal}^{\tau}(\mathcal{D}, \mathsf{sk}_{\mathcal{P}})$
3) Compute $\mathsf{rt} := \mathsf{MerkleCRH}(\mathcal{R})$
4) Set $\vec{x} := (\mathsf{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathsf{rt})$
5) Set $\vec{w} := (\mathsf{sk}_{\mathcal{P}}, \mathcal{D})$
6) Compute $\pi_{\mathsf{SEAL}} := \mathsf{SCIP.Prove}(\mathsf{pk}_{\mathsf{SEAL}}, \vec{x}, \vec{w})$
7) Output $\mathcal{R}$, rt, π_{SEAL}

Prove

- INPUTS:
  - prover *Proof-of-Storage* key $\mathsf{pk}_{\mathsf{POS}}$
  - replica $\mathcal{R}$
  - random challenge c
- OUTPUTS: a proof π_{POS}

1) Compute $\mathsf{rt} := \mathsf{MerkleCRH}(\mathcal{R})$
2) Compute path := Merkle path from rt to leaf $\mathcal{R}_c$
3) Set $\vec{x} := (\mathsf{rt}, c)$
4) Set $\vec{w} := (\mathsf{path}, \mathcal{R}_c)$
5) Compute $\pi_{\mathsf{POS}} := \mathsf{SCIP.Prove}(\mathsf{pk}_{\mathsf{POS}}, \vec{x}, \vec{w})$
6) Output π_{POS}

Verify

- INPUTS:
  - prover public key, $\mathsf{pk}_{\mathcal{P}}$
  - verifier SEAL and POS keys $\mathsf{vk}_{\mathsf{SEAL}}$, $\mathsf{vk}_{\mathsf{POS}}$
  - hash of data $\mathcal{D}$, $h_{\mathcal{D}}$
  - Merkle root of replica $\mathcal{R}$, rt
  - random challenge, c
  - tuple of proofs, $(\pi_{\mathsf{SEAL}}, \pi_{\mathsf{POS}})$
- OUTPUTS: bit b, equals 1 if proofs are valid

1) Set $\vec{x_1} := (\mathsf{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathsf{rt})$
2) Compute $b_1 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{SEAL}}, \vec{x_1}, \pi_{\mathsf{SEAL}})$
3) Set $\vec{x_2} := (\mathsf{rt}, c)$
4) Compute $b_2 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{POS}}, \vec{x_2}, \pi_{\mathsf{POS}})$
5) Output $b_1 \wedge b_2$

Filecoin PoSt protocol

Setup

- INPUTS:
  - prover key pair $(\mathsf{pk}_{\mathcal{P}}, \mathsf{sk}_{\mathcal{P}})$
  - prover POST key pair $\mathsf{pk}_{\mathsf{POST}}$
  - some data $\mathcal{D}$
- OUTPUTS: replica $\mathcal{R}$, Merkle root rt of $\mathcal{R}$, proof π_{SEAL}

1) Compute $\mathcal{R}$, rt, π_{SEAL} := PoRep.Setup($\mathsf{pk}_{\mathcal{P}}$, $\mathsf{sk}_{\mathcal{P}}$, $\mathsf{pk}_{\mathsf{SEAL}}$, $\mathcal{D}$)
2) Output $\mathcal{R}$, rt, π_{SEAL}

Prove

- INPUTS:
  - prover PoSt key $\mathsf{pk}_{\mathsf{POST}}$
  - replica $\mathcal{R}$
  - random challenge c
  - time parameter t
- OUTPUTS: a proof π_{POST}

1) Set $\pi_{\mathsf{POST}} := \perp$
2) Compute $\mathsf{rt} := \mathsf{MerkleCRH}(\mathcal{R})$
3) For $i = 0...t$:
   a) Set $c' := \mathsf{CRH}(\pi_{\mathsf{POST}} || c || i)$
   b) Compute $\pi_{\mathsf{POS}} := \mathsf{PoRep.Prove}(\mathsf{pk}_{\mathsf{POS}}, \mathcal{R}, c')$
   c) Set $\vec{x} := (\mathsf{rt}, c, i)$
   d) Set $\vec{w} := (\pi_{\mathsf{POS}}, \pi_{\mathsf{POST}})$
   e) Compute $\pi_{\mathsf{POST}} := \mathsf{SCIP.Prove}(\mathsf{pk}_{\mathsf{POST}}, \vec{x}, \vec{w})$
4) Output π_{POST}

Verify

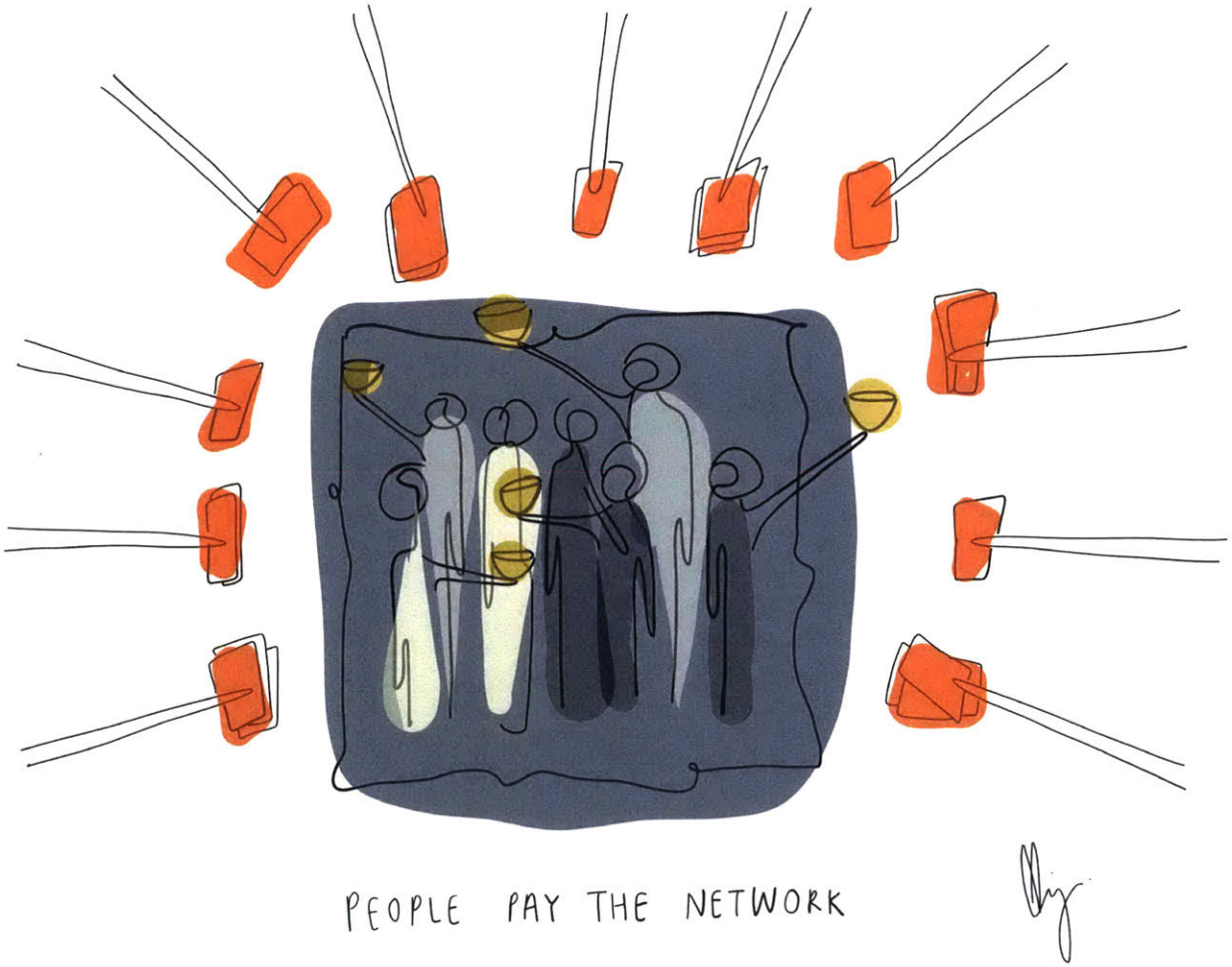
- INPUTS:
  - prover *public key* $\mathsf{pk}_{\mathcal{P}}$
  - verifier SEAL and POST keys $\mathsf{vk}_{\mathsf{SEAL}}$, $\mathsf{vk}_{\mathsf{POST}}$
  - hash of some data $h_{\mathcal{D}}$
  - Merkle root of some replica rt
  - random challenge c
  - time parameter t
  - tuple of proofs $(\pi_{\mathsf{SEAL}}, \pi_{\mathsf{POST}})$
- OUTPUTS: bit b, equals 1 if proofs are valid

1) Set $\vec{x_1} := (\mathsf{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathsf{rt})$
2) Compute $b_1 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{SEAL}}, \vec{x_1}, \pi_{\mathsf{SEAL}})$
3) Set $\vec{x_2} := (\mathsf{rt}, c, t)$
4) Compute $b_2 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{POST}}, \vec{x_2}, \pi_{\mathsf{POST}})$
5) Output $b_1 \wedge b_2$

Figure 6-2: *Proof-of-Replication* and *Proof-of-Spacetime* protocol sketches. Here CRH denotes a collision-resistant hash, $\vec{x}$ is the NP-statement to be proven, and $\vec{w}$ is the witness.

Chapter 7

Incentivizing File Storage



In this chapter, we give an overview of *Filecoin*. The following work has been presented in the Filecoin whitepaper [12] by Juan Benet and I in July 2017. Filecoin is a decentralized storage network that is *auditable*, *publicly verifiable* and designed on *incentives*. Clients pay a network of storage providers for data storage and retrieval and storage providers offer disk space and bandwidth in exchange of payments. Storage providers receive their payments only if the network can *audit* that their service was correctly provided. Filecoin does not require a new blockchain and can be implemented as a smart contract on top of existing blockchains. However, in Section 9.1, we show how we can create a useful *Proof-of-Work* based on storage, which will require a new blockchain layer.

- The Filecoin protocol is a *Decentralized Storage Network* designed on a blockchain and with a native protocol token (also called Filecoin). Clients spend tokens for storing and retrieving data and storage providers earn tokens by storing and serving data.
- The Filecoin DSN handle storage and retrieval requests respectively via two *verifiable markets*: the Storage Market and the Retrieval Market. The token is a market token and can be used by clients and storage providers to participate to the markets. Clients and storage providers set the prices for the services requested and offered and submit their orders to the markets.
- The markets are operated by the Filecoin network which employs *Proof-of-Spacetime* and *Proof-of-Replication* to guarantee that storage providers have correctly stored the data they committed to store.

7.1 Blockchain-based DSN

We personify all the users that run the Filecoin software as one single abstract entity: *The Network*. The Network acts as an intermediary that runs the Manage protocol; informally, at every new block in the Filecoin blockchain, full nodes manage the available storage, validate pledges, audit the storage proofs, and repair possible faults.

Ledger, $\mathcal{L}$. Our protocol is applied on top of a ledger-based currency; for generality we refer to this as the *Ledger*, $\mathcal{L}$. At any given time t (referred to as *epoch*), all users have access to $\mathcal{L}_t$, the ledger at epoch t, which is a sequence of *transactions*. The ledger is append-only[1]. The Filecoin DSN protocol can be implemented on any ledger that allows for the verification of Filecoin's proofs; we show how we can construct a ledger based on *useful work* in Section 9.1.

7.2 Participants

Any user can participate as a *Client*, a *Storage Miner*, and/or a *Retrieval Miner*.

- *Clients* pay to store data and to retrieve data in the DSN, via Put and Get requests.

- *Storage Miners* provide data storage to the network. Storage Miners participate in Filecoin by offering their disk space and serving Put requests. To become Storage Miners, users must *pledge* their storage by depositing collateral proportional to it. Storage Miners respond to Put requests by committing to store the client's data for a specified time. Storage Miners generate *Proofs-of-Spacetime* and submit them to the blockchain to prove to the Network that they are storing the data through time. In case of invalid or missing proofs, Storage Miners are penalized and loose part of their collateral. Storage Miners are also eligible to mine new blocks, and in doing so they hence receive the mining reward for creating a block and transaction fees for the transactions included in the block.

- *Retrieval Miners* provide data retrieval to the Network. Retrieval Miners participate in Filecoin by serving data that users request via Get. Unlike Storage Miners, they are not required to pledge, commit to store data, or provide proofs of storage. It is natural for Storage Miners to also participate as Retrieval Miners. Retrieval Miners can obtain pieces directly from clients, or from the Retrieval Market.

[1] $t < t'$ implies that $\mathcal{L}_t$ is a prefix of $\mathcal{L}'_t$

7.3 Markets

Demand and supply of storage meet at the two Filecoin Markets: Storage Market and Retrieval Market. In brief, clients and miners set the prices for the services they request or provide by submitting orders to the respective markets. The exchanges provide a way for clients and miners to see matching offers and initiate deals. By running the Manage protocol, the network guarantees that miners are rewarded and clients are charged if the service requested has been successfully provided. The detailed Storage Market's protocol is presented in Section 8.3 and the Retrieval Market's protocol in Section 8.4.

7.3.1 The Storage Market

The Storage Market is a *verifiable market* which allows clients (i.e. buyers) to request storage for their data and Storage Miners (i.e. sellers) to offer their storage.

We design the Storage Market protocol accordingly to the following requirements:

- **In-chain orderbook**: It is important that: (1) Storage Miners orders are public, so that the lowest price is always known to the network and clients can make informed decision on their orders, (2) client orders must be always submitted to the orderbook, even when they accept the lowest price, in this way the market can react to the new offer. Hence, we require orders to be added in clear to the Filecoin blockchain in order to be added to the orderbook.

- **Participants committing their resources**: We require both parties to commit to their resources as a way to avoid disservice: to avoid Storage Miners not providing the service and to avoid clients not having available funds. In order to participate to the Storage Market, Storage Miners must pledge, depositing a collateral proportional to their amount of storage in DSN (see Section 7.4.3 for more details). In this way, the Network can penalize Storage Miners that do not provide proofs of storage for the pieces they committed to store. Similarly, clients must deposit the funds specified in the order, guaranteeing in this way commitment and availability of funds during settlement.

one honest Retrieval Miner. The idea behind such gradual release protocols is that a party will only have a minimal advantage if she decides to cheat.

- **Payments channels**: Clients are interested in retrieving the pieces as soon as they submit their payments, Retrieval Miners are interested in only serving the pieces if they are sure of receiving a payment. Validating payments via a public ledger can be the bottleneck of a retrieval request, hence we must rely on efficient off-chain payments. The Filecoin blockchain must support payment channels which enable rapid, optimistic transactions and use the blockchain only in case of disputes. In this way, Retrieval Miners and Clients can quickly send the small payments required by our protocol. Future work includes the creation of a network of payment channels as previously seen in [47, 55].

7.4 Protocol Overview

In this section, we give an overview of the Filecoin DSN by describing the operations performed by the clients, the miners and the other nodes in the network. A detailed presentation of the protocol is found in Section 8. A sketch of the Filecoin protocol, using nomenclature defined later within this thesis, is shown in Figure 8-1 accompanied with an illustration in Figure 7-1.

7.4.1 Client Cycle

Below, we give a brief overview of the client cycle. An in-depth explanation of the following protocols is given in Section 8.3 and 8.4.

1. **Put**: *Client stores data in Filecoin.*

   Clients can store their data by paying Storage Miners in Filecoin tokens. The Put protocol is described in detail in Section 8.2.

   A client initiates the Put protocol by submitting a *bid* order to the Storage Market orderbook (by submitting their order to the blockchain). When a matching *ask* order

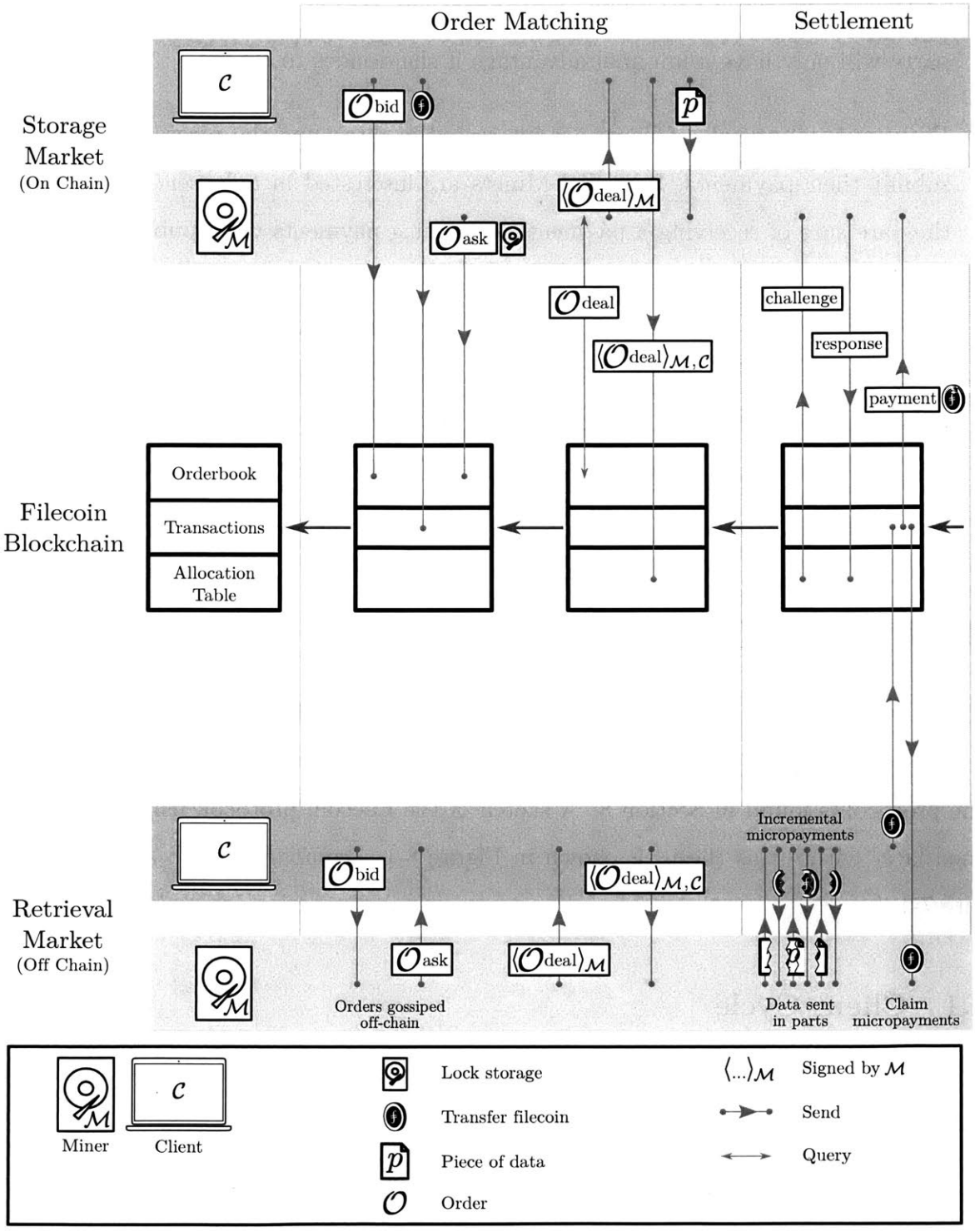


Figure 7-1: Illustration of the Filecoin Protocol, showing an overview of the Client-Miner interactions. The Storage and Retrieval Markets shown above and below the blockchain, respectively, with time advancing from the Order Matching phase on the left to the Settlement phase on the right. Note that before micropayments can be made for retrieval, the client must lock the funds for the microtransaction. Diagram by Dr. Evan Miyazono from [12].

from miners is found, the client sends the piece to the miner. Both parties sign a *deal* order and submit it to the Storage Market orderbook.

Clients should be able to decide the amount of physical replicas of their pieces either by submitting multiple orders (or specifying a replication factor in the order). Higher redundancy results in a higher tolerance of storage faults.

2. **Get**: *Client retrieves data from Filecoin.*

   Clients can retrieve any data stored in the DSN by paying Retrieval Miners in Filecoin tokens. The Get protocol is described in detail in Section 7.3.2.

   A client initiates the Get protocol by submitting a *bid* order to the Retrieval Market orderbook (by gossiping their order to the network). When a matching *ask* order from miners is found, the client receives the piece from the miner. When received, both parties sign a *deal* order and submit it to the blockchain to confirm that the exchange succeeded.

7.4.2 Mining Cycle (for Storage Miners)

Below, we give an informal overview of the mining cycle.

1. **Pledge**: *Storage Miners pledge to provide storage to the Network.*

   Storage Miners pledge their storage to the network by depositing collateral via a *pledge* transaction in the blockchain, via Manage.PledgeSector. The collateral is deposited for the time intended to provide the service, and it is returned if the miner generates proofs of storage for the data they commit to store. If some proofs of storage fail, a proportional amount of collateral is lost.

   Once the pledge transaction appears in the blockchain, miners can offer their storage in the Storage Market: they set their price and add an ask order to the market's orderbook.

2. **Receive Orders**: *Storage Miners get storage requests from the Storage Market.*

Once the pledge transaction appears in the blockchain (hence in the AllocTable), miners can offer their storage in the Storage Market: they set their price and add an ask order to the market's orderbook via Put.AddOrders.

Check if their orders are matched with a corresponding *bid* order from a client, via Put.MatchOrders.

Once orders are matched, clients send their data to the Storage Miners. When receiving the piece, miners run Put.ReceivePiece. When the data is received, both the miner and the client sign a *deal* order and submit it to the blockchain.

3. **Seal**: *Storage Miners prepare the pieces for future proofs.*

   Storage Miners' storage is divided in sectors, each sector contains pieces assigned to the miner. The Network keeps track of each Storage Miners' sector via the allocation table. When a Storage Miner sector is filled, the sector is *sealed.* Sealing is a slow, sequential operation that transforms the data in a sector into a replica, a unique physical copy of the data that is associated to the public key of the Storage Miner. Sealing is a necessary operation during the *Proof-of-Replication* as described in Section 6.4.

4. **Prove**: *Storage Miners prove they are storing the committed pieces.*

   When Storage Miners are assigned data, they must repeatedly generate proofs of replication to guarantee they are storing the data (for more details, see Section 6). Proofs are posted on the blockchain and the Network verifies them.

7.4.3 Mining Cycle (for Retrieval Miners)

Below, we give an informal overview of the mining cycle for Retrieval Miners.

1. **Receive Orders**: *Retrieval Miners get data requests from the Retrieval Market*

   Retrieval Miners announce their pieces by gossiping their *ask* orders to the network: they set their price and add an ask order to the market's orderbook.

   Then, Retrieval Miners check if their orders are matched with a corresponding *bid* order from a client.

2. **Send**: *Retrieval Miners send pieces to the client.*

   Once orders are matched, Retrieval Miners send the piece to the client (see Section 7.3.2 for details). When the piece is received, both the miner and the client sign a *deal* order and submit it to the blockchain.

7.4.4 Network Cycle

We give an informal overview of the operations run by the network.

1. **Assign**: *The Network assigns clients' pieces to Storage Miners' sectors.*

   Clients initiate the Put protocol by submitting a bid order in the Storage Market[2].

   When ask and bid orders match, the involved parties jointly commit to the exchange and submit a *deal* order in the market. At this point, the Network *assigns* the data to the miner and makes a note of it in the allocation table.

2. **Repair**: *The Network finds faults and attempt to repair them.*

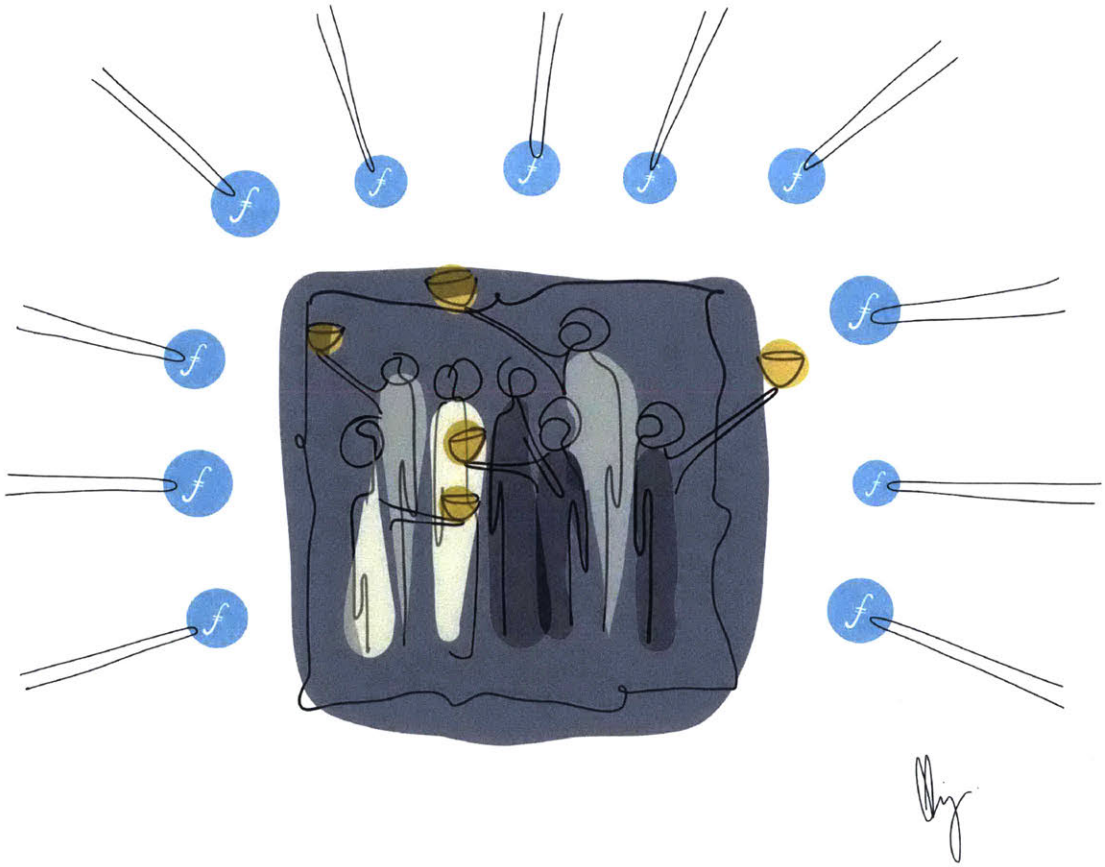
   All the storage allocations are public to every participant in the network. At every block, the Network checks if the required proofs for each assignment are present, checks that they are valid, and acts accordingly:

   - if any proof is missing or invalid, the network penalizes the Storage Miners by taking part of their collateral,
   - if a large amount of proofs are missing or invalid (defined by a system parameter Δ_{fault}), the network considers the Storage Miner *faulty*, settles the order as failed and reintroduces a new order for the same piece into the the market,
   - if every Storage Miner storing this piece is faulty, then the piece is lost and the client gets refunded.

[2]Storage orders are submitted via the blockchain, see Section 8.3.

Chapter 8

Filecoin Protocol



In this chapter, the final Filecoin protocol is presented in details, describing the data structures, the methods of the Get, Put and Manage protocol, and the protocols for the Storage and Retrieval Markets. The overall Filecoin Protocol is presented in Figure 8-1.

8.1 Data Structures

The Filecoin DSN data structures are described in this section and presented in details in Figure 8-2.

8.1.1 Pieces

A *piece* is some part of data that a client is storing in the DSN. For example, data can be deliberately divided into many pieces and each piece can be stored by a different set of Storage Miners.

8.1.2 Sectors

A *sector* is some disk space that a Storage Miner provides to the network. Miners store pieces from clients in their sectors and earn tokens for their services. In order to store pieces, Storage Miners must pledge their sectors to the network.

8.1.3 Allocation Table

The AllocTable is a data structure that keeps track of pieces and their assigned sectors. The AllocTable is updated at every block in the ledger and its Merkle root is stored in the latest block. Every node in the network will validate the updates to the table when receiving a new block. In practice, the table is used to keep the state of the DSN, allowing for quick look-ups during proof verification. For more details, see Figure 8-2.

8.1.4 Pledge

A *pledge* is a commitment to offer storage (specifically a *sector*) to the network. Storage Miners must submit their pledge to the ledger in order to start accepting orders in the Storage

Filecoin Protocol

Network

at each epoch t in the ledger $\mathcal{L}$:

1. for each new block:
   (a) check if the block is in the valid format
   (b) check if all transactions are valid
   (c) check if all orders are valid
   (d) check if all proofs are valid
   (e) check if all pledges are valid
   (f) discard block, if any of the above fails
2. for each new order $\mathcal{O}$ introduced in t
   (a) add $\mathcal{O}$ to the Storage Market's orderbook.
   (b) if $\mathcal{O}$ is a *bid*: lock $\mathcal{O}$.funds
   (c) if $\mathcal{O}$ is an *ask*: lock $\mathcal{O}$.space
   (d) if $\mathcal{O}$ is a *deal*: run Put.AssignOrders
3. for each $\mathcal{O}$ in the Storage Market's orderbook:
   (a) check if $\mathcal{O}$ has expired (or canceled):
   - remove $\mathcal{O}$ from the orderbook
   - return unspent $\mathcal{O}$.funds
   - free $\mathcal{O}$.space from AllocTable

   (b) if $\mathcal{O}$ is a *deal*, check if the expected proofs exist by running Manage.RepairOrders:
   - if one missing, penalize the $\mathcal{M}$'s pledge collateral
   - if proofs are missing for more than Δ_{fault} epochs, cancel order and re-introduce it to the market
   - if the piece cannot be retrieved and reconstructed from the network, cancel order and re-fund the client

Client

at any time:

1. submit new storage orders via Put.AddOrders
   (a) find matching orders via Put.MatchOrders
   (b) send file to the matched miner $\mathcal{M}$
2. submit new retrieval orders via Get.AddOrders
   (a) find matching orders via Get.MatchOrders
   (b) create a payment channel with $\mathcal{M}$

on receiving $\mathcal{O}_{\text{deal}}$ from Storage Miners $\mathcal{M}$

1. sign $\mathcal{O}_{\text{deal}}$
2. submit it to the blockchain via Put.AddOrders

on receiving (p_i) from Retrieval Miners $\mathcal{M}$:

1. verify that
2. send a micropayment to $\mathcal{M}$

Storage Miner

at any time:

1. renew expired pledges via Manage.PledgeSector
2. pledge new storage via Manage.PledgeSector
3. submit a new ask order via Put.AddOrder

at each epoch t:

1. for each $\mathcal{O}_{\text{ask}}$ in the orderbook:
   (a) find matched orders via Put.MatchOrders
   (b) start a new deal by contacting the matching client
2. for each sector pledged:
   (a) generate proof of storage via Manage.ProveSector
   (b) if time to post the proof (every Δ_{proof} epochs), submit it to the blockchain

on receiving piece p from client $\mathcal{C}$:

1. check if the piece is of the size specified in the order $\mathcal{O}_{\text{bid}}$
2. create $\mathcal{O}_{\text{deal}}$ and sign it and send it to $\mathcal{C}$
3. store the piece in a sector
4. if the sector is full, run Manage.SealSector

Retrieval Miner

at any time:

1. gossip *ask* orders to the network
2. listen to *bid* orders from the network

on retrieval request from $\mathcal{C}$:

1. start payment channel with $\mathcal{C}$
2. split data in multiple parts
3. only send parts if payments are received

Figure 8-1: Diagram of the Filecoin Protocol.

Market. A pledge consists of the size of the pledged sector and the collateral deposited by the Storage Miner (see Figure 8-2 for more details).

8.1.5 Orders and Orderbooks

An *order* is a statement of intent to request or offer a service. Clients submit *bid* orders to the markets to request a service (resp. Storage Market for storing data and Retrieval Market for retrieving data) and Miners submit *ask* orders to offer a service. The order data structures are shown in Figure 8-2.

Each market has an Order Book. The Storage Market's Order Book is stored *in-chain* and everyone has a shared view on all the orders (see "Exchange" in Section 4.5.1). The Retrieval Market's Order Book is not stored in-chain. Every node has a partial (and weakly consistent) view of the orders in the network.

Storage Market Orders

***bid* order**
$\mathcal{O}_{\mathsf{bid}} := \langle \mathsf{size}, \mathsf{funds}[, \mathsf{price}, \mathsf{time}, \mathsf{coll}, \mathsf{coding}] \rangle_{\mathcal{C}_i}$

- size, the size of the piece to be stored
- funds, the total amount that client $\mathcal{C}_i$ is depositing
- time, the maximum epoch time for which the file should be stored[a]
- price, the spacetime price in Filecoin[b]
- coll, the collateral specific to this piece that the miner is required to deposit
- coding, the erasure coding scheme for this piece

***ask* order**
$\mathcal{O}_{\mathsf{ask}}$: $\langle$ space, price $\rangle_{\mathcal{M}_i}$

- space, amount of space Storage Miner $\mathcal{M}_i$ is providing in the order
- price, the spacetime price in Filecoin

***deal* order**
$\mathcal{O}_{\mathsf{deal}}$: $\langle$ ask, bid, ts, hash $\rangle_{\mathcal{C}_i, \mathcal{M}_j}$

- ask, a cryptographic reference to $\mathcal{O}_{\mathsf{ask}}$ from $\mathcal{C}_i$
- order, a cryptographic reference to $\mathcal{O}_{\mathsf{bid}}$ from $\mathcal{M}_i$
- ts, timestamp epoch in which the order has been signed by $\mathcal{M}_i$
- hash cryptographic hash of the piece that $\mathcal{M}_j$ will store

[a]If not specified, the piece will be stored until expiration of funds.

[b]If not specified, when a Storage Miner is faulty, the network can re-introduce the order at the current host price

Retrieval Market Orders

***bid* order**
$\mathcal{O}_{\mathsf{bid}}$: $\langle$ piece, price $\rangle_{\mathcal{C}_i}$

- piece, the index of the piece requested[a]
- price, the price at which $\mathcal{C}_i$ is paying for one retrieval

***ask* order**
$\mathcal{O}_{\mathsf{ask}}$: $\langle$ piece, price $\rangle_{\mathcal{M}_i}$

- piece, the index of the piece requested
- price, the price at which $\mathcal{M}_j$ is serving the piece for

***deal* order**
$\mathcal{O}_{\mathsf{deal}}$: $\langle$ ask, order $\rangle_{\mathcal{C}_i, \mathcal{M}_j}$

- ask, a cryptographic reference to $\mathcal{O}_{\mathsf{ask}}$ from $\mathcal{C}_i$
- order, a cryptographic reference to $\mathcal{O}_{\mathsf{ask}}$ from $\mathcal{C}_i$

[a]Only pieces stored in Filecoin can be requested

DSN Data Structures

Pledge
pledge $:= \langle \mathsf{size}, \mathsf{coll} \rangle_{\mathcal{M}_i}$

- size, the size of the sector being pledged.
- coll, the collateral specific to this pledge that $\mathcal{M}_i$ deposits.

Orderbook
OrderBook: $(\mathcal{O}^1..\mathcal{O}^n)$

8.2 DSN Protocol Specifications

In this section, we introduce the three protocols (Put, Get and Manage) and their internal operations. In Figure 8-3 and 8-4, we show a possible implementation in pseudocode.

8.2.1 Put Protocol

The Put protocol consists of four algorithms: AddOrders, MatchOrders, SendPiece, ReceivePiece.

- AddOrders: Both Clients and Storage Miners run this operation to add orders to the Storage Market. This submits input orders to the ledger via a special transaction and on success, it adds the order to the Order Book.
- MatchOrders: Both Clients and Storage Miners run this operation to find matching orders, by querying the Order Book to find a matching order to an input order.
- SendPiece: Clients run this operation to send a piece from a *bid* order to the Storage Miner of the matching *ask* order. This sends the piece and the reference to a specific *bid* and *ask* orders and outputs a signed *deal* order from the miner.
- ReceivePiece: Storage Miners run this operation when receiving a piece from a Client. On receiving a piece, a *bid* and an *ask* order, this checks the validity of the orders and of the piece. On success, it signs a *deal* order which includes a reference to the *bid* and *ask* orders and the hash of the received piece.

8.2.2 Get Protocol

Similarly to the Put protocol, the Get protocol consists of four algorithms: AddOrders, MatchOrders, SendPiece, ReceivePiece.

- AddOrders: Both Clients and Retrieval Miners run this operation to add orders to the Retrieval Market. This gossips the input orders to the network.
- MatchOrders: Both Clients and Retrieval Miners run this operation to find matching orders, by querying the Order Book to find a matching order to an input order.

- ReceivePiece: Clients run this operation to request a piece from a Retrieval Miner. First, it creates a *deal* order for the retrieval and sets up a micropayment channel. Then, it sends small micropayments to the miner, receives small parts of the piece, validates the piece received and continues until the entire piece has been received.

- SendPiece: Retrieval Miners run this operation to send pieces to Clients. On receiving retrieval requests, this operation sets up a micropayment channel and sends a small part of the piece for each micropayment it receives, until the interacting Client stops sending payments.

8.2.3 Manage Protocol

- PledgeSector: Storage Miners run this operation to pledge a sector. This creates a special transaction announcing the amount of storage the miner is introducing and deposits a collateral proportional to the offered storage. On success, the new sector is added to the shared allocTable.

- SealSector: Storage Miners run this operation to seal a sector. This runs the PoSt.Setup that has a subroutine the slow Seal function. This outputs a special encoding, the *replica*, of the sector, the Merkle root hash of the replica and a proof that the encoding and its hash have been computer correctly.

- ProveSector: Storage Miners run this operation to prove they are storing their assigned data. This runs a PoSt.Prove on the replica and outputs a proof of storage.

- AssignOrders: The nodes in the network run this operation to register new *deal* orders to the allocation table. Given *deal* orders, this validates them, updates the allocTable and outputs it.

- RepairOrders: The nodes in the network run this operation to verify that all the proofs on the blockchain are valid and that no order has been missing proofs. For each entry in the allocTable, this verifies if proofs exists and if they are valid; if they are not, they are counted as missing and their assigned Storage Providers are penalized. If more than Λ faults exist, the order is canceled and re-introduced in the order book.

Put Protocol

Market

AddOrders

- INPUTS: list of orders $\mathcal{O}^1..\mathcal{O}^n$
- OUTPUTS: bit b, equals 1 if successful

1) Set $\mathsf{tx}_{\mathsf{order}} := (\mathcal{O}^1, .., \mathcal{O}^n)$
2) Submit $\mathsf{tx}_{\mathsf{order}}$ to $\mathcal{L}$
3) Wait for $\mathsf{tx}_{\mathsf{order}}$ to be included in $\mathcal{L}$
4) Output 1 on success, 0 otherwise

MatchOrders

- INPUTS:
  - the current Storage Market OrderBook
  - query order to match $\mathcal{O}^q$
- OUTPUTS: matching orders $\mathcal{O}^1..\mathcal{O}^n$

1) Match each $\mathcal{O}^i$ in OrderBook such that:
   a) If $\mathcal{O}^q$ is an *ask* order:
      i) Check if $\mathcal{O}^i$ is *bid* order
      ii) Check $\mathcal{O}^i$.price $\geq \mathcal{O}^q$.price
      iii) Check $\mathcal{O}^i$.size $\leq \mathcal{O}^q$.space
   b) If $\mathcal{O}^q$ is a *bid* order:
      i) Check if $\mathcal{O}^i$ is *ask* order
      ii) Check $\mathcal{O}^i$.price $\geq \mathcal{O}^q$.price
      iii) Check $\mathcal{O}^i$.space $\geq \mathcal{O}^q$.size
2) Output matched orders $\mathcal{O}^1...\mathcal{O}^n$

Exchange

SendPiece

- INPUTS:
  - an *ask* order $\mathcal{O}_{\mathsf{ask}}$
  - a *bid* order $\mathcal{O}_{\mathsf{bid}}$
  - a piece p
- OUTPUTS: a *deal* order $\mathcal{O}_{\mathsf{deal}}$ signed by $\mathcal{M}_i$

1) Get identity of $\mathcal{M}_i$ from $\mathcal{O}_{\mathsf{ask}}$ signature
2) Send ($\mathcal{O}_{\mathsf{ask}}$,$\mathcal{O}_{\mathsf{bid}}$,$p$) to $\mathcal{M}_i$
3) Receive $\mathcal{O}_{\mathsf{deal}}$ signed by $\mathcal{M}_i$
4) Check if $\mathcal{O}_{\mathsf{deal}}$ is valid according to Definition 8.3.1
5) Output $\mathcal{O}_{\mathsf{deal}}$

ReceivePiece

- INPUTS:
  - signing key for $\mathcal{M}_j$.
  - current orderbook OrderBook
  - *ask* order $\mathcal{O}_{\mathsf{ask}}$
  - *bid* order $\mathcal{O}_{\mathsf{bid}}$
  - piece p
- OUTPUTS: *deal* order $\mathcal{O}_{\mathsf{deal}}$ signed by $\mathcal{C}_i$ and $\mathcal{M}_j$

1) Check if $\mathcal{O}_{\mathsf{bid}}$ is valid:
   a) Check if $\mathcal{O}_{\mathsf{bid}}$ is in OrderBook
   b) Check if $\mathcal{O}_{\mathsf{bid}}$ is not referenced by other active $\mathcal{O}_{\mathsf{deal}}$
   c) Check if $\mathcal{O}_{\mathsf{bid}}$.size is equal to $|p|$
   d) Check if $\mathcal{O}$ is signed by $\mathcal{M}_i$
2) Store p locally
3) Set $\mathcal{O}_{\mathsf{deal}} := \langle\ \mathcal{O}_{\mathsf{ask}},\ \mathcal{O}_{\mathsf{deal}},\ \mathcal{H}(p)\ \rangle_{\mathcal{M}_i}$
4) Get identity of $\mathcal{C}_j$ from $\mathcal{O}_{\mathsf{bid}}$
5) Send $\mathcal{O}_{\mathsf{deal}}$ to $\mathcal{C}_j$
6) Output $\mathcal{O}_{\mathsf{deal}}$

Get Protocol

Market

AddOrders

- INPUTS: list of orders $\mathcal{O}^1..\mathcal{O}^n$
- OUTPUTS: none

1) Gossip $\mathcal{O}^1..\mathcal{O}^n$ to the network

MatchOrders

- INPUTS:
  - the current Retrieval Market OrderBook
  - query order to match $\mathcal{O}^q$
- OUTPUTS: matching orders $\mathcal{O}^1..\mathcal{O}^n$

1) Match each $\mathcal{O}^i$ in OrderBook such that:
   a) Check $\mathcal{O}^i$.piece is equal to $\mathcal{O}^q$.piece
   b) If $\mathcal{O}^q$ is an *ask* order:
      i) Check if $\mathcal{O}^i$ is *bid* order
      ii) Check $\mathcal{O}^i$.price $\geq \mathcal{O}^q$.price
   c) If $\mathcal{O}^q$ is a *bid* order:
      i) Check if $\mathcal{O}^i$ is *ask* order
      ii) Check $\mathcal{O}^i$.price $\geq \mathcal{O}^q$.price
2) Output matched orders $\mathcal{O}^1...\mathcal{O}^n$

Exchange

SendPiece

- INPUTS:
  - an *ask* order $\mathcal{O}_{\mathsf{ask}}$
  - a *bid* order $\mathcal{O}_{\mathsf{bid}}$
  - a piece p
- OUTPUTS: a *deal* order $\mathcal{O}_{\mathsf{deal}}$ signed by $\mathcal{C}_i$

1) Create $\mathcal{O}_{\mathsf{deal}}$:
   a) Set $\mathcal{O}_{\mathsf{deal}}$.ask := $\mathcal{O}_{\mathsf{ask}}$
   b) Set $\mathcal{O}_{\mathsf{deal}}$.bid := $\mathcal{O}_{\mathsf{deal}}$
2) Get identity of $\mathcal{C}_i$ from $\mathcal{O}_{\mathsf{bid}}$ signature
3) Setup a micropayment channel with $\mathcal{C}_i$
4) For each block of data p_j of p:
   a) Set π_j to be a merkle path from $\mathcal{H}(p)$ to p_j
   b) Send ($\mathcal{O}_{\mathsf{deal}}$,$p_j$, π_j) to $\mathcal{C}_i$
   c) Receive $\langle\ \mathcal{O}_{\mathsf{deal}},\ \mathrm{j}\ \rangle_{\mathcal{C}_i}$
5) Output $\mathcal{O}_{\mathsf{deal}}$

ReceivePiece

- INPUTS:
  - a client's key $\mathcal{C}_j$
  - an *ask* order $\mathcal{O}_{\mathsf{ask}}$
  - a *bid* order $\mathcal{O}_{\mathsf{bid}}$
  - merkle tree hash of p in the orders h_p
- OUTPUTS: a piece p

1) Create $\mathcal{O}_{\mathsf{deal}}$:
   a) Set $\mathcal{O}_{\mathsf{deal}}$.ask := $\mathcal{O}_{\mathsf{ask}}$
   b) Set $\mathcal{O}_{\mathsf{deal}}$.bid := $\mathcal{O}_{\mathsf{bid}}$
2) Get identity of $\mathcal{M}_i$ from $\mathcal{O}_{\mathsf{ask}}$ signature
3) Set up a micropayment channel with $\mathcal{M}_i$ (or re-using an existing one)
4) When receiving ($\mathcal{O}_{\mathsf{deal}}$, p_j, π_j) from $\mathcal{M}_i$:
   a) Check if $\mathcal{O}_{\mathsf{deal}}$ is valid and matches $\mathcal{O}_{\mathsf{ask}}$ and $\mathcal{O}_{\mathsf{bid}}$
   b) Check if π_j is a valid merkle-path with root hash h_p
   c) Send $\langle\ \mathcal{O}_{\mathsf{deal}},\ j\ \rangle_{\mathcal{C}_i}$
5) Output p

Figure 8-3: Description of the Put and Get Protocols in the Filecoin DSN

Manage **Protocol**

Network

AssignOrders

- INPUTS:
  - deal orders $\mathcal{O}^1_{\mathsf{deal}}..\mathcal{O}^n_{\mathsf{deal}}$
  - allocation table allocTable
- OUTPUTS: updated allocation table allocTable'

1) Copy allocTable in allocTable'
2) For each order $\mathcal{O}^i_{\mathsf{deal}}$:
   a) Check if $\mathcal{O}^i_{\mathsf{deal}}$ is valid according to Definition 8.3.1
   b) Get $\mathcal{M}_j$ from $\mathcal{O}^i_{\mathsf{deal}}$ signature
   c) Add details from $\mathcal{O}^i_{\mathsf{deal}}$ to allocTable'
3) Output allocTable'

RepairOrders

- INPUTS:
  - current time t
  - current ledger $\mathcal{L}$
  - table of storage allocations allocTable
- OUTPUTS: orders to repair $\mathcal{O}^1_{\mathsf{deal}}..\mathcal{O}^n_{\mathsf{deal}}$, updated allocation table allocTable

1) For each allocEntry in allocTable:
   a) If $t <$ allocEntry.last $+ \Delta_{\mathsf{proof}}$: skip
   b) Update allocEntry.last$= t$
   c) Check if π is in $\mathcal{L}_{t-\Delta_{\mathsf{proof}}:t}$ and PoSt.Verify(π)
   d) On success: update allocEntry.missing$= 0$
   e) On failure:
      i) update allocEntry.missing++
      ii) penalize collateral from $\mathcal{M}_i$'s pledge
   f) If allocEntry.missing $> \Delta_{\mathsf{fault}}$ then set all the orders from the current sector as failed orders
2) Output failed orders $\mathcal{O}^1_{\mathsf{deal}}..\mathcal{O}^n_{\mathsf{deal}}$ and allocTable'.

Miner

PledgeSector

- INPUTS:
  - current allocation table allocTable
  - pledge request pledge
- OUTPUTS: allocTable'

1) Copy allocTable to allocTable'
2) Set $\mathsf{tx}_{\mathsf{pledge}} :=$ (pledge)
3) Submit $\mathsf{tx}_{\mathsf{pledge}}$ to $\mathcal{L}$
4) Wait for $\mathsf{tx}_{\mathsf{pledge}}$ to be included in $\mathcal{L}$
5) Add new sector of size pledge.size in allocTable'
6) Output allocTable'

SealSector

- INPUTS:
  - miner public/private key pair $\mathcal{M}$
  - sector index j
  - allocation table allocTable
- OUTPUTS: a proof π_{SEAL}, a root hash rt

1) Find all the pieces $p_1..p_n$ in sector $\mathcal{S}_j$ in the pieceTable
2) Set $\mathcal{D} := p_1|p_2|..|p_n$
3) Compute $(\mathcal{R}, \mathsf{rt}, \pi_{\mathsf{SEAL}}) :=$ PoST.Setup(pp, $\mathsf{pk}_{\mathcal{M}}, \mathsf{sk}_{\mathcal{M}}, \mathcal{D}$)
4) Output π_{SEAL}, rt

ProveSector

- INPUTS:
  - miner public/private key pair $\mathcal{M}$
  - sector index j
  - challenge c
- OUTPUTS: a proof π_{POS}

1) Find $\mathcal{R}$ for sector j
2) Compute $\pi_{\mathsf{POS}} :=$ PoST.Prove(pp, $\mathcal{R}, c$)
3) Output π_{POS}

Figure 8-4: Description of the Manage Protocol in the Filecoin DSN

8.3 Storage Market Protocol

In brief, the Storage Market protocol is divided in two phases: *order matching* and *settlement*:

- *Order Matching*: Clients and Storage Miners submit their orders to the orderbook by submitting a transaction to the blockchain (step 1). When orders are matched, the client sends the piece to the Storage Miner and both parties sign a *deal* order and submit it to the orderbook (step 2).
- *Settlement*: Storage Miners seal their sectors (step 3a), generate proofs of storage for the sector containing the piece and submit them to the blockchain regularly (step 3b); meanwhile, the rest of the network must verify the proofs generated by the miners and repair possible faults (step 3c).

The Storage Market protocol is explained in detail in Figure 8-5.

8.3.1 Note on Valid Orders

Definition 8.3.1. Validity of *bid*, *ask*, *deal* orders is defined as follows.

(Valid *bid* order): A *bid* order from client $\mathcal{C}_i$, $\mathcal{O}_{\mathsf{bid}} := \langle \mathsf{size}, \mathsf{funds}[, \mathsf{price}, \mathsf{time}, \mathsf{coll}, \mathsf{coding}] \rangle_{\mathcal{C}_i}$ is valid if:

- $\mathcal{C}_i$ has at least the amount of funds available in their account.
- time is not set in the past
- The order must guarantee at least a minimum amount[1] of epochs of storage.

(Valid *ask* order): An *ask* order from Storage Miner $\mathcal{M}_i$, $\mathcal{O}_{\mathsf{ask}} := \langle \mathsf{space}, \mathsf{price} \rangle_{\mathcal{M}_i}$ is valid if:

- $\mathcal{M}_i$ has pledged to be a miner and the pledge will not expire before time epochs.
- space must be less than $\mathcal{M}_i$'s available storage: $\mathcal{M}_i$ pledged storage minus the storage committed in the orderbook (in *ask* and *deal* orders).

(Valid *deal* order): A *deal* order $\mathcal{O}_{\mathsf{deal}} := \langle \mathsf{ask}, \mathsf{bid}, \mathsf{ts} \rangle_{\mathcal{C}_i, \mathcal{M}_j}$ is valid if

[1]This will be a parameter of the system.

Storage Market Protocol

Order Matching

1. Storage Miner $\mathcal{M}_i$ and Client $\mathcal{C}_i$ add orders to the OrderBook:
   (a) $\mathcal{M}_i$ creates $\mathcal{O}_{\text{ask}}{}^1, \mathcal{O}_{\text{ask}}{}^2, \ldots$ and $\mathcal{C}_i$ creates $\mathcal{O}_{\text{bid}}{}^1, \mathcal{O}_{\text{bid}}{}^2, \ldots$

- ask references an order $\mathcal{O}_{\mathsf{ask}}$ such that: it is in the Storage Market OrderBook, no other deal orders in the Storage Market OrderBook mention it, it is signed by $\mathcal{C}_i$.
- bid references an order $\mathcal{O}_{\mathsf{bid}}$ such that: it is in the Storage Market OrderBook, no other deal orders in the Storage Market OrderBook mention it, it is signed by $\mathcal{M}_j$.
- ts is not set in the future or too far in the past.

Remark. If a malicious client receives a signed deal from a Storage Miner, but never adds it to the orderbook, then the Storage Miner cannot re-use the storage committed in the deal. The field ts prevents this attack because, after ts, the order becomes invalid and cannot be submitted in the orderbook.

8.4 Retrieval Market Protocol

In brief, the Retrieval Market protocol is divided in two phases: *order matching* and *settlement*:

- *Order Matching*: Clients and Retrieval Miners submit their orders to the orderbook by gossiping their orders (step 1). When orders are matched, the client and the Retrieval Miners establish a micropayment channel (step 2).
- *Settlement*: Retrieval Miners send a small parts of the piece to the client and for each piece the client sends to the miner a signed receipt (step 3). The Retrieval Miner presents the delivery receipts to the blockchain to get their rewards (step 4).

The protocol is explained in details in Figure 8-6.

Retrieval Market Protocol

Order Matching:

1. Retrieval Miners and Clients add orders to the Get.OrderBook:
   (a) Retrieval Miners $\mathcal{M}_i$ creates *ask* orders $(\mathcal{O}_{\mathsf{ask}}{}^1, \mathcal{O}_{\mathsf{ask}}{}^2, ..)$ and Client $\mathcal{C}_j$ creates *bid* orders $(\mathcal{O}_{\mathsf{bid}}{}^1, \mathcal{O}_{\mathsf{bid}}{}^2, ..)$.
   (b) Both $\mathcal{M}_i$ and $\mathcal{C}_j$ gossip their orders in the Filecoin network via Get.addOrders
   (c) Since there is no *commonly shared* orderbook, when users receive orders, they add them to their own orderbook's view. Differently from the Storage Market, these orders are not binding and no resource is committed (e.g. clients don't do any deposit).
2. When orders match, involved parties jointly create $\mathcal{O}_{\mathsf{deal}}$ and add it to the Get.OrderBook:
   (a) Retrieval Miner $\mathcal{M}_i$ and Client $\mathcal{C}_j$ independently run Get.matchOrders that queries their own current Get.OrderBook view.
   (b) Both $\mathcal{M}_i$ and $\mathcal{C}_j$ sign $\mathcal{O}_{\mathsf{deal}}$ and add it to their Get.OrderBook via Get.addOrders (as described before)
   (c) $\mathcal{C}_i$ and $\mathcal{M}_j$ setup a micropayment channel for $\mathcal{O}_{\mathsf{deal}}$

Settlement:

3. Both parties check whether the piece has been delivered:
   (a) $\mathcal{M}_i$ sends the piece p in parts via Get.SendPieces

8.5 Guarantees

The following is a brief analysis on how the Filecoin DSN achieves *integrity*, *retrievability*, *public verifiability* and *incentive-compatibility*.

- *Achieving Integrity*: Pieces are named after their cryptographic hash. After a Put request, clients only need to store this hash to retrieve the data via Get and to verify the integrity of the content received.

- *Achieving Retrievability*: In a Put request, clients specify the replication factor and the type of erasure coding desired, specifying in this way the storage to be (f, m)-*tolerant*. The assumption is that given m Storage Miners storing the data, a maximum of f faults are tolerated. By storing data in more than one Storage Miner, a client can increase the chances of recovery, in case Storage Miners go offline or disappear.

- *Achieving Public Verifiability and Auditability*: Storage Miners are required to submit their proofs of storage (π_{SEAL}, π_{POST}) to the blockchain. Any user in the network can verify the validity of these proofs, without having access to the outsourced data. Since the proofs are stored on the blockchain, they are a trace of operation that can be audited at any time.

- *Achieving Incentive Compatibility*: Informally, miners are rewarded for the storage they are providing. When miners commit to store some data, then they are required to generate proofs. Miners that skip proofs are penalized (by losing part of their collateral) and not rewarded for their storage. For the purpose of this thesis, we require a game theoretical analysis to be future work.

- *Achieving Confidentiality*: Clients that desire for their data to be stored privately, must encrypt their data before submitting them to the network.

Chapter 9

Future Work

The work presented in this thesis is the first milestone in a longer research path. It opens new directions of research in *Fair-Exchange*, *Proofs-of-Storage* and more broadly *Decentralized Storage Networks*. It is introductory and missing of implementation, security proofs and game theoretical analysis, all of which remains as future work. Nonetheless, it presents the elementary components for building an incentivized network for file storage.

In this chapter, we present two intuitions for future work: a *useful* Proof-of-Work protocol and the integration of smart contracts in Filecoin. The remaining of this section will summarize open problems and future directions.

9.1 Consensus Based on Useful Proof-of-Work

There have been several attempts to re-use mining power for useful computation. Some efforts require miners to perform a special computation alongside the standard *Proof-of-Work*. Other efforts replace *Proof-of-Work* with useful problems that are still hard to solve. For example, Primecoin [39] re-uses miners' computational power to find new prime numbers, Ethereum requires miners to execute small programs alongside with *Proof-of-Work*, and Permacoin [48] offers archival services by requiring miners to invert a hash function while proving that some data is being archived. Although most of these attempts do perform useful work, the amount of wasteful work is still a prevalent factor in these computations.

Wasteful Work

Solving hard puzzles can be really expensive in terms of cost of machinery and energy consumed, especially if these puzzles solely rely on computational power. When the mining algorithm is embarrassingly parallel, then the prevalent factor to solve the puzzle is computational power. Can we reduce the amount of wasteful work?

Ideally, the majority of a network's resources should be spent on useful work. Some efforts require miners to use more energy-efficient solutions. For example, Spacemint [53] requires miners to dedicate disk space rather than computation; while more energy efficient, theses disks are still "wasted", since they are filled with random data. Other efforts replace hard to solve puzzles with a traditional byzantine agreement based on *Proof-of-Stake*, where stakeholders vote on the next block is proportional to their share of currency in the system.

9.1.2 Modeling Mining Power

Power Fault Tolerance. In our technical report [13], we present *Power Fault Tolerance*, an abstraction that re-frames byzantine faults in terms of participants' influence over the outcome of the protocol. Every participant controls some *power* of which n is the total power in the network, and f is the fraction of power controlled by faulty or adversarial participants.

Power in Filecoin. In Filecoin, the *power* p_i^t of miner $\mathcal{M}_i$ at time t is the sum of the $\mathcal{M}_i$'s storage assignments. The *influence* I_i^t of $\mathcal{M}_i$ is the fraction of $\mathcal{M}_i$'s power over the total

power in the network.

In Filecoin, *power* has the following properties:

- *Public*: The total amount of storage currently in use in the network is public. By reading the blockchain, anyone can calculate the storage assignments of each miner - hence anyone can calculate the *power* of each miner and the total amount of power at any point in time.

- *Publicly Verifiable*: For each storage assignment, miners are required to generate *Proofs-of-Spacetime*, proving that the service is being provided. By reading the blockchain, anyone can verify if the power claimed by a miner is correct.

- *Variable*: At any point in time, miners can add new storage in the network by pledging with a new sector and filling the sector. In this way, miners can change their amount of power they have through time.

9.1.3 Accounting for Power with *Proof-of-Spacetime*

In order to account for power in Filecoin, we can user our *Proofs-of-Spacetime.* Every Δ_{proof} blocks[1], miners are required to submit *Proofs-of-Spacetime* to the network, which are only successfully added to the blockchain if the majority of power in the network considers them valid. At every block, every full node updates the AllocTable, adding new storage assignments, removing expiring ones and marking missing proofs.

The power of a miner $\mathcal{M}_i$ can be calculated and verified by summing the entries in the AllocTable, which can be done in two ways:

- **Full Node Verification**: If a node has the full blockchain log, it runs the NetworkProtocol from the genesis block to the current block and reads the AllocTable for miner $\mathcal{M}_i$. This process verifies every *Proof-of-Spacetime* for the storage currently assigned to $\mathcal{M}_i$.

[1] Δ_{proof} is a system parameter.

- **Simple Storage Verification**: Assume a light client has access to a trusted source that broadcasts the latest block. A light client can request from nodes in the network: (1) the current AllocTable entry for miner $\mathcal{M}_i$, (2) a Merkle path that proves that the entry was included in the state tree of the last block, (3) the headers from the genesis block until the current block. In this way, the light client can delegate the verification of the *Proof-of-Spacetime* to the network.

The security of the power calculation comes from the security of *Proof-of-Spacetime*. In this setting, PoSt guarantees that the miner cannot lie about the amount of assigned storage they have. Indeed, they cannot claim to store more than the data they are storing, since this would require spending time fetching and running the slow PoSt.Setup, and they cannot generate proofs faster by parallelizing the computation, since PoSt.Prove is a sequential computation.

9.1.4 Using Power to Achieve Consensus

We foresee multiple strategies for implementing the Filecoin consensus by extending existing (and future) Proof-of-Stake consensus protocols, where stake is replaced with assigned storage. Our strategy is to design a protocol such that the influence a user has over a block is proportional to each miner's assigned storage.

9.2 File Contracts and Bridges

Future work includes the support of *File Contracts* and *Bridges*. The incentivized DSNs proposed provides two basic primitives to the end users: Get and Put, respectively for submitting orders storing and retrieving data. These primitives are the minimum necessary to build more complex storage services which can be programmed via smart contracts. This would enable users to write new fine-grained storage/retrieval operations that we classify as *File Contracts*. More generically, *Smart Contracts* would enable users to write stateful programs that can spend tokens, request storage/retrieval of data in the markets and validate storage proofs.

9.2.1 File Contracts

Users can program the conditions for which they are offering or providing storage services. There are several examples worth mentioning: (1) contracting miners: clients can specify in advance the miners offering the service without participating in the market; (2) payment strategies: clients can design different reward strategies for the miners, for example a contract can pay the miner increasingly higher through time, another contract can set the price of storage informed by a trusted oracle; (3) ticketing services: a contract could allow a miner to deposit tokens and to pay for storage/retrieval on behalf of their users; (4) more complex operations, e.g. clients can create contracts that allow for data update.

9.2.2 Bridges

Future work should include a *Bridge* system to bring access to our system from other blockchain-based services and viceversa, in order to bring other blockchains' functionality in our system. Other blockchain systems such as Bitcoin [50], Zcash [60] and in particular Ethereum [19] and Tezos, allow developers to write smart contracts. However, these platforms provide very little storage capability and at a very high cost. Bridges provide a way to bring storage and retrieval support to these platforms. Future work could to provide *bridges* to connect other blockchain services with our DSN proposal. For example, integration with Zcash would allow support for sending requests for storing data in privacy.

[redacted]

[redacted]

[redacted]

[redacted]

- *Practical implementation of Filecoin*: Implementing Filecoin is a research effort which

[redacted]

Chapter 10

Conclusion

In this thesis, I have covered the necessary background and discussed the building blocks for designing decentralized infrastructures, in particular for the purpose of File Storage. This work expanded on the work presented in [12, 13, 14].

In the first part of the thesis, I reviewed current work on *protocol tokens*; I presented the notion of *Verifiable Markets* to model markets where anyone can participate in selling their services, as long as they are verifiable. Then, I introduced the notion of a *Decentralized Storage Network* to model a network of independent storage providers offering storage services. In the second part of the thesis, I have combined our work on Proofs-of-Storage and Verifiable Markets to construct an incentivized DSN: Filecoin. In this system, providers must be able to prove they have been storing data by generating Proofs-of-Storage and post them on the blockchain - the exchange of payment for the storage service is enforced by the blockchain network. Finally, I have presented directions of future work. Although implementation is left to future work, in this thesis I have shown, based on my original previous contribution [12], a practical construction for a decentralized infrastructure for file storage.

Bibliography

[1] Muneeb Ali, Jude C Nelson, Ryan Shea, and Michael J Freedman. Blockstack: A global naming and storage system secured by blockchains.

[2] Hunt Allcott and Matthew Gentzkow. Social media and fake news in the 2016 election. Technical report, National Bureau of Economic Research, 2017.

[3] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM computing surveys (CSUR)*, 36(4):335–371, 2004.

[4] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 443–458. IEEE, 2014.

[5] Nadarajah Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 7–17. ACM, 1997.

[6] Nadarajah Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in communications*, 18(4):593–610, 2000.

[7] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609. Acm, 2007.

[8] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, page 9. ACM, 2008.

[9] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, et al. Computational integrity with a public random string from quasi-linear pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 551–579. Springer, 2017.

[10] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013*, pages 90–108. Springer, 2013.

[11] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System. 2014.

[12] Juan Benet and Nicola Greco. Filecoin: A decentralized storage network. `https://filecoin.io/filecoin.pdf`, 2017.

[13] Juan Benet and Nicola Greco. Power Fault Tolerance (Work in Progress). `https://filecoin.io/power-fault-tolerance.pdf`, 2017.

[14] Juan Benet and Nicola Greco. Proof-of-Replication (Work in Progress). `https://filecoin.io/proof-of-replication.pdf`, 2017.

[15] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *International Cryptology Conference*, pages 421–439. Springer, 2014.

[16] Timothy J Berners-Lee. Information management: A proposal. Technical report, 1989.

[17] Nir Bitansky, Alessandro Chiesa, and Yuval Ishai. Succinct non-interactive arguments via linear interactive proofs. Springer, 2013.

[18] Kevin D Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 43–54. ACM, 2009.

[19] Vitalik Buterin. Ethereum, April 2014.

[20] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. Cryptology ePrint Archive, Report 2017/566, 2017. `http://eprint.iacr.org/2017/566`.

[21] Jeremy Clark, Joseph Bonneau, Edward W Felten, Joshua A Kroll, and Andrew Miller. On decentralizing prediction markets and order books. In *In WEIS*. Citeseer, 2014.

[22] Jeremy Clark, Joseph Bonneau, Edward W. Felten, Joshua A. Kroll, Andrew Miller, and Arvind Narayanan. On Decentralizing Prediction Markets and Order Books. In *WEIS '14: Proceedings of the 10th Workshop on the Economics of Information Security*, June 2014.

[23] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing privacy enhancing technologies*, pages 46–66. Springer, 2001.

[24] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 364–369. ACM, 1986.

[25] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[26] Brian F Cooper and Hector Garcia-Molina. Peer-to-peer data trading to preserve information. *ACM Transactions on Information Systems (TOIS)*, 20(2):133–170, 2002.

[27] Luis Cuende and Jorge Izquierdo. Aragon network: A decentralized infrastructure for value exchange. 2017.

[28] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Annual Cryptology Conference*, pages 585–605. Springer, 2015.

[29] C Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. *ACM Transactions on Information and System Security (TISSEC)*, 17(4):15, 2015.

[30] Juan A Garay, Markus Jakobsson, and Philip MacKenzie. Abuse-free optimistic contract signing. Springer, 1999.

[31] Barton Gellman and Ashkan Soltani. Nsa infiltrates links to yahoo, google data centers worldwide, snowden documents say. *The Washington Post*, Oct 2013.

[32] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[33] Steven Goldfeder, Joseph Bonneau, Rosario Gennaro, and Arvind Narayanan. Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin. In *International Conference on Financial Cryptography and Data Security*, 2017.

[34] Philippe Golle, Kevin Leyton-Brown, Ilya Mironov, and Mark Lillibridge. Incentives for sharing in peer-to-peer networks. In *Electronic Commerce*, pages 75–87. Springer, 2001.

[35] LM Goodman. Tezos: A self-amending crypto-ledger, 2014.

[36] Jarrad Hope. The status network. 2017.

[37] Markus Jakobsson. Ripping coins for a fair exchange. In *Advances in CryptologyâĂŤEUROCRYPTâĂŹ95*, pages 220–230. Springer, 1995.

[38] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*

[56] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In *Theory of Cryptography Conference*, pages 262–285. Springer, 2016.

[57] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE, 2001.

[58] Meni Rosenfeld. Overview of colored coins. 2012.

[59] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[60] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[61] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 90–107. Springer, 2008.

[62] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 325–336. ACM, 2013.

[63] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. 2017.

[64] Carmela Troncoso, George Danezis, Marios Isaakidis, and Harry Halpin. Systematizing decentralization and privacy: Lessons from 15 years of research and deployments. *arXiv preprint arXiv:1704.08065*, 2017.

[65] Fabian Vogelsteller. Openzeppelin solidity on github. `https://github.com/OpenZeppelin/zeppelin-solidity`.

[66] Fabian Vogelsteller. Erc20: Token standards. `https://github.com/ethereum/eips/issues/20`, 2016.

[67] David Vorick and Luke Champine. Sia: Simple decentralized storage. 2014.

[68] Liang Wang and Jussi Kangasharju. Measuring large-scale distributed systems: case of bittorrent mainline dht. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[69] Shawn Wilkinson, Tome Boshevski, Josh Brandoff, and Vitalik Buterin. Storj a peer-to-peer cloud storage network. 2014.