파일코인(Filecoin)

> *해당 번역본은 이용자가 디지털자산 관련 정보를 편리하게 확인할 수 있도록*
> *단순 참고용으로 번역된 것이며, 투자 권유를 목적으로 하지 않습니다.*
> *그 내용이 정확하지 않을 수 있으므로 정확한 정보 습득을 위해서는*
> *원문을 참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다.*
> *두나무는 해당 번역본 내용의 진실성 및 정확성을 보장하지 않고,*
> *제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며,*
> *투자 손실은 투자자에게 귀속됩니다.*

1. 서론

Filecoin은 *Proof-of-Spacetime*이라 불리는 새로운 형태의 증명 방식을 사용하는 프로토콜 토큰으로, 데이터 저장을 수행하는 채굴자들에 의해 블록이 생성되는 블록체인 위에서 구동된다. Filecoin 프로토콜은 단일 조정자(coordinator)에 의존하지 않는 독립적인 스토리지 제공자들의 네트워크를 통해 데이터 저장 및 검색 서비스를 제공하며, 그 구조는 다음과 같다:
(1) 클라이언트는 데이터를 저장하고 검색하기 위해 비용을 지불하고,
(2) 스토리지 채굴자(Storage Miner)는 저장 공간을 제공함으로써 토큰을 획득하며,
(3) 검색 채굴자(Retrieval Miner)는 데이터를 제공함으로써 토큰을 획득한다.

1.1 기본 구성 요소

Filecoin 프로토콜은 다음 네 가지 혁신적 구성 요소 위에 구축된다:

1. **탈중앙화 스토리지 네트워크(Decentralized Storage Network, DSN):**
   독립적인 스토리지 제공자들의 네트워크가 저장 및 검색 서비스를 제공할 수 있도록 하는 추상화를 제시한다(2장에서 설명). 이후 4장에서 Filecoin 프로토콜을 인센티브가 부여되고, 감사를 받을 수 있으며, 검증 가능한 DSN 구조로서 제시한다.
2. **혁신적인 저장 증명(Novel Proofs-of-Storage):**
   두 가지 새로운 *저장 증명(Proofs-of-Storage)* 을 제시한다(3장에서 설명):
   (1) *Proof-of-Replication*은 스토리지 제공자가 데이터를 자신만의 고유하고 전용된 물리적 저장소에 *복제(replication)* 했음을 증명할 수 있도록 한다. 고유한 물리적 사본을 강제함으로써 검증자는 증명자가 동일한 저장 공간에 여러 사본을 중복 저장(deduplication)하지 않았음을 확인할 수 있다.
   (2) *Proof-of-Spacetime*은 스토리지 제공자가 일정 기간 동안 데이터를 지속적으로 저장했음을 증명할 수 있도록 한다.
3. **검증 가능한 시장(Verifiable Markets):**

저장 요청(storage request)과 검색 요청(retrieval request)을 Filecoin 네트워크가 운영하는 두 개의 탈중앙화된 검증 가능한 시장에서 주문(order)으로 모델링한다(5장에서 설명). 검증 가능한 시장은 서비스가 올바르게 제공되었을 때만 결제가 이루어지도록 보장한다. 우리는 채굴자와 클라이언트가 각각 저장 및 검색 주문을 제출할 수 있는 *스토리지 마켓(Storage Market)* 과 *리트리벌 마켓(Retrieval Market)* 을 제시한다.

4. **유용한 작업 증명(Useful Proof-of-Work):**
   *Proof-of-Spacetime*을 기반으로 한 유용한 *작업 증명(Proof-of-Work)* 을 구성하는 방법을 제시하며, 이는 합의 프로토콜(consensus protocol)에서 사용될 수 있다. 채굴자들은 블록을 생성하기 위해 낭비적인 연산을 수행할 필요가 없으며, 대신 네트워크 내에서 데이터를 저장해야 한다.

1.2 프로토콜 개요

- Filecoin 프로토콜은 블록체인 위에 구축된 *탈중앙화 스토리지 네트워크(Decentralized Storage Network, DSN)* 구조로, 고유한 토큰을 사용한다. 클라이언트는 데이터를 저장하고 검색하기 위해 토큰을 지불하며, 채굴자는 데이터를 저장하고 제공함으로써 토큰을 획득한다.
- Filecoin DSN은 저장 요청과 검색 요청을 각각 두 개의 *검증 가능한 시장(verifiable markets)* — 즉, *스토리지 마켓(Storage Market)* 과 *리트리벌 마켓(Retrieval Market)* — 을 통해 처리한다. 클라이언트와 채굴자는 요청된 서비스와 제공되는 서비스의 가격을 설정하고, 해당 주문을 시장에 제출한다.
- 이 시장들은 Filecoin 네트워크에 의해 운영되며, 네트워크는 *Proof-of-Spacetime*과 *Proof-of-Replication*을 사용하여 채굴자가 저장하기로 약속한 데이터를 실제로 올바르게 저장했음을 보장한다.
- 마지막으로, 채굴자들은 기본 블록체인의 새로운 블록 생성에도 참여할 수 있다. 다음 블록 생성에 대한 채굴자의 영향력은 네트워크 내에서 현재 사용 중인 저장 용량의 크기에 비례한다.

본 논문에서 이후 정의될 용어들을 사용한 Filecoin 프로토콜의 개략도는 그림 1에, 그리고 클라이언트와 채굴자 간 상호작용을 시각적으로 보여주는 설명도는 그림 2에 제시되어 있다.

1.3 논문 구성

이 논문의 나머지 구성은 다음과 같다.
2장에서는 이론적 DSN 체계의 정의와 요구사항을 제시한다.
3장에서는 *Proof-of-Replication*과 *Proof-of-Spacetime* 프로토콜의 필요성을 논의하고, 그 정의 및 구체적인 설계를 제시한다. 이 두 가지 프로토콜은 Filecoin 내에서 데이터가 계약된 대로 지속적으로 저장되고 있음을 암호학적으로 검증하기 위해 사용된다.

4장에서는 Filecoin DSN의 구체적 구현을 다루며, 데이터 구조, 프로토콜, 그리고 참여자 간의 상호작용을 설명한다.
5장에서는 *검증 가능한 시장(Verifiable Markets)* 의 개념과 그 구현체인 스토리지 마켓 및 리트리벌 마켓을 정의하고 설명한다.
6장에서는 채굴자의 네트워크 기여도를 입증하고 평가하기 위한 *Proof-of-Spacetime* 프로토콜의 사용 목적과 절차를 설명하며, 이는 블록체인을 확장하고 블록 보상을 할당하는 데 필수적이다.

7장에서는 Filecoin 내 *스마트 컨트랙트(Smart Contracts)* 의 개요를 간략히 설명한다.
마지막으로, 8장에서 향후 연구 방향에 대해 논의하며 결론을 맺는다.

Filecoin Protocol Sketch

Network

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

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

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

Client

at any time:

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

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

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

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

1. verify that (p_i) is valid and it was requested
2. send a micropayment to $\mathcal{M}$

Storage Mine

at any time:

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

at each epoch t:

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

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

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

Retrieval Mine

at any time:

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

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

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

그림 1: Filecoin 프로토콜 개략도.

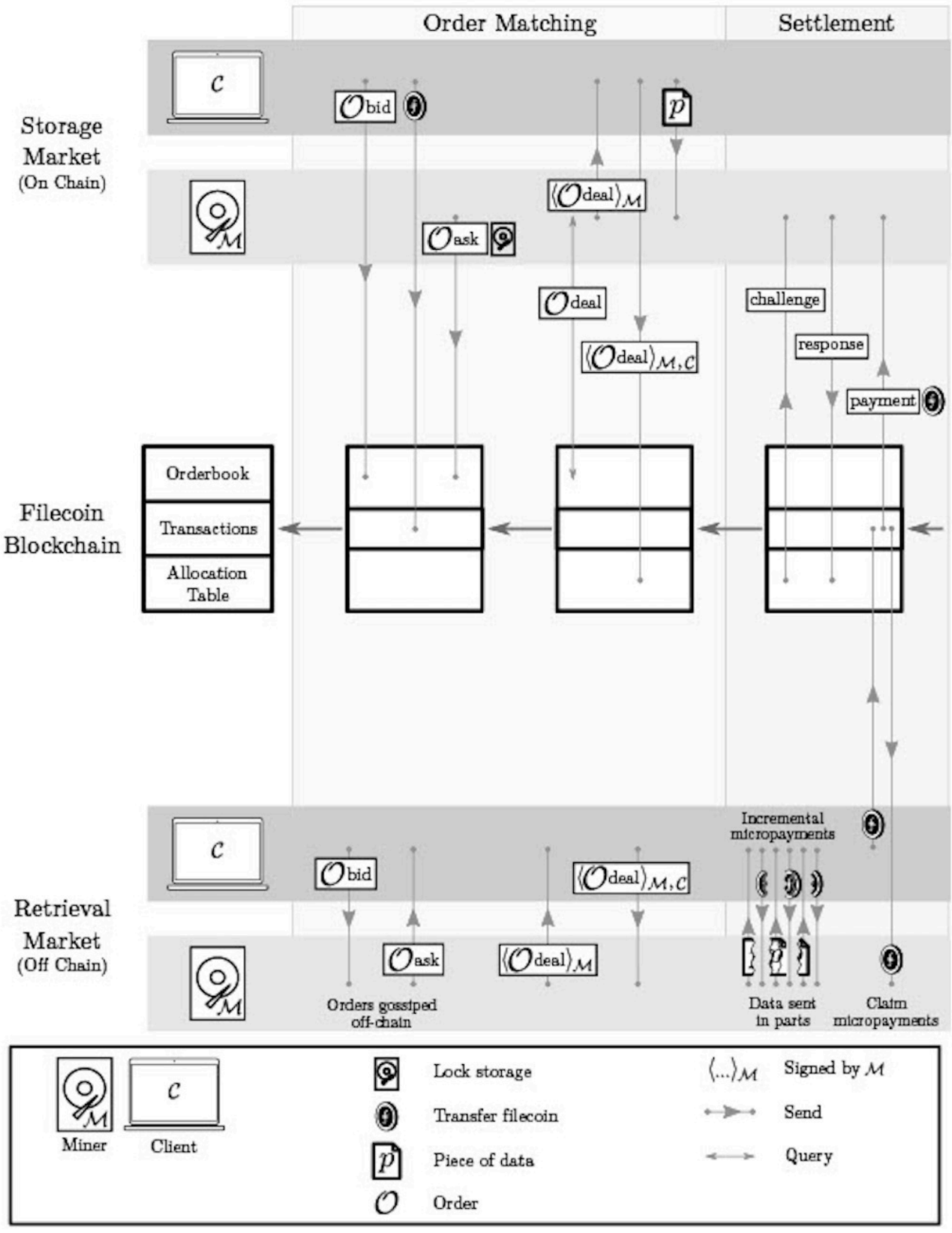


그림 2: Filecoin 프로토콜 설명도. 클라이언트와 채굴자 간 상호작용의 전체 개요를 보여준다. 스토리지 마켓과 리트리벌 마켓은 각각 블록체인의 위쪽과 아래쪽에 배치되어 있으며, 시간은 왼쪽의 주문 매칭(Order Matching) 단계에서 오른쪽의 결제(Settlement) 단계로 진행된다. 검색을 위한 마이크로페이먼트(micropayment)가 수행되기 전, 클라이언트는 반드시 해당 소액 거래에 필요한 자금을 미리 잠가(lock) 두어야 함에 유의한다.

2. 탈중앙화 스토리지 네트워크 (Decentralized Storage Network)의 정의

우리는 *탈중앙화 스토리지 네트워크*(Decentralized Storage Network, 이하 DSN) 체계의 개념을 제시한다.
DSN은 여러 독립적인 스토리지 제공자들이 제공하는 저장 공간을 집계(aggregate)하여, 클라이언트에게 데이터 저장 및 데이터 검색 서비스를 자율적으로 조정(self-coordinate)하며 제공한다.
이러한 조정은 탈중앙화되어 있으며 신뢰할 수 있는 제3자가 필요하지 않다. 즉, 시스템의 안전한 운영은 각 참여자의 작업을 조정하고 검증하는 프로토콜을 통해 달성된다.
DSN은 시스템의 요구사항에 따라 Byzantine 합의(Byzantine Agreement), 가십 프로토콜(gossip protocols), CRDTs 등 다양한 조정 전략을 사용할 수 있다.
이후 4장에서 우리는 Filecoin DSN의 구체적인 구조를 제시한다.

정의 2.1

DSN 체계 Π는 스토리지 제공자와 클라이언트가 실행하는 프로토콜의 튜플로 정의된다:

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

- **Put(data) → key:** 클라이언트는 *Put* 프로토콜을 실행하여 데이터를 고유 식별자 key 하에 *저장(store)* 한다.
- **Get(key) → data:** 클라이언트는 *Get* 프로토콜을 실행하여 key를 통해 현재 저장된 데이터를 *검색(retrieve)* 한다.
- **Manage():** 참여자 네트워크는 *Manage* 프로토콜을 통해 가용 저장 공간을 제어하고, 제공자의 서비스를 감사(audit)하며, 발생 가능한 오류를 복구(repair)한다. *Manage* 프로토콜은 스토리지 제공자들에 의해 주로 실행되며, 때로는 클라이언트나 감사자 네트워크와 함께 수행된다.

DSN 체계 Π는 **데이터 무결성(data integrity)** 과 **데이터 검색 가능성(retrievability)** 을 보장해야 하며, 이후 절에서 정의될 **관리(management)** 및 **저장(storage)** 오류를 허용할 수 있어야 한다.

2.1 장애 허용(Fault tolerance)

2.1.1 관리 장애(Management faults)

관리 장애는 *Manage* 프로토콜에 참여하는 노드들에 의해 발생하는 비잔틴(Byzantine) 장애로 정의된다.
DSN 체계는 기본적으로 사용하는 *Manage* 프로토콜의 장애 허용성(fault tolerance)에 의존한다.
이러한 관리 장애에 대한 허용 가정이 위반되면 시스템의 생존성(liveness)과 안전성(safety)이 훼손될 수 있다.

예를 들어, *Manage* 프로토콜이 스토리지 제공자를 감사하기 위해 비잔틴 합의(Byzantine Agreement, BA)를 요구하는 DSN 체계 Π를 고려하자.
이 프로토콜에서 네트워크는 스토리지 제공자로부터 저장 증명(proof of storage)을 수신하고, 해당 증명의 유효성에 대해 합의하기 위해 BA를 수행한다.
만약 BA가 전체 노드 수 n 중 f개의 장애를 허용한다면, DSN은 $f < n/2$개의 장애 노드를 허용할 수 있다.
이러한 가정이 위반되면 감사 과정이 손상될 수 있다.

2.1.2 저장 장애(Storage faults)

저장 장애는 클라이언트가 데이터를 검색할 수 없게 만드는 비잔틴 장애로 정의된다.
즉, 스토리지 채굴자가 저장한 조각(piece)을 분실하거나, 리트리벌 채굴자가 해당 조각의 제공을 중단하는 경우이다.

성공적인 *Put* 실행은 *(f, m)-내결함성(tolerant)* 을 가진다고 하며, 이는 입력 데이터가 m개의 독립적인 스토리지 제공자(전체 m개 중)에 저장되고, 최대 f개의 비잔틴 제공자를 허용할 수 있음을 의미한다.
매개변수 f와 m은 프로토콜 구현에 따라 달라지며, 프로토콜 설계자는 f와 m을 고정하거나 사용자가 직접 지정할 수 있도록 *Put(data)* 를 *Put(data, f, m)* 형태로 확장할 수도 있다.
저장된 데이터에 대한 *Get* 실행은 비잔틴 스토리지 제공자의 수가 f 미만일 경우 성공한다.

예를 들어, 각 스토리지 제공자가 모든 데이터를 저장하도록 설계된 간단한 *Put* 프로토콜을 고려하자.
이 경우 $m = n$ 이며, $f = m - 1$ 이다.
그러나 항상 $f = m - 1$ 인 것은 아니다.
어떤 체계는 소거 부호(erasure coding)를 사용할 수 있으며, 각 스토리지 제공자가 데이터의 특정 부분만 저장하도록 설계될 수 있다.
이 경우 전체 m개의 제공자 중 x개의 제공자만으로 데이터를 복원할 수 있으며, 이때 $f = m - x$ 가 된다.

2.2 속성(Properties)

이 절에서는 DSN 체계가 반드시 충족해야 하는 두 가지 기본 속성을 설명한 후, Filecoin DSN이 추가적으로 요구하는 속성들을 제시한다.

2.2.1 데이터 무결성(Data Integrity)

이 속성은 제한된 계산 능력을 가진 적대자 $\mathcal{A}$ 가 *Get* 실행의 결과로 변경되거나 위조된 데이터를 클라이언트가 수용하도록 만드는 것을 불가능하게 해야 함을 의미한다.

정의 2.2.
DSN 체계 Π가 *데이터 무결성(data integrity)* 을 제공한다는 것은 다음과 같은 경우를 의미한다:
일부 데이터 d가 키 k 하에 성공적으로 *Put* 실행되었다면, 어떤 계산적으로 제한된 적대자 $\mathcal{A}$ 도 *Get* 실행이 끝난 시점에서 식별자 k에 대해 $d' \neq d$ 인 데이터를 클라이언트가 받아들이도록 설득할 수 없다.

2.2.2 검색 가능성(Retrievability)

이 속성은 DSN 체계 Π의 내결함성(fault-tolerance) 가정 하에서, 데이터가 성공적으로 저장되었고 스토리지 제공자들이 프로토콜을 계속 준수한다면, 클라이언트가 결국 그 데이터를 검색할 수 있어야 함을 의미한다.

정의 2.3.
DSN 체계 Π가 *검색 가능성(retrievability)* 을 제공한다는 것은 다음과 같다:
데이터가 어떤 키 하에 성공적으로 *Put* 실행되었다면, 해당 키에 대해 클라이언트가 $data$를 검색할 수 있는 성공적인 *Get* 실행이 존재한다.

2.2.3 기타 속성(Other Properties)

DSN은 특정 응용 분야에 따라 추가적인 속성을 제공할 수 있다.
Filecoin DSN이 요구하는 세 가지 핵심 속성은 *공개 검증 가능성(public verifiability)*, *감사 가능성(auditability)*, *인센티브 정합성(incentive-compatibility)* 이다.

정의 2.4.
DSN 체계 Π가 *공개 검증 가능(publicly verifiable)* 하다는 것은 다음을 의미한다:
각 성공적인 *Put* 실행에 대해, 스토리지 제공자 네트워크가 현재 해당 데이터가 저장되어 있음을 증명하는 *저장 증명(Proof-of-Storage)* 을 생성할 수 있어야 한다.
이 저장 증명은 데이터에 접근할 수 없고 오직 키만 알고 있는 효율적인 검증자(efficient verifier)도 이를 신뢰할 수 있어야 한다.

정의 2.5.
DSN 체계 Π가 *감사 가능(auditable)* 하다는 것은 다음을 의미한다:
운영 과정에서 생성되는 *검증 가능한 이력(verifiable trace)* 이 존재하며, 이를 통해 미래의 감사 시 특정 데이터가 실제로 올바른 기간 동안 저장되어 있었음을 확인할 수 있어야 한다.

정의 2.6.
DSN 체계 Π가 *인센티브 정합적(incentive-compatible)* 이라는 것은 다음을 의미한다:
스토리지 제공자가 저장 및 검색 서비스를 성공적으로 제공할 경우 보상을 받고, 부정행위를 할 경우 처벌을 받도록 설계되어 있어, 스토리지 제공자의 우월한 전략(dominant strategy)이 데이터를 성실히 저장하는 것이다.

3. 복제 증명(Proof-of-Replication)과 시간-공간 증명(Proof-of-Spacetime)

Filecoin 프로토콜에서 스토리지 제공자는 자신이 보관료를 받고 저장하기로 약속한 데이터를 실제로 저장하고 있음을 클라이언트에게 증명해야 한다.
실제 운영에서는 스토리지 제공자가 *저장 증명(Proofs-of-Storage, PoS)* 을 생성하며, 이는 블록체인 네트워크(또는 클라이언트 자체)에 의해 검증된다.

이 절에서는 Filecoin에서 사용되는 *복제 증명(Proof-of-Replication, PoRep)* 과 *시간-공간 증명(Proof-of-Spacetime, PoSt)* 체계의 필요성, 구조, 그리고 구현 개요를 설명한다.

3.1 도입 배경(Motivation)

저장 증명(Proofs-of-Storage, PoS) 체계에는 *Provable Data Possession (PDP)* 과 *Proof-of-Retrievability (PoR)* 등이 있다.
이들은 데이터를 서버(증명자 P)에게 위탁한 사용자(검증자 V)가 해당 서버가 여전히 데이터 D를 저장하고 있는지 반복적으로 확인할 수 있도록 한다.
사용자는 데이터를 직접 다운로드하지 않고도 매우 효율적으로 데이터 무결성을 검증할 수 있다.
서버는 무작위로 선택된 데이터 블록 집합을 샘플링하여, 사용자와의 도전/응답(challenge/response) 프로토콜을 통해 확률적 보유 증명(probabilistic proof of possession)을 생성하고, 일정한 크기의 데이터만을 전송한다.

그러나 PDP 및 PoR 체계는 증명자가 도전/응답 시점에 일부 데이터를 **일시적으로 보유하고 있었음** 만을 보장한다.
Filecoin에서는 악의적 채굴자가 실제로 데이터를 저장하지 않고 보상을 얻는 것을 방지하기 위해 더 강력한 보장이 필요하다.
이를 위해 Filecoin은 다음 세 가지 공격 유형을 방지해야 한다: **Sybil 공격, 외주 공격, 생성 공격.**

- **Sybil 공격:** 악의적 채굴자가 여러 Sybil 신원을 생성하여 마치 다수의 복제본을 저장한 것처럼 속이고, 실제로는 데이터를 단 한 번만 저장한 채 더 많은 보상을 받으려 하는 공격.
- **외주 공격(Outsourcing Attacks):** 악의적 채굴자가 실제 저장 용량보다 더 많은 데이터를 저장하기로 약속하고, 다른 저장 제공자에게서 데이터를 빠르게 가져오는 방식으로 허위로 약속을 이행하는 공격.
- **생성 공격(Generation Attacks):** 악의적 채굴자가 방대한 데이터를 저장하는 것처럼 주장하지만, 실제로는 작은 프로그램을 이용해 데이터를 필요할 때마다 효율적으로 생성하는 공격.
  이 프로그램의 크기가 주장된 데이터보다 작다면, 채굴자는 Filecoin의 블록 보상 확률(저장 용량 비례)을 부당하게 높일 수 있다.

3.2 복제 증명(Proof-of-Replication)

복제 증명(Proof-of-Replication, PoRep) 은 서버(증명자 P)가 특정 데이터 D를 자신만의 고유하고 전용된 물리적 저장소에 *복제(replication)* 했음을 사용자(검증자 V)에게 입증할 수 있도록 하는 새로운 형태의 *저장 증명(Proof-of-Storage)* 이다.

이 체계는 상호작용형(interactive) 프로토콜로, 증명자 P는
(a) 데이터 D의 n개의 물리적으로 독립된 복제본을 저장하겠다고 약속(commit)하고,
(b) 도전/응답 프로토콜을 통해 검증자 V에게 실제로 모든 복제본을 저장하고 있음을 입증한다.

현재까지 알려진 바로는 PoRep은 PoR과 PDP 체계보다 개선된 형태로, **Sybil 공격, 외주 공격, 생성 공격**을 방지한다.

참고: PoRep의 공식 정의, 속성에 대한 상세 설명, 그리고 심층 연구는 참고 문헌 [5]를 참조할 수 있다.

정의 3.1 (복제 증명, Proof-of-Replication)
PoRep 체계는 효율적인 증명자 P가 검증자 V에게, 자신이 데이터 D의 물리적으로 독립된 복제본(replica) R을 저장하고 있음을 설득할 수 있도록 한다.

PoRep 프로토콜은 다음의 다항시간(polynomial-time) 알고리즘 튜플로 정의된다:

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

- **PoRep.Setup$(1^\lambda, D) \to R, S_P, S_V$**
  여기서 S_P와 S_V는 각각 증명자 P와 검증자 V의 체계별(setup-specific) 초기 변수이며, λ는 보안 매개변수(security parameter)이다.
  PoRep.Setup은 복제본 R을 생성하고, P와 V가 PoRep.Prove 및 PoRep.Verify를 실행하는 데 필요한 정보를 제공한다.
  일부 체계에서는 증명자 또는 제3자와의 상호작용을 통해 PoRep.Setup을 계산해야 할 수도 있다.
- **PoRep.Prove(S_P, R, c) → π^c**
  여기서 c는 검증자 V가 발행한 무작위 도전(challenge)이며, π^c는 증명자가 데이터 D의 특정 복제본 R에 접근할 수 있음을 입증하는 증거이다.
  PoRep.Prove는 증명자 P가 실행하여 검증자 V에게 제출할 증명 π^c를 생성한다.
- **PoRep.Verify(S_V, c, π^c) → ${0, 1}$**
  제출된 증명이 올바른지를 검증한다.
  PoRep.Verify는 검증자 V가 실행하며, P가 실제로 복제본 R을 저장하고 있는지를 판단한다.

3.3 시간-공간 증명(Proof-of-Spacetime)

저장 증명(Proof-of-Storage) 체계는 사용자가 스토리지 제공자가 도전 시점에 위탁된 데이터를 실제로 저장하고 있는지를 확인할 수 있게 한다.
그렇다면 *PoS 체계를 이용하여 어떤 데이터가 일정 기간 동안 지속적으로 저장되고 있었음을 어떻게 증명할 수 있을까?*

자연스러운 접근은 사용자가 일정 간격(예: 매 분마다)으로 스토리지 제공자에게 반복적으로 도전(challenge)을 보내는 것이다.
그러나 Filecoin과 같은 시스템에서는 각 상호작용에서 요구되는 통신 복잡도가 매우 커질 수 있으며, 이는 스토리지 제공자가 블록체인 네트워크에 주기적으로 증명을 제출해야 하는 구조에서 병목이 된다.

이 문제를 해결하기 위해 우리는 새로운 증명 방식인 *Proof-of-Spacetime (PoSt)* 을 도입한다.
PoSt에서는 검증자가 증명자(Prover)가 일정 기간 동안 데이터를 지속적으로 저장했는지를 확인할 수 있다.
핵심 아이디어는 증명자에게 다음을 요구하는 것이다:
(1) 시간의 경과를 측정하기 위해 순차적으로 *저장 증명(Proof-of-Storage)* (Filecoin에서는 *Proof-of-Replication*)을 생성하고,
(2) 이를 재귀적으로 합성하여 짧고 효율적인 단일 증명을 생성하는 것이다.

정의 3.2 (Proof-of-Spacetime)
PoSt 체계는 효율적인 증명자 P가 검증자 V에게 P가 데이터 D를 일정 기간 t 동안 저장하고 있음을 설득할 수 있게 한다.
PoSt는 다음과 같은 다항시간 알고리즘의 튜플로 구성된다:

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

- **PoSt.Setup($1^\lambda, D$) → S_P, S_V**
  여기서 S_P와 S_V는 각각 증명자 P와 검증자 V를 위한 체계별 초기 변수이며, λ는 보안 매개변수이다.
  PoSt.Setup은 P와 V가 PoSt.Prove 및 PoSt.Verify를 수행하는 데 필요한 정보를 제공한다.
  일부 체계에서는 증명자 또는 제3자와의 상호작용이 필요할 수 있다.
- **PoSt.Prove(S_P, D, c, t) → π^c**
  여기서 c는 검증자 V가 발행한 무작위 도전(challenge)이며, π^c는 증명자가 일정 기간 t 동안 데이터 D에 접근할 수 있었음을 입증하는 증거이다.
  PoSt.Prove는 증명자 P가 실행하여 검증자 V에게 제출할 증명 π^c를 생성한다.
- **PoSt.Verify(S_V, c, t, π^c) → ${0, 1}$**
  해당 증명이 올바른지를 확인한다.
  PoSt.Verify는 검증자 V가 실행하며, P가 주어진 기간 t 동안 데이터 D를 실제로 저장했는지를 검증한다.

3.4 실용적 PoRep 및 PoSt 구현

우리는 신뢰할 수 있는 제3자나 특수 하드웨어에 의존하지 않고, 기존 시스템에 배포 가능한 실용적인 PoRep 및 PoSt 구조에 관심이 있다.
우리는 [5]에서 제시된 *Seal 기반 Proof-of-Replication* 구조를 채택하며, 이 방식은 복제본을 생성하기 위해 Setup 단계에서 매우 느린 순차적 계산 과정인 *Seal*을 수행해야 한다.
PoRep과 PoSt의 프로토콜 개요는 그림 4에, PoSt의 증명 단계 내 핵심 메커니즘은 그림 3에 도식화되어

있다.

3.4.1 암호학적 구성 요소(Cryptographic Building Blocks)

- **충돌 저항 해시(Collision-resistant hashing):**
  우리는 충돌 저항 해시 함수 $CRH: {0,1}^* \to {0,1}^{O(\lambda)}$를 사용한다.
  또한 문자열을 여러 부분으로 분할한 후 이진 트리를 구성하고, 재귀적으로 CRH를 적용하여 루트를 출력하는 충돌 저항 해시 함수 *MerkleCRH*도 사용한다.

- **zk-SNARKs:**
  PoRep과 PoSt의 실용적 구현은 영지식 비대화 간결 지식 논증(*Succinct Non-interactive ARguments of Knowledge*, zk-SNARKs)에 의존한다.
  zk-SNARKs는 *간결성(succinctness)* 을 가지므로 증명이 매우 짧고 검증이 빠르다.

  보다 형식적으로, NP 언어 L과 이에 대한 판별 회로(decision circuit) C가 있다고 하자.
  신뢰할 수 있는 제3자는 일회성 설정(setup) 과정을 수행하여 두 개의 공개 키, 즉 증명 키 *pk*와 검증 키 *vk*를 생성한다.
  증명 키 *pk*를 사용하면 신뢰할 수 없는 증명자도 자신이 선택한 인스턴스 x에 대해 $x \in L$임을 입증하는 증명 π를 생성할 수 있다.

  이 비대화형 증명 π는 *영지식(zero-knowledge)* 이며 동시에 *지식 증명(proof-of-knowledge)* 이다.
  누구나 검증 키 *vk*를 이용하여 증명 π를 검증할 수 있으며, 특히 zk-SNARK 증명은 **공개 검증 가능(publicly verifiable)** 하다.
  즉, 증명을 생성한 증명자와 상호작용하지 않고도 누구나 π의 유효성을 검증할 수 있다.
  증명 π의 크기는 상수(constant)이며, 검증 시간은 입력 크기 $|x|$에 선형적으로 비례한다.

  회로 만족성(circuit satisfiability)에 대한 zk-SNARK은 다음의 다항시간 알고리즘 3개로 구성된다:

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

  - **KeyGen$(1^\lambda, C) \to (pk, vk)$:**
    보안 매개변수 λ와 회로 C를 입력받아, *pk*와 *vk*를 확률적으로 생성한다.
    두 키는 공개 매개변수로 배포되어 L_C 내의 멤버십 증명 및 검증에 사용된다.
  - **Prove$(pk, x, w) \to \pi$:**
    증명 키 *pk*와 입력 *x*, 그리고 NP 명제에 대한 증인 *w*를 입력받아, $x \in L_C$임을 입증하는 비대화형 증명 π를 출력한다.
  - **Verify$(vk, x, \pi) \to {0,1}$:**
    검증 키 *vk*, 입력 *x*, 증명 π를 입력받아, $x \in L_C$이면 1을 출력한다.

  zk-SNARK 시스템의 형식적 정의 및 구현에 대해서는 [6,7,8]을 참조할 수 있다.
  일반적으로 이러한 시스템은 신뢰할 수 있는 제3자에 의해 *KeyGen* 과정이 수행되어야 하지만,
  최근 *Scalable Computational Integrity and Privacy (SCIP)* 시스템 [9]은 이 초기 설정에 대한 신뢰 가정을 제거할 수 있는 가능성을 보여주고 있다.

3.4.2 Seal 연산(Seal operation)

Seal 연산의 역할은 다음 두 가지이다:
(1) 증명자(prover)로 하여금 자신의 공개키에 고유한 데이터 D의 의사난수적 순열(pseudo-random permutation)을 저장하도록 요구함으로써, 복제본(replica)들이 물리적으로 독립된 사본이 되도록 강제한다.
즉, *n*개의 복제본 저장을 약속하는 것은 *n*개의 독립된 복제본을 위해 디스크 공간을 실제로 *n*배로 할당해야 함을 의미한다.
(2) *PoRep.Setup* 단계에서 복제본을 생성하는 데 필요한 시간이 도전(challenge)에 응답하는 데 필요한 예상 시간보다 훨씬 오래 걸리도록 강제한다.

Seal 연산의 보다 형식적인 정의는 [5]를 참조한다.
이 연산은 *Seal${AES-256}$ 및 매개변수 τ를 이용하여 실현할 수 있으며*, *Seal*${AES-256}^\tau$ 의 실행 시간은 정직한 도전–증명–검증 시퀀스보다 약 10~100배 더 길게 설정된다.
특히 τ의 값은 *Seal*$_{AES}^\tau$ 실행이 복제본 R에 무작위 접근(random access)하여 *Prove*를 수행하는 것보다 명확히 더 비용이 크도록 선택하는 것이 중요하다.

3.4.3 실용적 PoRep 구성(Practical PoRep Construction)

이 절에서는 *PoRep* 프로토콜의 구성 과정을 설명하며, 그림 4에 단순화된 프로토콜 개요를 제시한다.
구체적인 구현 및 최적화 세부 사항은 생략한다.

복제본 생성(Creating a Replica)

Setup 알고리즘은 *Seal* 연산을 통해 복제본을 생성하고, 복제본이 올바르게 생성되었음을 증명하는 증거를 만든다.
증명자는 복제본을 생성한 후, 복제본 자체(R)를 제외한 출력값들을 검증자에게 전송한다.

```
Setup
● 입력:
  – 증명자 키 쌍 (pkP, skP)
  – 증명자 SEAL 키 pkSEAL
  – 데이터 D
● 출력:
  – 복제본 R
  – 복제본 R의 머클 루트 rt
  – Seal 증명 πSEAL
```

저장 증명(Proving Storage)

Prove 알고리즘은 복제본에 대한 저장 증명을 생성한다.
증명자는 검증자로부터 무작위 도전 *c*를 수신하며, 이는 복제본 R의 머클 트리 내 루트 *rt*로부터 특정 리프 노드 R_c를 결정한다.
증명자는 R_c와 그것이 루트 *rt*에 이르는 머클 경로에 대한 지식 증명(proof of knowledge)을 생성한다.

```
Prove
● 입력:
    – 증명자 Proof-of-Storage 키 pkPOS
    – 복제본 R
    – 무작위 도전 c
● 출력: 저장 증명 πPOS
```

증명 검증(Verifying the Proofs)

Verify 알고리즘은 복제본의 머클 루트와 원본 데이터의 해시를 기반으로 저장 증명의 유효성을 확인한다.
이 증명은 공개적으로 검증 가능하며, 원장을 유지하는 분산 시스템의 노드들과 특정 데이터에 관심 있는 클라이언트들이 직접 검증할 수 있다.

```
Verify
● 입력:
    – 증명자 공개키 pkP
    – 검증자 SEAL 및 POS 키 vkSEAL, vkPOS
    – 데이터 D의 해시 hD
    – 복제본 R의 머클 루트 rt
    – 무작위 도전 c
    – 증명 쌍 (πSEAL, πPOS)
● 출력: 비트 b (증명이 유효하면 1)
```

3.4.4 실용적 PoSt 구성(Practical PoSt Construction)

이 절에서는 *PoSt* 프로토콜의 구성 과정을 설명하며, 그림 4에 단순화된 프로토콜 개요를 제시한다.
구현 및 최적화 세부 사항은 생략한다.
Setup 및 *Verify* 알고리즘은 *PoRep* 구성과 동일하므로, 여기서는 *Prove* 알고리즘만을 설명한다.

공간 및 시간 증명(Proving Space and Time)

Prove 알고리즘은 복제본에 대한 *시간-공간 증명(Proof-of-Spacetime)* 을 생성한다.
증명자는 검증자로부터 무작위 도전을 수신한 후, 지정된 반복 횟수 t 동안 연속적으로 *복제 증명(Proofs-of-Replication)* 을 생성한다.
이때 각 증명의 출력을 다음 증명의 입력으로 사용한다(그림 3 참조).

```
Prove
● 입력:
    – 증명자 PoSt 키 pkPOST
    – 복제본 R
    – 무작위 도전 c
    – 시간 매개변수 t
● 출력: 시간–공간 증명 πPOST
```

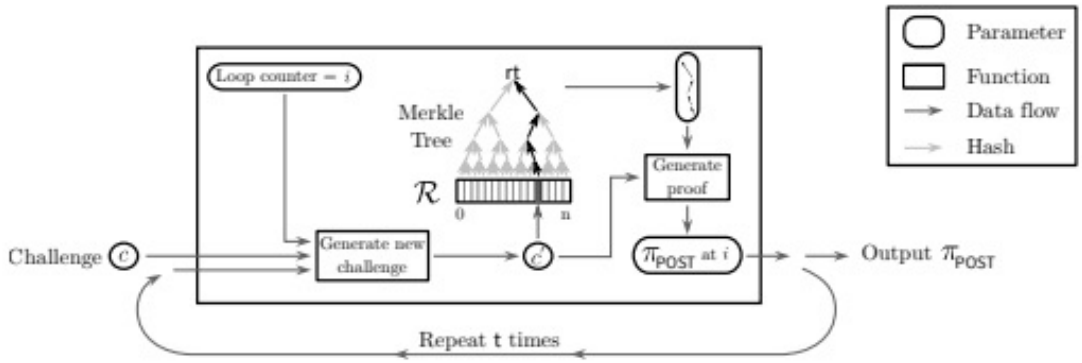
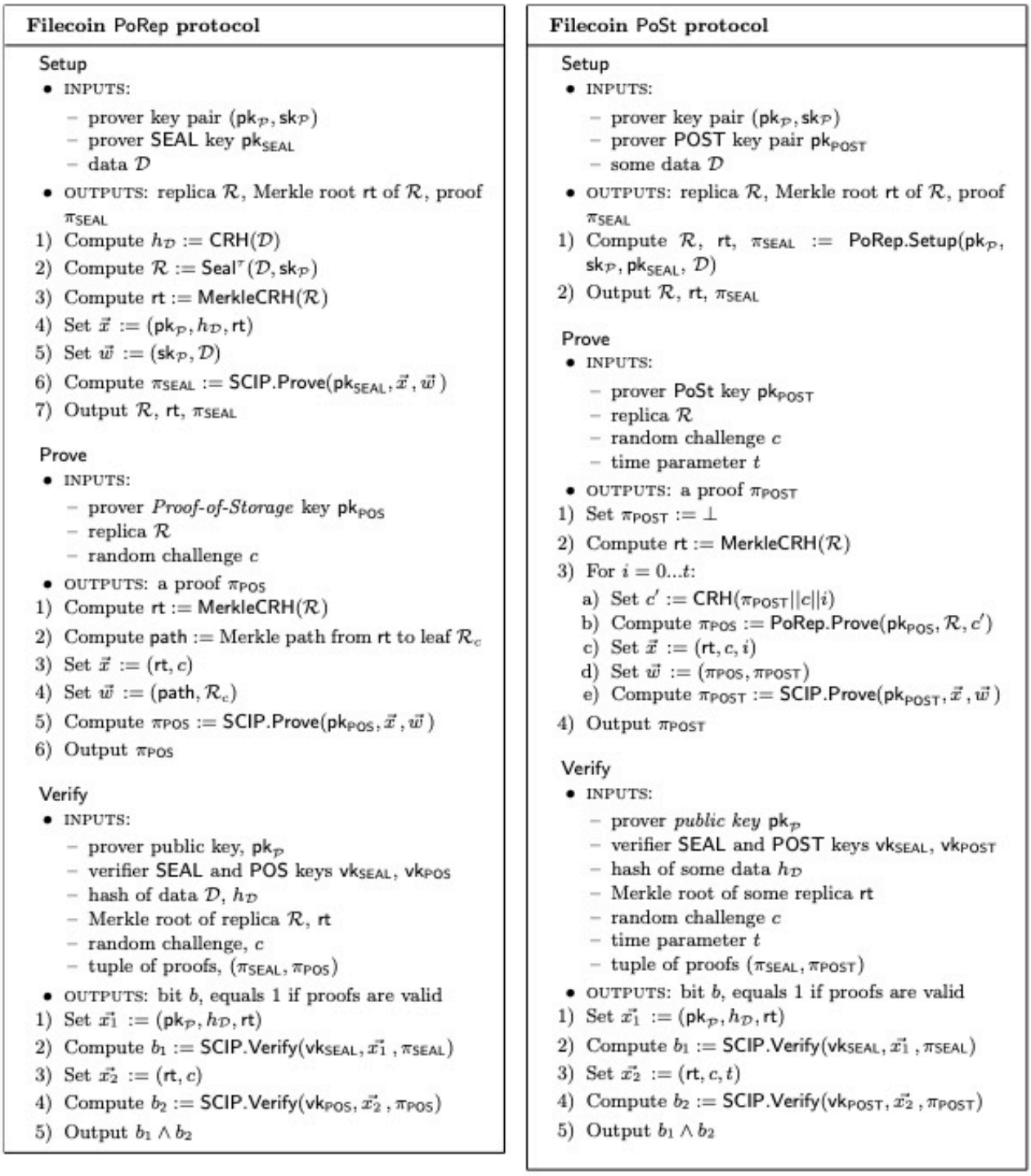


그림 3: PoSt.Prove의 기본 메커니즘을 나타낸 도식.
지속적 저장을 입증하기 위해 반복적으로 생성되는 증명 과정을 보여준다.

3.5 Filecoin에서의 활용(Usage in Filecoin)

Filecoin 프로토콜은 채굴자(miner)가 제공하는 저장 공간을 감사(audit)하기 위해 *Proof-of-Spacetime*을 사용한다.
Filecoin에서 PoSt를 활용하기 위해 우리는 지정된 검증자(verifier)가 존재하지 않으므로, 네트워크의 모든 참여자가 검증을 수행할 수 있도록 체계를 **비대화형(non-interactive)** 으로 수정하였다.
검증자는 공개 코인 모델(public-coin model)에서 동작하므로, 블록체인에서 무작위성을 추출하여 도전(challenge)을 발행할 수 있다.



Filecoin PoRep protocol

Setup

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

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

Prove

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

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

Verify

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

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

Filecoin PoSt protocol

Setup

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

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

Prove

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

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

Verify

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

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

그림 4: *Proof-of-Replication* 및 *Proof-of-Spacetime* 프로토콜 개요.
여기서 CRH는 충돌 저항 해시(collision-resistant hash)를 의미하며, $\vec{x}$는 증명해야 할 NP 명제(NP-statement), $\vec{w}$는 그에 대한 증인(witness)이다.

---

4. Filecoin: DSN 구조(Filecoin: a DSN Construction)

Filecoin DSN은 *감사 가능(auditable)* 하며, *공개 검증 가능(publicly verifiable)* 하고, *인센티브 기반(incentive-driven)* 으로 설계된 탈중앙화 스토리지 네트워크이다.
클라이언트는 네트워크 내 채굴자에게 데이터를 저장하고 검색하기 위해 비용을 지불하며, 채굴자는 디스크 공간과 대역폭을 제공하고 그 대가로 보상을 받는다.
채굴자는 네트워크가 자신들의 서비스가 올바르게 제공되었음을 *감사(audit)* 할 수 있을 때만 대금을 지급받는다.

이 절에서는 앞서 정의한 DSN 개념과 *Proof-of-Spacetime*을 기반으로 Filecoin DSN의 구체적인 구조를 제시한다.

4.1 환경 설정(Setting)

4.1.1 참여자(Participants)

모든 사용자는 **클라이언트(Client)**, **스토리지 채굴자(Storage Miner)**, **리트리벌 채굴자(Retrieval Miner)** 로 참여할 수 있다.

- **클라이언트(Clients)**
  DSN 내에서 데이터를 저장하거나 검색하기 위해 Put 및 Get 요청을 통해 비용을 지불한다.
- **스토리지 채굴자(Storage Miners)**
  네트워크에 데이터 저장 공간을 제공한다. 스토리지 채굴자는 디스크 공간을 제공하고 Put 요청을 처리함으로써 Filecoin에 참여한다.
  스토리지 채굴자가 되기 위해서는 제공할 저장 공간에 비례하는 담보(collateral)를 예치하여 *서약(pledge)* 해야 한다.
  스토리지 채굴자는 클라이언트의 데이터를 지정된 기간 동안 저장할 것을 약속하며, 데이터를 시간에 따라 지속적으로 저장하고 있음을 증명하기 위해 *Proofs-of-Spacetime*을 생성하여 블록체인에 제출한다.
  유효하지 않거나 누락된 증명을 제출할 경우, 스토리지 채굴자는 벌점을 받고 일부 담보를 잃는다.
  또한 스토리지 채굴자는 새로운 블록을 채굴할 자격이 있으며, 블록 생성 시 채굴 보상(block reward)과 해당 블록에 포함된 거래에 대한 수수료(transaction fee)를 함께 받는다.
- **리트리벌 채굴자(Retrieval Miners)**
  네트워크에 데이터 검색 서비스를 제공한다. 리트리벌 채굴자는 Get 요청을 통해 사용자가 요청하는 데이터를 제공함으로써 Filecoin에 참여한다.
  스토리지 채굴자와 달리, 리트리벌 채굴자는 담보 예치나 데이터 저장 약속, 저장 증명을 수행할 의무가 없다.
  스토리지 채굴자가 동시에 리트리벌 채굴자로 참여하는 것은 자연스러운 구조이다.
  리트리벌 채굴자는 데이터를 클라이언트로부터 직접 획득하거나, 리트리벌 마켓(Retrieval Market)을 통해 획득할 수 있다.

4.1.2 네트워크 $\mathcal{N}$(The Network)

Filecoin 전체 노드를 실행하는 모든 사용자를 하나의 추상적 실체로 표현할 수 있으며, 이를 *네트워크(Network)* 라고 한다.
네트워크는 *Manage* 프로토콜을 실행하는 중개자 역할을 하며, 비공식적으로 설명하면 Filecoin 블록체인에서 새 블록이 생성될 때마다 전체 노드들은 가용 저장 공간을 관리하고, 서약(pledge)을 검증하며, 저장 증명(storage proof)을 감사하고, 발생할 수 있는 오류를 복구한다.

4.1.3 원장(The Ledger)

Filecoin 프로토콜은 원장 기반 통화(ledger-based currency) 위에서 실행된다.
이를 일반화하여 우리는 이를 *원장(Ledger)*, 즉 $\mathcal{L}$ 로 표현한다.
특정 시점 *t* (이를 *epoch*이라 한다)에서 모든 사용자는 원장 $\mathcal{L}_t$ 에 접근할 수 있으며, 이는 *거래(transaction)* 의 순차적 목록이다.
원장은 append-only 구조로, 기존 데이터를 변경하지 않고 새 항목만 추가된다.
Filecoin DSN 프로토콜은 Filecoin의 증명 검증을 지원하는 어떤 원장 위에서도 구현될 수 있다.
6장에서 우리는 *유용한 작업(useful work)* 에 기반한 원장을 구성하는 방법을 제시한다.

4.1.4 시장(The Markets)

Filecoin의 저장 수요와 공급은 두 개의 시장에서 만난다: *스토리지 마켓(Storage Market)* 과 *리트리벌 마켓(Retrieval Market)*.
이 두 시장은 탈중앙화된 거래소(Decentralized Exchange, DEX)의 형태를 가지며, 세부 내용은 5장에서 설명된다.

요약하면, 클라이언트와 채굴자는 요청하거나 제공하려는 서비스의 가격을 설정하고, 해당 주문을 각 시장에 제출한다.
이 거래소들은 클라이언트와 채굴자가 서로의 제안을 확인하고 거래(deal)를 성사시킬 수 있는 메커니즘을 제공한다.
네트워크는 *Manage* 프로토콜을 실행함으로써, 요청된 서비스가 성공적으로 제공되었을 때 채굴자가 보상을 받고 클라이언트가 요금을 지불하도록 보장한다.

4.2 데이터 구조(Data Structures)

조각(Pieces)
조각(piece) 은 클라이언트가 DSN 내에 저장하는 데이터의 일부를 의미한다.
예를 들어, 데이터는 의도적으로 여러 조각으로 나뉘어 각각이 다른 스토리지 채굴자 집합에 의해 저장될 수 있다.

섹터(Sectors)
섹터(sector) 는 스토리지 채굴자가 네트워크에 제공하는 일정량의 디스크 공간을 의미한다.
채굴자들은 클라이언트로부터 받은 조각을 자신의 섹터에 저장하고, 그 대가로 토큰을 획득한다.
조각을 저장하기 위해 스토리지 채굴자는 자신의 섹터를 네트워크에 *서약(pledge)* 해야 한다.

할당 테이블(AllocationTable)
AllocTable 은 각 조각과 그것이 할당된 섹터를 추적하는 데이터 구조이다.
이 테이블은 원장의 매 블록마다 업데이트되며, 그 머클 루트(Merkle root)가 최신 블록에 저장된다.
실제로 이 테이블은 DSN의 상태를 유지하며, 증명 검증 시 빠른 조회를 가능하게 한다.
자세한 내용은 그림 5를 참조하라.

주문(Orders)
주문(order) 은 서비스 요청 또는 제공 의사를 명시한 문(statement)이다.
클라이언트는 서비스를 요청하기 위해 시장에 *매수 주문(bid order)* 을 제출하며
(데이터 저장의 경우 스토리지 마켓, 데이터 검색의 경우 리트리벌 마켓),
채굴자는 서비스를 제공하기 위해 *매도 주문(ask order)* 을 제출한다.
주문의 데이터 구조는 그림 10에 제시되어 있으며, 마켓 프로토콜의 세부 내용은 5장에서 설명된다.

주문장(Orderbook)
Orderbook 은 주문들의 집합을 의미한다.
스토리지 마켓의 주문장은 5.2.2절에, 리트리벌 마켓의 주문장은 5.3.2절에 자세히 설명되어 있다.

서약(Pledge)
서약(pledge) 은 네트워크에 저장 공간(특히 섹터)을 제공하겠다는 약속이다.
스토리지 채굴자는 스토리지 마켓에서 주문을 수락하기 시작하기 전에, 자신의 서약을 원장에 제출해야 한다.
서약에는 약속된 섹터의 크기와 스토리지 채굴자가 예치한 담보(collateral)가 포함된다(세부 내용은 그림 5 참조).

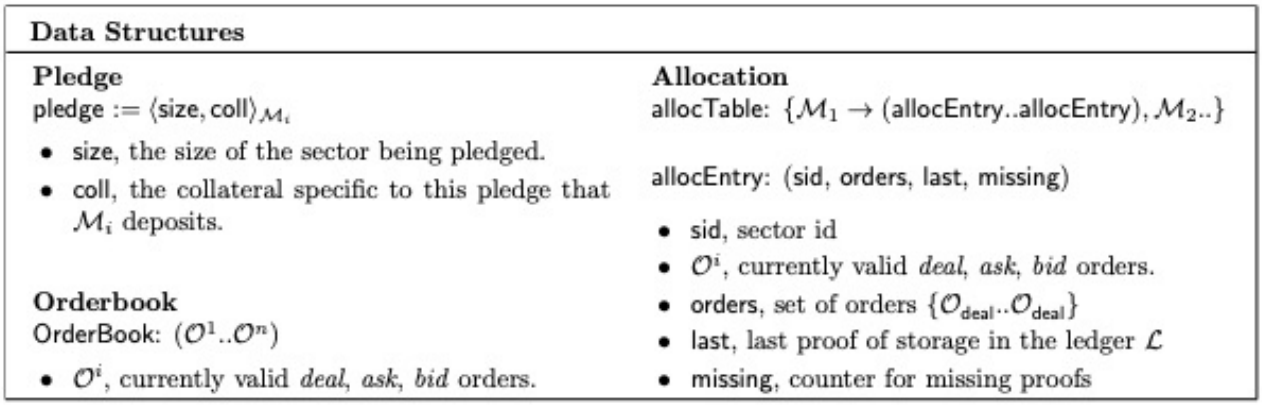


그림 5: DSN 체계의 데이터 구조들

4.3 프로토콜(Protocol)

이 절에서는 *클라이언트*, *네트워크*, *채굴자*가 수행하는 작업을 통해 Filecoin DSN의 전체적인 구조를 개관한다.
Get 및 *Put* 프로토콜의 절차는 그림 7에, *Manage* 프로토콜은 그림 8에 제시되어 있으며, 예시 실행 절차는 그림 6에 나타나 있다.
Filecoin 프로토콜 전체 개요는 그림 1에 제시되어 있다.

4.3.1 클라이언트 주기(Client Cycle)

클라이언트의 주기적 동작을 간략히 개관한다.
이후 프로토콜의 자세한 설명은 5장에서 다룬다.

1. Put: *클라이언트가 Filecoin에 데이터를 저장한다.*

클라이언트는 Filecoin 토큰으로 스토리지 채굴자에게 비용을 지불하여 자신의 데이터를 저장할 수 있다.
Put 프로토콜의 상세 절차는 5.2절에서 다룬다.

클라이언트는 스토리지 마켓의 주문장(orderbook)에 *매수 주문(bid order)* 을 제출(즉, 블록체인에 주문을 등록)함으로써 Put 프로토콜을 시작한다.
채굴자로부터 일치하는 *매도 주문(ask order)* 이 발견되면, 클라이언트는 해당 조각(piece)을 채굴자에게 전송한다.
양 당사자는 거래 세부 내용을 담은 *거래 주문(deal order)* 에 서명하고 이를 스토리지 마켓의 주문장에 제출한다.

클라이언트는 다중 주문을 제출하거나 주문 내에 복제 계수(replication factor)를 지정함으로써 자신의 조각을 몇 개의 물리적 복제본으로 저장할지 결정할 수 있다.
높은 중복도(redundancy)는 더 높은 저장 장애 허용(fault tolerance)을 제공한다.

---

2. Get: *클라이언트가 Filecoin에서 데이터를 검색한다.*

클라이언트는 Filecoin 토큰을 지불하여 DSN 내에 저장된 데이터를 리트리벌 채굴자로부터 검색할 수 있다.
Get 프로토콜의 상세 절차는 5.3절에서 다룬다.

클라이언트는 리트리벌 마켓의 주문장에 *매수 주문(bid order)* 을 제출함으로써(즉, 네트워크에 주문을 gossip 형태로 전파) Get 프로토콜을 시작한다.
채굴자로부터 일치하는 *매도 주문(ask order)* 이 발견되면, 클라이언트는 해당 조각(piece)을 채굴자로부터 수신한다.
데이터를 성공적으로 수신하면, 양 당사자는 *거래 주문(deal order)* 에 서명하고 이를 블록체인에 제출하여 교환이 성공적으로 이루어졌음을 확정한다.

4.3.2 채굴 주기(Mining Cycle, for Storage Miners)

이 절에서는 스토리지 채굴자의 채굴 주기를 비공식적으로 개관한다.

1. Pledge: *스토리지 채굴자가 네트워크에 저장 공간을 서약한다.*

스토리지 채굴자는 블록체인 상의 *pledge* 트랜잭션을 통해 담보를 예치함으로써 자신의 저장 공간을 네트워크에 서약한다.
이 과정은 `Manage.PledgeSector` 함수로 수행된다.
담보는 서비스를 제공하기로 한 기간 동안 예치되며, 채굴자가 자신이 저장하기로 약속한 데이터에 대해 올바른 저장 증명(proof of storage)을 생성할 경우 반환된다.
일부 증명이 실패할 경우, 그 비율에 해당하는 담보가 차감된다.

서약 트랜잭션이 블록체인에 포함되면, 채굴자는 스토리지 마켓에서 자신의 저장 공간을 제공할 수 있다.
채굴자는 가격을 설정하고, 시장의 주문장에 *매도 주문(ask order)* 을 추가한다.

```
Manage.PledgeSector
● 입력:
   – 현재 할당 테이블 allocTable
   – 서약 요청 pledge
● 출력: 업데이트된 할당 테이블 allocTable′
```

---

2. 주문 수신(Receive Orders): *스토리지 채굴자가 스토리지 마켓에서 요청을 수신한다.*

서약 트랜잭션이 블록체인(즉, AllocTable)에 기록되면, 채굴자는 자신의 저장 공간을 스토리지 마켓에서 제공할 수 있다.
채굴자는 가격을 설정하고, `Put.AddOrders` 함수를 통해 시장의 주문장에 *매도 주문* 을 추가한다.

```
Put.AddOrders
● 입력: 주문 목록 O1..On
● 출력: 비트 b (성공 시 1)
```

채굴자는 `Put.MatchOrders` 를 통해 자신의 주문이 클라이언트의 *매수 주문(bid order)* 과 일치하는지 확인한다.

```
Put.MatchOrders
● 입력:
   – 현재 스토리지 마켓 주문장(Storage Market OrderBook)
   – 일치 여부를 확인할 질의 주문 Oq
● 출력: 일치하는 주문들 O1..On
```

주문이 매칭되면, 클라이언트는 데이터를 스토리지 채굴자에게 전송한다.
채굴자는 데이터를 수신하면서 `Put.ReceivePiece` 함수를 실행한다.
데이터 수신이 완료되면, 채굴자와 클라이언트는 *거래 주문(deal order)* 에 서명하고 이를 블록체인에 제출한다.

```
Put.ReceivePiece
● 입력:
   – 채굴자 Mj의 서명 키
   – 현재 주문장 OrderBook
   – 매도 주문 Oask
   – 매수 주문 Obid
   – 조각 p
● 출력: 클라이언트 Ci와 채굴자 Mj가 서명한 거래 주문 Odeal
```

---

3. Seal: *스토리지 채굴자가 향후 증명을 위해 조각을 준비한다.*

스토리지 채굴자의 저장 공간은 섹터 단위로 나뉘며, 각 섹터에는 채굴자에게 할당된 조각들이 포함된다.
네트워크는 할당 테이블을 통해 각 스토리지 채굴자의 섹터 상태를 추적한다.
섹터가 채워지면, 해당 섹터는 *봉인(sealed)* 된다.
Sealing은 섹터 내 데이터를 복제본(replica)으로 변환하는 느리고 순차적인 연산으로,
이 복제본은 스토리지 채굴자의 공개키에 연관된 고유한 물리적 사본이 된다.
이 과정은 3.4절에서 설명한 *Proof-of-Replication* 수행에 필수적인 단계이다.

```
Manage.SealSector
● 입력:
   – 채굴자 공개/개인키 쌍 M
   – 섹터 인덱스 j
   – 할당 테이블 allocTable
● 출력: Seal 증명 πSEAL, 루트 해시 rt
```

4. Prove: *스토리지 채굴자가 약속된 데이터를 실제로 저장하고 있음을 증명한다.*

스토리지 채굴자가 데이터를 할당받으면, 해당 데이터를 실제로 저장하고 있음을 보장하기 위해 반복적으로 복제 증명(Proofs of Replication)을 생성해야 한다(자세한 내용은 3장을 참조).
이 증명들은 블록체인에 게시되며, 네트워크가 이를 검증한다.

```
Manage.ProveSector
● 입력:
   – 채굴자 공개/개인키 쌍 M
   – 섹터 인덱스 j
   – 도전값(challenge) c
● 출력: 저장 증명 πPOS
```

4.3.3 채굴 주기(Mining Cycle, for Retrieval Miners)

이 절에서는 리트리벌 채굴자의 채굴 주기를 비공식적으로 개관한다.

1. Receive Orders: *리트리벌 채굴자가 리트리벌 마켓에서 데이터 요청을 수신한다.*

리트리벌 채굴자는 네트워크에 자신의 *매도 주문(ask order)* 을 gossip 방식으로 전파하여 자신이 보유한 데이터 조각(pieces)을 알린다.
그들은 가격을 설정하고, 시장의 주문장(orderbook)에 매도 주문을 추가한다.

```
Get.AddOrders
● 입력: 주문 목록 O1..On
● 출력: 없음
```

이후 리트리벌 채굴자는 자신의 주문이 클라이언트의 *매수 주문(bid order)* 과 일치하는지 확인한다.

```
Get.MatchOrders
● 입력:
   – 현재 리트리벌 마켓 주문장(Retrieval Market OrderBook)
   – 일치 여부를 확인할 질의 주문 Oq
● 출력: 일치하는 주문들 O1..On
```

---

2. Send: *리트리벌 채굴자가 클라이언트에게 조각을 전송한다.*

주문이 일치하면, 리트리벌 채굴자는 해당 조각(piece)을 클라이언트에게 전송한다(세부 내용은 5.3절 참조).
데이터 수신이 완료되면, 채굴자와 클라이언트는 *거래 주문(deal order)* 에 서명하고 이를 블록체인에 제출하여 거래가 성공적으로 수행되었음을 확정한다.

```
Put.SendPiece
● 입력:
   – 매도 주문 Oask
   – 매수 주문 Obid
   – 조각 p
● 출력: 채굴자 Mi가 서명한 거래 주문 Odeal
```

4.3.4 네트워크 주기(Network Cycle)

이 절에서는 네트워크가 수행하는 작업을 비공식적으로 개관한다.

1. Assign: *네트워크가 클라이언트의 데이터를 스토리지 채굴자의 섹터에 할당한다.*

클라이언트는 스토리지 마켓에서 *매수 주문(bid order)* 을 제출함으로써 Put 프로토콜을 시작한다.
매수 및 매도 주문이 일치하면, 관련 당사자들은 교환 계약에 공동으로 서명하고 *거래 주문(deal order)* 을 시장에 제출한다.
이 시점에서 네트워크는 해당 데이터를 채굴자에게 *할당(assign)* 하고, 이를 할당 테이블에 기록한다.

```
Manage.AssignOrders
● 입력:
   – 거래 주문들 Odeal1..Odealm
   – 현재 할당 테이블 allocTable
● 출력: 업데이트된 할당 테이블 allocTable′
```

---

2. Repair: *네트워크가 오류를 탐지하고 이를 복구한다.*

모든 저장 할당 정보는 네트워크 내 모든 참여자에게 공개된다.
네트워크는 매 블록마다 각 데이터 할당에 필요한 증명(proof)이 존재하는지 확인하고, 해당 증명이 유효한지를 검증하며, 그 결과에 따라 다음과 같이 조치한다:

- 증명이 누락되거나 유효하지 않은 경우, 네트워크는 스토리지 채굴자의 담보 일부를 몰수한다.
- 시스템 매개변수 Δfault로 정의된 임계값 이상으로 증명이 누락되거나 유효하지 않은 경우, 네트워크는 해당 스토리지 채굴자를 *결함(faulty)* 으로 간주하고, 해당 주문을 실패로 처리한 뒤 동일한 조각에 대한 새로운 주문을 시장에 다시 등록한다.
- 해당 조각을 저장하던 모든 스토리지 채굴자가 결함 상태인 경우, 해당 조각은 손실된 것으로 간주되며, 클라이언트에게 환불이 이루어진다.

```
Manage.RepairOrders
● 입력:
   – 현재 시각 t
   – 현재 원장 L
   – 저장 할당 테이블 allocTable
● 출력:
   – 복구해야 할 거래 주문들 Odeal1..Odealm
   – 업데이트된 할당 테이블 allocTable′
```

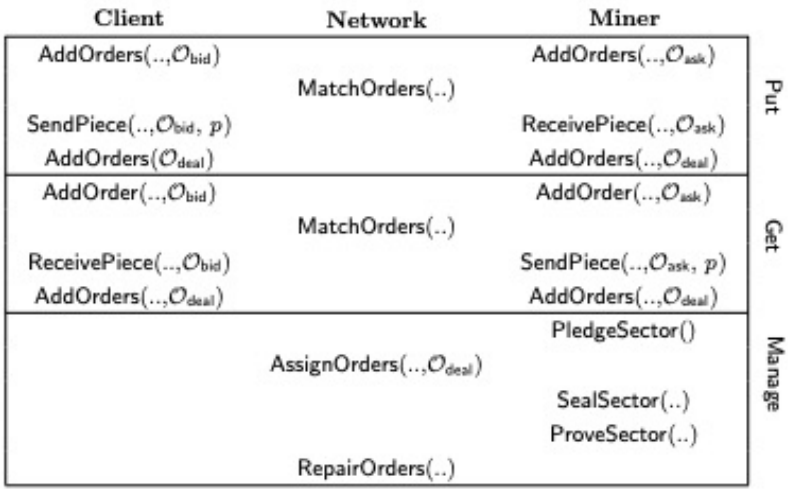


그림 6: Filecoin DSN의 예시 실행 흐름.
참여자별로 그룹화되어 있으며, 행(row) 단위로 시간 순서대로 정렬되어 있다.

4.4 보장과 요구사항(Guarantees and Requirements)

다음은 Filecoin DSN이 *무결성(integrity)*, *검색 가능성(retrievability)*, *공개 검증 가능성(public verifiability)*, *인센티브 정합성(incentive-compatibility)* 을 달성하는 방식에 대한 개념적 설명이다.

- **무결성 달성(Achieving Integrity):**
  각 조각(piece)은 그 암호학적 해시값을 이름으로 가진다.
  클라이언트는 Put 요청 후, Get을 통해 데이터를 검색하고 수신한 콘텐츠의 무결성을 검증하기 위해 해당 해시값만 저장하면 된다.
- **검색 가능성 달성(Achieving Retrievability):**
  Put 요청 시 클라이언트는 복제 계수(replication factor)와 원하는 소거 부호(erasure coding) 유형을 지정함으로써 저장을 (f,m)-내결함성(tolerant)으로 설정한다.
  즉, m명의 스토리지 채굴자가 데이터를 저장할 때 최대 f개의 장애가 허용된다.
  클라이언트는 데이터를 여러 스토리지 채굴자에게 분산 저장함으로써, 일부 채굴자가 오프라인이 되거나 사라지더라도 복구 가능성을 높일 수 있다.
- **공개 검증 및 감사 가능성 달성(Achieving Public Verifiability and Auditability):**
  스토리지 채굴자는 자신이 저장 중인 데이터에 대한 증명(π_{SEAL}, π_{POST})을 블록체인에 제출해야 한다.
  네트워크의 모든 사용자는 위탁된 데이터에 직접 접근하지 않고도 이러한 증명의 유효성을 검증할 수 있다.
  증명들은 블록체인에 저장되어 있기 때문에, 언제든 감사 가능한 운영 이력(trace)으로 남는다.
- **인센티브 정합성 달성(Achieving Incentive Compatibility):**
  비공식적으로, 채굴자는 실제로 제공하는 저장 공간에 대해 보상을 받는다.
  채굴자가 데이터를 저장하기로 약속하면, 이에 대한 증명을 지속적으로 생성해야 한다.
  증명을 누락한 채굴자는 담보 일부를 몰수당하고 보상을 받지 못한다.
- **기밀성 달성(Achieving Confidentiality):**
  자신의 데이터를 비공개로 저장하기를 원하는 클라이언트는 네트워크에 제출하기 전에 데이터를 암호화해야 한다.

Put Protocol

Market

AddOrders

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

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

MatchOrders

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

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

Exchange

SendPiece

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

1) Get identity of $\mathcal{M}_i$ from $\mathcal{O}_{\mathsf{ask}}$ signature
2) Send $(\mathcal{O}_{\mathsf{ask}},\mathcal{O}_{\mathsf{bid}},p)$ to $\mathcal{M}_i$
3) Receive $\mathcal{O}_{\mathsf{deal}}$ signed by $\mathcal{M}_i$
4) Check if $\mathcal{O}_{\mathsf{deal}}$ is valid according to Definition 5.2
5) Output $\mathcal{O}_{\mathsf{deal}}$

ReceivePiece

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

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

Get Protocol

Market

AddOrders

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

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

MatchOrders

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

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

Exchange

SendPiece

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

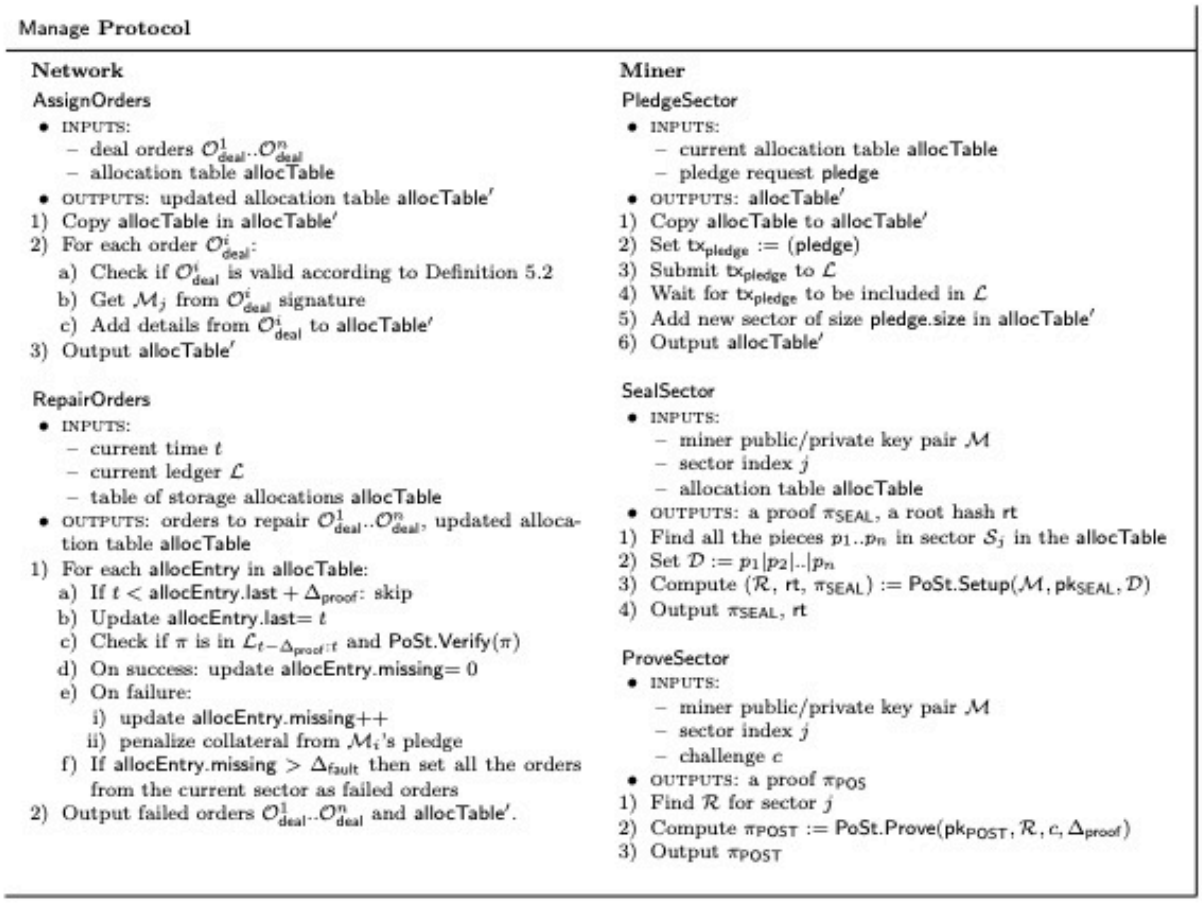
1) Create $\mathcal{O}_{\mathsf{deal}}$:
   a) Set $\mathcal{O}_{\mathsf{deal}}$.ask := $\mathcal{O}_{\mathsf{ask}}$
   b) Set $\mathcal{O}_{\mathsf{deal}}$.bid := $\mathcal{O}_{\mathsf{deal}}$
2) Get identity of $\mathcal{C}_i$ from $\mathcal{O}_{\mathsf{bid}}$ signature
3) Setup a micropayment channel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

ReceivePiece

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

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

그림 7: Filecoin DSN 내 *Put* 및 *Get* 프로토콜의 구조



Manage Protocol

Network

AssignOrders

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

1) Copy allocTable in allocTable′
2) For each order $\mathcal{O}^i_{\text{deal}}$:
   a) Check if $\mathcal{O}^i_{\text{deal}}$ is valid according to Definition 5.2
   b) Get $\mathcal{M}_j$ from $\mathcal{O}^i_{\text{deal}}$ signature
   c) Add details from $\mathcal{O}^i_{\text{deal}}$ to allocTable′
3) Output allocTable′

RepairOrders

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

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

Miner

PledgeSector

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

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

SealSector

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

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

ProveSector

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

1) Find $\mathcal{R}$ for sector j
2) Compute $\pi_{\text{POST}} := \text{PoSt.Prove}(\text{pk}_{\text{POST}}, \mathcal{R}, c, \Delta_{\text{proof}})$
3) Output π_{POST}

그림 8: Filecoin DSN 내 *Manage* 프로토콜의 구조

---

5. Filecoin 스토리지 및 리트리벌 마켓 (Filecoin Storage and Retrieval Markets)

Filecoin에는 **스토리지 마켓(Storage Market)** 과 **리트리벌 마켓(Retrieval Market)** 의 두 가지 시장이 존재한다.
두 시장은 구조적으로 동일하지만 설계 목적이 다르다.

- 스토리지 마켓에서는 클라이언트가 **스토리지 채굴자(Storage Miners)** 에게 비용을 지불하고 데이터를 저장한다.
- 리트리벌 마켓에서는 클라이언트가 **리트리벌 채굴자(Retrieval Miners)** 에게 비용을 지불하고 데이터를 검색한다.

두 경우 모두 클라이언트와 채굴자는 자신들의 공급/수요 가격을 설정하거나, 시장의 현재 가격을 수락할 수 있다.

이 교환들은 **네트워크(Network)** — 즉, Filecoin 전체 노드의 네트워크 — 에 의해 운영된다.
네트워크는 서비스가 올바르게 제공되었을 때, 채굴자가 클라이언트로부터 정당한 보상을 받도록 보장한다.

5.1 검증 가능한 시장(Verifiable Markets)

교환 시장(Exchange Markets)은 특정 상품 또는 서비스를 교환하도록 설계된 프로토콜이다.
이들은 구매자와 판매자가 거래를 수행할 수 있도록 함으로써 교환을 촉진한다.
Filecoin의 목적상, 이러한 교환은 **검증 가능(verifiable)** 해야 한다.
즉, 탈중앙화된 참여자 네트워크가 구매자와 판매자 간의 거래를 독립적으로 검증할 수 있어야 한다.

우리는 **검증 가능한 시장(Verifiable Markets)** 의 개념을 제시한다.
이 시장에서는 단일 중앙 기관이 교환을 통제하지 않으며, 거래는 투명하고 누구나 가명으로 참여할 수 있다.
검증 가능한 시장 프로토콜은 재화/서비스의 교환을 탈중앙화 방식으로 운영하며,
주문장(orderbook)의 일관성, 거래 정산(settlement), 서비스의 올바른 실행이 참여자들 — 즉,
Filecoin의 경우 채굴자와 전체 노드들 — 에 의해 독립적으로 검증된다.

검증 가능한 시장의 단순화된 구조는 다음과 같다.

정의 5.1.
검증 가능한 시장(Verifiable Market) 은 **주문 매칭(order matching)** 과 **정산(settlement)** 의 두 단계로 구성된 프로토콜이다.
주문(order) 은 특정 자산, 상품 또는 서비스의 매수 혹은 매도를 위한 의사 표현이며,
주문장(orderbook) 은 모든 활성 주문의 목록이다.

검증 가능한 시장 프로토콜(Verifiable Market Protocol)

주문 매칭(Order Matching):

1. 참여자들은 주문장에 *매수(buy)* 주문과 *매도(sell)* 주문을 추가한다.
2. 두 주문이 일치하면, 당사자들은 상호 교환을 약속하는 *거래 주문(deal order)* 을 공동으로 생성하고, 이를 네트워크에 전파하여 주문장에 추가한다.

정산(Settlement):

3. 네트워크는 판매자가 자신의 교환 또는 서비스를 올바르게 수행했음을 증명하는 암호학적 증명을 요구함으로써, 상품 또는 서비스의 전달이 정확히 이루어졌는지를 보장한다.
4. 검증이 완료되면, 네트워크는 결제를 처리하고 주문장에서 해당 주문들을 제거한다.

Verifiable Market Protocol

Order matching:

1. Participants add *buy* orders and *sell* orders to the orderbook.
2. When two orders match, involved parties jointly create a *deal* order that commits the two parties to the exchange, and propagate it to the network by adding it to the orderbook.

Settlement:

3. The network ensures that the transfer of goods or services has been executed correctly, by requiring sellers to generate cryptographic proofs for their exchange/service.
4. On success, the network processes the payments and clears the orders from the orderbook.

그림 9: *검증 가능한 시장(Verifiable Markets)* 의 일반적 프로토콜 구조

5.2 스토리지 마켓(Storage Market)

스토리지 마켓은 클라이언트(즉, 구매자)가 자신의 데이터를 저장할 수 있도록 요청하고, 스토리지 채굴자(즉, 판매자)가 저장 공간을 제공할 수 있게 하는 *검증 가능한 시장(verifiable market)* 이다.

5.2.1 요구사항(Requirements)

스토리지 마켓 프로토콜은 다음 요구사항을 충족하도록 설계된다.

- **온체인 주문장(In-chain orderbook):**
  다음 두 가지 사항이 중요하다.
  1. 스토리지 채굴자의 주문은 공개되어야 하며, 네트워크는 항상 최저가를 인식할 수 있어야 한다.
     이를 통해 클라이언트는 더 나은 정보에 기반해 자신의 주문을 결정할 수 있다.
  2. 클라이언트는 최저가를 수락하더라도 반드시 자신의 주문을 주문장에 제출해야 한다.
     이를 통해 시장은 새로운 제안에 즉시 반응할 수 있다.
     따라서, Filecoin 블록체인에 주문이 평문(clear) 형태로 기록되어야 주문장에 추가될 수 있다.
- **참여자의 자원 서약(Participants committing their resources):**
  서비스 불이행(disservice)을 방지하기 위해, 양측 모두 자신의 자원에 대한 서약(commitment)을 해야 한다.
  즉, 스토리지 채굴자가 서비스를 제공하지 않는 경우와 클라이언트가 충분한 자금을 보유하지 않은 경우를 모두 방지하기 위함이다.
  스토리지 마켓에 참여하려면 스토리지 채굴자는 DSN 내 제공할 저장 용량에 비례하는 담보(collateral)를 *서약(pledge)* 해야 한다(자세한 내용은 4.3.3절 참조).
  이를 통해 네트워크는 스토리지 채굴자가 자신이 저장하기로 약속한 조각(pieces)에 대한 저장 증명을 제출하지 않을 경우, 해당 채굴자에게 벌칙을 부과할 수 있다.
  마찬가지로 클라이언트 역시 주문에서 명시한 금액을 예치하여, 결제 시점(settlement)에서 자금의 가용성과 약속 이행을 보장해야 한다.

- **자체적 오류 복구(Self-organization to handle faults):**
  주문은 스토리지 채굴자가 합의된 기간 동안 지속적으로 저장 증명을 제출한 경우에만 정산(settlement)된다.
  네트워크는 이러한 증명이 존재하며 올바른지를 검증할 수 있어야 하며, 4.3.4절 *Repair* 부분에서 설명된 규칙에 따라 적절히 조치해야 한다.

5.2.2 데이터 구조(Datastructures)

Put 주문(Put Orders).
주문에는 세 가지 유형이 있다: *매수 주문(bid orders)*, *매도 주문(ask orders)*, 그리고 *거래 주문(deal orders)*.
스토리지 채굴자는 저장 공간을 제공하기 위해 *매도 주문(ask)* 을 생성하고,
클라이언트는 데이터를 저장하기 위해 *매수 주문(bid)* 을 생성한다.
양측이 가격에 합의하면, 함께 *거래 주문(deal)* 을 생성한다.
이러한 주문의 데이터 구조는 그림 10에 자세히 제시되어 있으며, 주문의 매개변수는 명시적으로 정의되어 있다.

Put 주문장(Put Orderbook).
스토리지 마켓의 주문장은 현재 유효(valid)하고 활성(open) 상태인 *매도(ask)*, *매수(bid)*, *거래(deal)* 주문의 집합이다.
사용자들은 그림 7에 정의된 *Put* 프로토콜 내 `AddOrders`, `MatchOrders` 메서드를 통해 주문장과 상호작용할 수 있다.

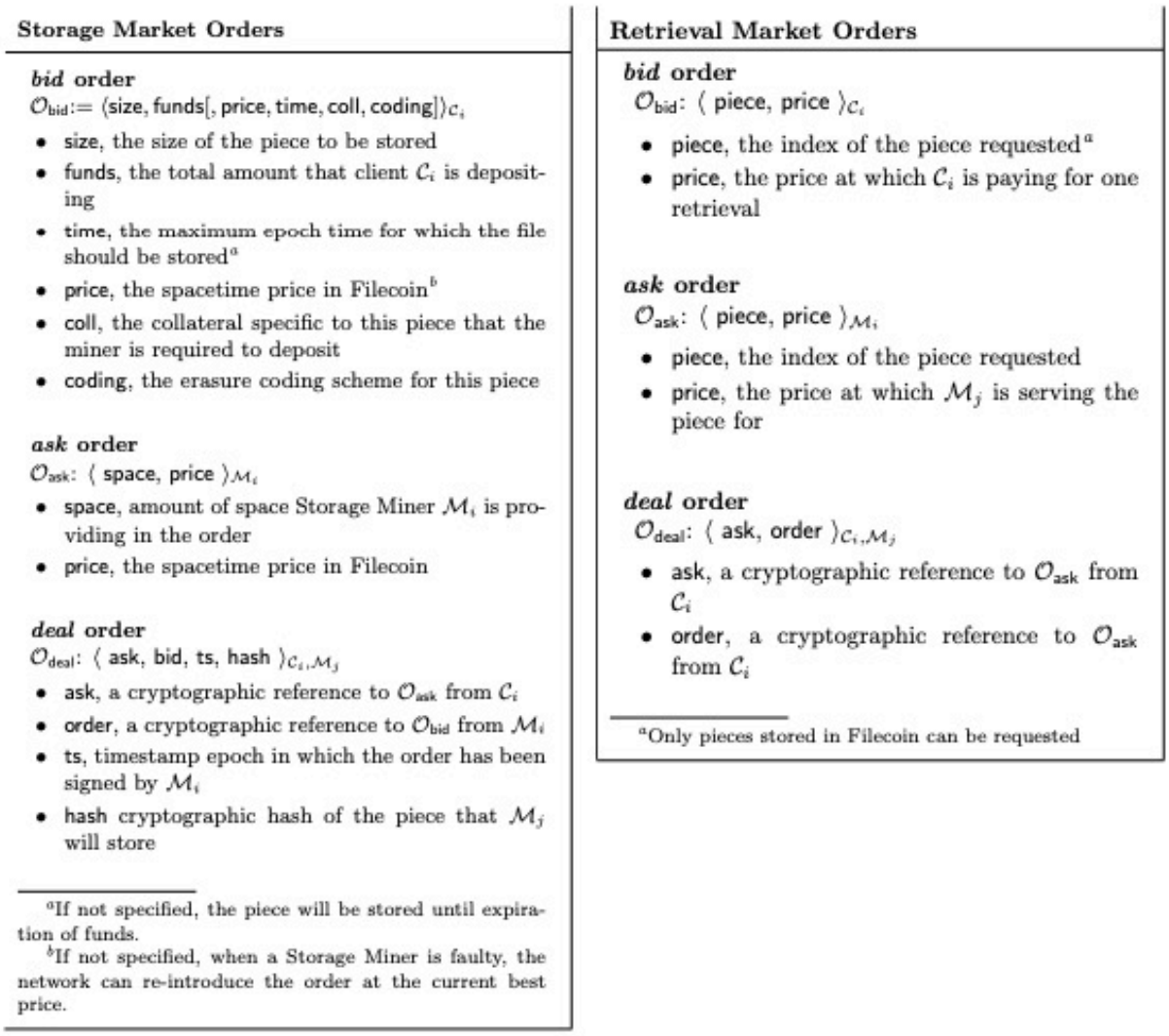
주문장은 공개되어 있으며, 모든 정직한 사용자는 동일한 주문장 상태를 본다.
각 epoch마다 새로운 블록에 새로운 주문 트랜잭션(t_{order})이 포함되면 주문장에 추가되고,
주문이 취소(canceled), 만료(expired), 혹은 정산(settled)되면 제거된다.
주문이 유효한 경우, 블록체인 블록에 기록되며, 그와 동시에 주문장에 반영된다.

정의 5.2.
매수 주문(bid), *매도 주문(ask)*, *거래 주문(deal)* 의 유효성(validity)은 다음과 같이 정의된다.

- **(유효한 매수 주문 Valid bid order):**
  클라이언트 C_i의 매수 주문 $O{bid} := (size, funds, f, price, time, coll, coding){C_i}$ 는 다음 조건을 만족할 때 유효하다.
  - C_i가 자신의 계정에 충분한 자금을 보유하고 있을 것.
  - `time`이 과거 시점으로 설정되지 않았을 것.
  - 주문이 최소한 특정 epoch 수 이상 저장을 보장해야 할 것.
- **(유효한 매도 주문 Valid ask order):**
  스토리지 채굴자 M_i의 매도 주문 $O{ask} := (space, price){M_i}$ 는 다음 조건을 만족할 때 유효하다.
  - M_i가 유효한 서약(pledge)을 완료했으며, 해당 서약이 지정된 기간(epoch) 내에 만료

되지 않을 것.
    - `space`가 M_i의 가용 저장 용량(즉, 서약된 저장 용량에서 이미 주문장에 등록된 저장량(ask 및 deal 주문)을 뺀 값)보다 작을 것.
- **(유효한 거래 주문 Valid deal order):**
  거래 주문 $O{deal} := (ask, bid, ts){C_i, M_j}$ 는 다음 조건을 만족할 때 유효하다.
    - `ask`가 스토리지 마켓 주문장 내 존재하는 O_{ask} 를 참조하며,
      해당 주문이 다른 어떤 거래 주문에서도 언급되지 않았고, C_i에 의해 서명되었을 것.
    - `bid`가 스토리지 마켓 주문장 내 존재하는 O_{bid} 를 참조하며,
      해당 주문이 다른 어떤 거래 주문에서도 언급되지 않았고, M_j에 의해 서명되었을 것.
    - `ts`(timestamp)가 미래 시점이 아니고, 너무 오래된 과거 시점으로 설정되지 않았을 것.

비고(Remarks).
만약 악의적인 클라이언트가 스토리지 채굴자로부터 서명된 거래 주문을 받은 뒤, 이를 주문장에 추가하지 않는다면,
스토리지 채굴자는 해당 거래에 약속된 저장 공간을 재사용할 수 없게 된다.
이러한 공격을 방지하기 위해, `ts` 필드는 일정 시간이 지나면 주문이 무효화되도록 설계되어 있으며,
`ts` 이후에는 주문을 주문장에 제출할 수 없게 된다.



Storage Market Orders

***bid* order**

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

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

***ask* order**

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

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

***deal* order**

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

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

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

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

Retrieval Market Orders

***bid* order**

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

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

***ask* order**

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

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

***deal* order**

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

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

[a]Only pieces stored in Filecoin can be requested

그림 10: 스토리지 및 리트리벌 마켓의 주문 데이터 구조

5.2.3 스토리지 마켓 프로토콜(The Storage Market Protocol)

간단히 요약하면, 스토리지 마켓 프로토콜은 **주문 매칭(Order Matching)** 과 **정산(Settlement)** 의 두 단계로 나뉜다.

- **주문 매칭(Order Matching):**
  클라이언트와 스토리지 채굴자는 블록체인에 트랜잭션을 제출하여 주문장을 갱신한다(1단계).
  주문이 매칭되면, 클라이언트는 해당 조각(piece)을 스토리지 채굴자에게 전송하고,
  양측은 공동으로 *거래 주문(deal order)* 에 서명하여 주문장에 제출한다(2단계).
- **정산(Settlement):**
  스토리지 채굴자는 자신의 섹터를 봉인(seal)한다(3a단계).
  이후 조각이 포함된 섹터에 대한 저장 증명(proof of storage)을 정기적으로 생성하여 블록체인에 제출한다(3b단계).
  한편, 네트워크의 다른 노드들은 채굴자가 생성한 증명을 검증하고, 오류가 발견될 경우 이를 복구한다(3c단계).

스토리지 마켓 프로토콜의 상세 절차는 그림 11에 제시되어 있다.

5.3 리트리벌 마켓(Retrieval Market)

리트리벌 마켓은 클라이언트가 특정 조각(piece)의 검색을 요청하고, 리트리벌 채굴자가 이를 제공할 수 있도록 하는 시장이다.
스토리지 채굴자와 달리 리트리벌 채굴자는 데이터를 장기간 저장하거나 저장 증명(proofs of storage)을 생성할 필요가 없다.
네트워크의 모든 사용자는 Filecoin 토큰을 대가로 조각을 제공함으로써 리트리벌 채굴자가 될 수 있다.
리트리벌 채굴자는 클라이언트로부터 직접 데이터를 수신하거나, 리트리벌 마켓에서 획득하거나, 스토리지 채굴자로서 저장 중이던 데이터를 활용할 수도 있다.

5.3.1 요구사항(Requirements)

리트리벌 마켓 프로토콜은 다음 요구사항을 만족하도록 설계된다.

- **오프체인 주문장(Off-chain orderbook):**
  클라이언트는 필요한 조각을 제공하는 리트리벌 채굴자를 찾아 가격에 합의한 후 데이터를 직접 교환할 수 있어야 한다.
  이를 위해 주문장은 블록체인을 통해 운영될 수 없으며, 블록체인 기반 운영은 빠른 데이터 검색 요청의 병목이 된다.
  따라서 각 참여자는 주문장의 *일부(partial view)* 만을 가지며, 모든 주문이 전체 네트워크에 완전히 공유되지 않는다.
  이에 따라 클라이언트와 리트리벌 채굴자는 서로의 주문을 **gossip 방식**으로 전파해야 한다.
- **신뢰 당사자 없는 데이터 교환(Retrieval without trusted parties):**
  "공정한 교환(fair exchange)"에 대한 불가능성 정리는 신뢰 당사자 없이 두 당사자가 교환을 수행하는 것이 불가능하다는 것을 상기시킨다.
  스토리지 마켓에서는 블록체인 네트워크가 탈중앙화된 신뢰 당사자로서 작동하며, 스토리지 채굴자가 제공한 저장 증명을 검증한다.
  그러나 리트리벌 마켓에서는 파일의 실제 전송을 네트워크가 직접 관찰하지 않는다.
  이를 해결하기 위해, 리트리벌 채굴자가 자신의 데이터를 여러 부분으로 나누어 전송하고, 클라이언트는 각 부분을 받을 때마다 소액 결제를 수행하도록 한다.
  이 방식에서는 클라이언트가 결제를 중단하거나 채굴자가 데이터를 보내지 않는 경우, 어느 쪽이든 교환을 즉시 중단할 수 있다.
  단, 이 시스템이 제대로 작동하려면 최소 한 명 이상의 정직한 리트리벌 채굴자가 존재한다는 가정을 둔다.
- **결제 채널(Payment channels):**
  클라이언트는 결제 후 즉시 데이터를 수신하기를 원하며, 리트리벌 채굴자는 결제가 보장될 때만 데이터를 제공하길 원한다.
  공개 원장을 통한 결제 검증은 데이터 검색 요청의 병목이 되므로, 효율적인 **오프체인 결제(off-chain payment)** 가 필요하다.
  Filecoin 블록체인은 낙관적 거래(optimistic transaction)를 가능하게 하는 결제 채널(payment channel)을 지원해야 하며, 분쟁이 발생할 경우에만 블록체인을 사용한다.
  이를 통해 리트리벌 채굴자와 클라이언트는 매우 빠르게 필요한 소액 결제를 교환할 수 있다.
  향후 연구 과제로는, [11, 12]에서 제시된 사례처럼 **결제 채널 네트워크(payment channel**

network) 의 구축이 포함된다.

5.3.2 데이터 구조(Data Structures)

Get 주문(Get Orders).
리트리벌 마켓에는 세 가지 유형의 주문이 존재한다.
클라이언트는 *매수 주문(bid order)* $O{bid}$ *를 생성하고, 리트리벌 채굴자는 매도 주문(ask order)* $O{ask}$ 를 생성한다.
그리고 양측이 거래 조건에 합의하면 공동으로 *거래 주문(deal order)* O_{deal} 을 생성한다.
주문의 데이터 구조는 그림 10에 상세히 제시되어 있다.

Get 주문장(Get Orderbook).
리트리벌 마켓의 주문장은 현재 유효(valid)하고 개방(open)된 *매수(bid)*, *매도(ask)*, *거래(deal)* 주문들의 집합이다.
스토리지 마켓과 달리, 리트리벌 마켓에서는 모든 사용자가 동일한 주문장을 보지 않는다.
주문이 네트워크 내에서 gossip 방식으로 전파되기 때문에, 각 채굴자와 클라이언트는 자신이 관심 있는 주문만 추적한다.

Storage Market Protocol

Order Matching

1. Storage Miner $\mathcal{M}_i$ and Client $\mathcal{C}_i$ add orders to the OrderBook:
   (a) $\mathcal{M}_i$ creates $\mathcal{O}_{\text{ask}}{}^1, \mathcal{O}_{\text{ask}}{}^2, ..$ and $\mathcal{C}_j$ creates $\mathcal{O}_{\text{bid}}{}^1, \mathcal{O}_{\text{bid}}{}^2, ...$
   (b) Orders are submitted to the blockchain via Put.addOrders($\mathcal{O}^1, \mathcal{O}^2, ..$)
   (c) On success, the orders are added to the OrderBook, the funds from $\mathcal{C}_j$ are deposited and the space from $\mathcal{M}_i$ is reserved.
2. When orders match, involved parties jointly create $\mathcal{O}_{\text{deal}}$ and add it to the OrderBook:
   (a) $\mathcal{M}_i$ and $\mathcal{C}_j$ independently query the OrderBook via Put.matchOrders($\mathcal{O}$).
   (b) If $\mathcal{M}_i$ and $\mathcal{C}_j$ have matching orders :
   - $\mathcal{C}_j$ sends the piece p to $\mathcal{M}_i$ via Put.SendPiece($\mathcal{O}_{\text{bid}}$, $\mathcal{O}_{\text{ask}}$, p)
   - $\mathcal{M}_i$ receives the piece p from $\mathcal{C}_j$ via Put.ReceivePiece($\mathcal{O}_{\text{bid}}$, $\mathcal{O}_{\text{ask}}$, p).
   - $\mathcal{M}_i$ signs $\mathcal{O}_{\text{deal}}$ and sends it to $\mathcal{C}_j$

   (c) $\mathcal{C}_j$ signs $\mathcal{O}_{\text{deal}}$ and adds it to the OrderBook via Put.addOrders($\mathcal{O}_{\text{deal}}$)

Settlement

3. The Network checks if the Storage Miners are correctly storing the pieces:
   (a) When a Storage Miner fills a sector, they *seal* it (they create a unique replica) via Manage.SealSector and submit the proof π_{SEAL} and rt to the blockchain.
   (b) Storage Miners generate new proofs at every epoch and add them to the Filecoin blockchain every Δ_{proof} epochs via Manage.ProveSectors.
   (c) The Network runs Manage.RepairOrders at every epoch. If proofs are missing or invalid, the network tries to repair in the following ways:
   - if any proofs are missing or invalid, it penalizes the Storage Miners by taking part of their collateral,
   - if a large amount of proofs are missing or invalid for more than Δ_{fault} epochs, it considers the Storage Miner *faulty*, settles the order as failed and reintroduces a new order for the same piece into the the market,
   - if every Storage Miner storing this piece is faulty, then the piece is lost and the client gets refunded.
4. When the time of the order is expired or funds run out, if the service was correctly provided, the Network processes the payments, and removes the orders.

그림 11: 스토리지 마켓 프로토콜 상세 절차

5.3.3 리트리벌 마켓 프로토콜(The Retrieval Market Protocol)

요약하면, 리트리벌 마켓 프로토콜은 **주문 매칭(order matching)** 과 **정산(settlement)** 의 두 단계로 구성된다.

- **주문 매칭(Order Matching):**
  클라이언트와 리트리벌 채굴자는 주문을 gossip 방식으로 전파하여 주문장에 추가한다(1단계).
  주문이 일치하면, 클라이언트와 리트리벌 채굴자는 **마이크로페이먼트 채널(micropayment channel)** 을 설정한다(2단계).
- **정산(Settlement):**
  리트리벌 채굴자는 조각을 여러 개의 작은 단위로 나누어 순차적으로 클라이언트에게 전송하고,
  각 단위에 대해 클라이언트는 채굴자에게 서명된 영수증(receipt)을 보낸다(3단계).
  리트리벌 채굴자는 이 영수증들을 블록체인에 제출하여 보상을 받는다(4단계).

프로토콜의 전체 절차는 그림 12에 자세히 설명되어 있다.

Retrieval Market Protocol

Order Matching:

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

Settlement:

3. Both parties check whether the piece has been delivered:
   (a) $\mathcal{M}_i$ sends the piece p in parts via Get.SendPiece
   (b) $\mathcal{C}_j$ receives the p in parts and for each part, $\mathcal{C}_j$ acknowledges delivery by sending a micropayment via Get.ReceivePiece
4. When the p has been received by $\mathcal{C}_j$, $\mathcal{M}_j$ can present the micropayments to the network and retrieve the payment, both parties remove their orders from the orderbooks.

그림 12: 리트리벌 마켓 프로토콜 상세 절차

6. 유용한 작업 합의(Useful Work Consensus)

Filecoin DSN 프로토콜은 Filecoin의 증명을 검증할 수 있는 어떤 합의 프로토콜 위에서도 구현될 수 있다.
이 절에서는 *유용한 작업(useful work)* 에 기반한 합의 프로토콜을 어떻게 부트스트랩할 수 있는지를 제시한다.
Filecoin에서는 낭비적인 *Proof-of-Work* 계산 대신, 채굴자들이 *Proof-of-Spacetime*을 생성하는 과정에서 수행하는 실제 저장 행위가 곧 합의 참여의 근거가 된다.

유용한 작업(Useful Work)
합의 프로토콜 내에서 채굴자가 수행하는 작업이 블록체인 보안을 넘어 네트워크 전체에 실제로 가치를 제공한다면, 그 작업을 *유용한 작업(useful work)* 으로 정의한다.

6.1 동기(Motivation)

블록체인 보안은 근본적으로 중요하지만, 전통적인 *Proof-of-Work (PoW)* 체계는 재사용할 수 없는 퍼즐을 풀기 위해 막대한 계산 자원을 낭비하는 문제를 가진다.

- **재사용 불가능한 작업(Non-reusable Work):**
  대부분의 퍼미션리스 블록체인은 해시 함수를 역산하는 것과 같은 계산적으로 어려운 퍼즐을 풀도록 채굴자에게 요구한다.
  그러나 이러한 퍼즐의 해답은 네트워크 보안을 제외하면 아무런 내재적 가치가 없으며, 결과물이 재활용될 수도 없다.
  그렇다면, 이 계산 자원을 보다 유용한 목적에 사용할 수는 없을까?

  유용한 계산으로의 전환 시도:
  몇몇 연구는 채굴자의 연산 능력을 유용한 계산에 재활용하려고 시도했다.
  어떤 연구는 표준 PoW와 병행하여 특별한 계산을 수행하도록 요구했고,
  다른 연구는 PoW 자체를 "해결은 어렵지만 유용한 문제"로 대체했다.
  예를 들어, Primecoin은 새로운 소수(prime number)를 찾는 데 채굴자의 연산력을 재활용하고,
  Ethereum은 PoW와 함께 짧은 프로그램을 실행하도록 요구하며,
  Permacoin은 데이터를 보관(archival)하는 증명과 함께 해시 함수를 역산하도록 설계했다.
  이러한 시도들은 일정 부분 유용한 작업을 수행하지만, 여전히 불필요한 연산 낭비가 크다는 공통적인 한계를 가진다.

- **낭비적인 작업(Wasteful Work):**
  단순히 계산 능력에 의존하여 퍼즐을 해결하는 방식은 장비 비용과 에너지 소비 면에서 비효율적이다.
  특히 병렬화가 쉬운 연산일수록 결과적으로는 계산력 경쟁으로 귀결된다.
  그렇다면, 낭비적인 연산의 양을 줄이는 것은 가능할까?

  낭비 감소 시도:
  이상적으로는 네트워크 자원의 대부분이 유용한 작업에 소비되어야 한다.
  일부 연구에서는 더 에너지 효율적인 방식의 채굴을 제안한다.
  예를 들어, Spacemint는 채굴자가 계산력 대신 디스크 공간을 제공하도록 하여 에너지 효율성을 개선했으나,
  여전히 그 디스크는 무작위 데이터로 채워져 있어 "낭비된 저장 공간"이라는 문제가 남는다.
  또 다른 접근은 PoW를 *Proof-of-Stake (PoS)* 기반 비잔틴 합의로 대체하여,
  시스템 내 지분 비율에 따라 블록 생성권을 투표로 결정하는 방식이다.

Filecoin은 이러한 문제를 해결하기 위해, **사용자 데이터를 저장하는 유용한 작업**을 기반으로 한 새로운 합의 프로토콜을 설계한다.

6.2 Filecoin 합의(Filecoin Consensus)

Filecoin은 *유용한 작업 기반 합의 프로토콜(useful work consensus protocol)* 을 제안한다.
여기서 네트워크가 채굴자를 블록 생성자로 선출할 확률(이를 **채굴자의 투표권(voting power)** 이라 한다)은
해당 채굴자가 네트워크 전체에서 현재 사용 중인 저장 용량(storage in use)에 비례한다.

Filecoin 프로토콜은 채굴자들이 계산력 증대를 위한 하드웨어 투자 대신, 저장 용량에 투자하도록 설계되어 있다.
즉, 채굴자는 데이터를 저장하고 그 저장을 입증하는 증명(Proof-of-Spacetime)을 생성함으로써 합의에 참여할 자격을 얻는다.

6.2.1 채굴 파워 모델링(Modeling Mining Power)

파워 결함 허용(Power Fault Tolerance)
기술 보고서 [13]에서는 *Power Fault Tolerance*라는 개념을 제시하며,
이는 비잔틴 결함(Byzantine fault)을 참여자의 "프로토콜 결과에 대한 영향력(influence)"으로 재정의한 것이다.
각 참여자는 네트워크 상에서 총 논리적 영향력을 나타내는 파워 p 를 가지며,
f 는 결함(또는 악의적) 참여자들이 보유한 파워의 비율을 의미한다.
즉, f 만큼의 파워가 악의적 주체에 의해 통제되는 상황을 고려한다.

Filecoin에서의 파워(Power in Filecoin)

Filecoin에서 채굴자 M_i의 시점 t에서의 파워 p_t^i 는
그 채굴자에게 할당된 모든 저장 공간의 합으로 정의된다.
채굴자의 영향력 I_t^i 는 네트워크 전체 파워 대비 해당 채굴자의 파워 비율로 표현된다.

Filecoin에서 **파워(power)** 는 다음 속성을 갖는다.

- **공개성(Public):**
  네트워크에서 현재 사용 중인 전체 저장 용량은 공개되어 있다.
  블록체인을 통해 각 채굴자의 저장 할당(storage assignment)을 확인할 수 있으므로,
  누구나 각 채굴자의 파워와 네트워크의 총 파워를 계산할 수 있다.
- **공개 검증 가능성(Publicly Verifiable):**
  각 저장 할당에 대해 채굴자는 *Proofs-of-Spacetime*을 생성해야 하며,
  이를 통해 해당 저장 서비스가 실제로 제공되고 있음을 증명한다.
  블록체인을 통해 누구나 채굴자가 주장한 파워가 올바른지 검증할 수 있다.
- **가변성(Variable):**
  채굴자는 언제든 새로운 섹터를 서약(pledge)하고 데이터를 채워 넣음으로써
  네트워크 내에서 자신의 파워를 증가시킬 수 있다.
  이러한 과정을 통해 채굴자는 새로운 저장 증명을 제출함으로써
  네트워크 내에서 자신의 영향력을 동적으로 조정할 수 있다.

6.2.2 Proof-of-Spacetime을 통한 파워 산정(Accounting for Power with Proof-of-Spacetime)

채굴자들은 매 Δproof 블록마다 *Proof-of-Spacetime (PoSt)* 을 네트워크에 제출해야 하며,
해당 증명은 네트워크의 과반(power majority)에 의해 유효하다고 판단될 때만 블록체인에 추가된다.
각 블록이 생성될 때마다 모든 풀노드(full node)는 *AllocTable* 을 업데이트하여 새로운 저장 할당을 추가하고, 만료된 항목을 제거하며, 누락된 증명을 표시한다.

채굴자 M_i의 파워는 *AllocTable* 내 기록을 합산함으로써 계산 및 검증할 수 있으며, 이 방법은 두 가지로 나뉜다.

- **풀노드 검증(Full Node Verification):**
  노드가 전체 블록체인 로그를 보유하고 있는 경우,
  제너시스(genesis) 블록부터 현재 블록까지 *NetworkProtocol* 을 실행하여 M_i의 *AllocTable* 을 읽는다.
  이 과정에서 M_i에게 현재 할당된 저장 공간에 대한 모든 *Proof-of-Spacetime* 이 검증된다.
- **간소화된 저장 검증(Simple Storage Verification):**
  라이트 클라이언트(light client)가 신뢰할 수 있는 출처로부터 최신 블록을 수신할 수 있다고 가정하자.
  이 클라이언트는 네트워크 노드들로부터 다음 정보를 요청할 수 있다.
  1. 채굴자 M_i에 대한 현재 *AllocTable* 항목,
  2. 해당 항목이 마지막 블록의 상태 트리(state tree)에 포함되어 있음을 증명하는 머클 경로(Merkle path),
  3. 제너시스 블록부터 현재 블록까지의 헤더(headers).
     이렇게 하면 라이트 클라이언트는 PoSt의 검증을 네트워크에 위임(delegate)할 수 있다.

파워 계산의 보안성은 *Proof-of-Spacetime*의 보안성에서 비롯된다.
PoSt는 채굴자가 자신에게 실제로 할당된 저장 용량을 속일 수 없음을 보장한다.
즉, 채굴자는 자신이 실제 저장 중인 데이터보다 더 많은 데이터를 저장한다고 주장할 수 없다.
그 이유는 더 많은 저장을 주장하려면 추가 데이터를 가져와 느린 *PoSt.Setup* 과정을 수행해야 하며,
또한 *PoSt.Prove* 연산은 순차적(sequential) 계산이므로 병렬화로 증명 생성을 가속화할 수도 없기 때문이다.

6.2.3 파워를 활용한 합의 달성(Using Power to Achieve Consensus)

Filecoin 합의는 기존 또는 향후의 *Proof-of-Stake (PoS)* 합의 프로토콜을 확장하여 구현할 수 있다.
단, PoS에서의 '지분(stake)'을 Filecoin에서는 '저장 할당(storage assignment)'으로 대체한다.
우리는 PoS의 자원 효율성을 개선하면서, 이전 연구 *Expected Consensus (EC)* [14]를 기반으로 한 구성을 제안한다.

이 전략에서 각 라운드(epoch)마다 한 명 이상의 채굴자가 리더로 선출되며,
리더로 선출될 확률은 각 채굴자의 저장 할당량에 비례한다.

일부 epoch에는 리더가 없을 수도 있고, 여러 명의 리더가 동시에 존재할 수도 있다.
리더들은 체인에 새로운 블록을 생성하여 네트워크에 전파하고,
각 epoch마다 하나 이상의 블록이 체인에 추가된다.
만약 리더가 없는(epoch leaderless) 경우, 빈 블록(empty block)이 체인에 추가된다.

Filecoin의 체인은 선형적으로 정렬될 수 있지만, 내부적으로는 **방향성 비순환 그래프(DAG, Directed Acyclic Graph)** 형태의 데이터 구조를 갖는다.
Expected Consensus는 확률적 합의(probabilistic consensus)로서,
각 epoch이 지나갈 때마다 이전 블록들에 대한 확신(confidence)이 점진적으로 높아진다.
결국 대체 가능한 다른 체인 이력이 존재할 확률이 무시할 만큼 작아지면 블록은 *확정(committed)* 상태가 된다.
참여자 과반이 특정 블록이 포함된 체인에 가중치(weight)를 더함으로써(체인을 연장하거나 블록에 서명함으로써) 블록이 확정된다.

채굴자 선출(Electing Miners)

각 epoch마다 채굴자는 자신이 리더로 선출되었는지 여부를 확인한다.
이 절차는 기존 합의 프로토콜(CoA [15], Snow White [16], Algorand [17])과 유사하다.

정의 6.1 (Filecoin에서의 EC 선출, EC Election in Filecoin)
채굴자 M_i가 시점 *t*에서 리더로 선출되기 위한 조건은 다음과 같다:

$$\mathcal{H}((\langle||rand(t)\rangle)M_i)/2^L \leq \frac{p_t^i}{\sum_j p_t^j}$$

여기서 *rand(t)* 는 epoch *t*에서 블록체인으로부터 추출 가능한 공개 난수(public randomness)이며,
p_t^i 는 채굴자 M_i의 파워이다.
임의의 메시지 *m*에 대해 $\mathcal{H}(m)$의 출력 크기는 *L*이라 가정한다.
$\mathcal{H}$ 는 안전한 암호학적 해시 함수이며,
$\langle m \rangle_{M_i}$ 는 채굴자 M_i의 개인키로 서명된 메시지 *m*으로 다음과 같이 정의된다:

$$\langle m \rangle_{M_i} := ((m), SIG_{M_i}(\mathcal{H}(m)))$$

그림 13은 채굴자(*ProveElect*)와 네트워크 노드(*VerifyElect*) 간의 리더 선출 프로토콜 절차를 나타낸다.

이 선출 방식은 **공정성(fairness)**, **비밀성(secrecy)**, **공개 검증 가능성(public verifiability)** 의 세 가지 주요 속성을 제공한다.

- **공정성(Fairness):**
  각 참여자는 매 선거마다 단 한 번의 시도만 할 수 있다.
  이는 서명이 결정적(deterministic)이며 *rand(t)* 가 고정되어 있기 때문이다.

$\mathcal{H}$ 가 안전한 해시 함수라면,
$\mathcal{H}(\langle||rand(t)\rangle M_i)/2^L$ 는 (0,1) 구간에서 균등하게 선택된 실수로 간주된다.
따라서 해당 식이 참이 될 확률은 $p_t^i / \sum_j p_t^j$ 와 같으며,
이는 채굴자의 네트워크 내 파워 비율과 동일하다.
이 확률은 파워의 분할(splitting)이나 풀링(pooling)에 대해 선형(linear)으로 보존된다.
또한 난수값 *rand(t)* 와 시간 *t*는 해당 epoch 이전에는 알 수 없다.

- **비밀성(Secret):**
  개인키 M_i를 보유하지 않은 효율적 적대자는 디지털 서명 가정 하에 서명을 생성할 확률이 무시할 만하다.
- **공개 검증 가능성(Public Verifiability):**
  선출된 리더 $i \in L^t$ 는 *(t, rand(t), $\mathcal{H}$(⟨||rand(t)⟩)/2^L)* 을 제시함으로써 효율적인 검증자에게 자신의 당선을 입증할 수 있다.
  앞선 조건들로 인해, 비밀키 없이 이러한 증명을 생성할 수 있는 효율적 적대자는 존재하지 않는다.

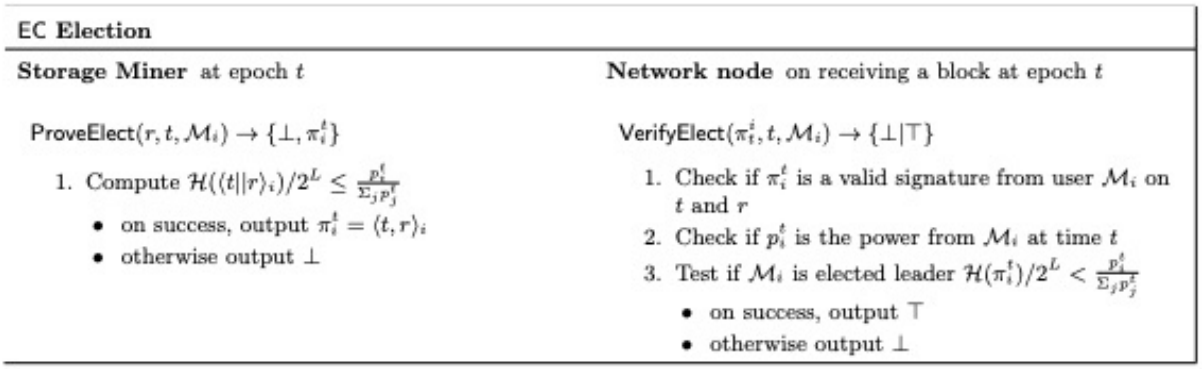


그림 13: Expected Consensus 프로토콜 내 리더 선출 과정

7. 스마트 컨트랙트(Smart Contracts)

Filecoin은 최종 사용자에게 두 가지 기본 연산을 제공한다: **Get** 과 **Put**.
이 두 연산은 클라이언트가 자신의 선호 가격에 따라 데이터를 저장하고 검색할 수 있도록 한다.
이 기본 연산들은 Filecoin의 주요 사용 사례를 모두 포괄하지만, Filecoin은 **스마트 컨트랙트(smart contracts)** 배포를 지원함으로써
이 두 연산을 기반으로 한 훨씬 복잡한 조작과 서비스 설계를 가능하게 한다.

사용자는 세밀한 수준의 저장/검색 요청을 프로그래밍할 수 있으며, 이를 **파일 컨트랙트(File Contracts)** 라고 부른다.
또한 일반적인 **스마트 컨트랙트(Smart Contracts)** 도 작성할 수 있다.
Filecoin은 [18]을 기반으로 한 *컨트랙트 시스템(Contracts System)* 과 *브리지 시스템(Bridge System)* 을 통합하여
Filecoin의 저장 기능을 다른 블록체인으로 확장하고, 반대로 다른 블록체인의 기능을 Filecoin 내로 가져올 수 있도록 설계되어 있다.

Filecoin 생태계에는 다양한 형태의 스마트 컨트랙트가 존재할 것으로 예상되며,
스마트 컨트랙트 개발자 커뮤니티의 활발한 참여가 기대된다.

7.1 Filecoin 내 컨트랙트(Contracts in Filecoin)

스마트 컨트랙트(Smart Contracts) 는 Filecoin 사용자들이 상태를 가진(stateful) 프로그램을 작성할 수 있도록 하며,
이를 통해 토큰을 지출하거나, 스토리지/리트리벌 마켓에서 데이터 저장 또는 검색을 요청하거나,
저장 증명(storage proof)을 검증할 수 있다.
사용자는 원장(ledger)에 트랜잭션을 전송하여 컨트랙트의 함수를 호출하고 상호작용할 수 있다.
Filecoin은 스마트 컨트랙트 시스템을 확장하여 Filecoin 고유의 기능(예: 마켓 연산, 증명 검증 등)을 지원한다.

Filecoin은 **데이터 저장에 특화된 컨트랙트(File Contracts)** 와 **범용 스마트 컨트랙트(Generic Smart Contracts)** 를 모두 지원한다.

- **파일 컨트랙트(File Contracts):**
  사용자는 자신이 저장 서비스를 제공하거나 요청하는 조건을 직접 프로그래밍할 수 있다.
  주요 예시는 다음과 같다.
  1. **특정 채굴자 지정 계약(Contracting miners):**
     클라이언트는 시장에 참여하지 않고도 미리 지정된 채굴자에게 서비스를 요청할 수 있다.
  2. **지불 전략(Payment strategies):**
     클라이언트는 다양한 보상 전략을 설계할 수 있다.
     예를 들어, 시간이 지남에 따라 채굴자에게 점차 더 많은 보상을 지급하거나,
     신뢰할 수 있는 오라클이 제공하는 데이터에 따라 저장 가격을 조정하는 계약을 만들 수 있다.
  3. **티켓 서비스(Ticketing services):**
     채굴자가 토큰을 예치하고, 자신을 대신해 이용자의 저장/검색 비용을 지불하는 계약을 만들 수 있다.
  4. **고급 기능(Complex operations):**
     클라이언트는 데이터를 업데이트할 수 있는 계약을 생성할 수 있다.
- **스마트 컨트랙트(Smart Contracts):**
  사용자는 Ethereum [18] 등과 같이, 트랜잭션에 프로그램을 연결하여 실행할 수 있다.
  이들은 저장과 직접적으로 관련되지 않은 범용 기능을 수행할 수 있으며,
  탈중앙화 네이밍 시스템, 자산 추적(asset tracking), 크라우드세일(crowdsale) 플랫폼 등 다양한 응용이 가능하다.

7.2 다른 시스템과의 통합(Integration with Other Systems)

브리지(Bridges) 는 서로 다른 블록체인을 연결하기 위한 도구이다.
Filecoin은 현재 개발 중인 브리지 시스템을 통해
Filecoin의 저장 기능을 다른 블록체인 기반 플랫폼으로 확장하고,
반대로 외부 블록체인의 기능을 Filecoin으로 가져오는 **크로스체인 상호운용성(cross-chain interoperability)** 을 지원할 예정이다.

- **다른 플랫폼에서의 Filecoin(Filecoin in other platforms):**
  Bitcoin [19], Zcash [20], Ethereum [18], Tezos 등 대부분의 블록체인 플랫폼은 스마트 컨트랙트를 지원하지만, 저장 기능이 매우 제한적이거나 비용이 높다.
  Filecoin은 이러한 플랫폼에 저장 및 검색 기능을 제공하기 위한 *브리지(bridge)* 를 개발하고 있다.
  참고로, IPFS는 이미 여러 스마트 컨트랙트 및 프로토콜 토큰에서 콘텐츠를 참조하고 배포하기 위한 수단으로 사용되고 있다.
  Filecoin 지원이 추가되면, 이들 시스템은 Filecoin 토큰을 사용하여 IPFS 콘텐츠의 실제 저장을 보장할 수 있다.
- **Filecoin 내의 타 플랫폼(Other platforms in Filecoin):**
  Filecoin은 다른 블록체인 서비스를 연결하기 위한 *브리지* 도 제공할 계획이다.
  예를 들어, Zcash와의 통합을 통해 **프라이버시 보호 저장 요청(private storage request)** 기능을 지원할 수 있다.

---

8. 향후 연구(Future Work)

본 논문은 Filecoin 네트워크 구축을 위한 명확하고 통합적인 경로를 제시하지만,
동시에 탈중앙화 스토리지 시스템에 대한 향후 연구의 출발점으로서의 역할도 수행한다.
이 절에서는 향후 연구 과제를 세 가지 범주로 구분하여 제시한다.

이는 (1) 이미 완료되었으나 공식적으로 기술 및 발표되지 않은 연구,
(2) 현재 프로토콜을 개선하기 위한 개방형 문제(open questions),
(3) 프로토콜의 형식적 정립(formalization)을 포함한다.

8.1 진행 중인 연구(On-going Work)

다음 주제들은 현재 진행 중인 Filecoin 관련 연구 항목들이다.

- 블록마다 기록되는 **Filecoin 상태 트리(state tree)** 의 명세화(specification).
- Filecoin 및 그 구성 요소들에 대한 **세부 성능 추정치(performance estimates)** 와 **벤치마크(benchmarks)**.
- **완전 구현 가능한 Filecoin 프로토콜 명세서(protocol specification)** 작성.
- **후원 기반 리트리벌(sponsored-retrieval) 티켓 모델** — 클라이언트 C_1이 조각 단위로 발행 가능한 토큰을 통해 다른 클라이언트 C_2의 다운로드를 후원(sponsor)할 수 있는 구조.
- **계층형 합의 프로토콜(Hierarchical Consensus)** — Filecoin 서브넷(subnet)이 일시적 혹은 영구적 네트워크 분할(partition) 중에도 독립적으로 트랜잭션을 처리할 수 있는 구조.

- **SNARK/STARK 기반 점진적 블록체인 스냅샷(incremental blockchain snapshotting)** 기술.
- **Filecoin-in-Ethereum 인터페이스 컨트랙트 및 프로토콜** 설계.
- **Braid를 이용한 블록체인 아카이브 및 체인 간 타임스탬핑(inter-blockchain stamping)**.
- 충돌 해결(conflict resolution)을 위한 경우에만 **Proof-of-Spacetime** 을 블록체인에 게시(post).
- Filecoin DSN 및 새로운 형태의 **저장 증명(Proofs-of-Storage)** 에 대한 실현 가능성(formal realization)의 형식적 증명(formal proof).

8.2 개방형 질문(Open Questions)

아래의 개방형 질문들은 네트워크 전체의 효율성과 안정성을 상당히 향상시킬 잠재력을 가지고 있지만, Filecoin의 초기 출시 이전에 반드시 해결되어야 하는 것은 아니다.

- **Proof-of-Replication의 Seal 함수 개선:**
  디코드(decode) 시 O(n) 복잡도를 갖고(O(mn) 아님), SNARK/STARK 없이도 공개 검증 가능한 새로운 원시 연산(primitive).
- **Proof-of-Replication의 Prove 함수 개선:**
  SNARK/STARK에 의존하지 않고 투명하며 공개 검증 가능한 새로운 증명 생성 원시 연산.
- **Proof-of-Retrievability 혹은 기타 Proof-of-Storage의 투명하고 공개 검증 가능한 형태.**
- 리트리벌 마켓의 새로운 전략: 예를 들어 확률적 결제(probabilistic payments) 또는 영지식(zero-knowledge) 기반 접근 방식.
- **Expected Consensus** 내에서, 매 epoch마다 정확히 한 명의 리더만 선출되는 **비밀 리더 선출(secret leader election)** 메커니즘.
- **SNARK 신뢰 설정(trusted setup) 확장 기법:**
  공개 매개변수를 점진적으로 확장할 수 있고, 연속적인 다자간 계산(MPC) 단계를 통해 이전 세대의 출력을 다음 세대 입력으로 사용할 수 있으며, 실패 확률을 줄이는 방식.

8.3 증명 및 형식 검증(Proofs and Formal Verification)

Filecoin 네트워크의 핵심 속성들을 **형식적으로 증명(formal proof)** 하고
형식 검증(formal verification) 된 프로토콜 명세를 개발하는 것은 매우 중요하다.
향후 수개월에서 수년에 걸쳐 Filecoin의 다양한 속성(확장성, 오프라인 기능 등)에 대한 증명을 단계적으로 완성할 계획이다.

이미 일부 증명은 진행 중이며, 여러 추가 증명들도 연구 계획에 포함되어 있다.
이러한 작업은 장기적이며 복잡하지만, Filecoin 프로토콜의 신뢰성과 수학적 안전성을 확보하기 위한 핵심적인 과제이다.

- **Expected Consensus 및 그 변형들에 대한 정확성 증명(Proofs of correctness).**
- **Power Fault Tolerance의 비동기 1/2 불가능성 결과(asynchronous 1/2 impossibility)** 를 회피하는 정확성 증명.

- **Filecoin DSN의 보편적 조합성(Univ. Composability, UC) 프레임워크 내 정식화:**
  Get, *Put*, *Manage* 를 이상적 기능(ideal functionality)로 정의하고,
  실제 프로토콜이 이들을 충실히 실현함을 증명.
- **자동 자가 복구(self-healing) 보장에 대한 형식 모델 및 증명.**
- **프로토콜 명세의 형식 검증(Formal verification of specifications)** — 예: TLA+, Verdi 등을 이용.
- **프로토콜 구현의 형식 검증(Formal verification of implementations)** — 예: Verdi 기반 검증.
- **Filecoin 인센티브 구조에 대한 게임이론적 분석(Game-theoretic analysis).**