FileCoin设计

Filecoin定义了一个去中心化的存储网络，从而为云存储构建一个交易市场。整个交易市场通过区块链实现，并以fileCoin为内部原生代币。在市场内，矿工通过为客户提供可靠存储服务进行挖矿，相对的，客户通过支付filecoin使用矿工提供的数据保存和数据分发的服务。

Filecoin主要模块

1. 去中心化存储网络
2. 基于存储证明的共识算法
3. 可验证的数据市场
4. 可利用的工作量证明

## Filecoin协议

**网络节点算法**

对于账本 L，在每个周期 t

1. 对每个新的区块
   1. 检查区块格式是否有效
   2. 检查区块中所有的交易是否有效
   3. 检查区块中所有的订单是否有效
   4. 检查区块中所有的数据证明是否有效
   5. 检查区块中所有的抵押是否有效
   6. 如果以上检查失败，丢弃区块
2. 对于时间t内的每个新的订单O
   1. 添加订单O到存储市场的orderbook
   2. 如果订单为bid，锁定订单的O.funds
   3. 如果订单为ask，锁定订单的存储空间O.space
   4. 如果订单为deal，运行Pub.AssignOrders完成订单
3. 对于存储市场orderbook的每个订单
   1. 检查订单O是否超时或者被取消
      1. 将订单O从orderbook中删除
      2. 返回未支付的订单O.funds
      3. 释放O.space到可用存储空间分配表AllocTable
   2. 如果O是deal，运行Manage.RepairOrders检查对应的状态证明是否有效
      1. 如果矿工方的证明无效，惩罚对应的抵押
      2. 如果超过Delta\_fault周期缺乏有效证明，取消订单，重新加到Market中
      3. 如果数据不可读取或者重建，取消order，并且赔偿用户

**客户端算法**

任意时刻：

1. 通过Put.AddOrders提交新的存储订单
   1. 通过Pub.MatchOrders匹配合适的订单
   2. 发送文件数据到对应的矿工
2. 通过Get.AddOrders提交新的数据校验订单
   1. 通过Get.MatchOrders匹配合适的订单
   2. 和对应的矿工M创建一个支付通道

当收到存储矿工M的订单O\_deal

1. 对O\_deal进行签名
2. 通过Put.AddOrders将签名后的订单提交到区块链

当收到数据验证矿工M的数据验证结果p\_i

1. 验证p\_i的有效性，而且是源自自己发送的验证请求
2. 支付到矿工M

**存储挖矿算法**

任意时刻：

1. 通过Manage.PledgeSector 更新超时的抵押
2. 通过Manage.PledgeSector 为新提供的存储提供抵押
3. 通过Pub.AddOrder提交新的ask订单

在每个周期t：

1. 对orderbook中的每个O\_ask：
   1. 通过Put.MatchOrders寻找批评的订单
   2. 和匹配的客户联系，请求成交
2. 对每个抵押的扇区：
   1. 通过Mange.ProveSector生成存储证明
   2. 如果到了发送存储证明的时间，将证明提交到区块链上

当收到客户发送过来的数据：

1. 检查数据的大小是否符合O\_bid中指定的数据大小
2. 创建O\_deal，并且签名，发送给客户端
3. 保存数据到存储区
4. 如果存储扇区已满，执行Manage.SealSector

**数据提取挖坑算法**

任意时刻：

1. 在网络中传播存储矿工的O\_ask订单
2. 监听网络中的O\_bid订单

当收到客户端发送的数据验证请求：

1. 和客户端建立支付通道
2. 将数据分片到多个部分
3. 如果收到支付，发送对应的数据

# 存储证明算法

存储证明算法允许用户将数据保存和处理的工作委托到外部服务器，同时可用持续地检查服务器始终保存着对应的数据。

服务器要证明自己保存的数据的有效性，将随机对数据进行采样，只传输一小部分数据从而证明自己的存储的有效性。

## Proof-of-Replication

Prover：

1. 保证对某份数据D保存 n个相互独立的备份
2. 通过challenge／response协议向verifier证明，自己保存了n个相互独立的备份

 其中 为安全参数

其中c为随机数

## PoRep的实现

基于zk-SNARKs实现存储证明。

在初始化阶段，基于zk-SNARKs生成两个公钥 proving key pk 和 verification key vk。其中proving key用于生成证明，可生成的非交互式的零知识证明 ，而任一个人都可以用vk来验证，而且验证时间与数据大小成正比。

### 创建一个备份

Setup：通过Seal操作为一个数据备份生成一个数据备份证明。prover将生成的Merkle根和备份证明发送给verifier

输入：

1. prover的密钥
2. Prover的seal公钥
3. 数据D

输出：

1. 数据的备份R
2. 数据的Merkle根：rt
3. 数据备份证明

### 证明数据存储

Prove：为备份的数据生成一个存储证明。prover收到一个来自verifier的随机挑战c，prover需要根据c选择出R对应的分片，然后将和对应的Merkel路径构建出此次的数据存储证明回复。

输入：

1. prover的proof­-of-storage公钥
2. 备份的数据R
3. 随机数 c

输出：

1. 数据存储证明

### 验证数据存储证明

Verify

输入：

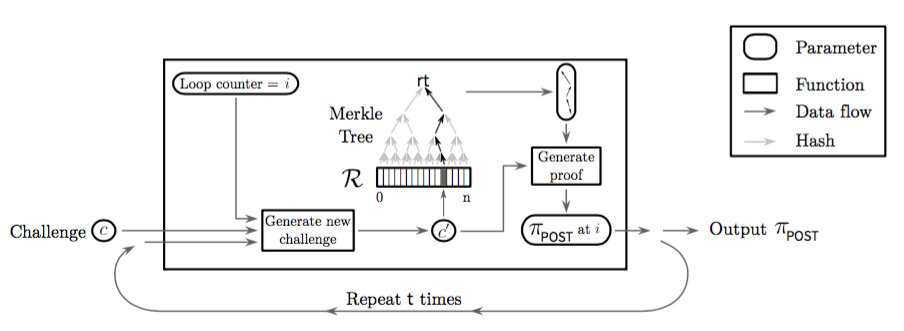
1. 证明者的公钥
2. 验证者的公钥
3. 数据的哈希
4. 数据的Merkle根 rt
5. 数据存储证明

输出：

验证结果｛true, false｝

## PoSt的实现

Setup与Verify类似，Prove部分改进为：



每一次随机验证的结果作为下一次随机验证的输入参数，通过迭代保证数据的持续稳定存储。

## 具体协议实现

### 初始化算法

输入：

1. prover的密钥
2. Prover的seal公钥
3. 数据D

算法：

1. 计算
2. 计算
3. 计算
4. 计算
5. 输出

### 证明生成算法

输入：

1. prover的proof­-of-storage公钥
2. 备份的数据R
3. 随机数 c

算法：

1. 计算
2. 计算
3. 计算
4. 输出

### 证明验证算法

输入：

1. 证明者的公钥
2. 验证者的公钥
3. 数据的哈希
4. 数据的Merkle根 rt
5. 数据存储证明

算法：

1. 计算
2. 计算
3. 返回