Filecoin：一个去中心式存储网络

（白皮书）

译：IPFS 原力区

目录

当今的互联网正处于一股浪潮之中：中心专有式服务正逐渐被去中心式服务所取代；中心式信任方逐渐被可验证式分布计算取代；脆弱的位置寻址逐渐被弹性的内容寻址取代；低效的整体服务逐渐被点对点算法市场取代。比特币、以太坊及其他区块链产品已经证明了去中心化交易分账的有效性。这些公共账本可以处理精密而智能的合同，以加密的方式交易价值数百亿美金的资产。这些系统是开放式互联网最早的实体，去中心化网络的参与者们在没有中心管理或中心式信任方的情况下，提供了很有用处的支付服务。IPFS 通过对全球性点对点网络所使用的数十亿文件提供服务，证明了去中心化网络中内容寻址的效用。

Filecoin 是一个去中心化的存储网络，它可以将云存储转变为算法市场。这个市场运作在一个拥有本地协议记号（也叫做“Filecoin”）的区块链上，在这个市场上，矿工们通过对客户提供存储服务赚取 Filecoin。相对地，客户可以使用 Filecoin 来雇佣矿工存储或分发数据。同比特币相似，Filecoin 矿工们会为了追求回报而竞相开采区块，但 Filecoin 的开采能力与存储积极性正相关，这可以为客户提供更有效用的服务（而不像比特币，为了维持区块链的一致性而限制其效用）。如此就激励了矿工们尽可能多地积累存储空间并租借给客户。本协议可以将积累起来的资源组织成任何人都可信赖的、有自愈功能的存储网络。这个网络通过复制和分发内容建立自身的鲁棒性，同时还可以自动侦测和修复复制错误。客户可以通过选择复制参数防范不同的风险类型。由于协议在客户方对内容进行了端对端加密，存储空间的提供者无法得到密匙，所以这种云存储网络可以提供足够的安全性。Filecoin 作为 IPFS 顶端的激励层，可以为任意数据提供存储架构，在保存去中心化数据、构建和运行分发应用以及执行智能合约的情况下格外有用。

本文包含以下内容：

- 介绍 Filecoin 网络，概述协议并详细介绍几个重要组件。
- 概述去中心化网络（DSN）的方案和特点，然后通过 Filecoin 构建一个 DSN。
- 基于存储证明方案，介绍一个名为“复制证明”的新方案，该方案可以使任意的复制数据储存在独立的物理空间中。
- 介绍一个基于复制证明和存储的新型可工作的一致性作为强度的度量。
- 建立可验证市场概念并构建两个市场：存储市场和检索市场。他们分别管理写入和读取来自 Filecoin 的数据。
- 讨论应用场景、与其他系统的连通性以及如何使用协议。

1. 介绍

Filecoin 是一种协议标记，它是一个运作在叫做“时空证明”的新型证明上的区块链，在这种协议上，矿工通过存储数据来创造区块。Filecoin 通过一系列相互独立的存储提供商来提供存储和读取服务，而非通过单一的协调器。其中：（1）客户通过支付 Filecoin 来存储和读取数据。（2）存储矿工通过提供存储服务获得 Filecoin。（3）检索矿工通过提供数据获得 Filecoin。

1.1 基本构成

Filecoin 协议由以下四个新型组件构成：

- 去中心化存储网络（Decentralized Storage Network, DSN）：我们通过独立的存储提供商构成的抽象网络来提供存取服务（详见第 2 节）。而后我们会阐述 Filecoin 协议是一个有激励性的、可审计的、可被证实的 DSN 架构（详见第 4 节）。
- 新型存储证明：我们会介绍两种新兴的存储证明（Proofs-of-Storage）（详见第 3 节）：（1）复制证明（Proof-of-Replication）允许存储提供商来证明数据已经被复制到单一的物理存储器上。强制性单一物理拷贝可以检查和确保提供商没有将多余拷贝放到同一存储器。（2）时空证明（Proof-of-Spacetime）允许存储提供商来证明他们在指定的时间内存储了某些数据。
- 可验证市场（Verifiable Markets）我们在两个基于 Filecoin 网络的去中心可验证市场上对存储请求和检索请求进行了建模（详见第 5 节）。可验证市场可以确保当服务被正常提供的时候执行支付操作。我们还会展示矿工和客户可以分别独立提交存储和检索命令的存储市场和检索市场。
- 有效的工作量证明（Proof-of-Work）：我们会展示如何在时空证明的基础上构建一个可以用在共识协议上的有效工作证明。矿工们不需要浪费计算能力来开采区块，他们只需要在网络中对数据进行存储即可。

1.2 协议概览

- Filecoin 是一个建立在区块链上的，拥有本地记号的去中心化存储网络架构。客户通过消费这些记号存储和读取数据，矿工通过存储和提供数据赚取标记。
- Filecoin DSN 分别通过两个可验证市场进行读取请求：存储市场和检索市场。客户和矿工协商服务定价，而后将将订单上传到市场。
- 这些市场由 Filecoin 网络来运作，Filecoin 网络通过时空证明和复制证明来确保矿工可以执行承诺，正确地存储数据。
- 最后，矿工们可以参与到新区块的建造中。矿工在新区块中的影响力与他们在网络中提供的存储量正相关。

1.3 白皮书组织

- 第二节中我们展示我们在理论上对 DSN 网络的定义和要求。

- 第三节中我们发展、定义并展示我们的复制证明和时空证明协议，并按照交易约定，使用 Filecoin 来加密地核实数据被不断地存储。
- 第四节描述了精确的 Filecoin DSN 实例，包括数据结构、协议以及参与者交互。
- 第五节中我们对可验证市场的概念进行了定义和描述，以及他们的概念实现—存储市场和检索市场。
- 第六节中描述了时空协议的使用，并展示了如何评估矿工对网络的贡献，这对扩大区块链以及分发区块回报至关重要。
- 第七节简要描述了 Filecoin 中的智能合约。第八节我们对未来工作做了一些讨论来作为总结。

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 Δ_{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_{deal}

1. 签署 o_{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_{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 Δ_{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_{bid} 中制定的尺寸
2. 创建 o_{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

图一 Filecoin 协议草图

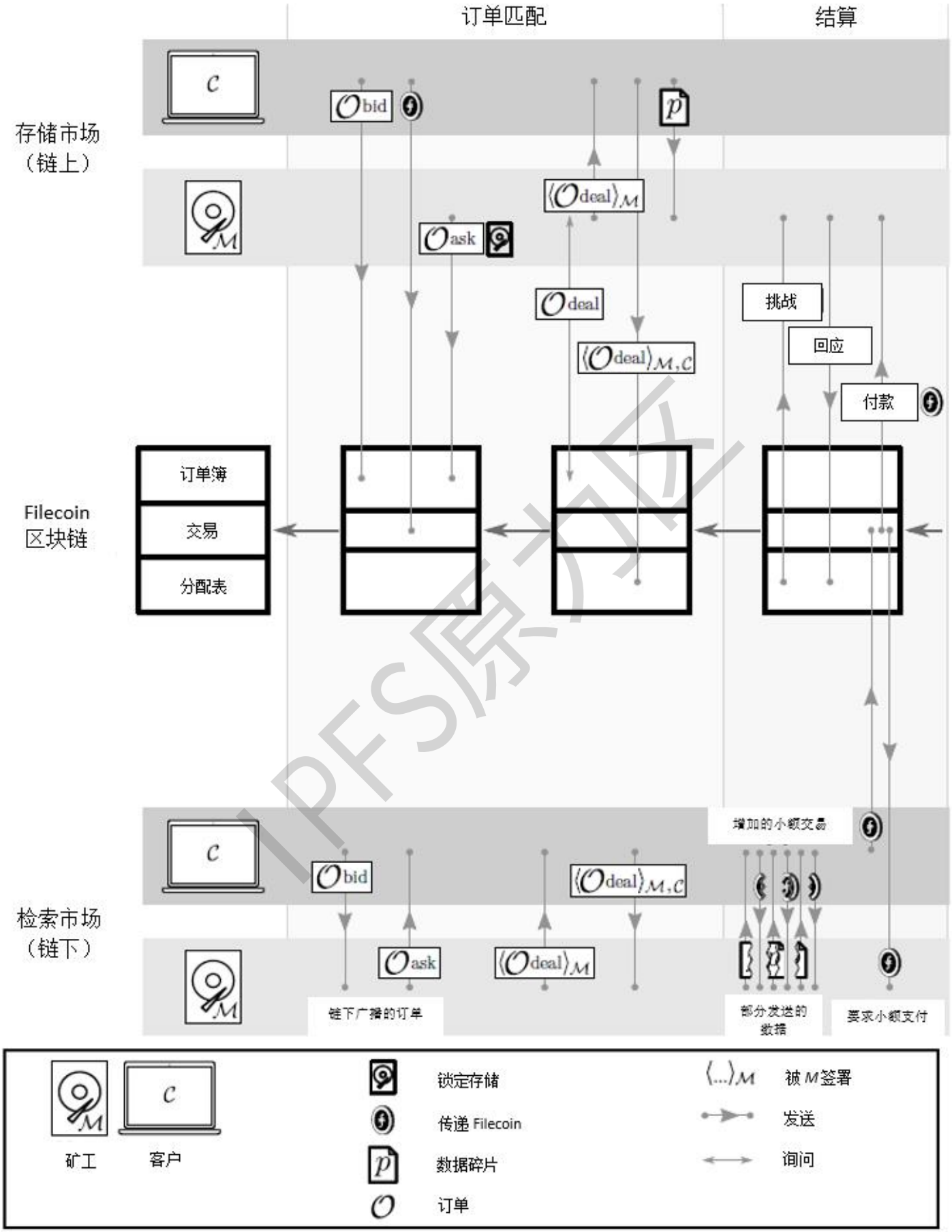
订单匹配
结算
存储市场
（链上）
Filecoin
区块链
检索市场
（链下）
订单簿
交易
分配表
挑战
回应
付款
增加的小额交易
链下广播的订单
部分发送的
数据
要求小额支付
矿工
客户
锁定存储
传递 Filecoin
数据碎片
订单
被 M 签署
发送
询问

图 2 Filecoin 协议插图

图 2 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储存市场和检索市场，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

本节中我们要介绍去中心化存储网络（DSN）的概念。 DSNs 汇聚来自众多独立提供者的存储能力，并通过自我协调的方式为客户提供数据读取服务。这种协调是去中心化的，不需要中心式信任方的参与：协议通过协调和查证独立个体的操作来实现系统的安全运行。根据系统要求，DSNs 可以调用不同的协调策略，包括拜占庭协议（Byzantine Agreement），流言算法（Gossip Protocols）或者 CRDTs 等。稍后在第四节中，我们会提供 Filecoin DSN 的具体架构。

【定义 2.1】DSN 方案 Π 是由存储提供商和客户运行的协议元组:

(Put; Get; Manage)

- Put(data)→ key: 客户执行 Put 协议，在特定的身份密匙（key）下存储数据（data）。
- Get(key) → data: 客户执行 Get 协议，读取利用密匙存储的数据。
- Manage(): 各参与者形成的网络通过 Manage 协议进行协作，以此来控制可用的存储空间，审查存储提供者供应的服务，以及修复一些可能存在的错误。Manage 协议由存储提供商（通常也有客户参与）或者审查网络来运作 [1]。DSN 方案Π必须确保数据的完整性和可恢复性，以及在管理和存储中的容错性。这些概念会在后文中给出定义。

2.1 容错

2.1.1 管理错误

我们将存储故障定义为 Manage 协议参与者导致的拜占庭错误（Byzantine Faults）。基于 Manage 协议的 DSN 方案应当具有容错性。如果管理错误违反了容错性假设，可能会影响系统的活跃度和安全性。

举例来说，假设有一个 DSN 方案Π，它的 Manage 协议需要拜占庭协议（Byzantine Agreement，BA）来审核存储提供者。在这样一个协议中，网络从存储提供者那里收到存储证明，然后运行 BA 来验证这些有效性。假节点总数为 n，BA 最多可以容纳 f 个错误节点，那么我们的 DSN 可以容忍的故障节点数为 f<n/2 个。如果违反这些假设，审计就要做出妥协。

2.1.2 存储错误

我们将存储错误定义为阻止用户获取数据的拜占庭错误。也就是说，这些错误可能是存储矿工丢失了一些数据片段，或是检索矿工在某些片段中停止了服务。一个成功的 Put 操作应当是(f,m)-tolerant

的形式—如果输入的数据被储存在 m 个独立存储提供者处（总数为 n），可以容纳的拜占庭提供者最多为 f。f 和 m 两个参数取决于协议的实现。协议的设计者可以自行设定这两个参数，也可以把选择交给用户，将 Put(data)扩展为 Put(data, f, m)。在 Get 命令中，如果错误的存储提供者数量小于 f，该命令就可被成功执行。

举例来说，设想一个简单的方案，其中 Put 协议被设计为每一个存储提供者都要存储所有的数据。在这个方案中，m=n，f=m-1。但是 f=m-1 永远成立吗？不，有些方案可能会采用可擦除代码设计，这样的话每个存储提供者都储存整体数据的一部分，这样的话如果 m 个存储供应者中有 x 个需要检索数据，就有 f=m-x。

2.2 特性

这里我们价绍 DSN 方案应有的两种特性，以及 Filecoin DSN 所具备的其他特性。

2.2.1 数据完整性

该属性要求没有任何有限对手(bounded adversary) A 可以使客户在 Get 操作结束的时候接受被更改或伪造的数据。

【定义 2.2】DSN 方案Π在以下情况中提供数据完整性：对任意在密匙 k 下的数据 d 进行成功的 Put 操作，都没有可计算边界的对手(computationally-bounded adversary) A 可以让客户在对密匙 k 执行 Get 操作结束的时候接受 d’，其中 d’≠ d。

2.2.2 数据恢复性

该属性要求：给出的Π具有容错假设，如果有些数据已经被成功存储在Π中，并且存储提供者持续遵循协议，那么客户最终应当能够检索到数据。

【定义 2.3】一个具有可恢复性的 DSN 方案Π要求：如果在密匙下的数据有任意成功的 Put 操作，其同样存在一个成功的 Get 操作能让客户检索到数据[1]。

2.2.3 其他特性

DSNs 可以针对具体应用提供特定的其他属性。本文定义了 Filecoin DSN 所需要的三个关键属性：公开可验证性、可审查性和激励兼容性。

【定义 2.4】 对于每个成功的 Put 操作，存储网络提供者可以生成数据已被存储的证明。这个存储证明必须说服任何知道密匙但无法访问其对应数据的有效审核人。满足这种情况的 DSN 方案Π是公开可验证的。

【定义 2.5】 如果方案能够生成可验证的操作记录，并且该记录在未来可以被查证数据的确在特定的时间内被有效存储，那么一个 DSN 方案Π是可审查的。

【定义 2.6】 如果存储提供者可以通过存储和检索数据而获得回报，或者因为作弊而得到惩罚一体并且所有存储提供者的最优策略就是存储数据。那么一个 DSN 方案Π是激励可兼容的。

3. 复制证明与时空证明

在 Filecoin 协议中，存储供应商必须让客户相信，他们的数据已经被完好存储。在实际操作中，存储供应商会生成存储证明（Proofs-of-Storage，POS）给区块链网络（或客户本身）进行验证。

本节中，我们将展示和概括在 Filecoin 中所使用的复制证明（Proof-of-Replication，PoRep）和时空证明（Proof-of-Spacetime，PoSt）实现方案。

3.1 动机

存储证明（POS）方案，例如可证明数据拥有（PDP）以及可恢复性证明（PoR）方案，它允许用户（即审核人 V）将数据外包给服务器（既证明人 P）以检查服务器是否依然存储数据 D。用户可以用比下载数据还便捷的方式，来验证外包给服务器数据的完整性。服务器通过对一组区块进行随机采样和传送少量数据，来生成概率性的拥有证明，以此作为给用户的怀疑/响应协议。

PDP 和 PoR 方案只允许证明人在怀疑/响应的时候拥有某些数据。在 Filecoin 中，我们需要更强大的保障，来阻止恶意矿工通过攻击获得不应得的奖励。常见的三种攻击类型有：女巫攻击（Sybil attack）、外包攻击（outsourcing attacks）、生成攻击（generation attacks）。

- 女巫攻击：恶意矿工可能通过创建多个女巫身份，假装存储了很多拷贝（并从中获取奖励），但实际上只存储了一次。
- 外包攻击：由于可以快速地从其他存储提供商获取数据，恶意矿工可能提交比他们实际的物理存储容量更大的存储能力。
- 生成攻击：恶意矿工会宣称要存储大量的数据，但相反，他们使用一些小体积程序有效地生成请求。如果这个程序体积小于他们所宣称的储存容量，就会导致恶意矿工在 Filecoin 获取区块奖励的可能性增加。奖励本身是应该和当前的存储量正相关的。

3.2 复制证明

复制证明（PoRep）是一个新型的存储证明，它允许服务器（证明人 P)向用户（审核人 V）证明：数据 D 已被复制到它专有的物理存储上了。我们的方案是一种交互式协议，证明人 P:（a）承诺存储数据 D 的 n 个不同的拷贝（独立物理拷贝），然后（b）通过怀疑/响应协议来使审核人 V 相信 P 确实已经存储了每个拷贝。我们目前已经知道，PoRep 在 PDP 和 PoR 方案中，阻止女巫攻击、外包攻击、生成攻击的能力会有所提高。

注意，*想要 PoRep 正式的定义，以及了解它的属性和深入研究的，我们推荐参考[5]*

【定义 3.1】 PoRep 方案允许有效的证明人 P 来说服审核人 V：数据 D 的独立物理副本 R 已被 p 存储。PoRep 协议是多项式时间算法的元组：

(Setup, Prove, Verify)

- Setup(1^{λ}, D) → R, S_P, S_V, 其中 S_P 和 S_V 是为方案专门设定的 P 和 V 变量，λ是一个安全参数。PoRep.Setup 用来生成副本 R，以及给予 P 和 V 必要的信息来运行 PoRep.Prove 和 PoRep.Verify。一些方案可能会要求证明人或者第三方交互来运算 PoRep.Setup。
- Prove(S_P, R, c) → π^c，其中 c 是审核人 V 发出的随机质疑， π^c 是证明人产生的访问 R 的证明，R 是 D 的特定副本。PoRep.Prove 由 P 运行，为 V 生成πc。
- Verify(S_V, c, π^c)→ {0, 1}，用来检测证明是否正确。PoRep.Verify 由 V 运行，并使 V 相信 P 已经存储了 R。

3.3 时空证明

存储证明方案允许用户检查存储提供商当时是否存储了外包数据。我们如何使用 PoS 方案来证明在一段时间内数据被存储了？一个非常自然的答案是要求用户重复（例如，每分钟）对存储提供商发送请求。然而这样的交互所需要的通信复杂度会成为一些系统的瓶颈。像 Filecoin 这样，存储提供商会被要求提交证明到区块链网络。

为了解决这个问题，我们介绍一个新的证明，时空证明（Proof-of-Spacetime），它可以让审核人检查存储提供商是否在一段时间内存储了他的外包数据。这就对证明人产生了直观的要求：（1）生成连续的存储证明（在本文中是“复制证明”）来作为确定时间的一种方法。（2）递归执行来生成简单的证明。

【定义 3.2】 （时空证明）Post 方案可使有效的证明人 P 能够说服一个审核人 V 相信 P 在一段时间内存储了数据 D。PoSt 是多项式时间算法的元组：

(Setup, Prove, Verify)

- Setup(1^{λ}, D) → R, S_P, S_V, 其中 S_P 和 S_V 是为方案专门设定的 P 和 V 变量，λ是一个安全参数。PoRep.Setup 用来生成副本 R，以及给予 P 和 V 必要的信息来运行 PoRep.Prove 和 PoRep.Verify。一些方案可能会要求证明人或者第三方交互来运算 PoRep.Setup。
- Prove(S_P, R, c) → π^c，其中 c 是审核人 V 发出的随机质疑， π^c 是证明人产生的访问 R 的证明，R 是 D 的特定副本。PoRep.Prove 由 P 运行，为 V 生成πc。
- Verify(S_V, c, π^c)→ {0, 1}，用来检测证明是否正确。PoRep.Verify 由 V 运行，并使 V 相信 P 已经存储了 R。

3.4 PoRep 和 PoSt 实际应用

我们对 PoRep 和 PoSt 在现有系统上的应用很感兴趣，希望能构建一个实用性系统，但不依赖任何的中心式第三方或者硬件。我们给出了一个 PoRep 架构（见“密封的复制证明”[5]），它需要在 Setup 过程中执行缓慢的顺序计算密封来生成副本。PoRep 和 PoSt 的协议草图详见图 4，Post 证明步骤的底层机制的见图 3。

3.4.1 构建加密区块

防碰撞散列。我们使用一个防碰撞散列函数：CRH : {0, 1}* → {0, 1}$^{O(\lambda)}$。以及另一个一个防碰撞散列函数 MerkleCRH，它可以将字符串分割成多个部分，构造二叉树并将递归应用到 CRH，最后输出根。

zk-SNARKs。我们对 PoRep 和 PoSt 的实现依赖于零知识的简洁非交互式知识论（zero-knowledge Succinct Noninteractive ARguments of Knowledge，zk-SNARKs)[6,7,8]。因为 zk-SNARKs 非常简洁，证明短并且容易验证。形式上来说，就是让 L 作为 NP 语言，并用 C 做为 L 的判决电路。中心式信任方执行一次设置会产生两个公共密钥：证明密钥 pk 和验证密钥 vk。证明密钥 pk 可使任何（非信任的）的证明人都能生成证明π来证明 x∈L，x 是任一实例。非交互式证明π是零知识的，也是知识证明的。任何人都可以使用验证密钥 vk 来验证π证明。尤其 zk-SNARK 的证明是可公开验证的：任何人都可以验证π，而不需要与生成π的证明者进行交互。π证明具有恒定的大小，并且可以在| x |的线性时间内得到及时验证。

zk-SNARKs 是一个三项式时间算法：

(KeyGen, Prove, Verify)

- KeyGen(1λ,C)→(pk, vk)。输入安全参数为λ，回路为 C，pk 和 vk 是 KeyGen 的概率样本。这两个密匙是公共参数，可用于证明/审核 Lc 上的成员。
- Prove(pk, x, w)→π。在输入 pk、输入 x 并看到 NP 声明 w 后，如果 x∈LC，证明人 Prove 会输出非交互式证明π。
- Verify(vk, x,π)→{0, 1}。当输入 vk，x 和证明 π，如果有 x∈LC，审核人 verifier 输出 1。

建议有兴趣的读者阅读[6，7，8]，那里有对 zk-SNARK 系统的概念和实现更详细的介绍。通常来说系统会要求 KeyGen 是由中心式可信任参与方来运行的。新型可扩展计算的完整性和隐私性（SCIP）系统[9]展示了一个很有前景的、可以避免这个步骤的发展方向，才有了上面的信任假设。

3.4.2 密封操作

密封操作的作用是：

（1）通过要求证明人存储公钥下数据 D 的伪随机序列，强制副本成为成为相互独立的拷贝，这样提交 n 个副本后就会产生 n 个独立的磁盘空间（并占用副本大小 n 倍的储存空间）。

（2）在运行 PoRep.Setup 的时候强制生成副本，实质上会比预估的质疑响应花费更多的时间。有关密封操作的更正式定义，请参见[5]。上述的操作可以用 Seal$^{\tau}$AES−256 实现，Seal$^{\tau}$AES−256 中的τ

需要比常规的“质疑-证明-审核”序列多花费 10-100 倍的时间。所以对τ的选择非常重要的，因为运行 Seal[τ]BC 可能比证明人随机访问 R 要花费更多时间。

3.4.3 PoRep 构建实践

本节将描述 PoRep 协议的组成，并在图 4 展示简化的协议草图。我们略过了具体实现方法和优化细节。

创建副本。Setup 算法通过密封操作，和正确生成副本的证明，来实现副本的生成。证明人生成副本，并将输出（不包括 R）发送给审核人。

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 π_{SEAL}

证明存储。Prove 算法生成副本存储的证明。证明人收到来自审核人的随机质疑 c。该审核人在树根为 rt 的 Merkle 树 R 中确定叶子节点 Rc。证明人生成关于 Rc，和其延伸到叶子 Rc 的 Merkle 路径的知识证明。

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 π_{POS}

审核证明。考虑到副本的 Merkle 树根，和原始数据的散列，Verify 算法会审核存储证明的有效性。证明是公开可验证的：分布式系统的节点保持了账本和客户对特定数据的关注，这样就能验证这些证明。

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 PoSt 构建实践

本节将描述 Post 协议架构，并在图 4 中给出一个简单协议草图。本节忽略实现过程和优化细节。Setup 和 Verify 算法和前文的 PoRep 架构相同，所以这里只描述 Prove。

证明时空。Prove 算法为副本生成时空证明。证明人接收来自于审核人的随机质疑，并顺序生成复制证明，证明输出后，经过特定次数的迭代 t，可作为其他的输入（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π_{POST}

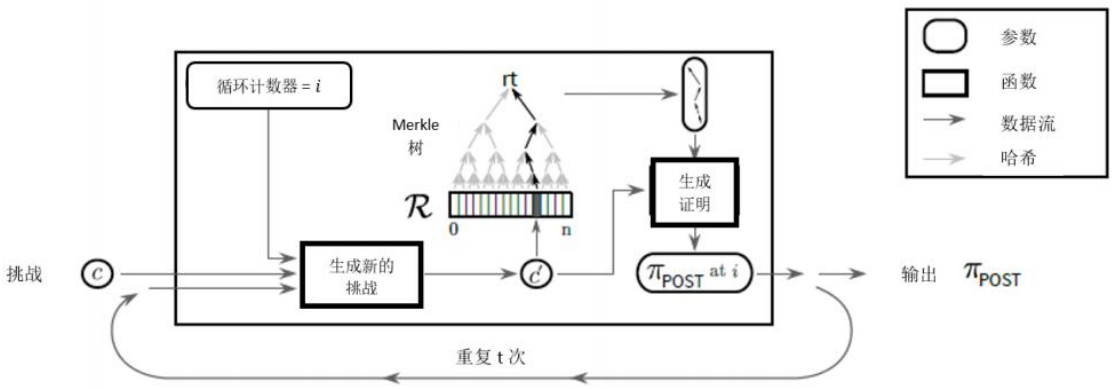

图 3 Post.Prove 的基础机制图示显示了随时间推移存储的迭代证明

3.5 在 Filecoin 中的应用

Filecoin 协议采用时空证明来审核矿工提供的存储。因为没有指定的审核人，并且我们希望网络中的任何成员都有审核权，所以为了在 Filecoin 中使用 PoSt，我们把方案改为了非交互式的。我们的审核人在公开的代币模型中运行，所以我们可以从区块链中随机地发出质疑。

[1]这个定义并不保证每个 Get 操作都是成功的，但如果每次 Get 操作最终都能返回数据，那这个方案就是可行的。

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 π_{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 π_{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 π_{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 π_{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，π_{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π_{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 $\pi_{\mathrm{POS}} \coloneqq \mathrm{PoRep.Prove}(\mathrm{pk}_{\mathrm{POS}}, \mathcal{R}, \mathrm{c}')$
   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π_{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协议草图这里 CRH 表示了防碰撞的哈希，$\bar{x}$ 是等待证明的 NP 声明，$\bar{w}$ 是证人。

4. Filecoin:一个 DSN 架构

Filecoin DSN 是一个建立在激励机制上，可审计、可查证的去中心化存储网络。客户向矿工支付代币来获取数据存储和检索数据服务，矿工提供存储空间和传输带宽来获得回报。矿工们只有当网络能够查证他们在正确提供服务的时候才能获得报酬。

在本节中，我们通过 DSN 的定义和“时空证明”来梳理 Filecoin DSN 架构。

4.1 设置

4.1.1 参与者

任何用户都可以成为、存储矿工和/或检索矿工。

- 客户在 DSN 中通过 Put 和 Get 请求存储或检索数据，并支付代币。
- 存储矿工为网络提供数据存储。存储矿工通过提供磁盘空间和响应 Put 请求来参与 Filecoin 运作。要想成为存储矿工，用户必须用与存储空间成比例的抵押品来抵押。存储矿工通过在特定时间内存储数据，来响应用户的 Put 请求。存储矿工生成时空证明并提交到区块链网络，来证明他们在特定时间内存储了数据。如果数据失效或丢失，存储矿工将被罚没部分抵押品。存储矿工也可以挖掘新区块。如果挖到了新区块，矿工就能获得挖取新块的奖励和新区块中的交易费用。
- 检索矿工为网络提供数据检索服务。检索矿工通过提供用户 Get 请求所需要的数据来参与 Filecoin 运作。和存储矿工不同，他们不需要抵押品，不需要提交存储数据，也不需要提供存储证明。存储矿工同样可以担任检索矿工。检索矿工可以直接从客户或者从检索市场赚取收益。

4.1.2 网络 N

我们将运行在 Filecoin 各个节点上的全体用户抽象为一个实体：网络。这个网络作为中介运行整个管理协议。简单来说，区块链中的每个新区块，都由全节点构成的网络来管理。网络可以管理可用存储，验证抵押品，审核存储证明以及修复可能存在的故障。

4.1.3 账本

我们的协议应用在基于账本的货币上。一般来说，我们管这个叫账本 L。在任意给定的时间 t（也叫时期）内，所有的用户都能访问 t 时期的账本 L_t，它是这段时期内交易序列的记录。账本只能被追加，不能被篡改。Filecoin DSN 协议可以运行在任意可验证 Filecoin 证明的账本上。在第六节中我们会展示了如何基于有效工作建立一个账本。

4.1.4 市场

存储的需求和供给在两个 Filecoin 市场相遇：存储市场和检索市场。这两个市场属于去中心化交易所，我们第 5 节中对其进行阐述。简而言之，客户和矿工们通过向各自的市场提交订单来为服务定价。交易所为客户和矿工们提供了匹配交易和牵线的方法。运行管理协议后，如果服务请求被成功提供，网络会确保矿工得到奖励，客户得到服务。

4.2 数据结构

片段

片段指客户在 DSN 所存数据的某一部分。例如，数据可以被随意划分为许多片段，每个片段可以由不同的存储矿工来存储。

扇区

扇区指存储矿工向网络提供的磁盘空间。矿工将客户的数据片段存储到扇区，并以此赚取代币。为了存储片段，矿工们必须向网络抵押他们的扇区。

分配表

分配表是一个数据表，记录数据片段的流向和其分配的扇区。分配表在账目下的区块中都会更新，它的 Merkle 根存储在最新的区块中。在实际操作中，该表用来维持 DSN 的状态，证明验证的过程中能够保证快速查找。详细内容请参考图 5。

订单

订单是请求或提供服务的声明。客户向市场提交出价来请求服务（分别在存储数据的存储市场和检索数据的检索市场），矿工们提交应答订单来提供服务。订单数据结构如图 10 所示。市场协议将在第 5 节详细介绍。

订单簿

订单簿是订单的集合。存储市场订单簿请查看第 5.2.2 节，检索市场订单簿第 5.3.3 节。

抵押

抵押是向网络提供存储（特别是扇区）的承诺。想要在存储市场接受订单，存储矿工必须将抵押提交给总账。抵押包括了抵押扇区的大小和存储矿工的抵押品（详见图 5）。

数据结构

Pledge 抵押

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

- 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

本节中我们通过对客户、网络和矿工的介绍，对 Filecoin DSN 有了一个整体的概括。我们在图 7 中给出了 Get 和 Put 协议的方法，在图 8 中给出了 Manage 协议，并用图 6 给出了一个协议实例。Filecoin 整体的协议概览请参见图 1。

4.3.1 客户周期

本节会简要介绍客户周期的概念；下文中各协议的深入解释请参考第五节。

1. **Put：客户在 Filecoin 中存储数据。**

客户通过向矿工支付代币可以对数据进行存储。Put 协议在节中有详细介绍。

客户向存储市场的订单簿发起投标（通过向区块链提交订单），就启动了 Put 协议。当有匹配的矿工应答时，客户就可以将数据片段发送给矿工。双方签署交易订单，并将其发送到存储市场的订单簿。

客户应当能够通过提交多重订单（或者在订单中指定复制扇区）来决定数据的拷贝数量。更高的冗余度可以提高储存的容错率。

1. **Get：客户从 Filecoin 中取回数据。**

客户可以通过向检索矿工支付代币来获取 DSN 中的任何数据。Get 协议在 5.3 节中有更详细的解释。

客户向检索市场的订单簿投标（向网络提交订单），就开始了 get 协议。当有匹配的矿工应答被找到，客户就会从矿工处得到数据片段。收到片段后，双方签署交易协议并提交到区块链，证明交易已完成。

4.3.2 挖矿周期（对存储矿工）

我们简单介绍一下挖矿周期。

1.抵押：存储矿工向网络抵押存储。

存储矿工通过 **Manage.PledgeSector** 在区块链中存放抵押品，来保证向网络提供稳定的存储。抵押品为了保证服务而存在，如果矿工为所存储的数据生成了存储量证明，抵押品就会被退回。如果没有成功生成存储量证明，矿工就得不到抵押品了。

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存。矿工们设置价格，并响应市场订单簿中的订单要求。

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（在 AllocTable 中），矿工就能在存储市场中提供存储。他们设定价格并通过 Put.AddOrders 向市场订单簿响应订单。

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

通过 **Put.MatchOrders** 来对客户的报价订单进行匹配。

Put.MatchOrders
- 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 $O^1 \dots O^n$

一旦订单匹配，客户就将数据发给存储矿工。存储矿工接收到数据后，运行 **Put.ReceivePiece** 。数据接收完成后，矿工和客户签署交易订单并提交到区块链。

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：存储矿工为数据片段准备未来证明。

存储矿工的存储空间被切分为很多扇区，每个扇区包括了分配给矿工的数据片段。网络通过分配表来记录每个存储矿工的扇区。当一个扇区被填满了，这个扇区就被密封（sealed）起来。密封是一种缓慢的顺序操作。密封可以将扇区中的数据转换成为唯一的物理副本，并与存储矿工的公钥相关联。密封在复制证明中是一项必须的操作，相关问题可参见第 3.4 节。

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 π_{SEAL}，一个树根哈希 rt

4.证明：存储矿工证明他们正在存储数据。

当存储矿工被分配了数据时，必须重复生成复制证明来确保他们正在存储数据（详情参看第 3 节）。证明被发布在区块链中，并由网络来验证。

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 π_{POS}

4.3.3 挖矿周期（对于检索矿工）

本节简要概括检索矿工的挖矿周期。

1.收到订单：检索矿工从检索市场得到获取数据的请求。

检索矿工设置价格并发送到市场订单簿，并通过向网络发送报价。

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

然后检索矿工检查是否与客户的订单报价匹配。

Get.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 $O^1 \ldots O^n$

2.发送：检索矿工向客户发送数据碎片。

一旦订单匹配，检索矿工就将数据发送给客户（第 5.3 节）。数据接收完成后，矿工和客户就签署交易订单提交到区块链。

Put.SendPieces
- 输入：
  --- 一个询价订单 O_{ask}
  --- 一个报价订单 O_{bid}
  --- 一个碎片 p
- 输出：一个由 M_i 签署的成交订单 O_{deal}

4.3.4 网络周期

本节给出简单的网络操作概述。

1.分配：网络将客户的数据片段分配给存储矿工的扇区。

客户通过向存储市场提交订单来启动 Put 协议。当询价单和报价单匹配的时候，参与各方共同为交易担保并向市场提交订单。此时，网络将数据分配给矿工，并将其记录到分配表中。

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

2.修复：网络发现故障并试图修复

所有的存储分配都是全网公开的。在每个区块中，网络都会检查储存分配的证明是否存在，以确保它们有效，并按照下列要求正确工作：

- 如果有部分证明丢失或失效，网络会罚没部分抵押品来惩罚存储矿工。
- 如果存在大量的证明丢失或失效（由系统参数Δfault 定义），网络会认定存储矿工存在故障，将订单设定为失效，并为这部分数据引入新订单，重新进入市场。
- 如果所有该数据的存储矿工都发生故障，则认定该数据丢失，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 $\mathcal{O}^1_{\text{deal}} \dots \mathcal{O}^n_{\text{deal}}$，更新分配表 allocTable

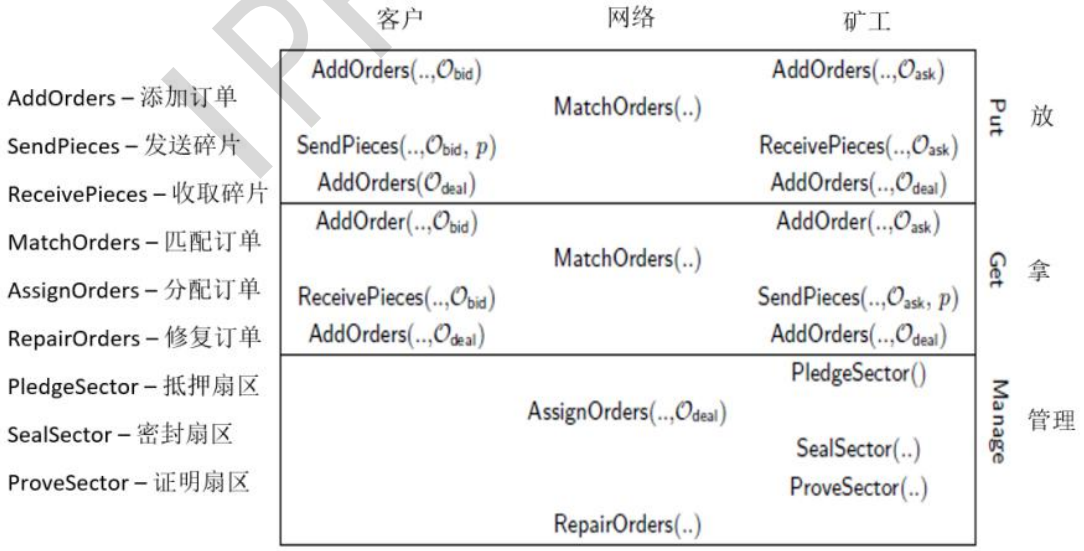

图 6 Filecoin DSN 的执行示例，按群分组并按照行及时间顺序排序

4.4 担保和要求

以下是关于 Filecoin DSN 如何实现完整性、可恢复性、公开可验证性和激励兼容性的相关条款。

- 完整性的实现：数据片段以加密的哈希值来命名，在 Put 请求后，客户只需要存储哈希即可通过 Get 操作来检索数据，并可对收到的数据进行完整性检查。
- 可恢复性的实现：在 Put 请求中，客户可以指定复制参数和纠删码的类型，还可以将存储方式指定为(f, m)-tolerant。它的意思是如果给定 m 个存储矿工存储数据时，最多可以容忍 f 个故障。使用更多的存储矿工进行存储，可以让客户增加数据恢复的机会，以防存储矿工下线或消失。
- 公开可验证和可审核性的实现：存储矿工需要提交其存储量证明 (π_{SEAL}, π_{POST})到区块链。网络中任意用户都可以在不访问数据本身的情况下，验证这些证明的有效性。由于这些证明都存储在区块链上，所有的操作过程都可以被随时查验。
- 激励兼容性的实现：一般来说，矿工通过提供存储数据获得回报。当矿工承诺存储数据时，他们必须生成证明。如果忽略了证明，矿工就会被惩罚（通过罚没部分抵押品），并且不会得到储存回报。
- 保密性的实现：如果客户希望维护数据的隐私性，就必须在提交到网络之前，对数据进行加密。

Put 协议

市场

AddOrders 添加订单

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

1) 设定 $\text{tx}_{\text{order}} := (O^1, \dots, O^n)$
2) 提交 tx_{order} 到 $\mathcal{L}$
3) 等待 tx_{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 $O^1 \dots O^n$

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 $O^i.\text{price} \geq O^q.\text{price}$
      iii) 检查大小 $O^i.\text{size} \leq O^q.\text{space}$
   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 $O^i.\text{price} \leq O^q.\text{price}$
      iii) 检查大小 $O^i.\text{space} \geq O^q.\text{size}$
2) 输出匹配订单 $O^1 \dots O^n$

交易

SendPieces 发送碎片

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

1) 从 O_{ask} 的签名中得到 $\mathcal{M}_i$ 的身份
2) 发送 $(O_{\text{ask}}, O_{\text{bid}}, p)$ 给 $\mathcal{M}_i$
3) 收到由 $\mathcal{M}_i$ 签署的 O_{deal}
4) 根据定义 5.2. 检查 O_{deal} 是否有效
5) 输出 O_{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_{ask}
  - 报价订单 O_{bid}
  - 碎片 p
- 输出：$\mathcal{C}_i$ 和 $\mathcal{M}_j$ 签署的成交订单 O_{deal}

1) 检查 O_{bid} 是否有效：
   a) 检查 O_{bid} 是否在订单簿中
   b) 检查 O_{bid} 没有被其他活跃的 O_{deal} 涉及
   c) 检查大小 $O_{\text{bid}}.\text{size}$ 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 $O_{\text{deal}} := \langle O_{\text{ask}}, O_{\text{deal}}, \mathcal{H}(p) \rangle_{\mathcal{M}_i}$
4) 从 O_{bid} 获得 $\mathcal{C}_j$ 的身份
5) 发送 O_{deal} 给 $\mathcal{C}_j$
6) 输出 O_{deal}

Get 协议

市场

AddOrders 添加订单

- 输入：订单目录 $O^1 \dots O^n$
- 输出：无

1) 向网络广播 $O^1 \dots O^n$

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 $O^1 \dots O^n$

1) 匹配订单簿中的每个 O^i，以至于：
   a) 检查 $O^i.\text{piece}$ 等于 $O^q.\text{piece}$
   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 $O^i.\text{price} \geq O^q.\text{price}$
   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 $O^i.\text{price} \leq O^q.\text{price}$
2) 输出匹配订单 $O^1 \dots O^n$

交易

SendPieces 发送碎片

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

1) 创建 O_{deal}：
   a) 设定 $O_{\text{deal}}.\text{ask} := O_{\text{ask}}$
   b) 设定 $O_{\text{deal}}.\text{bid} := O_{\text{deal}}$
2) 从 O_{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 π_j 设为从 $\mathcal{H}(p)$ 到 p_j 的 Merkle 路径
   b) 发送 $(O_{\text{deal}}, p_j, \pi_j)$ 到 $\mathcal{C}_i$
   c) 接收 $\langle O_{\text{deal}}, j \rangle_{\mathcal{C}_i}$
5) 输出 O_{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_{ask}
  - 一个报价订单 O_{bid}
  - 订单中 p 的 Merkle 树哈希 h_p
- 输出：一个碎片 p

1) 创建 O_{deal}
   a) 设定 $O_{\text{deal}}.\text{ask} := O_{\text{ask}}$
   b) 设定 $O_{\text{deal}}.\text{bid} := O_{\text{bid}}$
2) 从 O_{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 $(O_{\text{deal}}, p_j, \pi_j)$：
   a) 检查 O_{deal} 是有效并且匹配 O_{ask} 和 O_{bid} 的
   b) 用树根哈希 h_p 检查 π_j 是否为一有效 Merkle 路径
   c) 发送 $\langle O_{\text{deal}}, j \rangle_{\mathcal{C}_i}$
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 $O^1_{\text{deal}} \dots O^n_{\text{deal}}$
  - 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 O^i_{deal}：
   a) 根据定义 5.2. 检查 O^i_{deal} 是否有效
   b) 从 O^i_{deal}的签名中得到 $\mathcal{M}_j$ 的身份
   c) 将细节从 O^i_{deal} 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 $O^1_{\text{deal}} \dots O^n_{\text{deal}}$，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 π 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 $O^1_{\text{deal}} \dots O^n_{\text{deal}}$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 π_{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 $p_1 \dots p_n$
2) 设定 $\mathcal{D} := p_1|p_2|\dots|p_n$
3) 计算 $(\mathcal{R}, \text{rt}, \pi_{\text{SEAL}}) := \text{PoSt.Setup}(\mathcal{M}, \text{pk}_{\text{SEAL}}, \mathcal{D})$
4) 输出π_{SEAL}, 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 π_{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 π_{POST}

图 8 Filecoin DSN 中 Manage 协议的描述

5.Filecoin 的存储市场和检索市场

Filecoin 有两个市场：存储市场和检索市场。这两个市场结构相同但设计不同。存储市场让客户通过付费使得存储矿工储存数据。检索市场让客户向检索矿工付费来取回数据。这两种情况下，客户和矿工都可以设置报价或接受报价。整个交易是由网络来运行—Filecoin 中的所有节点构成了拟人化的网络。网络保证了矿工在提供服务时可以得到客户的奖励。

5.1 验证市场

交易市场是一种协议，它们通过促使买家和买家达成交易，促进特定商品和服务的交换。对我们而言，交易应当是可验证的：参与者们构成的去中心化网络必须能够在买家和卖家间对交易进行验证。

这里我们提出验证市场的概念。它没有单一的实体来管理交易，交易是完全透明的，任何人都可以匿名参与。可验证市场协议可以去中心化地完成服务和货品的交易：订单簿的一致性、订单的处理和服务的正确执行都可以由参与者，也就是 Filecoin 里面的矿工和全部节点，来进行独立验证。简化的可验证市场模型如下：

定义 5.1：可验证市场协议分两个阶段：订单匹配和处理。订单是购买或出售抵押品、商品或服务的解决方案，订单簿就是所有订单的集合列表。

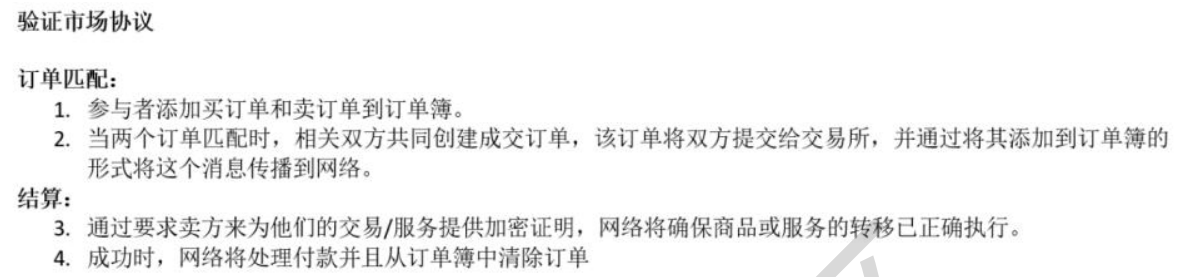

图 9 可验证市场的一般性协议

5.2 存储市场

存储市场是可验证的市场，它允许客户（即买家）请求存储数据以及存储矿工（即卖家）提供存储空间。

5.2.1 需求

我们根据以下需求来设计存储市场协议：

- **链式订单簿：**重点（1）存储矿工的订单公开的，所以最低价是透明的，客户可以对订单做出最优抉择。（2）客户订单必须提交给订单簿，即使他们选择了最低价格，这样市场就可以对新的订单做出反应。因此为了能被添加到订单簿，我们要求订单必须被添加到 Filecoin 区块链。
- **参与者投入抵押：**为了避损，防止存储矿工不提供存储或者客户没有支付能力，我们要求参与双方投入抵押资源。为了参与存储市场，存储矿工必须保证在 DSN 中存入与其存储量成比例的抵押品（更多细节请参考 3.3 节）。通过这种方式，网络对承诺存储数据但又不提供存储证明的矿工进行惩罚。同样的，客户必须预存一定数量的资金，以保证其支付能力。
- **故障自处理：**只有在存储矿工反复证明他们已经在约定时间内存储了数据的情况下，订单才会进行结算。网络必须能够验证这些证明的存在性和正确性，并且保证它们按 3.4 节修复部分所讲的规则运行。

5.2.2 数据结构

Put 订单 有三种类型的订单：出价订单，询价订单和交易订单。存储矿工创建询价订单来提供存储，客户创建出价订单以及请求存储，当双方谈妥价格时，他们共同创建交易订单。订单的数据结构和参数的细节如图 10 所示。

Put 订单簿 存储市场的订单簿是对当前情况有效的，并且对询价，出价和交易订单都是开放的。用户可以通过 Put 协议中的方法（AddOrders,MatchOrders）与订单簿进行交互。具体方法如图 7 所示。

订单簿是公开的，对所有用户都有同样的可见性。在每个周期，如果最新的区块中出现新的交易，新订单就会立即被添加到订单簿。如果订单被取消，失效或已被完成，就会被删除。如果订单是有效的，它将被添加到区块链中，也就同时被添加到订单簿：

定义 5.2：我们定义出价，询价，交易订单的有效性：

（有效出价单）：从客户发出的投标单 Ci,Obid:= (hsize, funds[, price,time, coll, coding])>Ci，如果满足下面的条件就是有效的：

- Ci 在账户里有可用的资金。
- 时间没有超时。
- 订单必须保证最少的存储周期（这是个系统参数）。

（有效询价单）：从存储矿工发出的询价单 Mi，Oask:= (hspace, pricei)Mi，如果满足下面的条件就是有效的：

- Mi 承诺做矿工，并且在订单周期内不会反悔。
- 空间必须小于 Mi 的可用存储：Mi 承诺的存储减去预留给订单簿的部分（在询价订单和交易订单中）。

（有效交易订单）：交易订单 Odeal:= (hask, bid,ts)Ci,Mj，如果满足下面的条件就是有效的：

- 询价（ask）参考这样的订单 Oask：它在存储市场的订单簿中储存，并不涉及其他订单。它由 Ci 签署。
- 出价（bid）订单参考这样的订单 Obid：它在存储市场的订单簿中储存，并不涉及其他订单。它由 Mj 签署。
- ts 不能设置为将来的时间，或者太为过去的时间

评论：如果恶意的客户从存储矿工收到了签名的交易，但从来没有将其添加到订单簿，那么存储矿工就无法重新使用交易中涉及的存储。ts 可以防止这种攻击，因为超过 ts 之后，订单就会失效，并无法在订单簿中进行提交。

存储市场订单

报价订单
$\mathcal{O}_{\text{bid}} \coloneqq \langle \text{size, funds}[, \text{price, time, coll, coding}] \rangle_{\mathcal{C}_i}$
- 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
$\mathcal{O}_{\text{ask}} \coloneqq \langle \text{space, price} \rangle_{\mathcal{M}_i}$
- 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
$\mathcal{O}_{\text{deal}} \coloneqq \langle \text{ask, bid, ts} \rangle_{\mathcal{C}_i,\mathcal{M}_i}$
- 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
$\mathcal{O}_{\text{bid}}$: $\langle \text{piece, price} \rangle_{\mathcal{C}_i}$
- 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
$\mathcal{O}_{\text{ask}}$: $\langle \text{piece, price} \rangle_{\mathcal{M}_i}$
- piece：被请求碎片的索引
- price：$\mathcal{M}_i$ 提供碎片的价格

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

图 10 检索和存储市场的命令数据结构

5.2.3 存储市场协议

简单来说，存储市场协议分两个部分：订单匹配和结算：

- 订单匹配：客户和存储矿工将交易提交到区块链，这样可以使订单同步到订单簿（步骤 1）。当订单匹配时，客户发送数据给存储矿工，双方签署交易协议并提交到订单簿（步骤 2）。
- 结算： 存储矿工将扇区密封（步骤 3a），为存储有数据片段的扇区生成存储证明，并定期将其提交到区块链（步骤 3b)；同时，其余的网络需要验证矿工生成的证明并修复可能存在的故障（步骤 3c）。

存储市场协议的详细情况请参考图 11。

5.3 检索市场

检索市场允许客户请求特定的数据，并由检索矿工提供该数据。与存储矿工不同的是，检索矿工不要求在特定时间内存储数据或者生成存储证明。网络中的任何用户都可以成为检索矿工，通过提供数据片段来赚取 Filecoin 代币。检索矿工可以直接从客户接收数据片段，也可以成为存储矿工存储它们。

5.3.1 需求

我们根据以下的需求来设计检索市场协议：

- **链下订单簿**：客户必须能够找到提供所需要数据片段的检索矿工，并在商定价格之后直接交换数据。这意味着订单簿不能通过区块链来运行—否则这将成为快速检索请求的瓶颈—不按此操作的参与者只能看到订单簿的一部分。因此，我们要求双方都参与订单。
- **无信任方检索**：公平交换不可能性的结果[10]提醒我们，交易双方不可能在没有信任方的情况下进行交换。在存储市场中，区块链网络作为（去中心化的）信任方来验证存储矿工提供的存储。在检索市场中，检索矿工和客户交换文件的时候并没有网络的监督。为了实现这种结果，我们要求检索矿工将数据分割成多个部分，每当客户收到一个片段时，矿工们会收到一定的报酬。这种方式中，遇到客户停止付款，或者矿工停止发送数据，任何一方都可以终止交易。值得注意的是，我们必须假设至少有一个检索矿工是诚实的。
- **支付通道**：客户希望付款时可以立即收到数据，而检索矿工只有在确认收到付款后才会提供数据片段。通过公共账本确认交易可能会成为检索请求的瓶颈，所以，我们必须依赖有效的链下支付。Filecoin 区块链必须支持快速的支付通道，只有在出现纠纷的情况下才使用区块链，来使交易高效无误地进行。通过这种方式，检索矿工和客户可以快速发送协议所要求的小额支付。在我们未来的工作中，包含了创建一个如[11,12]所述的支付通道网络。

5.3.2 数据结构

获取订单

检索市场中包含三种类型的订单：客户创建的出价订单 O_{bid}，检索矿工创建的询价订单 O_{ask}，以及存储矿工和客户达成的交易订单 O_{deal}。订单的数据结构如图 10 所示。

获取订单簿

检索市场的订单簿包含公开有效的出价订单，询价订单和交易订单。与存储市场不同，每个用户所能看到的订单簿都不同。因为订单在网络中传播，每个矿工和客户只会追踪他们所感兴趣那一部分。

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 π_{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 Δ_{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 Δ_{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 存储市场协议详情

5.3.3 检索市场协议

简单来说，检索市场协议分为两个部分：订单匹配和结算：

- **订单匹配**：客户和检索矿工通过广播将订单提交给订单簿（步骤 1）。订单匹配后，客户和检索矿工创建小额支付通道（步骤 2）。
- **结算**：检索矿工发送小部分的数据片段发送给客户，然后对每个碎片客户会向矿工发送签名收据（步骤 3）。检索矿工向区块链提交收据并获得回报（步骤 4）。

该协议在图 12 中有详细解释。

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 检索市场协议详情

6.有用工作共识

Filecoin DSN 协议可以应用在允许验证 Filecoin 证明的任何共识协议上。在本节中，我们将展示如何引导基于有用工作的共识协议。Filecoin 矿工通过生成“时空证明”来参与共识，取代了浪费计算量的工作量证明。

有用工作 如果在保证了区块链安全性的前提下，计算的输出对网络来说是有价值的，我们就认为矿工在共识协议中的工作是有用的。

6.1 动机

虽然确保区块链的安全是至关重要的。但工作量证明证明方案往往会求解一些不能重复的问题，或者需要浪费大量的计算才能找到解决方案的问题。

- **不可重复的工作**：很多区块链要求矿工解决一些难以计算的问题，例如反向求解一个哈希函数。通常情况下这些问题都没有什么用处，除了保护网络安全之外也没有其他价值。我们可以让这件事变得有用吗？

可重复使用工作的尝试：已经有一些人尝试重复使用挖矿能力来进行更有用的计算。有些尝试是要求矿工沿用工作量证明的标准，来进行一些特殊计算。还有其他一些尝试希望用一些目前难以解决但非常有用的问题来替代工作量证明。例如，质数币（Primecoin）重复利用矿工的计算能力来寻找新的质数；以太坊要求矿工沿用工作量证明来执行小程序；Permacoin 提供档案管理服务，它要求矿工反向求解哈希函数来证明数据已被归档。虽然这些试探性工作大多很有用，但这些工作中计算能力浪费的现象仍然很普遍。

- **浪费的工作**：解决困难问题在机器损耗和能源消耗方面是十分昂贵的，特别是如果这些问题完全依赖计算能力。当挖矿算法难以并发计算的时候，解决难题的普遍因素就是计算能力。我们可以减少浪费的工作吗？

减少浪费工作的尝试：理想情况下，大部分网络资源应该花费在有用的工作上。一些尝试方案要求矿工使用更节能的解决方案。例如，Spacemint 要求矿工着力于磁盘空间而不是计算；虽然更加节能，因为磁盘空间被随机数据填满，所以仍然很浪费。其他的尝试是用传统的基于权益证明的拜占庭协议来取代难题，其中权益拥有者在下一个区块中，按其代币的占有比例进行投票。

我们来设计一个基于用户数据存储的有用工作共识协议。

6.2 Filecoin 共识

我们提出了有用工作的共识协议，这个协议中矿工被选出开发新区块的概率（我们称其为矿工的投票权）与其当前储存量在全网中的占比正相关。我们设计了 Filecoin 协议，这个协议中的矿工更愿意投身于存储而不是计算能力，并以此来平衡挖矿计算。矿工提供存储，并利用自己的计算能力生成数据已被存储的证明，以此参与到这个共识协议中。

6.2.1 挖矿能力建模

功率容错

在我们的技术报告[13]中，我们提出了功率容错，这是对在参与者对协议结果的影响方面进行拜占庭故障重构的抽象。每个参与者控制网络总功率 n 中的一部分，其中 f 是故障或竞争参与者控制的部分。

Filecoin 功率

在 Filecoin 中，功率 P^t_i 是 t 时间内，矿工 Mi 所分配到的存储任务的总和。影响因子 I^t_i 是 Mi 的功率在全网功率中的占比。

在 Filecoin 中，功率有以下特性：

- **公开性**：网络中正在使用的存储总量是公开的。通过读取区块链，每个人都可以计算任意矿工被分配的存储任务—因此任何人在任意时间点都可以计算出的每个矿工的功率和全网总功率。
- **公开验证性**：对于每个存储任务，矿工都需要生成时空证明来证明自己确实在提供存储服务。通过读取区块链，任何人都可以验证矿工声明的功率是否正确。
- **可变性**：在任意时间点，矿工都可以通过新增抵押扇区和填充扇区来增加新的存储。这样矿工就能对功率大小进行调整。

6.2.2 功率会计与时空证明

在每个Δ_{proof}区块（Δ_{proof}为一个系统参数）中，矿工们都必须向网络提交时空证明，如果网络中大多数功率认为它们有效，证明就可以被成功添加到区块链。在每个区块中，每个完整节点都会会更新 AllocTable，添加新的存储任务分配，删除过期和缺少标记的证明。

矿工 Mi 的功率可以通过分析 AllocTable 来计算和验证。分析主要有两种方式：

- **全节点验证**：如果节点拥有完整的区块链记录，则可以从初始区块开始,运行 NetworkProtocol 读取 Mi 的 AllocTable 直到当前区块。这个过程可以验证当前每一个分配给 Mi 的存储的时空证明。
- **简单存储验证**：假设一个轻客户端可以访问一个可以广播最新区块的信任源。轻客户端可以从网络节点中请求：（1）矿工 Mi 当前 AllocTable 的入口 （2）能够证明入口包含在最新区块的状态树中的 Merkle 路径（3）从初始区块到当前区块的头文件。这样轻客户端就可以将时空证明的验证托付给网络。

功率计算的安全性来自于时空证明的安全性。在这个设置里面，PoSt 保证了矿工无法对他们所分配的存储量说谎。事实上，他们不能声称存储过量的数据，因为这会耗费很多时间来获取和运行 PoSt.Setup。并且 PoSt.Prove 是顺序算法，他们无法通过并行计算快速生成证明。

6.2.3 使用功率达成共识

我们预见了通过扩展权益证明共识协议的当下规模（和未来预期）来实现 Filecoin 共识的多种策略，其中权益被替换为分配式存储。同时我们预见了权益证明协议的改进，我们提出了一个架构，这个架构基于我们前期“预期共识”的工作 [14]。我们的策略是在每一轮中选出一个（或多个）矿工，赢得选举的概率与矿工被分配的存储量成正例。

预期共识

预期共识（Expected Consensus，EC）的基本直觉是确切地、不可预测地、并且秘密地在每个周期中选举一小组领导人集合。在预期中，每个周期内当选的领导人数是 1，但也有一些周期内可能有零个或者许多领导人。每个周期中，区块连网络都会扩展一个或多个区块。如果某个周期内没有领导人，网络就会被添加一个空的区块。虽然链中的区块可以被线性排列，但其数据结构是一个有向无环图。EC 是一个概率共识，每个周期都比前面的区块更具有确定性，所以最终可以达到足够的确定性，使得出现不同历史的可能性足够小。如果大多数的参与者都通过扩展区块链或签署区块来将自己的权重附加到链上，这个区块就稳固了。

选举矿工

在每个周期，每个矿工都要检查自己是否被选为领导人，这类似于前面的协议：CoA[15]，Snow White16]，和 Algorand[17]。

定义 6.1（Filecoin 中的 EC 选举）如果下面的条件是满足的，则在时段 t 内矿工 Mi 成为领导人：

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

其中 rand(t)是在时间 t 内可以从区块链中提取出来的公开的随机变量，P^t_i 是 Mi 的功率。考虑对于任意的 m，L 是 H(m)的大小，H 是一种安全的加密哈希函数， <m>Mi 是由 Mi 签名的信息 m，例如：

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

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种特性：公平性，保密性和公开可验证性性。

- **公平性**：每次选举中每个参与者只有一次机会，因为签名是确定性的并且 t 和 rand(t)是固定的。假设 H 是安全的加密哈希函数，那么 $H(<t||rand(t)>Mi)/2^L$ 必须是从（0，1）均匀选择的实数，因此，使等式为真的可能性为 $P^t_i/\Sigma jp^t_j$，这与矿工在在网络中的功率相等。因为这个概率对功率是线性的，这种可能性在分割或者汇总功率的情况下就会被保留。注意随机值 rand(t)在没有给出 t 之前是未知的。
- **保密性**：在给出的数字签名假设中，即便是非常高效的攻击者，在没有密匙 Mi 的情况下计算出签名的概率也是极小的。
- **公开可验证性**：被选出的领导人 $i \in L^t$ 可以通过给出 t、rand(t)、$H(<t||rand(t)>Mi)/2^L$ 来说服一个有效的验证者；鉴于前面给出的观点，再有能力的攻击者在没有获胜秘钥的情况下也不能生成证明。

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 $\mathcal{H}(\langle t||r\rangle_i)/2^L \leq \frac{p_i^t}{\Sigma_j p_j^t}$
   - 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 π_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 $\mathcal{H}(\pi_i^t)/2^L < \frac{p_i^t}{\Sigma_j p_j^t}$
   - 成功输出 $\top$
   - 否则输出 $\perp$

图 13 预期共识协议中的领导人选举

7. 智能合约

Filecoin 给终端用户提供了两种基本操作：存（Put）和取（Get）。这两种操作使用户可以用他们认为合适的价格在市场中存取数据。尽管这些操作已经可以覆盖 Filecoin 的基本使用场景，我们仍然希望能在存取操作之上，通过部署智能合约来完成一些更复杂的操作。使用者们可以编写新的细分类存储/检索请求，与一般的智能合约类似，我们将其归类为“文件合约”。我们整合出一个合约系统 [18] 和一个桥梁系统，来将 Filecoin 的存储带入到其它区块链中。反之亦然，我们也可以将其它区块链的功能带入进 Filecoin。

我们希望 Filecoin 生态系统中的智能合约越多越好，并希望更多智能合约的开发者们参与进来，组件一个开发社区平台。

7.1 Filecoin 中的合约

智能合约允许用户们编写有状态的程序，这些程序可以通过消费代币在市场中进行数据存储/检索，并生成存储证明。用户可以通过向总账发送交易信息触发合约中的函数，来同智能合约进行交互。我们扩展了智能合约系统，使其可以支持 Filecoin 特定的一些操作（例如市场运营、证明审核等）。

Filecoin 特定地支持数据存储合约，同时也支持更多类型的智能合约：

- **Filecoin 合约**：我们允许用户对他们提供的存储服务情况进行编程。值得一提的例子有：（1）承包矿工：客户可以在不参与市场的情况下，预先指定提供服务的矿工。（2）支付策略：客户可以给矿工设计不同的回报策略。例如薪酬随时间增长的的合约，或者由可信赖的谕示监督下设置存储价格的合约。（3）票据服务：一种允许矿工积累代币，并代表他们的用户支付存储/检索服务的合约。（4）更复杂的操作：客户可以搭建允许数据更新的合约。
- **智能合约**：用户可以像在其它系统（例如以太坊[18]）中一样，不直接依赖于存储的使用而将程序和交易关联起来。我们预测这样的应用可能有：去中心化命名系统、资产追踪和 crowdsale 平台。

7.2 与其他系统的集成

桥（Bridges）是用来连接不同区块连的工具；虽然桥工具仍在发展之中，我们仍然计划开发跨链交互系统，以便于将 Filecoin 存储带入到其它区块链技术平台，同时也将其他平台的功能特性带入到 Filecoin 中。

- **其他平台中的 Filecoin**：其他系统，例如比特币（Bitcoin[19]）、Zcash[20]，特别是以太坊（Ethereum）[18]和 Tezos，这些系统都允许开发人员编写智能合约；然而这些平台存储能力低且成本高昂。我们计划提供一个桥梁，将存储和检索功能提供给这些平台。我们注意到 IPFS 已经被作为一种引用和分发内容的方式，应用于一些智能合约（以及协议代币）中。增加对 Filecoin 的支持会令这些系统确保 IPFS 内容的存储以换取 Filecoin 代币。
- **Filecoin 中的其他平台**：我们计划为 Filecoin 和其它区块链服务提供桥接。例如，与 Zcash 集成可以让 Filecoin 私密地发送存储请求。

8. 未来的工作

我们的工作为 Filecoin 网络构架的发展方向指明了清晰而有方向性的道路；当然，这些工作只是我们对去中心化存储系统研究的开始。在本节中我们会指出和说明我们未来要做的三部分工作。这包括了已经完成的和即将发表的一部分工作，一些公开的关于提升现有协议的问题，以及协议的形式化问题。

8.1 正在进行的工作

我们正在进行的工作主要分为以下几点：

- Filecoin 状态树的规格说明。
- Filecoin 及其组件的详细性能评估和基准测试。
- 完整可执行的 Filecoin 协议规范。
- 可赞助检索券模型：这个模型可以使客户 C1 通过发放单张的可用代币券，赞助另一位客户 C2 进行数据下载。
- 分层共识协议：Filecoin 的子网可以进行划分，并且在临时或永久划分的时候能持续处理交易。
- 使用 SNARK / STARK 的增量区块链快照。
- Filecoin 与以太坊交互（Filecoin-in-Ethereum）的合约接口和协议。
- 实现区块链归档，用 Braid 进行跨链标记。
- 只有在区块链解决冲突的时候才发起"时空证明"。
- 形式化证明 Filecoin DSN 的实现和新型存储证明。

8.2 开放型问题

有一些开放型问题，如果被解决了，可能会对网络有整体的改善。但这些问题并不紧急，并不是必须在项目启动前就要解决的。

- 重构更好的复制证明的密封函数，解码时间复杂度是 O(n)（目前是 O(nm)），无需 SNARK/STARK 即可公开验证。
- 重构更好的复制证明的证明函数，无需 SNARK/STARK 即可公开验证。
- 一个透明的、公开课验证的检索证明或其他存储证明。
- 在检索市场中新检索策略（例如基于概率的支付，零知识条件支付）
- 为预期共识设计更好的秘密领导人选举，使得每个周期中都有唯一确定的领导人当选。
- 更好更可信的 SNARK 方案，允许扩容公共参数（该方案可以运行 MPC 序列，其中每个附加 MPC 都严格降低故障率，并且每个 MPC 的输出都是有效的）。

8.3 证明和形式化验证

由于证明和形式化验证具有明确的价值，我们计划在未来几个月至几年内证明 Filecoin 网络的许多特性，并开发正式的验证协议规范。其中有几个证明正在实施，还有一些尚在构思当中。但要证明 Filecoin 的众多属性（如比例缩放，离线使用）将是非常艰难和长期的工作。

- 预期共识及其变体的正确性证明。
- 功率容错异步 1/2 不可能性结果回避正确性的证明。
- 通用可组合框架中 Filecoin DSN 的公式化，描述如何将 Get，Put 和 Manage 作为理想功能，并证明其实现方法。
- 形式化建模和证明自动化自愈系统。
- 形式化验证协议的描述（例如 TLA +或 Verdi）。
- 形式化验证的实现（例如 Verdi）。
- Filecoin 激励机制的博弈理论分析。

致谢

这项工作是 Protocol Labs 团队众人汗水的积累，尤其是 Protocol Labs 的合作者和顾问的帮助、评论和审查。Juan Benet 在 2014 年编写了初始版本的 Filecoin 白皮书，为本项工作奠定了基础。他和 Nicola Greco 开发了新的协议，并与团队其他成员合作编写了这份白皮书，团队成员们贡献了评论、总结和审核等工作。特别地，“大卫老爸”David Dalrymple 提出了订单范例和其他想法，Matt Zumwalt 改进了本文的结构，Evan Miyazono 绘制了插图，并总结了文章，Jeromy Johnson 在设计协议时提供了很多见解，Steven Allen 贡献了富有洞察力的问题和说明。我们也感谢所有的合作者和顾问，尤其是 Andrew Miller 和 Eli Ben-Sasson，与他们的交流对我们提供了很大帮助。

早先版本：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 les. 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 G artner. 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 o -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 e cient 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.