# Filecoin：一种去中心化的存储网络

协议实验室

Neo Ge 译

2017 年 8 月 14 日

**摘要**

互联网正处于一场革命当中：中心化专有服务正在被去中心化开放式服务所取代；可信中心被可验证计算取代；脆弱的位置地址被弹性内容地址所取代；低效的整体式服务被点对点算法市场所取代；比特币、以太坊和其他的区块链网络已经证明了去中心化交易账本的有效性。这些公共账本处理复杂的智能合约应用程序并且交易价值数百亿美金的加密货币资产。参与者形成一个提供有效支付服务的、没有中心化管理或信任中心的去中心化网络，这些系统是互联网开放服务的第一个实例。IPFS 根据去中心化网络自身已经证实了内容寻址的有效性，即在全球点对点网络上提供数十亿文件使用的服务。它解放孤岛数据、留活网络分区、脱机工作、绕开审查并且给予了数字信息永久性。

Filecoin 是一个去中心化的存储网络，它让云存储变身成了一个算法市场。这个市场基于一个本地协议通证（也叫 Filecoin）来运行，在这里矿工们可以通过向客户提供存储获取 Filecoin。反过来，客户花费 Filecoin 雇佣矿工们来存储或分发数据。和比特币一样，Filecoin 矿工们为了可观的奖励而竞争挖区块，但是 Filecoin 的挖矿功率是和有效存储成比例的，即直接为客户提供有用的服务（这里不像比特币挖矿仅限于维护区块链的共识）。这种设计为矿工们提供了巨大的激励，使他们尽可能的集聚存储以便出租给客户。Filecoin 协议将这些集聚来的资源编织成一个世界上任何人都可以依靠的自我修复存储网络。这个网络通过复制和分散内容实现稳健型，同时自动检测与修复复制品故障。客户可以选择复制参数来针对不同的威胁进行保护。Filecoin 协议的云存储网络是一个安全的网络，因为内容在客户端是端到端加密的，而存储提供者并不能访问到解密的秘钥。Filecoin 是运行在可以为任何数据提供存储基础设施的 IPFS [1] 之上的激励层。它对去中心化数据，构建及运行分布式应用程序，以及实现智能合约具有巨大的作用。

这些工作包括：

(a) 介绍 Filecoin 网络，概述这个协议以及详细介绍几个组件。
(b) 形式化去中心化存储网络（DSN）的方案和属性，然后将 Filecoin 构造为 DSN。

(c) 介绍一种叫做复制证明的新型存储证明方案，它将可以证明任何数据副本都存储在实际的独立存储器中。
(d) 介绍一种新型的基于有序复制证明和存储之上的有效工作共识，以作为功率的一种衡量方式。
(e) 形式化可验证市场，并构建存储市场和检索市场，它们将分别管理如何从 Filecoin 写入和读取数据。
(f) 讨论实例，与其他系统的连接，以及如何使用这个协议。

注意：Filecoin 是一项正在进行中的工作。目前正在进行积极的研究，本文新的版本将出现在 https://filecoin.io. 如有意见和建议，请发送电子邮件至 research@filecoin.io.

本文由 IPFSMain 星际大陆公司 Neo Ge 翻译，中文译本会持续更新在 QmZycT24gXmR1tJFHUW3HABBb94RhwNEzVEWR8nuVx9bfk

在此特别感谢 Kejie Wang 先生对此 Filecoin 白皮书的校对工作，如对中文译文有任何建议或对 IPFS 技术有兴趣，请发送电子邮件到 nge@ipfsmain.ca

# 目录

# 图表清单

# 1. 介绍

Filecoin 是一种协议通证，它的区块链运行在一种叫做“时空证明”的新型证明机制上，它的区块将被存储数据的矿工创建出来。Filecoin 协议通过不依赖于单个协调的独立存储提供者组成的网络来提供数据存储和检索服务，其中：(1) 用户为数据存储和检索支付通证，(2) 存储矿工通过提供存储空间赚取通证，(3) 检索矿工提供数据服务赚取通证。

## 1.1. 基本组件

Filecoin 协议建立在四个新型组件之上。

1. **去中心化存储网络（DSN）**：我们提出一个由独立存储提供者组成的网络的抽象概念来提供存储和检索服务（见第 2 章）。接着我们将 Filecoin 作为一个可激励的、可审计并且可验证的 DSN 构架来展示（见第 4 章）。
2. **新型的存储证明**：我们提出两种新型的存储证明（见第 3 章）：(1) 复制证明允许存储提供者证明数据确实被复制到了其独特的专用物理存储设备上。强制执行独特的物理副本使验证者可以检验证明者不是在同一个存储空间中将多个重复数据副本删除；(2) 时空证明允许存储提供者证明他们在指定的时间内持续存储了某些数据。
3. **可验证市场**：我们将存储请求和检索（检索与取回英文相同）请求建模成由 Filecoin 网络运行的两个去中心化的可验证市场内的订单（见第 5 章）。可验证市场确保了当一种服务被正确提供的时候，相应的款项会被支付。我们展示的存储市场和检索市场中，矿工和客户可以分别地提交存储订单和检索订单。
4. **有效的工作证明**：我们展示了如何基于“时空证明”来构建一个有效的工作证明来应用于共识协议之中。矿工将不再需要花费不必要的计算资源来挖掘区块，而是必须在网络中存储数据。

## 1.2. 协议概述

- Filecoin 协议是一个构建在区块链和本地通证之上的去中心化存储网络。用户为存储和检索数据花费通证，矿工以存储和提供数据赚取通证。
- Filecoin 的 DSN 通过两个可验证的市场来分别处理存储和检索请求：即存储市场和检索市场。用户和矿工为所要求的和提供的服务设定价格，并将订单提交到市场上。
- 市场由采用了时空证明和复制证明的 Filecoin 网络来操作，以确保矿工准确无误地存储他们承诺存储的数据。
- 最后，矿工可以参与区块链中新区块的创造中。一个矿工对下一个区块的影响力与它在网络中当前存储的使用量成正比。

图 1 是使用了特定术语的 Filecoin 协议草图，这些术语将会在本文后面阐述，图 2 是一个插图。

## 1.3. 论文组织

本文的其余内容安排如下：我们将在第 2 章中阐述一个理论上的 DSN 方案的定义与要求。在第 3 章中，我们将会激励、定义并且展示我们的复制证明和时空证明协议，他们将在 Filecoin 系统中用加密的方式保证数据按照订单的要求被持续不断地存储。第 4 章则描述了 Filecoin DSN 的具体事例，包括描述数据结构、协议以及参与者之间的相互作用。第 5 章将定义及描述了可验证市场的概念，包括了存储市场和验证市场是如何实施的。第 6 章为演示及评估矿工对网络的贡献描述了时空证明协议的使用情况，这对扩展区块链以及分配奖励区块是必要的。第 7 章则提供了 Filecoin 智能合约的简要介绍，第 8 章则以对未来工作的讨论结束。

**Filecoin 协议草图**

**网络**

在每一个纪元 $t$ 的账本 $\mathcal{L}$ 中：

1. 对于每一个新区块：
   (a) 检查区块是否为有效格式
   (b) 检查所有的交易都有效
   (c) 检查所有的订单都有效
   (d) 检查所有的证明都有效
   (e) 检查所有的抵押物都有效
   (f) 如上述任何一个失败则丢弃区块
2. 对于在 $t$ 中引入的每个新订单 $o$
   (a) 添加 $o$ 到存储市场订单簿
   (b) 如果 $o$ 是报价：锁定 $o.\text{funds}$
   (c) 如果 $o$ 是询价：锁定 $o.\text{space}$
   (d) 如果 $o$ 是成交订单：运行 Put.AssignOrders
3. 对于存储市场订单簿中的每一个 $o$
   (a) 检查 $o$ 如果过期（或取消）了：
      - 从订单簿中移除 $o$
      - 退换未动用的资金 $o.\text{funds}$
      - 从分配表中解放 $o.\text{space}$
   (b) 如果 $o$ 是成交订单，通过运行 Manage.RepairOrders 检查预期证明是否存在：
      - 如果有一个失踪，则惩罚 $\mathcal{M}$ 的抵押物
      - 如果证明已经失踪了 $\Delta_{\text{fault}}$ 个纪元以上，取消订单并且重新将其推向市场
      - 如果无法从网络中取回和重建该碎片，则取消订单并为客户退款

**客户**

在任何时候：

1. 通过 Put.AddOrders 提交新的存储订单
   (a) 通过 Put.MatchOrders 寻找匹配订单
   (b) 向匹配成功的矿工 $\mathcal{M}$ 发送文件
2. 通过 Get.AddOrders 提交新的检索订单
   (a) 通过 Get.MatchOrders 寻找匹配订单
   (b) 与 $\mathcal{M}$ 构建支付通道

从存储矿工 $\mathcal{M}$ 收到 $o_{\text{deal}}$

1. 签署 $o_{\text{deal}}$
2. 通过 Put.AddOrders 将其提交到区块链

从检索矿工 $\mathcal{M}$ 收到 $(p_i)$

1. 签署它
2. 向 $\mathcal{M}$ 发送一个小额款项

**存储矿井**

在任何时候

1. 通过 Manage.PledgeSector 更新过期的抵押
2. 通过 Manage.PledgeSector 抵押新的存储
3. 通过 Put.AddOrder 提交新的询价订单

在每一个纪元 $t$：

1. 对于订单簿中的每一个 $o_{\text{ask}}$：
   (a) 通过 Put.MatchOrders 寻找匹配订单
   (b) 通过联系匹配的客户开始新的交易
2. 对于每一个被抵押的扇区：
   (a) 通过 Manage.ProveSector 生成存储证明
   (b) 如果有时间发布证明（每个 $\Delta_{\text{fault}}$ 纪元），将其提交到区块链

从客户 $\mathcal{C}$ 接受到碎片 $p$：

1. 检查碎片是否具有订单 $o_{\text{bid}}$ 中制定的尺寸
2. 创建 $o_{\text{deal}}$ 并签署、发送给 $\mathcal{C}$
3. 在扇区中存储碎片
4. 如果扇区满了，则运行 Manage.SealSector

**检索矿井**

在任何时候

1. 向网络广播询价订单
2. 从网络收听出价订单

从客户 $\mathcal{C}$ 接受到检索请求：

1. 与 $\mathcal{C}$ 开始搭建支付通道
2. 将数据分为多份
3. 只有在收到付款时才发送

**图 1 Filecoin 协议草图**

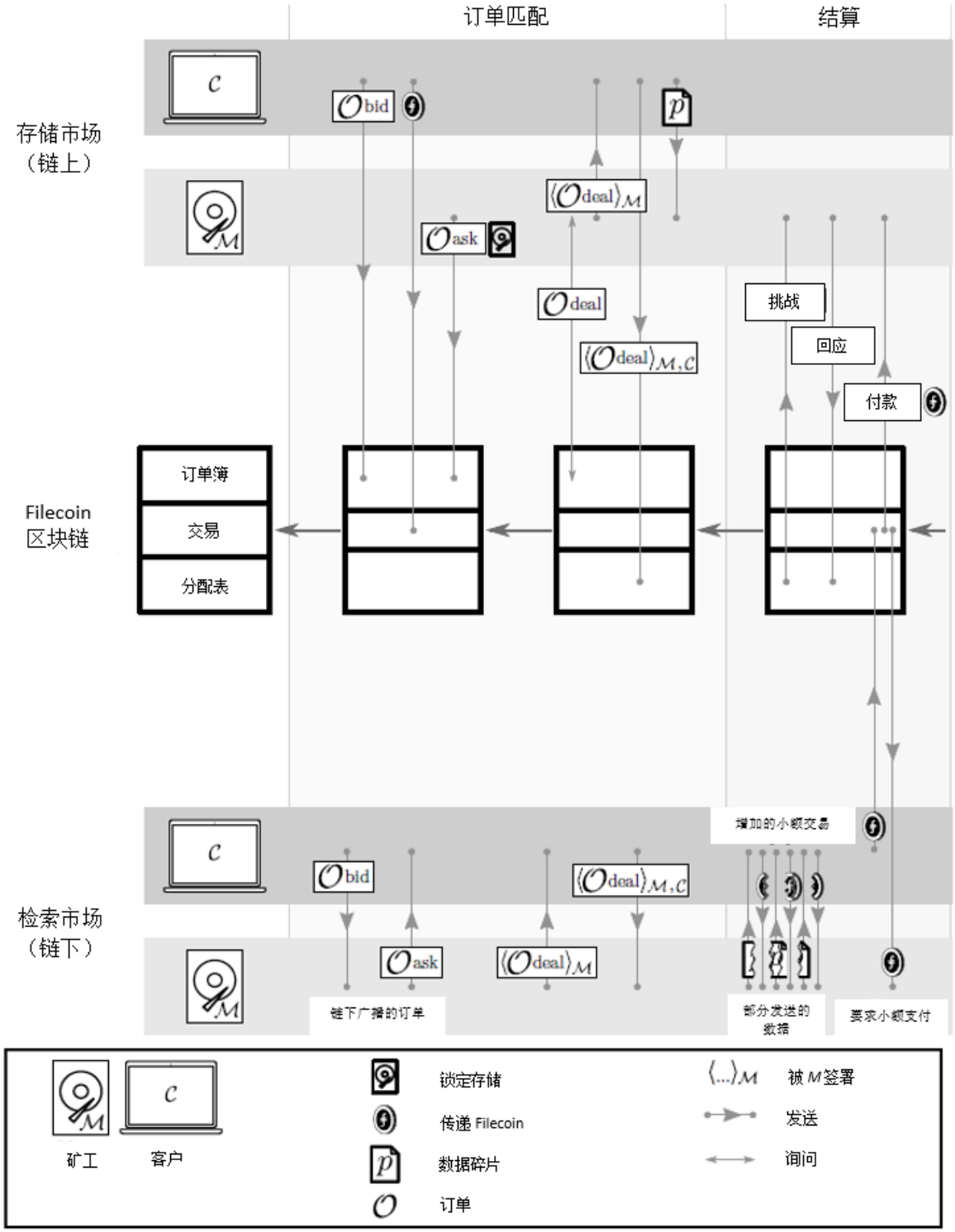


**图 2 Filecoin 协议插图，展示了客户和矿工之间的互动。此图分别在“区块链”的上面和下面展示了存储市场和检索市场，随着时间推进从左侧的订单匹配阶段过渡到右侧的结算阶段。请注意，在为检索进行小额支付之前，用户必须为小额支付锁定资金。**

# 2. 去中心化存储网络的定义

我们介绍了去中心化存储网络（DSN）方案的概念。DSNs 聚集了多个独立存储供应商提供的存储空间，并且它能自我协调以对用户提供数据存储和检索服务。这种协调是去中心化并且不需要信任方的：即通过协议调节及验证个体方的操作来达到安全运行整个系统的目的。DSNs 可以根据系统的需求采用不同的调节策略，包括拜占庭协议、流言协议以及无冲突可复制数据类型（CRDTs）。在后面的第 4 章中，我们将会提供一个 Filecoin DSN 的架构。

**定义 2.1.** 一个 DSN 方案 $\prod$ 是一个由存储供应商和客户运行的协议元组：

（Put, Get, Manage）

- Put (data) → key：客户执行 Put 协议在唯一的标识秘钥下存储数据。
- Get (key) → data：客户执行 Get 协议用秘钥检索当前存储的数据。
- Manage ()：网络的参与者通过 Manage 协议来进行协调：即控制可用的存储，审核供应商提供的服务以及修复可能出现的故障。Manage 协议常常由存储提供商联合客户或者审计网络一同运行[1]。

一个 DSN 方案 $\prod$ 必须确保数据的完整性和可回收性，并且能够容许下面这些管理和存储上的故障。

## 2.1. 故障容错

### 2.1.1. 管理故障

我们将管理故障定义为由 Manage 协议中参与者引起的拜占庭故障。一个 DSN 方案依赖于它 Manage 协议的故障容错性。违反故障容错性的管理故障假设会对系统的活跃度和安全性进行妥协。

例如，考虑一个 DSN 方案 $\prod$，其中 Manage 协议需要拜占庭协议来审核存储供应商。在这样的协议中，网络接收存储供应商提供的存储证明并运行拜占庭协议来对这些证明的有效性达成共识。如果在 $n$ 个所有节点之中，拜占庭协议能容许最多 $f$ 个故障节点，那么我们的 DSN 可以容许 $f < n/2$ 个故障节点。在违反这些假设的情况下，审计上就要做出妥协。

### 2.1.2. 存储故障

我们将存储故障定义为阻止客户检索数据的拜占庭故障：例如存储矿工丢掉了他们的碎片，检索矿工停止了服务碎片。一个成功的 Put 操作是$(f, m)$，即它的输入数据存储在 $m$ 个独立的存储供应商上（一共有 $n$ 个），而且它可以容许最多 $f$ 个拜占庭供应商。参数 $f$ 和 $m$ 取决于协议的实现情况；协议设计者可以将 $f$ 和 $m$ 设置为定值，或是把选择权交给使用者，将 Put (data)扩展为 Put (data, $f$, $m$)。如果故障存储供应商的数目比 $f$ 小，那么存储数据的 Get 操作便是成功的。

[1] 在 Manage 协议依赖于区块链的情况下，我们认为矿工即是审计人员，因为他们验证并调节存储供应商。

例如，考虑一个简单的方案，Put 协议被设计成了每一个存储供应商需要存储所有的数据。在这个方案中 $m = n$ 并且 $f = m - 1$ 。但 $f$ 一直都会等于 $m - 1$ 吗？并不是，有些方案可能采用可擦除式编码，每一个存储供应商将存储一段数据的特定部分，就像 $m$ 个存储供应商中有 $x$ 个被要求检索数据；在这种情况下 $f = m - x$ 。

## 2.2. 属性

我们描述了 DSN 方案中所必需的两个属性，然后将提出 Filecoin DSN 所需要的额外属性。

### 2.2.1. 数据完整性

该属性需要没有限制的对手 $\mathcal{A}$ 在 Get 操作结束的时候能够说服客户接受改变的或伪造的数据。

**定义 2.2.** 一个 DSN 方案 ∏ 可以提供数据完整性如果：对任意成功的数据 $d$ 下私钥 $k$ 的 Put 操作，不存在计算有限的对手 $\mathcal{A}$ 在 Get 操作结束时说服客户接受 $d'$，这里 $d'$ 不等于 $d$ 。

### 2.2.2. 可恢复性

该属性满足了以下要求：给定我们的 ∏ 容错假设，如果数据被成功地存储在了 ∏ ，并且存储供应商继续遵循协议，那么用户最终可以检索数据。

**定义 2.3.** 一个 DSN 方案 ∏ 可以提供数据完整性如果：对任意成功的数据下私钥的 Put 操作，存在一个成功的客户针对私钥检索数据的 Get 操作[2]

### 2.2.3. 其他属性

**定义 2.4.** 一个 DSN 方案 ∏ 是可以公开验证的，如果：对于每一个成功的 Put 操作，存储网络供应商可以生成数据当前正在被存储的证明。存储证明必须能够说服任意的知晓私钥但不能访问数据的有效验证者。

**定义 2.5.** 一个 DSN 方案 ∏ 是可以审查的，如果它生成了可验证的操作轨迹，并且在未来的时间点上能够确认数据当时确实在正确的时间线内被存储了。

**定义 2.6.** 一个 DSN 方案 ∏ 具备可兼容激励性，如果：存储供应商由于成功提供了存储和检索服务而获得了奖励，或者因为作弊而受到惩罚，这样的存储供应商的优势策略是存储数据。

[2] 这个定义不保证每一个 Get 操作都能成功：如果每次 Get 操作最终都能取回数据，那么这个方案就是公平的。

# 3. 复制证明和时空证明

在 Filecoin 协议中，存储供应商必须让他们的客户相信，客户付费的数据已经被他们存储了；在实践中，存储供应商将生成存储证明供给区块链网络或者客户自己来进行验证。

在这一章中，我们将会介绍并勾勒出复制证明和时空证明的实现方案。

## 3.1. 动机

存储证明（PoS）方案例如数据持有性验证（PDP） [2] 和可恢复性证明（PoR） [3] [4] 允许用户（即验证者 $\mathcal{V}$ ）把数据 $\mathcal{D}$ 外包给服务器（即证明者 $\mathcal{P}$ ）来反复检查服务器是否持续存储 $\mathcal{D}$ 。用户可以通过一个很高效的方式来验证外包数据的完整性，比下载数据更高效。服务器通过对一组随机数据块的采样并且与用户之间通过挑战/响应协议的形式发送少量的恒定数据这两个方法来生成所有权的概率证明。

PDP 和 PoR 方案只能保证在挑战/响应的时候证明者的数据所有权。在 Filecoin 中，我们需要更强大的保障以阻止作恶矿工利用下面三种攻击来在不提供存储的情况下作弊获得奖励：女巫攻击、外包攻击以及生成攻击。

- 女巫攻击：作恶矿工可能通过创建多个女巫身份来假装存储（并且获得奖励）了很多份物理存储副本，但实际上只存储了一份。
- 外包攻击：依赖于可以从其他存储供应商处快速获取数据，作恶矿工可能承诺存储比他们实际物理存储容量大得多的数据。
- 生成攻击：作恶矿工可能会宣称要存储大量的数据，但他们反而使用小程序有效地生成请求。如果这个小程序小于所宣称要存储的数据，那么作恶矿工在赢取 Filecoin 区块的可能性上就增加了，因为可能性是与矿工当前使用中的存储成正比的。

## 3.2. 复制证明

复制证明（PoRep）是一个新型的存储证明。它让服务器（即证明者 $\mathcal{P}$ ）说服用户（即验证者 $\mathcal{V}$ ）数据 $\mathcal{D}$ 已经被复制到了它的唯一专用的物理存储上了。我们的方案是一种交互式的协议，证明者 $\mathcal{P}$ ：(a) 承诺存储数据 $\mathcal{D}$ 的 $n$ 个不同副本（即独立物理副本），然后 (b) 通过一个挑战/响应协议说服验证者 $\mathcal{V}$ ： $\mathcal{P}$ 确实已经存储了每一个副本。据我们所知，PoRep 改进了 PoR 和 PDP 方案，阻止了女巫攻击、外包攻击和生成攻击的发生。

请注意，对于复制证明正式的定义以及其属性的描述和它的深入研究，我们推荐阅读者去阅读协议实验室的另一份学术报告 [5].

**定义 3.1.** （复制证明）PoRep 方案使得一个有效证明者 $\mathcal{P}$ 说服验证者 $\mathcal{V}$ ，数据 $\mathcal{D}$ 的、专属于 $\mathcal{P}$ 的、一个独立物理副本 $\mathcal{R}$ 已被存储在 $\mathcal{P}$ 上。PoRep 协议的特征是多项式时间算法的元组：

(Setup, Prove, Verify)

- PoRep.Setup$(1^{\lambda}, \mathcal{D}) \rightarrow \mathcal{R}, S_{\mathcal{P}}, S_{\mathcal{V}}$， 其中 $S_{\mathcal{P}}$ 和 $S_{\mathcal{V}}$ 是 $\mathcal{P}$ 和 $\mathcal{V}$ 的方案特定的设置变量，$\lambda$ 是一个安全参数。PoRep.Setup 被用来生成副本 $\mathcal{R}$，并且给 $\mathcal{P}$ 和 $\mathcal{V}$ 必要的信息来运行 PoRep.Prove 和 PoRep.Verify。一些方案可能需要证明人或者与第三方的相互作用来运算 PoRep.Setup。
- PoRep.Prove$(S_{\mathcal{P}}, \mathcal{R}, c) \rightarrow \pi^{c}$， 其中 $c$ 是验证者 $\mathcal{V}$ 发出的一个随机挑战，并且 $\pi^{c}$ 是证明者产生的对于数据 $\mathcal{D}$ 的特定副本 $\mathcal{R}$ 的访问权的证明。PoRep.Prove 由 $\mathcal{P}$ 运行来为 $\mathcal{V}$ 生成 $\pi^{c}$。
- PoRep.Verify$(S_{\mathcal{V}}, c, \pi^{c}) \rightarrow \{0,1\}$，用来检测证明是否正确。PoRep.Verify 由 $\mathcal{V}$ 运行并且说服 $\mathcal{V}$： $\mathcal{P}$ 是否存储了 $\mathcal{R}$。

## 3.3. 时空证明

时空证明方案允许用户检测在挑战期间存储供应商是否存储了外包数据。我们如何使用 PoS 方案来证明数据在于短时间内被存储了呢？对这个问题的一个自然的回答是要求用户重复（如每分钟）对存储供应商发出挑战。然而，每次交互所需要的通讯复杂程度将会成为类似 Filecoin 这类系统的瓶颈，因为存储供应商也会被要求提交他们的证明到区块链网络。

为了回答这个问题，我们引出一种新的证明，时空证明，其中验证者可以检验在一段时间内证明者是否存储了他/她的外包数据。如此对证明者的要求则是 (1)生成顺序的存储证明（在这里是复制证明），来作为一种 (2)递归执行生成简短证明的确定时间的方法。

**定义 3.2.** （时空证明）PoSt 方案使得有效的证明者 $\mathcal{P}$ 去说服验证者 $\mathcal{V}$：在一段时间 $t$ 内 $\mathcal{P}$ 存储了数据 $\mathcal{D}$。PoSt 协议的特征是多项式时间算法的元组：

(Setup, Prove, Verify)

- PoSt.Setup$(1^{\lambda}, \mathcal{D}) \rightarrow S_{\mathcal{P}}, S_{\mathcal{V}}$，其中 $S_{\mathcal{P}}$ 和 $S_{\mathcal{V}}$ 是 $\mathcal{P}$ 和 $\mathcal{V}$ 的方案特定的设置变量，$\lambda$ 是一个安全参数。PoSt.Setup 被用来给 $\mathcal{P}$ 和 $\mathcal{V}$ 提供必要的信息来运行 PoSt.Prove 和 PoSt.Verify。一些方案可能需要证明人或者与第三方的相互作用来运算 PoSt.Setup。
- PoSt.Prove$(S_{\mathcal{P}}, \mathcal{D}, c, t) \rightarrow \pi^{c}$，其中 $c$ 是验证者 $\mathcal{V}$ 发出的一个随机挑战，并且 $\pi^{c}$ 是证明者在一段时间 $t$ 内对数据 $\mathcal{D}$ 的访问权的证明。PoSt.Prove 由 $\mathcal{P}$ 运行来为 $\mathcal{V}$ 生成 $\pi^{c}$。
- Post.Verify$(S_{\mathcal{V}}, c, t, \pi^{c}) \rightarrow \{0,1\}$，用来检测证明是否正确。PoSt.Verify 由 $\mathcal{V}$ 运行并且说服 $\mathcal{V}$： $\mathcal{P}$ 是否在一段时间 $t$ 内存储了 $\mathcal{D}$。

## 3.4. PoRep 和 PoSt 的实际应用

我们对 PoRep 和 PoSt 在线用系统中的应用构建感兴趣，而不是依赖于硬件或是信任方。我们给出了一个 PoRep 的系统架构（详见复制证明学术报告中“以密封为基础的复制

证明”），它在 Setup 过程中需要一个非常慢的顺序计算来密封以生成副本。PoRep 和 PoSt 的协议草图在图 4 中给出，PoSt 底层机制的证明步骤则在图 3 中。

### 3.4.1. 构建加密区块

**防碰撞哈希运算：**我们使用了一个防碰撞的哈希函数 CRH：$\{0,1\}^* \rightarrow \{0,1\}^{O(\lambda)}$。我们还使用了一个防碰撞哈希函数 MerkleCRH，它将字符串分割成了多个部分，再构造出一个二叉树并递归采用 CRH，然后输出树根。

**zk-SNARKs：**我们 PoRep 和 PoSt 的实际实施依赖于零知识证明 [6] [7] [8]。因为 zk-SNARKs 是简洁的，证明非常短而且易于验证。更正式一些，让 $\mathcal{L}$ 成为一种 NP 语言，$\mathcal{C}$ 是 $\mathcal{L}$ 的决策流程。一个信任方引导了一个一次性的设置阶段，这产生了两个公钥：一个证明秘钥 pk 和一个验证秘钥 vk。证明秘钥 pk 使任何（不受信任的）证明者生成一个证明 $\pi$ 来证明，对于她选择的一个实例 $x$，$x \in \mathcal{L}$。这个非交互式证明 $\pi$ 既是零知识又是知识证明。任何人都可以使用验证秘钥 vk 验证 $\pi$；特别是 zk-SNARK 的证明是可以公开验证的：任何人都可以验证 $\pi$，而不与产生 $\pi$ 的证明者进行交互。证明 $\pi$ 具有恒定的大小并且可以在 $|x|$ 中线性的时间内验证。

对于 zk-SNARK 的流程满足性是多项式时间算法的三重元组

(KeyGen, Prove, Verify)

- $\text{KeyGen}(1^\lambda, \mathcal{C}) \rightarrow (\text{pk}, \text{vk})$。在输入安全参数 $\lambda$ 和流程 $\mathcal{C}$ 之上，KeyGen 产生概率样本 pk 和 vk。这两个秘钥作为公钥被公布，并且可以在 $\mathcal{L}_\mathcal{C}$ 中证明/验证成员资格。
- $\text{Prove}(\text{pk}, x, w) \rightarrow \pi$。在输入 pk、$x$ 和 NP 声明的证人 $w$ 之上，证明者 Prove 为声明 $x \in \mathcal{L}_\mathcal{C}$，输出一个非交互式的证明 $\pi$。
- $\text{Verify}(\text{vk}, x, \pi) \rightarrow \{0,1\}$。在输入 vk、$x$ 以及证明 $\pi$ 之上，如果 $x \in \mathcal{L}_\mathcal{C}$，验证者 Verify 输出 1.

我们建议感兴趣的读者参考 [6] [7] [8]，了解 zk-SNARK 系统的正式介绍和实例。通常来讲，这些系统需要一个信任方来运行 KeyGen 操作；创新的可扩展计算的完整性和隐私性系统（SCIP） [9] 展示了在以上假设信任的前提下，为避免这个第一步，展示了一个有希望的方向。

### 3.4.2. 密封操作

密封操作所扮演的角色是(1) 通过要求证明者存储一个对于他们公钥来说独特的伪随机排列 $\mathcal{D}$，强迫复制品成为物理独立副本，这样，承诺存储 $n$ 个复制品将会导致 $n$ 个独立副本的专用空间（因此复制品的存储大小是 $n$ 倍），并且(2) 在 PoRep.Setup 过程中强迫生成副本实质上会花费比预计响应挑战更多的时间。有关密封操作更正式的定义，请参见 [5] 。以上的操作可以用 $\text{Seal}^\tau_{\text{AES}-256}$ 来实现，并且 $\tau$ 使得 $\text{Seal}^\tau_{\text{AES}-256}$ 需要花费比诚实的挑战验证证明序列多 10-100 倍的时间。请注意，选择 $\tau$ 是极为重要的，因为与以随机访问 $\mathcal{R}$ 来运行 Prove 相比，运行 $\text{Seal}^\tau_{\text{BC}}$ 将毫无疑问地更花时间。

### 3.4.3. 实用的 PoRep 架构

本章将描述 PoRep 协议的架构，并且在图 4 中包含了一个简单的协议草图；实现和优化的细节被省略了。

**创建副本：**Setup 算法通过密封操作生成一个副本并且提供证明。证明者生成副本并将输出（不包括 $\mathcal{R}$ ）发送给验证者。

Setup

- 输入：
  --- 证明者秘钥对 $(\mathrm{pk}_{\mathcal{P}}, \mathrm{sk}_{\mathcal{P}})$
  --- 证明者 SEAL 秘钥 $\mathrm{pk}_{\mathrm{SEAL}}$
  --- 数据 $\mathcal{D}$
- 输出：副本 $\mathcal{R}$ ，$\mathcal{R}$ 的 Merkle 树根 rt，证明$\pi_{\mathrm{SEAL}}$

**证明存储：**Prove 算法生成对于副本的存储证明。证明者收到来自验证者的随机挑战 $c$ ，要求在以 rt 为树根 $\mathcal{R}$ 的 Merkle 树中确认特定的叶子 $\mathcal{R}_c$ ；证明者生成关于 $\mathcal{R}_c$ 的、Merkle 路径通往 rt 的知识证明。

Prove

- 输入：
  --- 证明者存储证明秘钥 $\mathrm{pk}_{\mathrm{POS}}$
  --- 副本 $\mathcal{R}$
  --- 随机挑战 $c$
- 输出：一个证明 $\pi_{\mathrm{POS}}$

**验证证明：**Verify 算法检查副本给定 Merkle 树根和原始数据哈希的存储证明的合法性。证明是可以公开验证的：维护去中心化系统账本的节点和对特定数据感兴趣的客户可以验证这些证明。

Verify

- 输入：
  --- 证明者公钥，$\mathrm{pk}_{\mathcal{P}}$
  --- 验证者 SEAL 和 POS 秘钥 $\mathrm{vk}_{\mathrm{SEAL}}, \mathrm{vk}_{\mathrm{POS}}$
  --- 数据 $\mathcal{D}$ 的哈希，$h_{\mathcal{D}}$
  --- 副本 $\mathcal{R}$ 的 Merkle 树根，rt
  --- 随机挑战， $c$
  --- 证明的元组，$(\pi_{\mathrm{SEAL}}, \pi_{\mathrm{POS}})$
- 输出：比特 $b$ ，在证明有效时等于 1

### 3.4.4. 实用的 PoSt 架构

本章将描述 PoSt 协议的架构，并且在图 4 中包含了一个简化的协议草图；实现和优化的细节被省略了。Setup 和 Verify 算法和上面 PoRep 中的架构相同，因此这里只描述 Prove。

**空间与时间的证明：**Prove 算法为副本生成时空证明。证明者从验证者处收到随机挑战，并为此顺序生成复制证明，同时使用证明的输出作为另一个输入做指定 t 次的迭代（见图 3）。

Prove

- 输入：
  --- 证明者 PoSt 秘钥 $\mathrm{pk}_{\mathrm{POST}}$
  --- 副本 $\mathcal{R}$
  --- 随机挑战 $c$
  --- 时间参数 $t$
- 输出：一个证明$\pi_{\mathrm{POST}}$

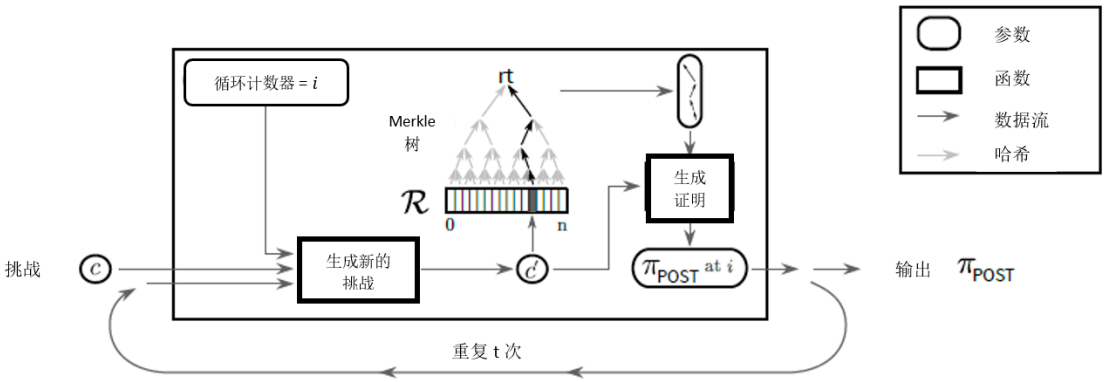


**图 3 PoSt.Prove 的基础机制图示显示了随时间推移存储的迭代证明**

## 3.5. 在 Filecoin 中的运用

Filecoin 协议采用了时空证明来审核矿工提供的存储。为了在 Filecoin 中使用 PoSt，出于无指定验证者和我们想要任何网络成员都可以验证的原因，我们将我们的方案修改成了非交互模式。由于我们的验证者是在公共通证模型下运行的，我们可以从区块链中提取随机性来发出挑战。

**Filecoin PoRep 协议**

**Setup 建立**

- 输入：
  - 证明者秘钥对 $(\mathrm{pk}_{\mathcal{P}}, \mathrm{sk}_{\mathcal{P}})$
  - 证明者 SEAL 秘钥 $\mathrm{pk}_{\mathrm{SEAL}}$
  - 数据 $\mathcal{D}$
- 输出：副本 $\mathcal{R}$ ，$\mathcal{R}$ 的 Merkle 树根 rt，证明 $\pi_{\mathrm{SEAL}}$

1) 计算 $h_{\mathcal{D}} \coloneqq \mathrm{CRH}(\mathcal{D})$
2) 计算 $\mathcal{R} \coloneqq \mathrm{Seal}^{\mathrm{r}}(\mathcal{D}, \mathrm{sk}_{\mathcal{P}})$
3) 计算 $\mathrm{rt} \coloneqq \mathrm{MerkleCRH}(\mathcal{R})$
4) 设定 $\vec{x} \coloneqq (\mathrm{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathrm{rt})$
5) 设定 $\vec{\omega} \coloneqq (\mathrm{sk}_{\mathcal{P}}, \mathcal{D})$
6) 计算 $\pi_{\mathrm{SEAL}} \coloneqq \mathrm{SCIP.Prove}(\mathrm{pk}_{\mathrm{SEAL}}, \vec{x}, \vec{\omega})$
7) 输出 $\mathcal{R}, \mathrm{rt}, \pi_{\mathrm{SEAL}}$

**Prove 证明**

- 输入：
  - 证明者存储证明秘钥 $\mathrm{pk}_{\mathrm{POS}}$
  - 副本 $\mathcal{R}$
  - 随机挑战 $c$
- 输出：一个证明 $\pi_{\mathrm{POS}}$

1) 计算 $\mathrm{rt} \coloneqq \mathrm{MerkleCRH}(\mathcal{R})$
2) 计算路径 path ≔从 rt 到叶子 $\mathcal{R}_c$的 Merkle 路径
3) 设定 $\vec{x} \coloneqq (\mathrm{rt}, c)$
4) 设定 $\vec{\omega} \coloneqq (\mathrm{path}, \mathcal{R}_c)$
5) 计算 $\pi_{\mathrm{POS}} \coloneqq \mathrm{SCIP.Prove}(\mathrm{pk}_{\mathrm{POS}}, \vec{x}, \vec{\omega})$
6) 输出 $\pi_{\mathrm{POS}}$

**Verify 验证**

- 输入：
  - 证明者公钥，$\mathrm{pk}_{\mathcal{P}}$
  - 验证者 SEAL 和 POS 秘钥 $\mathrm{vk}_{\mathrm{SEAL}}, \mathrm{vk}_{\mathrm{POS}}$
  - 数据 $\mathcal{D}$ 的哈希，$h_{\mathcal{D}}$
  - 副本 $\mathcal{R}$ 的 Merkle 树根，rt
  - 随机挑战，$c$
  - 证明的元组，$(\pi_{\mathrm{SEAL}}, \pi_{\mathrm{POS}})$
- 输出：比特 $b$ ，在证明有效时等于 1

1) 设定 $\overrightarrow{x_1} \coloneqq (\mathrm{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathrm{rt})$
2) 计算 $b_1 \coloneqq \mathrm{SCIP.Verify}(\mathrm{vk}_{\mathrm{SEAL}}, \overrightarrow{x_1}, \pi_{\mathrm{SEAL}})$
3) 设定 $\overrightarrow{x_2} \coloneqq (\mathrm{rt}, c)$
4) 计算 $b_2 \coloneqq \mathrm{SCIP.Verify}(\mathrm{vk}_{\mathrm{POS}}, \overrightarrow{x_2}, \pi_{\mathrm{POS}})$
5) 输出 $b_1 \wedge b_2$

**Filecoin PoSt 协议**

**Setup 建立**

- 输入：
  - 证明者秘钥对$(\mathrm{pk}_{\mathcal{P}}, \mathrm{sk}_{\mathcal{P}})$
  - 证明者 POST 秘钥对$\mathrm{pk}_{\mathrm{POST}}$
  - 一些数据 $\mathcal{D}$
- 输出：副本 $\mathcal{R}$ ，$\mathcal{R}$ 的 Merkle 树根 rt，证明 $\pi_{\mathrm{SEAL}}$

1) 计算 $\mathcal{R}$，rt，$\pi_{\mathrm{SEAL}} \coloneqq$ $\mathrm{PoRep.Setup}(\mathrm{pk}_{\mathcal{P}}, \mathrm{sk}_{\mathcal{P}}, \mathrm{pk}_{\mathrm{SEAL}}, \mathcal{D})$
2) 输出 $\mathcal{R}$，rt，$\pi_{\mathrm{SEAL}}$

**Prove 证明**

- 输入：
  - 证明者 PoSt 秘钥 $\mathrm{pk}_{\mathrm{POST}}$
  - 副本 $\mathcal{R}$
  - 随机挑战 $c$
  - 时间参数 $t$
- 输出：一个证明$\pi_{\mathrm{POST}}$

1) 设定 $\pi_{\mathrm{POST}} \coloneqq \perp$
2) 计算 $\mathrm{rt} \coloneqq \mathrm{MerkleCRH}(\mathcal{R})$
3) 对于 $i = 0 \ldots t$:
   a) 设定 $c' \coloneqq \mathrm{CRH}(\pi_{\mathrm{POST}} || c || i)$
   b) 计算 $\pi_{\mathrm{POS}} \coloneqq \mathrm{PoRep.Prove}(\mathrm{pk}_{\mathrm{POS}}, \mathcal{R}, c')$
   c) 设定 $\vec{x} \coloneqq (\mathrm{rt}, c, i)$
   d) 设定 $\vec{\omega} \coloneqq (\pi_{\mathrm{POS}}, \pi_{\mathrm{POST}})$
   e) 计算 $\pi_{\mathrm{POST}} \coloneqq \mathrm{SCIP.Prove}(\mathrm{pk}_{\mathrm{POST}}, \vec{x}, \vec{\omega})$
4) 输出$\pi_{\mathrm{POST}}$

**Verify 验证**

- 输入：
  - 证明者公钥，$\mathrm{pk}_{\mathcal{P}}$
  - 验证者 SEAL 和 POST 秘钥 $\mathrm{vk}_{\mathrm{SEAL}}, \mathrm{vk}_{\mathrm{POST}}$
  - 数据 $\mathcal{D}$ 的哈希，$h_{\mathcal{D}}$
  - 副本的 Merkle 树根，rt
  - 随机挑战，$c$
  - 时间参数 $t$
  - 证明的元组，$(\pi_{\mathrm{SEAL}}, \pi_{\mathrm{POST}})$
- 输出：比特 $b$ ，在证明有效时等于 1

1) 设定 $\overrightarrow{x_1} \coloneqq (\mathrm{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathrm{rt})$
2) 计算 $b_1 \coloneqq \mathrm{SCIP.Verify}(\mathrm{vk}_{\mathrm{SEAL}}, \overrightarrow{x_1}, \pi_{\mathrm{SEAL}})$
3) 设定 $\overrightarrow{x_2} \coloneqq (\mathrm{rt}, c, t)$
4) 计算 $b_2 \coloneqq \mathrm{SCIP.Verify}(\mathrm{vk}_{\mathrm{POST}}, \overrightarrow{x_2}, \pi_{\mathrm{POST}})$
5) 输出 $\mathrm{b}_1 \wedge \mathrm{b}_2$

**图 4 复制证明和时空证明的协议草图。这里 CRH 表示了防碰撞的哈希，$\vec{x}$ 是等待证明的 NP 声明，$\vec{\omega}$ 是证人**

# 4. Filecoin：一个 DSN 架构

Filecoin DSN 是一个可审查的、可公开验证并且为激励所设计的去中心化存储网络。客户为了存储和检索数据向矿工付费；矿工提供硬盘空间和带宽来赚取费用。矿工只有在网络审核他们确实提供了服务后才会收到付款。

在本章中，我们将展示基于 DSN 定义以及时空证明的 Filecoin DSN 架构。

## 4.1. 设置

### 4.1.1. 参与者

任何使用者都可以作为客户、存储矿工或检索矿工来进行参与。

- 客户在 DSN 中通过 Put 和 Get 请求来付费进行存储及检索数据。
- 存储矿工向网络提供数据存储。存储矿工在 Filecoin 中提供他们的硬盘空间并且服务 Put 请求。想要成为存储矿工，用户必须用与存储空间成比例的抵押物来抵押自己的硬盘空间。存储矿工用承诺在特定时间存储客户数据的方式来响应 Put 请求。存储矿工生成时空证明并将他们提交到区块链来向网络证明他们一直在存储数据。在证明无效或者丢失的情况下，存储矿工将被惩罚并失去一部分的抵押物。存储矿工也有资格挖取新的区块，如果挖到新的区块，矿工将获得创建新区块的奖励以及包含在区块中的交易费。
- 检索矿工为网络提供数据检索服务。检索矿工在 Filecoin 中提供给客户 Get 请求所需的数据。与存储矿工不同，他们不需要抵押、承诺存储数据或是提供存储证明。存储矿工可以很自然同时兼做检索矿工。检索矿工可以从客户或者检索市场上获取碎片。

### 4.1.2. 网络 N

我们将运行 Filecoin 全节点的所有用户设想为一个单一的抽象实体：网络。网络充当运行 Manage 协议的中介；非正式地，在 Filecoin 区块链的每个新区块中，全节点将管理可用存储，验证抵押物，审核存储证明以及修复可能的故障。

### 4.1.3. 账本

我们的协议适用于基于账本的货币；一般而言，我们称之为 Ledger，$\mathcal{L}$。在任何给定时间 $t$（称为纪元），所有用户都可以访问 $\mathcal{L}_t$，在纪元 $t$ 时候下，账本的交易是顺序的。账本是只可附加式的[3]。Filecoin DSN 协议可以在任何允许验证 Filecoin 证明的账本上实现；在第 6 章中我们展示了我们如何基于有用的工作构建一个账本。

### 4.1.4. 市场

存储的供给和需求组成了两个 Filecoin 市场：存储市场和检索市场。这两个市场是两个去中心化交易所，这会在第 5 章中详细解释。简言之，客户和矿工为他们请求的服务设定价格或是通过向两个市场分别提交订单。交易所为客户和矿工提供了一种方式来查看匹配报价并发起交易。如果请求的服务被成功提供，通过运行 Manage 协议，网络将保证矿工得到报酬的同时客户也被收取了费用。

[3] $t < t'$ 意味着 $\mathcal{L}_t$ 是 $\mathcal{L}'_t$ 的前缀

## 4.2. 数据结构

**碎片：**一个碎片是客户存储在 DSN 的数据的一部分。例如，数据可以被有意分为很多个碎片，每一个碎片由不同的存储矿工保存。

**扇区：**一个扇区是存储矿工提供给网络的一些硬盘空间。矿工将从客户那里得来的碎片存储在他们的扇区中并为他们的服务获取通证。为了存储碎片，存储矿工必须向网络抵押他们的扇区。

**分配表：**分配表是一个用来跟踪碎片以及分配扇区的数据结构。分配表在账本中每一个区块都会更新一次，并且它的 Merkle 树根将存储在最新的区块中。在实践中，该表用来保持 DSN 的状态，允许在证明验证过程中快速查找。更多细节请看图 5。

**订单：**一个订单是请求或提供服务的意向声明。客户向市场提交报价订单以请求服务（存储市场以存储数据或检索市场以检索数据），矿工提交询价订单来提供服务。订单的数据结构如图 10 所示。市场的协议将在第 5 章中详细介绍。

**订单簿：**订单簿是订单的集合。请查看章节 5.2.2.中的存储市场订单簿和章节 5.3.2.中的检索市场订单簿。

**抵押：**抵押是向网络提供存储（特别是扇区）的一种承诺。存储矿工必须将他们的抵押提交给账本才能开始在存储市场中接受订单。一个抵押包含了抵押扇区的大小和存储矿工存放的抵押物（更多详细信息请看图 5）。

**数据结构**

**Pledge 抵押**

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

- size，抵押的扇区大小
- coll，$\mathcal{M}_i$ 存放的针对这个抵押的抵押物

**订单簿**

订单簿 ( $\mathcal{O}^1 \ldots \mathcal{O}^n$)

- $\mathcal{O}^i$，当前有效的成交、询价和报价订单

**分配**

分配表：$\{\mathcal{M}_1 \rightarrow (\text{allocEntry} \ldots \text{allocEntry}), \mathcal{M}_2 \ldots\}$

allocEntry 分配输入：$(\text{sid}, \text{orders}, \text{last}, \text{missing})$

- sid，扇区 ID
- $\mathcal{O}^i$，当前有效的成交、询价和报价订单
- orders，订单集 $\{\mathcal{O}_{\text{deal}} \ldots \mathcal{O}_{\text{deal}}\}$
- last，账本 $\mathcal{L}$ 中的最后的存储证明
- missing，失踪证明的计数器

**图 5 DSN 方案中的数据结构**

## 4.3. 协议

在这一章中，我们将通过描述客户、网络和矿工执行的操作来概述 Filecoin DSN。我们在图 7 中介绍了 Get 和 Put 协议的方法，在图 8 中介绍了 Manage 协议。图 6 展示了一个协议执行的示例。Filecoin 协议的概览则是在图 1 中。

### 4.3.1. 客户周期

我们给出了客户周期的概览；对于接下来协议的深层次的解释将在第 5 章中给出。

1. **Put：** 客户将数据存储在 Filecoin
   客户可以通过向存储矿工支付 Filecoin 通证来存储他们的数据。Put 协议的细节将在接下来的章节 5.2.中。
   一个客户通过向存储市场订单簿提交一个报价订单（提交订单到区块链）来启动 Put 协议。当匹配到从矿工而来的询价订单时，客户将碎片发给矿工。双方签署成交订单并且将它提交到存储市场订单簿。
   客户们可以通过提交多份订单（或在订单中指定复制因子）来决定他们碎片的物理副本数量。冗余度越高，存储故障的容许度就越高。
2. **Get：** 客户从 Filecoin 检索数据
   客户可以通过向检索矿工支付 Filecoin 通证来检索在 DSN 中存储的任何数据。Get 协议在章节 5.3.中有详细介绍。
   一个客户通过向检索市场订单簿提交一个报价订单（向整个网络广播他们的订单）来启动 Get 协议。当匹配到从矿工而来的询价订单时，客户将收到从矿工而来的碎片。收到后，双方签署成交订单并且将它提交到区块链来确认交易成功。

### 4.3.2. 挖矿周期（对于存储矿工）

我们给出了一个非正式的挖矿周期概述

1. **抵押：** 存储矿工向网络抵押存储
   存储矿工通过在区块链抵押交易（通过 Manage.PledgeSector.）中存放抵押物来向整个网络抵押他们的存储。抵押物将在提供服务期间被抵押，并且会在矿工为他们承诺存储数据提供存储证明时返还给他们。如果一些存储证明失败了，那么成比例的抵押物就会损失掉。
   一旦抵押交易在区块链中出现，矿工就可以在存储市场中提供他们的存储了。

   Manage.PledgeSector
   - 输入：
     --- 当前的分配表 allocTable
     --- 抵押请求 pledge
   - 输出：allocTable’

2. **接收订单：** 存储矿工从存储市场获取存储请求
   一旦抵押交易出现在区块链中（因为在分配表中），矿工可以在存储市场中提供他们的存储：他们设定价格并且通过 Put.AddOrders.在市场的订单簿中添加一个询价订单。

   Put.AddOrders
   - 输入：订单目录 $\mathcal{O}^1 \ldots \mathcal{O}^n$
   - 输出：比特 $b$，如果成功则等于 1

   通过 Put.MatchOrders 检查他们的订单是否和客户的报价订单匹配一致。

Put.MatchOrders
- 输入：
  --- 当前的存储市场订单簿
  --- 查询订单以匹配 $\mathcal{O}^q$
- 输出：匹配订单 $\mathcal{O}^1 \ldots \mathcal{O}^n$

一旦订单匹配，客户将发送他们的数据给存储矿工。矿工收到碎片时运行 Put.ReceivePiece。当数据被接收后，矿工和客户双方签署一个成交订单并将它提交到区块链。

Put.ReceivePiece
- 输入：
  --- 为 $\mathcal{M}_j$ 签署秘钥
  --- 当前的订单簿 OrderBook
  --- 询价订单 $\mathcal{O}_{\text{ask}}$
  --- 报价订单 $\mathcal{O}_{\text{bid}}$
  --- 碎片 $p$
- 输出： $\mathcal{C}_i$ 和 $\mathcal{M}_j$ 签署的成交订单 $\mathcal{O}_{\text{deal}}$

3. **密封**：存储矿工为未来的证明准备碎片

存储矿工的存储被切分扇区，每个扇区包含了分配给矿工的碎片。网络通过分配表来跟踪每个存储矿工的扇区。当一个存储矿工的扇区被填满了，这个扇区就被密封起来了。密封是一种缓慢的顺序操作，它将扇区中的数据转换成复制品，即一个与矿工公钥相关联的唯一的物理副本。密封如章节 3.4.中描述的一样，是一个复制证明期间的必须操作。

Manage.SealSector
- 输入：
  --- 矿工公/私钥对 $\mathcal{M}$
  --- 扇区指数 $j$
  --- 分配表 allocTable
- 输出：一个证明 $\pi_{\text{SEAL}}$，一个树根哈希 rt

4. **证明**：存储矿工证明他们正在存储所承诺的碎片

当存储矿工被分配数据时，他们必须重复生成复制证明来保证他们在存储数据（更多细节请参阅第 3 章）。证明被公布在区块链上并由网络来验证它们。

Manage.ProveSector
- 输入：
  --- 矿工公/私钥对 $\mathcal{M}$
  --- 扇区指数 $j$
  --- 挑战 $c$
- 输出：一个证明 $\pi_{\text{POS}}$

### 4.3.3. 挖矿周期（对于检索矿工）

我们给出一个对于检索矿工非正式的挖矿周期概述。

1. **收到订单：**检索矿工从检索市场得到数据请求
   检索矿工通过向网络广播他们的询价订单来宣布他们的碎片：他们设置价格并向市场订单簿添加询价订单。

   Get.AddOrders
   - 输入：订单目录 $\mathcal{O}^1 \ldots \mathcal{O}^n$
   - 输出：无

   然后检索矿工检查是否与客户的报价订单匹配一致。

   Get.MatchOrders
   - 输入：
     --- 当前的检索市场订单簿
     --- 查询订单来匹配 $\mathcal{O}^q$
   - 输出：匹配订单 $\mathcal{O}^1 \ldots \mathcal{O}^n$

2. **发送：**检索矿工发送碎片给客户
   一旦订单匹配，检索矿工发送数据碎片给客户（详情参见章节 5.3.）。当碎片接收完毕，矿工和客户双方签署成交订单并将它提交到区块链。

   Put.SendPieces
   - 输入：
     --- 一个询价订单 $\mathcal{O}_{\text{ask}}$
     --- 一个报价订单 $\mathcal{O}_{\text{bid}}$
     --- 一个碎片 $p$
   - 输出：一个由 $\mathcal{M}_i$ 签署的成交订单 $\mathcal{O}_{\text{deal}}$

### 4.3.4. 网络周期

我们给出一个非正式的网络操作概述。

1. **分配：**网络将客户的碎片分配给存储矿工的扇区。
   客户通过在存储市场[4]上提交报价订单来启动 Put 协议。
   当询价与报价订单相匹配时，有关各方共同承诺交易并在市场上提交成交订单。
   此时，网络将数据分配给矿工并将其记录在分配表中。

   Manage.AssignOrders
   - 输入：
     --- 成交订单 $\mathcal{O}^1_{\text{deal}} \ldots \mathcal{O}^n_{\text{deal}}$
     --- 分配表 allocTable
   - 输出：更新分配表 allocTable'

[4] 存储订单会通过区块链提交，见第 5 章。

2. **修复：**网络发现故障并尝试修复它们。
   所有的存储分配对于网络中的每个参与者都是公开的。在每一个区块，网络都会检查每一个分配的所需证明是否存在，检查它们是否有效，并采取相应措施：
   - 如果有任何证明丢失或是无效，网络会通过扣除部分抵押物来惩罚存储矿工，
   - 如果大量证明丢失或是无效（由系统参数 $\Delta_{\text{fault}}$ 定义），网络会认定存储矿工存在故障并将订单设定为失败，并且重新推出同一份碎片的新订单进入市场，
   - 如果每一个所有存储该碎片的存储矿工都有故障，则该碎片丢失，客户获得退款。

   Manage.RepairOrders
   - 输入：
     --- 当前时间 $t$
     --- 当前账本 $\mathcal{L}$
     --- 存储分配表 allocTable
   - 输出：要修复的订单 $\mathcal{O}^1_{\text{deal}} \ldots \mathcal{O}^n_{\text{deal}}$，更新分配表 allocTable

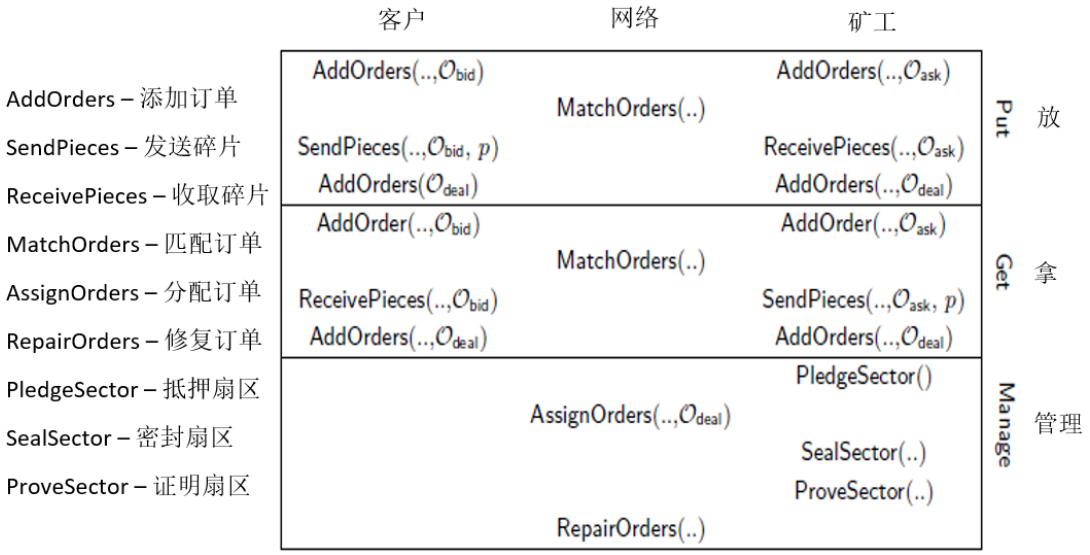


**图 6 Filecoin DSN 的执行示例，按群分组并按照行及时间顺序排序**

## 4.4. 保证和要求

以下是 Filecoin DSN 如何实现完整性、可检索性、可公开验证性和激励兼容性。

- **实现完整性：**碎片以加密哈希值命名。在一个 Put 请求之后，客户只需要存储哈希值即可通过 Get 取回数据并且验证收到的数据的完整性。
- **实现可恢复性：**在一个 Put 请求中，客户指定复制因子和所需的擦除编码类型，以这种方式指定存储可以容许 $(f, m)$ 。这个假设是说有 $m$ 个存储矿工存储数据，且容许最多 $f$ 个故障。客户可以通过在 1 个以上的存储矿工处存储数据提高数据恢复的可能性，以防止存储矿工下线或者消失。
- **实现可公开验证性和可审核性：**存储矿工需要向区块链提交他们的存储证明 $(\pi_{\text{SEAL}}, \pi_{\text{POST}})$。网络中的任意用户都可以在不访问外包数据的情况下验证这些证明的有效性。由于这些证明都是存储在区块链上的，所以一丝一毫的操作都是可以被随时审核的。
- **实现激励兼容性：**非正式地，矿工通过提供存储而获得奖励。当矿工承诺存储数据时，他们需要生成证明。忽略证明的矿工会受到惩罚（通过损失部分抵押物），并且不会收到存储的奖励。
- **实现保密性：**如果客户希望他们的数据被私密存储，他们必须在上传数据之前进行数据加密。

**Put 协议**

市场

AddOrders 添加订单

- 输入：订单目录 $O^1 \ldots O^n$
- 输出：比特 $b$，如果成功则等于 1

1) 设定 $\mathrm{tx}_{\mathrm{order}} \coloneqq (O^1, \ldots, O^n)$
2) 提交 $\mathrm{tx}_{\mathrm{order}}$ 到 $\mathcal{L}$
3) 等待 $\mathrm{tx}_{\mathrm{order}}$ 被添加到 $\mathcal{L}$ 中
4) 成功则输出 1，否则输出 0

MatchOrders 匹配订单

- 输入：
  - 当前的存储市场订单簿
  - 查询订单以匹配 $O^q$
- 输出：匹配订单 $O^1 \ldots O^n$

1) 匹配订单簿中的每个 $O^i$,以至于：
   a) 如果 $O^q$ 是询价订单：
      i) 检查 $O^i$ 是否是报价订单
      ii) 检查价格 $O^i.\mathrm{price} \geq O^q.\mathrm{price}$
      iii) 检查大小 $O^i.\mathrm{size} \leq O^q.\mathrm{space}$
   b) 如果 $O^q$ 是报价订单：
      i) 检查 $O^i$ 是否是询价订单
      ii) 检查价格 $O^i.\mathrm{price} \leq O^q.\mathrm{price}$
      iii) 检查大小 $O^i.\mathrm{space} \geq O^q.\mathrm{size}$
2) 输出匹配订单 $O^1 \ldots O^n$

**交易**

SendPieces 发送碎片

- 输入：
  - 一个询价订单 $O_{\mathrm{ask}}$
  - 一个报价订单 $O_{\mathrm{bid}}$
  - 一个碎片 $p$
- 输出：一个由 $\mathcal{M}_i$ 签署的成交订单 $O_{\mathrm{deal}}$

1) 从 $O_{\mathrm{ask}}$ 的签名中得到 $\mathcal{M}_i$ 的身份
2) 发送 $(O_{\mathrm{ask}}, O_{\mathrm{bid}}, p)$ 给 $\mathcal{M}_i$
3) 收到由 $\mathcal{M}_i$ 签署的 $O_{\mathrm{deal}}$
4) 根据定义 5.2. 检查 $O_{\mathrm{deal}}$ 是否有效
5) 输出 $O_{\mathrm{deal}}$

ReceivePiece 收取碎片

- 输入：
  - 为 $\mathcal{M}_j$ 签署秘钥
  - 当前的订单簿 OrderBook
  - 询价订单 $O_{\mathrm{ask}}$
  - 报价订单 $O_{\mathrm{bid}}$
  - 碎片 $p$
- 输出：$\mathcal{C}_i$ 和 $\mathcal{M}_j$ 签署的成交订单 $O_{\mathrm{deal}}$

1) 检查 $O_{\mathrm{bid}}$ 是否有效：
   a) 检查 $O_{\mathrm{bid}}$ 是否在订单簿中
   b) 检查 $O_{\mathrm{bid}}$ 没有被其他活跃的 $O_{\mathrm{deal}}$ 涉及
   c) 检查大小 $O_{\mathrm{bid}}.\mathrm{size}$ 是否等于 $|p|$
   d) 检查 $O$ 是否是被 $\mathcal{M}_i$ 签署的
2) 在本地存储 $p$
3) 设定 $O_{\mathrm{deal}} \coloneqq \langle O_{\mathrm{ask}}, O_{\mathrm{deal}}, \mathcal{H}(p) \rangle_{\mathcal{M}_i}$
4) 从 $O_{\mathrm{bid}}$ 获得 $\mathcal{C}_j$ 的身份
5) 发送 $O_{\mathrm{deal}}$ 给 $\mathcal{C}_j$
6) 输出 $O_{\mathrm{deal}}$

**Get 协议**

市场

AddOrders 添加订单

- 输入：订单目录 $O^1 \ldots O^n$
- 输出：无

1) 向网络广播 $O^1 \ldots O^n$

MatchOrders

- 输入：
  - 当前的检索市场订单簿
  - 查询订单来匹配 $O^q$
- 输出：匹配订单 $O^1 \ldots O^n$

1) 匹配订单簿中的每个 $O^i$，以至于：
   a) 检查 $O^i.\mathrm{piece}$ 等于 $O^q.\mathrm{piece}$
   b) 如果 $O^q$ 是一个询价订单：
      i) 检查 $O^i$ 是否是报价订单
      ii) 检查价格 $O^i.\mathrm{price} \geq O^q.\mathrm{price}$
   c) 如果 $O^q$ 是一个报价订单：
      i) 检查 $O^i$ 是否是询价订单
      ii) 检查价格 $O^i.\mathrm{price} \leq O^q.\mathrm{price}$
2) 输出匹配订单 $O^1 \ldots O^n$

**交易**

SendPieces 发送碎片

- 输入：
  - 一个询价订单 $O_{\mathrm{ask}}$
  - 一个报价订单 $O_{\mathrm{bid}}$
  - 一个碎片 $p$
- 输出：一个由 $\mathcal{C}_i$ 签署的成交订单 $O_{\mathrm{deal}}$

1) 创建 $O_{\mathrm{deal}}$：
   a) 设定 $O_{\mathrm{deal}}.\mathrm{ask} \coloneqq O_{\mathrm{ask}}$
   b) 设定 $O_{\mathrm{deal}}.\mathrm{bid} \coloneqq O_{\mathrm{deal}}$
2) 从 $O_{\mathrm{bid}}$ 的签名中得到 $\mathcal{C}_i$ 的身份
3) 与 $\mathcal{C}_i$ 之间设立小额支付通道
4) 对于数据 $p$ 的每一份 $p_j$
   a) 将 $\pi_j$ 设为从 $\mathcal{H}(p)$ 到 $p_j$ 的 Merkle 路径
   b) 发送 $(O_{\mathrm{deal}}, p_j, \pi_j)$ 到 $\mathcal{C}_i$
   c) 接收 $\langle O_{\mathrm{deal}}, j \rangle_{\mathcal{C}_i}$
5) 输出 $O_{\mathrm{deal}}$

ReceivePiece 收取碎片

- 输入：
  - 一个客户的秘钥 $\mathcal{C}_j$
  - 一个询价订单 $O_{\mathrm{ask}}$
  - 一个报价订单 $O_{\mathrm{bid}}$
  - 订单中 $p$ 的 Merkle 树哈希 $h_p$
- 输出：一个碎片 $p$

1) 创建 $O_{\mathrm{deal}}$
   a) 设定 $O_{\mathrm{deal}}.\mathrm{ask} \coloneqq O_{\mathrm{ask}}$
   b) 设定 $O_{\mathrm{deal}}.\mathrm{bid} \coloneqq O_{\mathrm{bid}}$
2) 从 $O_{\mathrm{ask}}$ 获得 $\mathcal{M}_i$ 的身份
3) 与 $\mathcal{M}_i$ 之间设立小额支付通道（或重新利用一条现有的）
4) 当从 $\mathcal{M}_i$ 接收到 $(O_{\mathrm{deal}}, p_j, \pi_j)$：
   a) 检查 $O_{\mathrm{deal}}$ 是有效并且匹配 $O_{\mathrm{ask}}$ 和 $O_{\mathrm{bid}}$ 的
   b) 用树根哈希 $h_p$ 检查 $\pi_j$ 是否为一有效 Merkle 路径
   c) 发送 $\langle O_{\mathrm{deal}}, j \rangle_{\mathcal{C}_i}$
5) 输出 $p$

**图 7 Filecoin DSN 中 Put 和 Get 协议的描述**

**Manage 管理协议**

网络

AssignOrders 分配订单

- 输入：
  - 成交订单 $\mathcal{O}_{\text{deal}}^{1} \ldots \mathcal{O}_{\text{deal}}^{n}$
  - 分配表 allocTable
- 输出：更新分配表 allocTable'

1) 在 allocTable'中复制 allocTable
2) 对于每一个订单 $\mathcal{O}_{\text{deal}}^{i}$：
   a) 根据定义 5.2. 检查 $\mathcal{O}_{\text{deal}}^{i}$ 是否有效
   b) 从 $\mathcal{O}_{\text{deal}}^{i}$的签名中得到 $\mathcal{M}_j$ 的身份
   c) 将细节从 $\mathcal{O}_{\text{deal}}^{i}$ 添加到 allocTable'中
   d) 输出 allocTable'

RepairOrders 修复订单

- 输入：
  - 当前时间 $t$
  - 当前账本 $\mathcal{L}$
  - 存储分配表 allocTable
- 输出：要修复的订单 $\mathcal{O}_{\text{deal}}^{1} \ldots \mathcal{O}_{\text{deal}}^{n}$，更新分配表 allocTable

1) 对于分配表中的每一个条目 allocEntry
   a) 如果 $t < \text{allocEntry.last} + \Delta_{\text{proof}}$：跳过
   b) 更新 $\text{allocEntry.last} = t$
   c) 检查 $\pi$ 是否在 $\mathcal{L}_{t-\Delta_{\text{proof}}:t}$ 和 $\text{PoSt.Verify}(\pi)$ 中
   d) 成功：更新 $\text{allocEntry.missing} = 0$
   e) 失败：
      i) 更新 allocEntry.missing++
      ii) 从 $\mathcal{M}_i$ 的抵押中扣除抵押物
   f) 如果 $\text{allocEntry.missing} > \Delta_{\text{fault}}$，则将当前扇区中的所有订单设为失败订单
2) 输出失败订单 $\mathcal{O}_{\text{deal}}^{1} \ldots \mathcal{O}_{\text{deal}}^{n}$和 allocTable'

矿工

PledgeSector

- 输入：
  - 当前的分配表 allocTable
  - 抵押请求 pledge
- 输出：allocTable'

1) 将 allocTable 复制到 allocTable'
2) 设定 $\text{tx}_{\text{pledge}} := (\text{pledge})$
3) 将 $\text{tx}_{\text{pledge}}$ 提交到 $\mathcal{L}$
4) 等待 $\text{tx}_{\text{pledge}}$ 被添加到 $\mathcal{L}$ 中
5) 添加新的扇区大小 pledge.size 到 allocTable'中
6) 输出 allocTable'

SealSector 密封扇区

- 输入：
  - 矿工公/私钥对 $\mathcal{M}$
  - 扇区指数 $j$
  - 分配表 allocTable
- 输出：一个证明 $\pi_{\text{SEAL}}$，一个树根哈希 rt

1) 在分配表中找到扇区 $\mathcal{S}_j$ 中的所有碎片 $p_1 \ldots p_n$
2) 设定 $\mathcal{D} := p_1 | p_2 | \ldots | p_n$
3) 计算 $(\mathcal{R}, \text{rt}, \pi_{\text{SEAL}}) := \text{PoSt.Setup}(\mathcal{M}, \text{pk}_{\text{SEAL}}, \mathcal{D})$
4) 输出$\pi_{\text{SEAL}}, \text{rt}$

ProveSector 证明扇区

- 输入：
  - 矿工公/私钥对 $\mathcal{M}$
  - 扇区指数 $j$
  - 挑战 $c$
- 输出：一个证明 $\pi_{\text{POS}}$

1) 为扇区 $j$ 找到 $\mathcal{R}$
2) 计算 $\pi_{\text{POST}} := \text{PoSt.Prove}(\text{pk}_{\text{POST}}, \mathcal{R}, c, \Delta_{\text{proof}})$
3) 输出 $\pi_{\text{POST}}$

**图 8 Filecoin DSN 中 Manage 协议的描述**

# 5. Filecoin 存储和检索市场

Filecoin 有两个市场：存储市场和检索市场。这两个市场拥有相同的结构与不同的设计。存储市场允许客户向存储矿工付费来存储数据。检索市场允许客户向检索矿工付费来享受数据的检索传递服务。在这两种情况下，客户和矿工可以设置他们的报价和要价或是接受当前的报价。这些交易是由网络这个 Filecoin 全节点网络的拟人化概念所运行的。网络保证矿工在提供服务时可以得到来自客户的奖励。

## 5.1. 可验证市场

交易市场是促进特定商品或服务交易的协议。它们使得买家和卖家促成交易。出于我们的目的，我们要求交易是可以验证的：即一个去中心化的参与者的网络必须能够在买家和卖家之间验证交易。

我们提出了可验证市场的概念，其中没有任何的实体来管理交易所，交易是透明的，任何人都可以匿名参与。可验证市场协议使得对于商品和服务的交易完全去中心化：订单簿的一致性、订单结算和服务的正确执行都是通过矿工和 Filecoin 全节点这些参与者独立验证的。我们将可验证市场简化为以下架构：

**定义 5.1.** 一个可验证市场是一个有着两个阶段的协议：订单匹配与订单结算。订单是购买或出售抵押物、商品或服务的意向声明，订单簿则是所有可用订单的清单。

**验证市场协议**

**订单匹配：**

1. 参与者添加买订单和卖订单到订单簿。
2. 当两个订单匹配时，相关双方共同创建成交订单，该订单将双方提交给交易所，并通过将其添加到订单簿的形式将这个消息传播到网络。

**结算：**

3. 通过要求卖方来为他们的交易/服务提供加密证明，网络将确保商品或服务的转移已正确执行。
4. 成功时，网络将处理付款并且从订单簿中清除订单

**图 9 可验证市场的通用协议**

## 5.2. 存储市场

存储市场是一个允许客户（如买家）为了他们的数据请求存储也允许存储矿工（如卖家）提供他们的存储空间的可验证市场。

### 5.2.1. 要求

我们根据以下要求设计了存储市场协议：

- **链上订单簿：**重要的是： (1) 存储矿工的订单是公开的，所以网络始终知晓最低价格，客户可以对其订单做出明智的决定，(2) 即使客户接受了最低的价格，

他们的订单还是要必须始终提交到订单簿中，这样市场可以对新报价作出反应。因此，我们要求将订单明确地添加到 Filecoin 区块链中，以便添加到订单簿中。

- **参与者对他们的资源做出承诺：**我们要求双方都承诺将他们的资源作为避免损害的一种方式：避免存储矿工不提供服务，也可以避免客户没有可用资金。为了参与存储市场，存储矿工必须抵押与其在 DSN 中的存储量成比例的抵押物（详情请参阅章节 4.3.3.）。通过这种方式，网络可以惩罚那些承诺存储数据但又不提供存储证明的存储矿工。同样的，客户必须向订单存入指定的资金，以这种方式保证在结算期间的承诺与资金的可用性。
- **故障的自处理：**只有在存储矿工反复证明他们已经在约定时间内存储了碎片的情况下，订单才会结算给矿工。网络必须能够验证这些证明的存在与正确性，并且按照章节 4.3.4.中的修复部分中的规则行事。

### 5.2.2. 数据结构

**Put 订单：**有三种类型的订单：出价订单，询价订单和成交订单。存储矿工创建询价订单来添加存储，客户创建出价订单来请求存储，当双方对价格达成一致时，他们便一起创建成交订单。订单的数据结构细节图 10 所示，其中对于订单的参数被明确地定义了。

**Put 订单簿：**存储市场中的订单簿是当前有效且开放的询价、出价以及成交订单的集合。用户可以通过图 7 中描述的 Put 协议：AddOrders 和 MatchOrders 的方法来与订单簿交互。

订单簿是公开的，并且每一个诚实的用户都有着相同的订单簿视图。在每一个纪元，如果新的订单交易 $(\mathrm{tx}_{\mathrm{order}})$ 出现在新的区块链区块中，那么这个新的订单就会被添加到订单簿中；订单在被取消、过期或是结算后移除出订单簿。订单被添加在区块链中的区块中，因此在订单簿中的订单都是有效的。

**定义 5.2.** 我们定义了出价、询价以及成交订单的有效性：

**（有效的出价订单）：**一个客户 $\mathcal{C}_i$ 的出价订单，

$\mathcal{O}_{\mathrm{bid}} \coloneqq \langle \mathrm{size}, \mathrm{funds}[, \mathrm{price}, \mathrm{time}, \mathrm{coll}, \mathrm{coding}] \rangle_{\mathcal{C}_i}$ 如果满足以下条件就是有效的：

- $\mathcal{C}_i$ 的账户中有可用资金
- 时间不是过去设定的
- 订单必须保证最少[5]的存储周期

**（有效的询价订单）：**一个存储矿工 $\mathcal{M}_i$ 的询价订单，

$\mathcal{O}_{\mathrm{ask}} \coloneqq \langle \mathrm{space}, \mathrm{price} \rangle_{\mathcal{M}_i}$ 如果满足以下条件就是有效的：

- $\mathcal{M}_i$ 已经抵押成为了一名矿工，并且抵押不会在纪元结束前到期
- 空间必须小于 $\mathcal{M}_i$ 的可用存储：即 $\mathcal{M}_i$ 抵押的存储减去在订单簿（询价订单和成交订单）中承诺的存储

[5] 这将是一个系统参数。

**（有效的成交订单）：**一个成交订单，

$\mathcal{O}_{\text{deal}} \coloneqq \langle \text{ask}, \text{bid}, \text{ts} \rangle_{\mathcal{C}_i, \mathcal{M}_i}$ 如满足以下条件则有效：

- 参考询价订单 $\mathcal{O}_{\text{ask}}$：它由 $\mathcal{C}_i$ 签署且在存储市场订单簿中，且没有其他的成交订单涉及它。
- 参考出价订单 $\mathcal{O}_{\text{bid}}$：它由 $\mathcal{M}_i$ 签署且在存储市场订单簿中，且没有其他的成交订单涉及它。
- ts 不能设置为未来时间或者过去很久的时间。

**备注：**如果一个作恶客户从存储矿工那里收到了一份已经签署的合同，但从来没有将其添加到订单簿中，那么存储矿工就无法重新使用订单中提交的存储。ts 将阻止这种攻击，因为，在 ts 之后，订单将变为无效并且无法被添加到订单簿中。

**存储市场订单**

报价订单

$\mathcal{O}_{\text{bid}} \coloneqq \langle \text{size}, \text{funds}[, \text{price}, \text{time}, \text{coll}, \text{coding}] \rangle_{\mathcal{C}_i}$

- size：要存储的碎片大小
- funds：客户 $\mathcal{C}_i$ 的总储蓄金额
- time：文件将被存储的最长纪元时间[a]
- price：以 Filecoin 计算的时空价格[b]
- coll：针对于这块碎片，矿工需要存放的特定抵押物
- coding：这块碎片的擦除编码方案

询价订单

$\mathcal{O}_{\text{ask}} \coloneqq \langle \text{space}, \text{price} \rangle_{\mathcal{M}_i}$

- space：矿工按照订单需要提供的存储空间大小
- price：以 Filecoin 计算的时空价格

成交订单

$\mathcal{O}_{\text{deal}} \coloneqq \langle \text{ask}, \text{bid}, \text{ts} \rangle_{\mathcal{C}_i, \mathcal{M}_i}$

- ask：来自 $\mathcal{C}_i$ 对 $\mathcal{O}_{\text{ask}}$ 的加密参考
- order：来自 $\mathcal{M}_i$ 对 $\mathcal{O}_{\text{bid}}$ 的加密参考
- ts：纪元时间戳，其中订单已由 $\mathcal{M}_i$ 签署
- hash：$\mathcal{M}_i$ 将要存储碎片的加密哈希

**检索市场订单**

报价订单

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

- piece：被请求碎片的索引[c]
- price：$\mathcal{C}_i$ 对一次索引支付的价格

询价订单

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

- piece：被请求碎片的索引
- price：$\mathcal{M}_i$ 提供碎片的价格

成交订单

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

1. ask：来自$\mathcal{C}_i$ 对 $\mathcal{O}_{\text{ask}}$ 的加密参考
2. order：来自$\mathcal{C}_i$ 对 $\mathcal{O}_{\text{ask}}$ 的加密参考

[a] 如果没有指定，碎片将被存储到资金耗尽

[b] 如果没有指定，当存储矿工出现故障时，网络可以当前最好的价格重新引入订单

[c] 只有用 Filecoin 存储的碎片才能被请求

**图 10 存储市场和检索市场的订单数据结构**

### 5.2.3. 存储市场协议

简而言之，存储市场协议被分为两个阶段：订单匹配和订单结算。

- **订单匹配：**客户和存储矿工通过提交交易到区块链来将他们的订单提交到订单簿中（第 1 步）。当订单匹配完成，客户将碎片发送给存储矿工，随后双方签署成交订单并且将它提交到订单簿（第 2 步）。
- **结算：**存储矿工封存他们的扇区（第 3a 步），为包含碎片的扇区生成存储证明并且定期将它们提交到区块链（第 3b 步）；与此同时，其余的网络必须验证矿工所生成的证明并且修复可能的故障（第 3c 步）。

存储市场协议细节请参阅图 11。

## 5.3. 检索市场

检索市场允许客户请求检索特定的碎片，并由检索矿工提供这个服务。与存储矿工不同，检索矿工不需要在特定时间周期内存储数据或者生成存储证明。在网络中的任何用户都可以成为检索矿工，通过提供检索服务来赚取 Filecoin 通证。检索矿工可以直接从客户那里获取碎片，也可以通过检索市场获取它们，或者作为存储矿工直接存储它们。

### 5.3.1. 要求

我们根据以下要求设计了检索市场协议：

- **链下订单簿：**客户必须能够找到提供所需碎片的检索矿工，并且在定价之后直接交易。这意味着订单簿不能通过区块链来运行，因为这将成为快速检索请求的瓶颈，相反，参与者将只能看到订单簿的一部分。因此，我们要求双方广播自己的订单。
- **无信任方检索：**公平交换 [10] 的不可能性提醒我们要让双方在没有信任方的情况下进行交易是不可能的。在存储市场中，区块链网络将作为一个验证存储矿工提供的存储的（去中心化）信任方。在检索市场中，检索矿工和客户将在没有网络见证文件交易的前提下来进行数据交易。我们通过要求检索矿工将数据分割成多个部分来解决这个问题，并且对于发送给客户的每个部分，矿工都会收到付款。这样，如果客户停止付款，或者矿工停止发送数据，任何一方都可以停止交易。值得注意的是，要想让这个办法管用，我们必须假设始终有一个诚实的检索矿工。
- **支付通道：**客户希望在提交付款之后立即可以取回碎片，检索矿工则希望只有在确认会收到付款之后才会提供碎片。通过公共账本验证付款可能是检索请求的瓶颈，因此我们必须依赖于高效的链下付款。Filecoin 区块链必须支持可以进行快速并且乐观的交易通道，并且仅在出现纠纷的情况下才使用区块链。通过这种方式，检索矿工和客户可以快速发送我们协议要求的小额支付。包括创建支付通道网络在内的未来工作在前面的 [11] [12] 中可以见到。

### 5.3.2. 数据结构

**Get 订单：**在检索市场中存在三种订单：客户创建的报价订单 $\mathcal{O}_{\text{bid}}$，检索矿工创建的询价订单 $\mathcal{O}_{\text{ask}}$，与当检索矿工和客户成交时共同创建的成交订单 $\mathcal{O}_{\text{deal}}$。订单的数据结构详情请参阅图 10。

**Get 订单簿：**检索市场中的订单簿是当前有效且开放的询价、出价以及成交订单的集合。不同于存储市场，每一个用户都会有不同的订单簿视图，因为订单是在网络中广播的，而每一个矿工和客户只会跟踪他们感兴趣的订单。

**存储市场协议**

订单匹配

1. 存储矿工 $\mathcal{M}_i$ 和客户 $\mathcal{C}_i$ 添加订单到订单簿：
   (a) $\mathcal{M}_i$ 创建 $\mathcal{O}^1_{\text{ask}}, \mathcal{O}^2_{\text{ask}}, \ldots$，$\mathcal{C}_i$ 创建 $\mathcal{O}^1_{\text{bid}}, \mathcal{O}^2_{\text{bid}}, \ldots$
   (b) 订单通过 Put. addOrders$(\mathcal{O}^1, \mathcal{O}^2, \ldots)$ 将订单提交到区块链上
   (c) 成功后，订单将被添加到订单簿中，$\mathcal{C}_j$ 的资金被存入，$\mathcal{M}_i$ 的空间被预留
2. 当订单匹配的时候，相关双方共同创建成交订单 $\mathcal{O}_{\text{deal}}$，并将其添加到订单簿中：
   (a) $\mathcal{M}_i$ 和 $\mathcal{C}_j$ 通过 Put. matchOrders$(\mathcal{O})$ 独立查询订单簿
   (b) 如果 $\mathcal{M}_i$ 和 $\mathcal{C}_j$ 有匹配订单：
   - $\mathcal{C}_j$ 通过 Put. SendPieces$(\mathcal{O}_{\text{bid}}, \mathcal{O}_{\text{ask}}, p)$ 将碎片 $p$ 发送给 $\mathcal{M}_i$
   - $\mathcal{M}_i$ 通过 Put. ReceivePieces$(\mathcal{O}_{\text{bid}}, \mathcal{O}_{\text{ask}}, p)$ 从 $\mathcal{C}_j$ 收取碎片 $p$
   - $\mathcal{M}_i$ 签署 $\mathcal{O}_{\text{deal}}$ 并将其发送给 $\mathcal{C}_j$

   (c) $\mathcal{C}_j$ 签署 $\mathcal{O}_{\text{deal}}$，并通过 Put. addOrders$(\mathcal{O}_{\text{deal}})$ 将其添加到订单簿

结算

3. 网络监察存储矿工是否正确地存储着碎片：
   (a) 当一个存储矿工填充一个扇区的时候，他们通过 Manage. SealSector 将其密封（他们创建一个独特的副本），并提交证明 $\pi_{\text{SEAL}}$ 和 rt 到区块链
   (b) 存储矿工在每一个纪元生成新的证明，并在每一个 $\Delta_{\text{proof}}$ 纪元，通过 Manage. ProveSectors 将它们提交到区块链
   (c) 网络在每一个纪元都会运行 Manage. RepairOrders。如果证明失踪或者无效，网络将会尝试用以下方式修复：
   - 如果有任何的证明丢失或者无效，存储矿工将被扣除部分抵押物作为惩罚
   - 如果大量证明丢失或是无效长达 $\Delta_{\text{fault}}$ 个纪元，网络会认定存储矿工存在故障并将订单设定为失败，并且重新推出同一份碎片的新订单进入市场
   - 如果每一个所有存储该碎片的存储矿工都有故障，则该碎片丢失，客户获得退款。
4. 当订单的时间到期或者资金用完的时候，如果服务被正确地提供了，网络会处理付款并移除订单。

**图 11 详细的存储市场协议**

### 5.3.3. 检索市场协议

简而言之，检索市场协议分为两个阶段：订单匹配和订单结算。

- **订单匹配：**客户和检索矿工通过广播他们的订单将订单添加到订单簿（第 1 步）。当订单匹配完成，客户和检索矿工之间建立一条小额支付通道（第 2 步）。
- **结算：**检索矿工发送碎片的一小部分给客户，然后客户针对每一份碎片都会向矿工发送一份签署的收据（第 3 步）。检索矿工向区块链出示送达收据从而获得奖励（第 4 步）。

该协议将会在图 12 中详细解释。

**检索市场协议**

订单匹配：

1. 检索矿工和客户添加订单到 Get.OrderBook：
   (a) 检索矿工 $\mathcal{M}_i$ 创建询价订单 $\mathcal{O}^1_{\text{ask}}, \mathcal{O}^2_{\text{ask}}, \ldots$，客户 $\mathcal{C}_i$ 创建报价订单 $\mathcal{O}^1_{\text{bid}}, \mathcal{O}^2_{\text{bid}}, \ldots$
   (b) $\mathcal{M}_i$ 和 $\mathcal{C}_i$ 都在 Filecoin 网络中通过 Get.addOrders 广播他们的订单
   (c) 由于没有共同的订单簿，当用户收到订单时，他们会将订单添加到他们自己的订单簿视图中。不同于存储市场，这些订单没有约束力，也没有资源的保证（如客户不会进行任何存款）。
2. 一旦订单匹配，相关双方共同创建成交订单 $\mathcal{O}_{\text{deal}}$，并将其添加到 Get.OrderBook 中：
   (a) 检索矿工 $\mathcal{M}_i$ 和客户 $\mathcal{C}_i$ 独立运行 Get.matchOrders 来查询他们各自的当前 Get.OrderBook 视图
   (b) $\mathcal{M}_i$ 和 $\mathcal{C}_i$ 双方签署 $\mathcal{O}_{\text{deal}}$，并通过 Get.addOrders 将其添加到他们的 Get.OrderBook（如前面描述）
   (c) $\mathcal{M}_i$ 和 $\mathcal{C}_i$ 为 $\mathcal{O}_{\text{deal}}$ 建立小额支付通道

结算：

3. 双方检查碎片是否已送达：
   (a) $\mathcal{M}_i$ 通过 Get.SendPieces 将碎片 $p$ 分为部分发送
   (b) $\mathcal{C}_i$ 接收部分的 $p$，并对于每一小部分的 $p$，$\mathcal{C}_i$ 通过 Get.ReceivePiece 发送一次小额支付的方式承认送达
4. 当 $p$ 被 $\mathcal{C}_i$ 接收后，$\mathcal{M}_i$ 可以向网络展示小额付款并回收款项，双方将他们的订单从订单簿中移除。

**图 12 详细的检索市场协议**

# 6. 有用的工作共识

Filecoin DSN 协议可以在任何允许 Filecoin 证明验证的共识协议之上实现。在本章中，我们将介绍如何基于有用的工作来引导共识协议。代替掉浪费的工作证明计算，由工作的 Filecoin 矿工所生成的时空证明是允许他们参与共识的原因。

**有用的工作：**如果计算的输出是对整个网络有价值的，而不仅仅是保卫区块链的安全，我们便认为矿工在共识协议中所做的工作是有用的。

## 6.1. 动机

确保区块链的安全是至关重要的，工作证明方案常常需要解决其答案不可再用或者需要大量的浪费计算的难题。

- **不可重复利用的工作：**大多数无权限的区块链要求矿工解决一个难以解决的计算难题，例如反转哈希函数。通常这些难题的解决方案都是无用的，并且除了保护网络之外没有任何固定的价值。我们能够重新将这项工作重新用于有用的事情吗？

  **尝试重复使用的工作：**业内已经有了数次尝试再利用挖矿功率进行有用的计算。有些尝试要求矿工同时与标准的工作证明进行一些特殊计算。其他的尝试想用有用的问题取代工作证明，但依然很难解决。例如，Primecoin 重新利用矿工的计算能力来寻找新的素数，以太坊要求矿工与工作证明一起执行小程序，Permacoin 通过要求矿工反转哈希函数同时证明某些数据正在存档来提供存档服务。尽管这些尝试中的绝大多数都能执行有用的工作，但是这些计算中浪费的工作量仍然很普遍。
- **浪费的工作：**解决难题在机器成本和能源成本方面的消耗是非常昂贵的，特别是如果这些谜题完全依赖于计算能力。当挖矿算法令人尴尬地平行时，解决难题的普遍因素是计算能力。我们可以减少浪费的工作吗？

  **尝试减少浪费：**理想情况下，网络资源的大部分应该花在有用工作上。一些尝试是要求矿工使用更节能的解决方案。例如，Spacemint 要求矿工用硬盘而不是算力来挖矿；虽然更加节能，但是这些硬盘空间依然被浪费了，因为它们被随机数据填满了。其他的尝试会用基于权益证明的传统拜占庭协议来代替解决难题的困难，其中利益相关方在下一个区块中的投票数与它们在系统中所占有的货币份额成正比。

我们着手设计了一个基于用户数据存储的有用的工作共识协议。

## 6.2. Filecoin 共识

我们提出了一种有用的工作共识协议，其中网络选择一个矿工来创建新区块（我们称之为矿工的投票权）的概率与当前这个矿工使用中的存储和网络其余部分相关的存储的关系成正比。我们设计了 Filecoin 协议，以便矿工宁愿投资存储而不是算力来并行挖掘计算。矿工提供存储并重新使用计算以证明数据被存储以参与共识。

### 6.2.1. 挖矿功率建模

**功率容错：**在我们的技术报告 [13] 中，我们提出了功率容错，即一个根据参与者对协议结果的影响重新构建拜占庭故障的抽象化概念。每个参与者控制网络总功率 n 中的一些功率，f 是故障或作恶参与者所控制的功率占比。

**Filecoin 功率：**在 Filecoin 中，在 $t$ 时刻，矿工$\mathcal{M}_i$的功率 $p_i^t$ 是 $\mathcal{M}_i$ 存储任务的总和。$\mathcal{M}_i$的影响力 $I_i^t$ 是 $\mathcal{M}_i$ 功率除以全网功率总和的分数。

在 Filecoin 中，功率有以下属性：

- **公开：**网络中当前正在使用的存储总量是公开的。通过读取区块链，任何人都可以计算每个矿工的存储任务，因此任何人都可以在任意时间点计算出每一个矿工的功率与网络总功率。
- **可公开验证的：**对于每一个存储任务，矿工被要求生成时空证明以证明服务在被持续提供。通过读取区块链，任何人都可以验证矿工所声明的功率是否正确。
- **可变的：**在任意时间点，矿工可以通过抵押新扇区并且填充新扇区的方式来添加新的存储。这样矿工便可以改变他们的功率。

### 6.2.2. 用时空证明来衡量功率

对于每一个 $\Delta_{\text{proof}}$ 区块[6]，矿工们必须提交时空证明到网络，如果网络中的绝大部分功率都认为这些证明是有效的，那么这些证明将被成功地添加到区块链中。在每一个区块中，每个全节点都会更新分配表、添加新的存储任务、移除过期的存储任务并且标记缺失的证明。

一个矿工 $\mathcal{M}_i$ 的功率可以通过分配表中的条目来进行计算和验证，这些可以通过两种方式来实现：

- **全节点验证：**如果一个节点拥有完整的区块链记录，则可以从创世区块运行网络协议直到当前区块并为矿工 $\mathcal{M}_i$ 读取分配表。这一过程验证当前分配给 $\mathcal{M}_i$ 存储的每一个时空证明。
- **简单存储验证：**假设一个小型客户可以访问一个广播了最新区块的可信源。小型客户可以从网络的节点中请求：(1) 矿工 $\mathcal{M}_i$ 在当前分配表中的条目，(2) 一个可以证明该条目包含在最后一个区块的状态树中的 Merkle 路径，(3) 从创世区块到当前区块的区块头。这样，小型客户就可以将时空证明的验证委托给网络。

功率计算的安全性来自于时空证明的安全性。在这个设定里面，PoSt 保证矿工无法对他们被分配的存储量说谎。实际上，他们不能声称能够存储比当前存储的数据更多的数据，因为这需要花时间来获取并运行缓慢的 PoSt.Setup，并且由于 PoSt.Prove 是一个顺序计算，他们不能通过并行计算来更快地生成证明。

[6] $\Delta_{\text{proof}}$ 是一个系统参数。

### 6.2.3. 使用功率达成共识

我们通过扩展现有的（和未来的）权益证明共识协议来预见实施 Filecoin 共识的多种策略，其中权益被指定的存储所替换。但是我们预见了权益证明协议的改进，所以我们我们提出了一项基于我们之前的工作的架构，称为预期共识 [14]。我们的策略是在每一轮选出一名（或多名）矿工，这样赢得选举的概率和每一个矿工所被分配的存储成正比。

**预期共识：**预期共识 EC 的基础在于在每个纪元确定地、不可预测地并且秘密地选举一小部分领袖。我们的期望是每个纪元都只选出 1 个领袖，但一些纪元内可能会出现 0 个或者多个领袖。领袖通过创建一个区块并将其传播到网络的方式来拓展区块链条。在每一个纪元，区块链将被延伸一个或多个区块。在没有领袖的纪元里，一个空的区块将被添加到区块链中。虽然区块链中的区块可以线性排序，但它的数据结构是一个有向无环图。EC 是一个概率共识，每个纪元都比前面的区块更加确定，最终因为不同历史可能性足够小的缘故达到足够的确定性。如果绝大部分的参与者通过扩展链或签名区块的方式将他们的权重添加到区块所属的链上，那么这个区块就被确定了。

**选举矿工：**在每一个纪元，每个矿工都会检查它们是否当选为领袖，这与先前的协议类地完成：CoA [15]，Snow White [16]，Algorand [17]。

**定义 6.1.（Filecoin 中的 EC 选举）**如果满足以下条件，则在时刻 $t$ 矿工 $\mathcal{M}_i$ 当选领袖：

$$\mathcal{H}\left(\langle t||rand(t)\rangle_{\mathcal{M}_i}\right)/2^L \leq \frac{p_i^t}{\sum_j p_j^t}$$

其中 $\text{rand}(t)$ 是一个可以在纪元 $t$ 从区块链中提取的公开的随机变量，$p_i^t$ 是矿工 $\mathcal{M}_i$ 的功率。对于任何 $m$，$\mathcal{H}(m)$ 的大小都是 $L$，$\mathcal{H}$ 是一种安全的加密哈希函数，$\langle m\rangle_{\mathcal{M}_i}$ 是一个由 $\mathcal{M}_i$ 签署的消息 $m$，使得：

$$\langle m\rangle_{\mathcal{M}_i} \coloneqq \left((m), \text{SIG}_{\mathcal{M}_i}(\mathcal{H}(m))\right)$$

在图 13 中，我们描述了矿工（ProveElect）和网络节点（VerifyElect）之间的协议。

这种选举方案提供了三个属性：公平、保密和可公开验证性。

- **公平：**由于签名是确定的，并且 $t$ 和 $\text{rand}(t)$ 是固定的，每个参与者每次选举都只有一次机会。假设 $\mathcal{H}$ 是一种安全的加密哈希函数，那么 $\mathcal{H}\left(\langle t||rand(t)\rangle_{\mathcal{M}_i}\right)/2^L$ 一定是一个从 (0,1) 均匀选择的实数。因此，方程式为真的概率必须是 $p_i^t/\sum_j p_j^t$，这等于矿工在网络中功率的那部分。因为这个概率在功率上是线性的，所以这种可能性在分裂或汇集功率下得以保留。请注意，随机值 $\text{rand}(t)$ 在时间 $t$ 之前都是未知的。
- **保密：**在给定数字签名的假设情况下，一个没有秘钥 $\mathcal{M}_i$ 的有能力的攻击者可以通过计算来获取签名的可能性是可以忽略不计的。
- **可公开验证的：**一个当选的领袖 $i \in L^t$ 可以通过展示 $t$、$\text{rand}(t)$ 和 $\mathcal{H}(\langle t||rand(t)\rangle_i)/2^L$ 来说服一个有效验证者。鉴于前面的观点，有能力的攻击者在没有获胜秘钥的情况下是不能生成证明的。

**EC 选举**

在纪元 $t$ 的**存储矿工**

$\text{ProveElect}(r, t, \mathcal{M}_i) \rightarrow \{\perp, \pi_i^t\}$

1. 计算$\mathcal{H}(\langle t||r\rangle_i)/2^L \leq \frac{p_i^t}{\sum_j p_j^t}$
   - 成功输出 $\pi_i^t = \langle t, r\rangle_i$
   - 否则输出 $\perp$

在纪元 $t$ 收到一个区块的**网络节点**

$\text{VerifyElect}(\pi_i^t, t, \mathcal{M}_i) \rightarrow \{\perp \mid \top\}$

1. 检查 $\pi_i^t$ 在 $t$ 和 $r$ 上是否是来自于用户 $\mathcal{M}_i$ 的有效签名
2. 在时刻 $t$，检查 $p_i^t$ 是否是 $\mathcal{M}_i$ 的功率
3. 测试 $\mathcal{M}_i$ 是否是当选领袖 $\mathcal{H}(\pi_i^t)/2^L < \frac{p_i^t}{\sum_j p_j^t}$
   - 成功输出 $\top$
   - 否则输出 $\perp$

**图 13 在期望共识协议中的领袖选举**

# 7. 智能合约

Filecoin 为终端用户提供了两个基本的原语：Get 和 Put。这些原语让客户可以在市场中以他们自己偏爱的价格存储和检索数据。虽然原语涵盖了 Filecoin 的默认使用案例，但我们还是通过支持智能合约的部署，允许在 Get 和 Put 之上设计更复杂的操作。用户可以编写新的精细的存储/检索请求，我们将其称为文件合约以及通用智能合约。我们整合了合约系统（基于 [18]）和一个桥接系统将 Filecoin 存储带入其他区块链之中，反之亦然，也将其他的区块链的功能带入 Filecoin 之中。

我们期望在 Filecoin 生态系统中存在大量的智能合约，我们也期待一个由智能合约开发者组成的社区。

## 7.1. Filecoin 中的合约

智能合约使得 Filecoin 的用户能够编写可以花费通证、在市场中请求存储/检索数据和验证存储证明的有状态的程序。用户可以通过将交易发送到账本以触发合约中的功能函数来与智能合约进行交互。我们拓展了智能合约系统以支持 Filecoin 的特定操作（如市场操作和证明验证）。

FIlecoin 支持特定于数据存储的合同，以及更通用的智能合约：

- **文件合约：**我们允许用户对他们提供的存储服务进行编程。有几个例子值得一提：(1) 承包矿工：客户可以提前指定提供服务的矿工而无需参与市场，(2) 支付策略：客户可以针对矿工设计不同的奖励策略，比如一个可以随着时间的推移费用越付越高的合约，另一个合约则由一个值得信任的预报器来设定存储价格，(3) 票务服务：合同允许矿工代表他们的用户存入通证并且支付存储/检索服务， (4) 更复杂的操作：客户可以创建允许数据更新的合约。
- **智能合约：**用户可以将程序与其他系统中的事物相关联（如以太坊 [18]），这并不直接依赖于存储的使用。我们预见了一些应用如：去中心化命名系统、资产追踪和众筹平台。

## 7.2. 与其他系统的集成

桥接系统是连接不同区块链的工具；虽然仍在进行中，但我们计划支持跨链交互，以便将 Filecoin 存储带入其他基于区块链的平台，并将其他平台的功能也带入 Filecoin。

- **Filecoin 在其他平台上：**其他的区块链系统如比特币 [19]、Zcash [20]，特别是以太坊 [18] 和 Tezos 允许开发者编写智能合约；然而这些平台只提供很少的存储能力和非常高的成本。我们计划提供一种桥接系统以将存储和检索支持带入这些平台。我们注意到 IPFS 已经被一些智能合约（和协议通证）当做引用和分发内容的方式使用了。添加对 Filecoin 的支持将允许这些系统能够以交易 Filecoin 通证的方式来保证 IPFS 上存储的内容。
- **其他平台在 Filecoin 上：**我们计划提供桥接系统以链接 FIlecoin 和其他区块链的服务。例如，与 Zcash 的集成将支持发送隐私存储数据的请求。

# 8. 未来的工作

这项工作为 Filecoin 网络的构建提供了一条清晰而有凝聚力的道路；然而，我们也认为这项工作是未来去中心化存储系统研究的起点。在本章中，我们确定并填充了三类未来的工作。这包括已完成的工作，它们只是等待描述和发布、提出改进当前协议的开放式问题以及协议的形式化等方面。

## 8.1. 正在进行的工作

以下主题代表正在进行的工作

- 一个描述每个区块中 Filecoin 状态树的规范
- Filecoin 及其组件的详细性能评估和基准
- 一个完全可实现的 Filecoin 协议规范
- 一个赞助检索票务模型，其中通过发行每片可承载且可花费的通证，任何客户 C1 都可以赞助另一个客户 C2 的下载
- 一个分层共识协议，其中 Filecoin 子网可以在临时或永久分区进行分区操作或继续处理交易。
- 使用 SNARK/STARK 增加区块链快照
- 基于以太坊的 Filecoin 接口合约和协议
- 与 Braid 的区块链归档和跨区块链盖戳
- 仅在解决冲突的时候在区块链上发布时空证明
- 正式证明 Filecoin DSN 和新型存储证明的实现

## 8.2. 开放性问题

这里有许多未解决的问题，尽管他们不会耽误上线时间，但其答案有可能大大改善整个网络。

- 一个描述复制证明密封功能更好的原语，其中理想情况下是 O(n)解码（而不是 O(nm)解码），并且无需 SNARK/STARK 就可以公开验证
- 一个描述复制证明 Prove 功能更好的原语，其中无需 SNARK/STARK 就可以公开验证且透明
- 一个可以公开验证且透明的检索证明或是其他的存储证明
- 检索市场中针对检索的新策略（例如，基于概率支付、零知识条件支付）
- 一个更好的期望共识秘密领袖选举，其中每一个纪元只有一个领袖当选
- 一个更好的 SNARKs 可信设置方案，允许公共参数的增量扩展（可以运行一系列 MPCs 的方案，其中每个额外的 MPC 严格降低故障概率，并且每个 MPC 的输出可用于系统）

## 8.3. 证明和正式验证

由于证明和正式验证的明确价值，我们计划证明 Filecoin 网络的很多属性，并在未来几个月和几年内开发正式验证协议的规范。我们已经有了一些正在进行中的证明，还有更多的还在思考中。但是要证明 Filecoin 的很多属性（如缩放和离线）将是艰难且长期的工作。

- 预期协议和变体的正确性证明

- 功率容错的正确性证明，其中异步 1/2 将不会导致分叉
- 在通用组合框架中制定 Filecoin DSN，将 Get、Put 和 Manage 作为理想的功能来描述，并证明我们的确实现了它
- 自动自愈保证的正式模型和证明
- 正式地验证协议描述（例如 TLA+或者 Verdi）
- 正式验证实现（例如 Verdi）
- 对 Filecoin 激励的博弈论分析

# 致谢

这项工作是协议实验室团队中多个人的累计工作，如果没有协议实验室的合作者和顾问的帮助、评论和审查是不可能完成的。Juan Benet 在 2014 年书写的原始的 Filecoin 白皮书为这项工作奠定了基础。他和 Nicola Greco 开发了新的协议并且与提供了有用的贡献、评论、审查和交流的剩余团队成员合作写完了这份白皮书。特别是，David “Davidad” Dalrymple 提出了订单簿范例和其他想法，Matt Zumwalt 改进了白皮书的结构，Evan Miyazono 创作了插图并完成了论文，Jeromy Johnson 在设计协议时提出了深刻见解，Steven Allen 提出了富有洞察力的问题和说明。我们同样也感谢所有的合作者和顾问们有用的交流；尤其是 Andrew Miller 和 Eli Ben-Sasson。

上一个版本：QmZ5fLHwK9d55iyxpke6DJTWxsNR3PHgi6F43jSUgEKx31

# 参考文献


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

[2] 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.

[3] 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*, pages 584–597. Acm, 2007.

[4] 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.

[5] Protocol Labs. Technical Report: Proof-of-Replication. 2017.

[6] 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.

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

[8] 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.

[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] Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany, 1999.

[11] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2015.

[12] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. *arXiv preprint arXiv:1702.05812*, 2017.

[13] Protocol Labs. Technical Report: Power Fault Tolerance. 2017.

[14] Protocol Labs. Technical Report: Expected Consensus. 2017.

[15] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y. *ACM SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.

[16] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. 2016.

[17] Silvio Micali. Algorand: The efficient and democratic ledger. *arXiv preprint arXiv:1607.01341*, 2016.

[18] Vitalik Buterin. Ethereum <https://ethereum.org/>, April 2014. URL `https://ethereum.org/`.

[19] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[20] 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.