

라이터(Lighter)

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.

라이터 프로토콜(Lighter Protocol): 투명하고 검증 가능한 연산을 통한 오더북 매칭 및 청산

Elliot Technologies, Inc. (Lighter 상호로 영업)

2025년 10월

초록(Abstract)

디지털 자산 거래에 대한 전 세계적 관심이 지속적으로 증가함에 따라, 기존 블록체인은 증가하는 수요를 충족할 만큼 효율적으로 확장하는 데 어려움을 겪어 왔다. 이러한 한계로 인해 많은 사용자가 불투명한 수탁형 거래 플랫폼으로 이동하게 되었다. 라이터 프로토콜(Lighter Protocol)은 확장 가능하고, 안전하며, 투명하고, 비수탁적이며, 공정한 거래 인프라를 제공함으로써 이러한 문제를 해결한다. 라이터는 고급 암호기법, 새로운 데이터 구조, 그리고 블록체인 기술을 결합하여 악의적이거나 중앙화된 운영자가 초래할 수 있는 위험을 제거하는 동시에 효율적인 시장 가격 발견을 가능하게 한다. 나아가 본 접근법은 차세대 고성능·고보안 디지털 거래 플랫폼을 구축하기 위한 기초 프레임워크를 확립한다.

1. 서론(Introduction)

디지털 거래 환경은 여전히 보안, 투명성, 그리고 신뢰 최소화 측면에서 지속적인 과제에 직면해 있다. 전통적인 거래 시스템은 중앙화 리스크, 불투명한 운영, 신뢰된 운영자에 대한 의존성으로 인해 제약을 받고 있으며, 이는 거래자를 잠재적인 사기, 불공정한 관행, 그리고 시스템적 취약성에 노출시킨다. 이러한 문제는 시장의 효율성과 신뢰성 또한 저해한다.

핵심 과제는 신뢰에 의존하지 않으면서도 보안성, 투명성, 효율성을 보장하는 거래 인프라를 구축하는 것이다. 이를 달성하기 위해서는 운영자의 오작동을 방지하고, 모든 행위가 공개적으로 정의된 규칙 집합을 따르도록 보장하는 공개 검증 가능한 연산 엔진이 필요하다.

검증 가능한 연산(verifiable computation)을 구현하기 위한 다양한 접근 방식이 제안되어 왔으며, 각각 고유한 트레이드오프를 가진다.

- 탈중앙화 합의를 사용하는 블록체인(Blockchains with Decentralized Consensus)** — 다수 노드에 의한 중복 실행을 통해 검증 가능성을 달성한다. 충분히 탈중앙화된 경우, 검열과 변조에 대한 강력한 보호를 제공하는 매우 높은 보안성을 갖는다. 그러나 합의 오버헤드와 네트워크 동기화로 인해 지연 시간과 처리량에 한계가 존재한다. 또한 중복 실행을 위해 데이터가 공개되어야 하므로, 데이터 프라이버시에 대한 유연성도 제한된다. 이더리움(Ethereum) 블록체인은 이러한 모델을 대표하는 사례로, 탈중앙화를 통해 강력한 보안을 제공하는 대신 성능상의 트레이드오프를 수반한다.
- 신뢰 실행 환경(Trusted Execution Environments, TEEs)** — 하드웨어 기반 연산 증명을 제공하지만, 벤더 신뢰에 의존하며 사이드 채널 공격에 취약하다. 사용자가 실행 결과를 독립적으로 검증할 수 없기 때문에 일정 수준의 신뢰를 다시 도입하게 된다.

- **간결 증명(Succinct Proofs, SNARKs)** — 연산의 정확성을 간결하고 비대화형 방식으로 검증할 수 있는 암호학적 증명이다. 과거에는 비용이 높고 구현이 어려웠으나, 최신 SNARK는 합의 기반 시스템의 한계를 넘어서는 강력한 보안성, 프라이버시 유연성, 그리고 확장성을 제공한다.

탈중앙화 블록체인은 신뢰 없는 금융을 위한 기반을 확립하며, 거래에서 요구되는 보안성과 투명성을 충족시켰다. 그러나 높은 거래 수수료와 긴 블록 생성 시간을 가진 이더리움과 같은 블록체인은 저지연과 비용 효율성이 핵심인 고빈도 거래(high-frequency trading)에 적합하도록 설계되지 않았다. 이러한 한계로 인해 탈중앙화 오더북은 실용성이 낮아졌고, 혁신은 자동화된 시장조성자(Automated Market Makers, AMMs)로 이동했다. AMM은 거래를 단순화하고 접근성을 확대하지만, 구조적인 비효율성을 내포한다. 직접적인 주문 매칭 대신 알고리즘 기반 가격 산정을 사용하며, 가격 발견을 위해 차익거래나 외부 오라클에 의존하고, 효율적인 작동을 위해 일반적으로 대규모 유동성 풀을 요구한다. 이러한 제약은 특히 유동성이 낮거나 변동성이 높은 시장에서 더 큰 가격 슬리피지를 초래하는 경우가 많다.

이러한 한계를 극복하기 위해, 일부 프로젝트는 오프체인 오더북을 사용하고 블록체인에서 결제하는 방식을 실험해 왔다. 이 구조에서는 연산 부담이 큰 작업을 오프체인으로 이전하면서도, 온체인 검증을 유지한다. 일반적으로 이더리움은 결제 레이어로 사용되며, 오프체인에서 매칭된 거래는 이후 온체인에서 검증되고 확정된다. 중앙화된 운영자가 오프체인 오더북을 관리하며, 마켓 메이커는 서명된 주문 메시지를 운영자에게 전송하고, 운영자는 이를 오프체인 매칭 엔진에서 집계한다. 주문이 교차(cross)된다고 판단되면, 운영자는 거래 데이터를 블록체인에 제출하여 결제를 요청한다. 스마트 컨트랙트는 서명 검증과 주문 교차 여부 확인을 통해 주문의 유효성을 검증한 후 거래를 실행한다. 이 아키텍처는 주문 저장과 매칭을 오프체인에 두어 주문당 가스 비용을 회피함으로써 비용을 절감한다. 그러나 새로운 문제도 발생한다. 오더북 상태가 오프체인에만 존재하기 때문에, 스마트 컨트랙트는 가격-시간 우선순위(price-time priority)나 주문 공정성을 독립적으로 검증할 수 없다. 참여자는 운영자가 주문을 검열, 재정렬, 지연시키지 않을 것이라는 점을 신뢰해야 하며, 이는 탈중앙화를 목표로 한 시스템에 다시 중앙화 리스크를 도입한다.

범용 레이어 2 기술의 발전으로 이더리움과 같은 블록체인에서의 거래 비용은 크게 감소했으며, 스테이블코인 페어와 같은 저변동성 시장에서는 완전한 온체인 오더북이 가능해졌다. 그러나 변동성이 더 높은 시장에서는 현재의 레이어 2 솔루션이 여전히 충분하지 않다. 마켓 메이커는 가격 변동에 대응하기 위해 주문을 빈번하게 업데이트해야 하며, 현존하는 범용 레이어 2의 처리량과 비용 제약은 타이트한 스프레드를 유지하는 것을 경제적으로 어렵게 만든다. 근본적인 문제는 범용 설계에 있다. 범용 가상머신은 유연하지만 거래에 최적화되어 있지 않다. 금융 워크로드를 위해 특별히 설계된 전용 가상머신은 현대 오더북 시장이 요구하는 속도, 결정성, 비용 효율성을 달성할 수 있다.

한편, Hyperliquid 및 dYdX v4와 같은 금융 특화 레이어 1 블록체인이 등장하여 금융 사용 사례에 대한 검증 가능한 주문 매칭 성능을 개선하고자 했다. 그러나 이러한 아키텍처는 검증 가능성을 합의 메커니즘에 의존한다. 탈중앙화 수준이 제한될 경우 합의는 상대적으로 쉽게 훼손될 수 있으며, 이는 전체 보안을 약화시킨다. 탈중앙화를 강화하면 보안성은 높아지지만, 블록 시간이 길어지고 지연 시간이 증가하여 전통적인 거래소 수준의 성능을 달성하기 어렵다. 또한 검증자나 시퀀서가 트랜잭션을 재정렬하거나 삽입하여 수익을 추출하는 최대 추출 가능 가치(Maximum Extractable Value, MEV)에 대한 노출도 커진다. 애플리케이션 전용 블록체인으로 독립 운영될 경우 유동성이 분절되고, 이더리움과 같은 기존 생태계가 가진 자본과 네트워크 효과로부터 고립된다. 더 나아가 안전한 자산 출금을 위해서는 견고한 비수탁 환경이 필요하지만, 도착 체인의 보안성 또한 동일하게 중요하다. 브리지를 통해 연결된 고립된 블록체인은 침해 시 신뢰할 수 있는 비수탁적 대안을 제공하지 못한다. 확장성 제약에도 불구하고, 이더리움은 가장 충분히 검증된 검증 가능 실행 플랫폼으로 남아 있으며, 사용자 출구를 위한 중립적이고 안전한 결제 레이어를 제공한다.

신뢰 없는 금융 시스템 구축이라는 과제를 해결하기 위해, 라이터는 가격-시간 우선순위 주문 매칭, 리스크 관리, 계정 업데이트를 포함한 전체 금융 연산에 대해 실행 증명을 생성하는 애플리케이션 특화 SNARK 기반 프로버를 도입한다. 금융을 위해 특별히 설계된 이더리움 상의 애플리케이션 특화 레이어 2로서, 라이터는 시스템 상태를 검증 가능하게 추적·갱신하며, 이더리움의 보안성과 유동성에 라이터 프로토콜의 수평·수직 확장성을 결합한다. 이 아키텍처는 고빈도 금융에 준하는 비용 및 지연 특성을 가지는 검증 가능한 온체인 거래를 제공하는 동시에, 이더리움 수준의 보안성과 모든 참여자에 대한 공정한 실행을 보장한다.

이러한 아키텍처는 우선순위 변경과 같은 대부분의 악의적 행위를 완화하며, MEV 추출에 대한 경제적 유인을 실질적으로 감소시킨다. 라이터는 극히 낮은 지연 시간으로 동작하기 때문에, 트랜잭션 재정렬로부터 발생할 수 있는 잠재적 이점은 밀리초 단위의 차이로 제한되며, 실제로는 악용이 어렵고 사용자에게 미치는 영향도 무시할 수 있는 수준에 그친다. 향후 개선을 통해 이러한 잔여 효과마저 제거하는 것을 목표로 하며, 이는 8장에서 논의한다.

이러한 설계를 통해 라이터는 온체인 거래를 위한 검증 가능하고 효율적인 기반을 제공하며, 고급 금융 애플리케이션에 적합한 공정한 실행과 높은 성능을 보장한다.

2. 라이터 프로토콜(Lighter Protocol)

본 섹션에서는 라이터 생태계의 첫 번째 제품인 무기한 선물(perpetual futures) 거래를 중심으로 라이터 프로토콜(Lighter Protocol)을 설명한다. 본 프로토콜은 시퀀서(Sequencer), 프로버(Prover), 인덱서(Indexer), API 서버(API Servers), 그리고 스마트 컨트랙트(Smart Contracts)라는 주요 구성 요소로 이루어져 있다.

시퀀서는 저지연 실행 엔진으로서, 댄풀(mempool)로부터 사용자 트랜잭션을 수신하여 이를 순차적으로 실행하고 블록 형태로 구성한다. 이후 트랜잭션 영수증과 상태 변경 사항을 다수의 데이터 피드를 통해 인덱서와 프로버 서비스로 전파한다.

인덱서는 시퀀서의 데이터 피드를 수신하여 이를 사용자 친화적인 구조로 재구성한다. API 서버는 이렇게 인덱싱된 데이터를 소비하여 사용자에게 실시간 정보를 제공하며, 프로토콜과 상호작용하기 위한 주요 인터페이스 역할을 수행한다. 사용자는 API 서버를 통해 트랜잭션 요청을 제출하거나 프로토콜의 현재 상태를 조회할 수 있다.

프로버는 시퀀서의 실행 피드를 입력으로 받아, 가격-시간 우선순위(price-time-priority) 주문 매칭과 청산(liquidations)을 포함한 모든 거래소 연산에 대해 간결한 실행 증명을 생성한다. 프로버는 이전 상태와 사용자 트랜잭션 배치를 입력값으로 사용하여, 결과적으로 발생한 상태 전이가 정확함을 검증하는 증명을 생성한다.

마지막으로, 이더리움(Ethereum) 상의 스마트 컨트랙트는 예치된 자산과 라이터의 정규 상태 루트(canonical Lighter state root)를 보관한다. 이 상태 루트는 사용자 자산과 포지션, 퍼블릭 풀(public pools), 오더북(order books), 그리고 기타 핵심 시스템 구성 요소를 포괄한다. 라이터 프로토콜은 처리된 트랜잭션 배치를 나타내는 상태 업데이트 제안을 주기적으로 이더리움에 게시하며, 각 제안에는 계정별 업데이트와 같은 상세한 상태 전이 데이터를 포함한 데이터 블롭(blob)이 함께 첨부된다. 이를 통해 사용자는 자신의 상태를 독립적으로 재구성하고 검증할 수 있다. 프로버가 상태 업데이트 제안에 대한 증명을 생성하고, 해당 증명이 이더리움에서 검증되면, 스마트 컨트랙트는 정규 거래소 상태를 갱신한다. 이러한 증명은 상태 업데이트 제안의 정확성뿐만 아니라, 함께 게시된 온체인 데이터의 유효성과 완전성까지 입증하며, 이더리움의 데이터 블롭에 포함된 정보만으로도 검열 저항적인 온체인 데이터에 기반해 모든 사용자 상태를 완전히 재구성할 수 있음을 보여준다.

2.1. 배치(Batches)

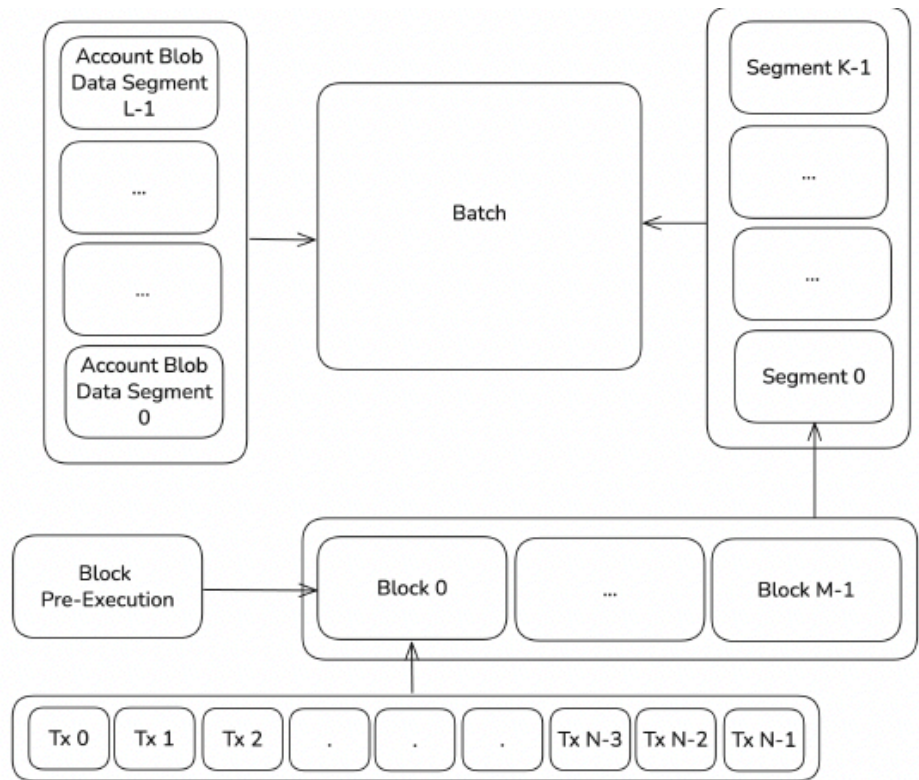


그림 1: 라이터 프로토콜 배치

트랜잭션(Transactions): 트랜잭션은 사용자 또는 시스템에 의해 발생하는 거래소의 단위 연산이다. 예로는 주문 생성, 자산 인출, 청산 개시 등이 있다.

블록 사전 실행(Block Pre-Execution): 오라클 입력 적용이나 무기한 선물 시장 전반에 대한 프리미엄 계산과 같이, 거래소 전역 연산을 수행하는 라이터 시스템 내의 단위 연산이다.

블록(Block): 하나의 블록은 블록 사전 실행과 연속된 트랜잭션 시퀀스로 구성된다.

세그먼트(Segment): 세그먼트는 여러 개의 연속된 블록으로 구성된다.

계정 블롭 데이터 세그먼트(Account Blob Data Segment): 연속된 계정 전반에 걸쳐 발생한 비어 있지 않은 상태 변경을 직렬화하고 압축한 블롭 데이터이다.

배치(Batch): 배치는 연속된 세그먼트들과 함께, 실행 과정에서 생성된 모든 퍼블릭 계정 상태 변경 및 시장 관련 퍼블릭 정보를 나타내는 블롭 데이터를 포함한다. 퍼블릭 시장 데이터는 세그먼트 단위로 집계되며, 퍼블릭 계정 데이터는 계정 블롭 데이터 세그먼트로부터 집계된다.

라이터 프로토콜의 배치는 분기 없는 선형 시퀀스로 배열된다. 이 시퀀스의 시작을 나타내는 제네시스 배치(genesis batch)는 높이(height) 0으로 지정되며, 이후 체인에 추가되는 각 배치는 이전 배치보다 1씩 증가한 배치 높이를 갖는다.

각 배치는 이더리움 데이터 블롭을 통해 상태 업데이트 제안을 게시할 때 함께 온체인에 기록되는 퍼블릭 데이터를 포함한다. 이 데이터는 각 사용자가 예치 자산, 미결제 포지션, 라이터의 퍼블릭 풀에 대한 지분을 포함한 자신의 개별 계정 상태를 재구성하기에 충분하다. 이러한 재구성은 사용자가 자신의 자산을 독립적으로 검증할 수 있도록 하며, 이스케이프 해치(Escape Hatch) 메커니즘을 제공하여, 중개자 없이 이더리움 상에서 자신의 총 계정 가치를 증명하고 안전하게 시스템을 이탈할 수 있도록 한다.

라이터는 초당 수만 건의 사용자 연산을 실행하며, 이는 이더리움에 전부 게시하기에는 과도한 데이터 양을 생성한다. 이러한 확장성 병목을 해결하기 위해, 라이터 프로토콜은 하이브리드 데이터 가용성(hybrid data availability) 모델을 채택하여, 안전한 이스케이프 해치를 위해 필요한 최소한의 데이터만 게시하고 중복되는 세부 정보는 생략한다. 고빈도 거래는 대부분 실제 체결로 이어지지 않는 주문 연산이 지배적이기 때문에, 이러한 설계는 검증 가능한 주문 매칭을 유지하면서도 데이터 볼륨을 크게 줄여준다.

라이터는 배치 전체에 걸쳐 사용자의 총 계정 가치에 영향을 미치는 모든 계정 상태 변경을 집계하고, 이러한 집계된 업데이트를 표현하는 머클 트리(Merkle Tree)를 구성한다. 이후 이 트리에서 비어 있지 않은 리프 노드만을 직렬화하여 압축된 바이트 데이터로 생성하고, 이를 모든 무기한 시장에 대한 펀딩 비율(funding rates)과 마크 가격(mark prices)과 같은 관련 시장 데이터와 결합한다. 이렇게 결합된 데이터는 해당 배치에 첨부되어, 블롭을 포함한 트랜잭션을 통해 이더리움에 퍼블릭 레코드의 일부로 게시된다.

2.2. 트랜잭션(Transactions)

라이터 프로토콜(Lighter Protocol)은 사용자 트랜잭션을 큐에 적재하기 위해 두 가지 상이한 메커니즘을 구현한다.

2.2.1. 롤업 트랜잭션(Rollup Transactions)

롤업 트랜잭션은 사용자가 자산을 예치한 이후 수행되는 모든 거래소 연산을 포함한다. 이러한 트랜잭션은 API 서버를 통해 시퀀서에 의해 실행되도록 큐에 적재된다. 롤업 트랜잭션은 시퀀서가 직접 소비하며, 어떠한 트랜잭션 수수료도 발생하지 않는다.

2.2.2. 우선 트랜잭션(Priority Transactions)

우선 트랜잭션은 주로 검열 저항성을 요구하는 출금 관련 연산으로 구성되며, 예를 들어 포지션 종료, 자산의 이더리움(Ethereum)으로의 인출, 퍼블릭 풀로부터의 지분 상환 등이 이에 해당한다. 이러한 트랜잭션은 이더리움 상의 스마트 컨트랙트를 통해 직접 큐에 적재되며, 이를 통해 검열 저항성과 사전에 정의된 시간 내 실행이 보장된다. 만약 이더리움에 제출된 우선 트랜잭션을 시퀀서가 실행하지 못할 경우, 스마트 컨트랙트는 거래소 상태를 동결하고 사용자가 이스케이프 해치(Escape Hatch) 메커니즘을 통해 안전하게 시스템을 이탈할 수 있도록 허용한다. 우선 트랜잭션 프레임워크는 확장 가능하도록 설계되어 있어, 향후 추가적인 우선 연산을 통합할 수 있으며, 라이터와 보다 광범위한 이더리움 생태계 간의 조합성(composability) 인터페이스 역할을 수행한다.

2.3. 시퀀서(Sequencer)

시퀀서는 우선 트랜잭션과 롤업 트랜잭션을 모두 선입선출(first-come, first-served) 큐에 저장하고 모니터링한다. 시퀀서는 이러한 트랜잭션을 처리하여 블록으로 구성하고, 해당 블록들을 배치로 집계한 뒤, 이를 상태 업데이트 제안의 형태로 스마트 컨트랙트에 제출한다. 또한 시퀀서는 무기한 선물 시장 연산에 사용되는 활성 무기한 시장의 지수 가격(index price)을 얻기 위해 탈중앙화 오라클 네트워크와 연동된다.

동시에 시퀀서는 배치와 관련된 데이터를 병렬화된 위트니스(Witness) 서비스¹로 전달하며, 이 서비스는 프로버(Prover)가 실행 증명을 생성하는 데 필요한 입력값을 생성한다. 더불어 시퀀서는 데이터를 인덱서(Indexer) 서비스에도 위임하는데, 인덱서는 수신된 데이터를 처리하고 재구성하여 API 서버 집합이 운영하는 API 및 WebSocket 인터페이스에 제공한다. 이러한 인터페이스는 시장 상태와 계정 상태에 대한 업데이트를 포함한 실시간 롤업 데이터를 사용자에게 빠르고 안정적으로 제공한다. 시퀀서가 게시하는 저지연 데이터 피드에 대해 암호학적 보장을 제공하기 위해, 사전 커밋(pre-commitment) 스킴이 도입될 예정이다. 이 메커니즘은 브로드캐스트된 데이터가 이후 온체인에 커밋되는 상태와 정확히 일치함을 보장하여, 트랜잭션에 대해 신속한 암호학적 최종성(finality)을 제공한다. 또한 시퀀서는 배치를 빈번하게 이더리움에 커밋함으로써, 사용자가 확정된 온체인 상태를 최소한의 지연으로 관찰할 수 있도록 한다.

라이터 프로토콜의 설계는 본질적으로 시퀀서의 권한과 영향력을 제한한다. 시퀀서의 핵심 기능은 트랜잭션의 순서를 결정하는 것이다. 트랜잭션 실행과 주문 매칭은 프로버에 의해 검증되므로, 모든 유효한 상태 전이는 트랜잭션 순서와 오라클 데이터로부터 결정론적으로 도출될 수 있다. 이러한 책임 분리는, 매칭이 시퀀서 계층에서 직접 수행되고 증명이 단순히 두 주문이 교차했음을 확인하는 기존 zk-롤업 오더북 아키텍처에 비해, 시퀀서 서비스의 일부를 더 쉽게 탈중앙화할 수 있도록 한다. 기존 설계에서는 시퀀서가 매칭을 직접 통제하기 때문에 상당한 검열 리스크가 발생한다. 시퀀서는 특정 메이커 주문을 선택적으로 우대하거나, 다른 주문의 우선순위를 낮추거나, 매칭 결과를 조작할 수 있다. 이는 테이커 주문이 덜 유리한 메이커 주문과 체결되도록 유도할 수 있으며, 가격-시간 우선순위의 무결성을 훼손한다. 이러한 위험은 특히 낮은 변동성 또는 좁은 스프레드 시장에서 동일한 가격 수준에 다수의 메이커 주문이 존재할 때 더욱 두드러진다. 증명 계층에서 시간 기반 우선순위가 강제되지 않는다면, 시퀀서는 거래를 임의로 재정렬하거나 선별적으로 우대할 수 있으며, 이는 공정성과 시장 중립성을 침해한다.

2.4. 프로버(Prover)

프로버는 배치 실행과 그에 대응하는 상태 전이에 대해 SNARK 증명²을 생성한다. 프로버는 트랜잭션 실행으로 인해 발생한 상태 변경을 검증하는 커스텀 산술 회로(custom arithmetic circuits)와, 모든 배치 입력을 단일한 간결 증명으로 결합하는 다층 집계 시스템을 사용한다. 본 문서에서 회로(circuits)라는 용어는 프로버 서비스를 위해 개발된 이러한 커스텀 산술 회로를 구체적으로 지칭한다.

이 회로들은 공개된 수학적 명제들의 집합을 정의한다. 이러한 명제를 만족하는 입력값의 집합은 트랜잭션 배치에 대해 라이터 프로토콜이 정확히 실행되었음을 검증한다. 이 입력값들 중 정확히 하나는 공개 입력(public input), 즉 커밋먼트(commitment)로 지정된다. 검증자가 프로버가 생성한 증명을 검증할 때, 증명 생성 시 사용된 것과 동일한 커밋먼트를 함께 제공한다. 이 커밋먼트에는 새로운 상태 루트와 프로토콜 운영에 필요한 기타 보조 데이터를 포함하여, 상태 전이를 나타내는 핵심 데이터가 담겨 있으며, 이를 통해 검증자는 증명의 정확성과 온체인 상태와의 대응 관계를 확인할 수 있다.

시퀀서가 스마트 컨트랙트에 상태 업데이트 제안을 제출할 때, 해당 제안에는 배치 내 트랜잭션 실행 결과로 생성된 새로운 상태 루트를 포함한 새로운 프로토콜 상태 정보가 담긴다. 또한 스마트 컨트랙트에 큐잉된 우선 요청이 처리되었음을 표시하기 위한 데이터와, 출금 연산과 같이 라이터 프로토콜에서 이더리움으로 전달되는 메시지도 포함된다. 여기에 더해, 배치에 대한 계정 단위 상태 업데이트를 나타내는 블록 데이터와, 펀딩 비율 및 마크 가격과 같은 글로벌 시장 정보도 제안에 포함된다. 스마트 컨트랙트는 제안을 수신하면, 이에 대응하는 커밋먼트를 계산하여 저장한다. 이후 프로버 서비스는 커밋된 배치에 대한 증명을 생성한다. 각 증명은 해당 배치의 커밋먼트(공개 입력)가 회로에서 정의된 수학적 명제를 만족하는 입력 중 하나에 해당함을 검증한다. 그 결과, 증명은 배치 내 모든 트랜잭션이 정확히 실행되었고 롤업 상태가 올바르게 갱신되었음을 확인한다.

라이터 프로버는 오더북 트리(Order Book Tree)와 같이, 본 프로토콜을 위해 특별히 설계된 새로운 데이터 구조 집합을 활용한다. 가격-시간 우선순위 주문 매칭과 청산에 대한 실행 증명을 생성하는 방법은 다양하게 존재할 수 있으나, 라이터는 이러한 특수 데이터 구조를 통해 이를 효율적으로 달성한다. 프로버는 오더북 상태 저장에 머클 트리(Merkle Tree)를 사용하는 데서 발생하는 비효율성을 해결하기 위해, 매칭 연산 시 단 하나의 리프 접근만으로도 가격-시간 우선순위를 엄격하게 강제할 수 있는 하이브리드 데이터 구조를 도입한다. 라이터 프로토콜의 상태 트리(State Tree)는 3장에서 상세히 설명한다.

2.5. 스마트 컨트랙트(Smart Contracts)

스마트 컨트랙트의 주요 기능은 사용자 자산과 롤업의 현재 상태 루트(state root)를 유지하는 것이다. 시퀀서는 트랜잭션 배치를 스마트 컨트랙트에 커밋하며, 컨트랙트는 이러한 커밋을 저장하고 이후 검증을 위해 큐에 적재한다. 프로버(Prover) 서비스가 증명을 생성하면, 해당 증명은 스마트 컨트랙트로 제출되어 검증된다. 컨트랙트에 저장된 대응 커밋먼트(공개 입력, public input)와 증명을 사용한 검증이 성공하면, 현재 상태 루트가 갱신되고, 출금 시 자산 이전과 같이 라이터 프로토콜에서 전달된 온체인 메시지가 실행된다.

또한 스마트 컨트랙트는 검열 저항적 우선 트랜잭션을 위한 큐잉 메커니즘을 구현하며, 시퀀서가 사전에 정의된 시간 내에 이러한 트랜잭션을 이후 블록에 반드시 포함하도록 강제한다. 시퀀서가 이 의무를 이행하지 못해 —기한 이전에 커밋된 블록에서 큐에 적재된 우선 트랜잭션 중 하나라도 누락될 경우— 컨트랙트는 이스케이프 해치(Escape Hatch, 6장)로 알려진 비상 출구 프로토콜을 활성화한다. 이스케이프 해치가 활성화되면 롤업 상태는 동결되어, 추가적인 블록 커밋이나 검증이 중단된다. 이러한 동결 상태에서 사용자는 스마트 컨트랙트를 통해 자신의 자산을 직접 인출할 수 있다.

3. 상태 트리(State Tree)

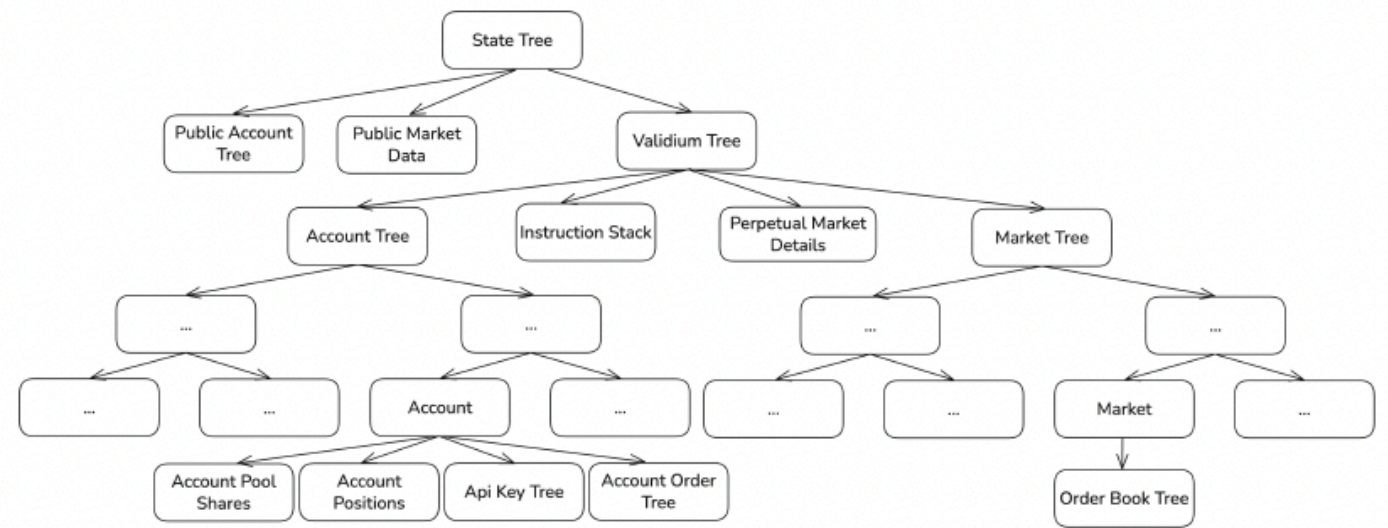


그림 2: 라이터 프로토콜 상태 트리

라이터 프로토콜의 전체 상태는 상태 트리(State Tree)라 불리는 데이터 구조에 캡슐화되어 있다. 상태 트리는 머클화(merkelized)³ 된 데이터 구조로, 각 노드는 해당 서브트리에 포함된 모든 데이터를 나타내는 해시를 포함한다. 이에 따라 상태 트리의 루트 해시는 라이터 프로토콜 전체 상태에 대한 암호학적 커밋먼트로 기능한다.

상태 트리 내의 데이터 구조는 효율적이고 검증 가능한 오더북 매칭, 청산 실행, 그리고 계정 관리 연산을 가능하게 하도록 설계되어 있다. 상태 트리의 핵심 구성 요소인 오더북 트리(Order Book Tree)는 기존 머클 트리 모델에 여러 혁신을 도입한다. 이는 오더북의 전체 상태를 저장하고 오더북 연산을 효율적으로 수행할 수 있도록 하는 통합 데이터 구조로 작동한다. 내부 노드는 각 노드 하위에 포함된 모든 리프를 대표하는 집계 주문 데이터를 포함한다. 또한 이 트리는 접두 트리(prefix tree)와 유사한 인덱싱 스킴을 사용하여, 주문의 우선순위를 리프 인덱스에 직접 인코딩한다.

3.1. 계정 트리(Account Tree)

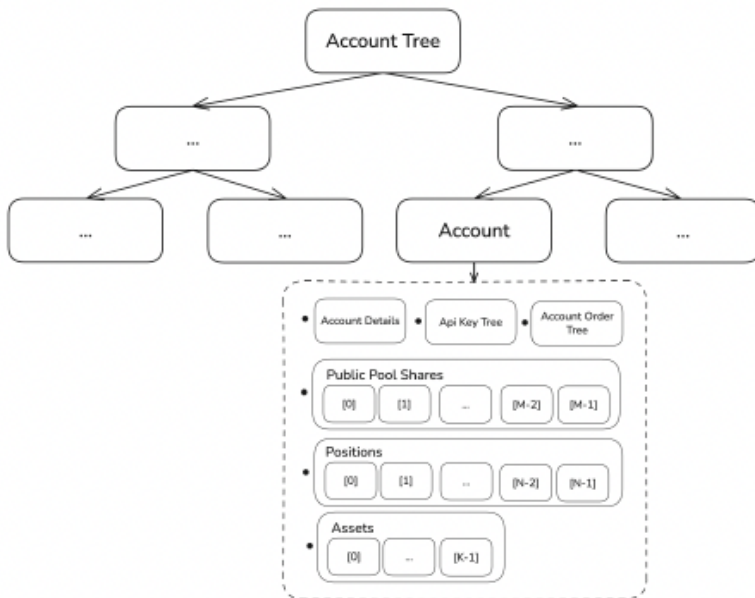


그림 3: 계정 트리

상태 트리 내의 계정 트리(Account Tree)는 자산 보유 현황, 포지션 정보, 퍼블릭 풀 지분 정보 등을 포함한 계정 관련 데이터를 저장하도록 설계되어 있다. 계정 트리의 리프는 다음과 같은 데이터를 포함하되, 이에 한정되지는 않는다.

계정 세부 정보(Account Details): 계정과 연계된 L1 주소, 서브 계정 및 퍼블릭 풀을 위한 메인 계정 인덱스, 계정 유형, 그리고 해당 유형에 따라 필요한 추가 데이터를 저장한다. 또한 주문 수, DMS(Dead man's switch) 트리거 타임스탬프와 같은 주문 관련 정보도 포함한다.

계정 주문 트리(Account Order Tree): 계정과 연관된 모든 활성 및 대기 주문을 저장하며, 여기에는 대기 중인 조건부 주문, 오더 북에 제출된 주문, TWAP 주문이 포함된다. 라이터는 사용자가 주문에 클라이언트 주문 인덱스를 할당할 수 있도록 하여, 주문 관리를 용이하게 한다. 계정 주문 트리는 거래소가 부여한 주문 인덱스와 사용자가 지정한 클라이언트 주문 인덱스 간의 매핑도 유지한다.

API 키 트리(API Key Tree): 사용자 생성 API 키에 대한 논스(nonce) 및 공개키 정보를 저장한다. 각 계정은 요청 서명을 위해 여러 개의 API 키를 가질 수 있다.

퍼블릭 풀 지분(Public Pool Shares): 퍼블릭 풀 내에서 사용자가 보유한 지분과 진입 시점의 가격 정보를 저장한다. 퍼블릭 풀은 기능이 제한된 계정으로, 다른 사용자가 지분을 매수할 수 있다.

포지션(Positions): 사용자가 보유한 무기한 선물 포지션에 대한 정보를 저장한다. 격리 포지션(isolated position)의 경우, 해당 포지션에 할당된 자산도 함께 저장한다.

자산(Assets): 사용자의 계정에 보유된 USDC 수량을 저장한다. 이 목록은 향후 추가적인 자산 유형의 예치를 지원하도록 확장될 예정이다.

라이터는 서로 다른 목적을 수행하는 네 가지 계정 유형을 지원한다.

1. **메인 계정(Main Account):** 기능에 제한이 없는 라이터 프로토콜의 표준 계정으로, 각 메인 계정은 하나의 L1 주소와 고유하게 연결된다.
2. **서브 계정(Sub-Account):** 동일한 L1 주소를 공유하는 메인 계정으로부터 파생된 격리 계정이다. 서브 계정은 하나의 L1 주소 하에서 여러 개의 격리된 환경을 운영할 수 있도록 한다.
3. **퍼블릭 풀(Public Pool):** 다른 사용자가 지분을 매수할 수 있는 서브 계정의 한 유형이다. 퍼블릭 풀은 기능이 제한되어 있어, 보유한 자산을 이전하거나 인출할 수는 없지만 시장에서 거래는 가능하다. 퍼블릭 풀을 통해 사용자는 다른 계정의 거래 전략에 참여할 수 있다.
4. **보험 기금(Insurance Fund):** 시장 구성에 참여하는 동시에, 오더북 유동성이 부족하여 부실 계정을 청산할 수 없는 경우에 대비해 백스톱 유동성을 제공하는 특수한 퍼블릭 풀 유형이다. 보험 기금은 보험 기금 운영자(Insurance Fund Operator)로 알려진 지정된 프로토콜 계정에 의해 관리된다.

3.2. 퍼블릭 계정 트리(Public Account Tree) 및 계정 델타 트리(Account Delta Tree)

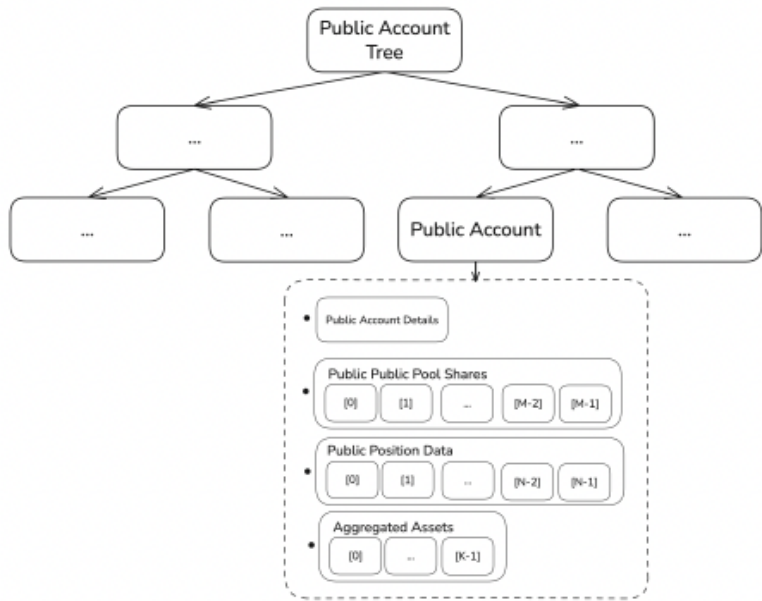


그림 4: 퍼블릭 계정 트리

퍼블릭 계정 트리(Public Account Tree)는 계정 트리(Account Tree)와 유사한 구조를 가지며, 각 계정에 대응되는 퍼블릭 정보를 저장하는 역할을 수행한다. 이 트리는 이더리움(Ethereum)에 게시된 블록 데이터만을 사용하여 완전히 재구성할 수 있다. 트리의 각 리프에 포함된 필드는, 계정이 자신의 총 자산 가치를 검증하는 데 필요한 최소한의 데이터 집합을 나타낸다.

이를 위해 격리 포지션(isolated positions)과 크로스 포지션(cross positions)에 포함된 자산과 해당 포지션의 진입 가격(entry quotes)은 단일화된 자산 가치로 집계된다. 그 결과, 퍼블릭 포지션 리스트에는 각 포지션의 규모(size)와 관련된 펀딩 정보만이 포함된다. 유사하게, 퍼블릭 풀 포지션의 경우 각 리프에는 지분 수량만 저장된다.

각 리프에는 또한 이스케이프 해치(Escape Hatch) 모드에서 자산의 수령자를 결정하기 위해 사용되는 소유권 정보가 포함된다. 더불어 퍼블릭 풀 및 보험 기금 계정의 경우, 전체 지분 수(total number of shares)도 함께 저장되며, 이는 참여자와 운영자 모두의 지분 가치를 검증하는 데 사용된다.

계정 델타 트리(Account Delta Tree)는 라이터 상태(State)의 일부가 아니라 배치(Batch)에 속하는 구조이다. 각 배치의 시작 시점에 라이터 프로버(Lighter Prover)는 비어 있는 계정 델타 트리를 초기화하고, 퍼블릭 계정 트리에 적용되는 것과 동일한 업데이트를 이 트리에 적용한다. 배치가 종료될 때쯤, 계정 델타 트리에는 해당 기간 동안 퍼블릭 계정 트리에서 발생한 모든 상태 변경이 누적된다.

이후 계정 델타 트리는 직렬화 및 압축되어 데이터 블록을 통해 이더리움으로 전송된다. 모든 배치의 블록 데이터를 순차적으로 읽어 초기에는 비어 있는 퍼블릭 계정 트리에 적용함으로써, 퍼블릭 계정 트리의 전체 상태를 완전히 재구성할 수 있다.

3.3. 무기한 시장 세부 정보(Perpetual Market Details)

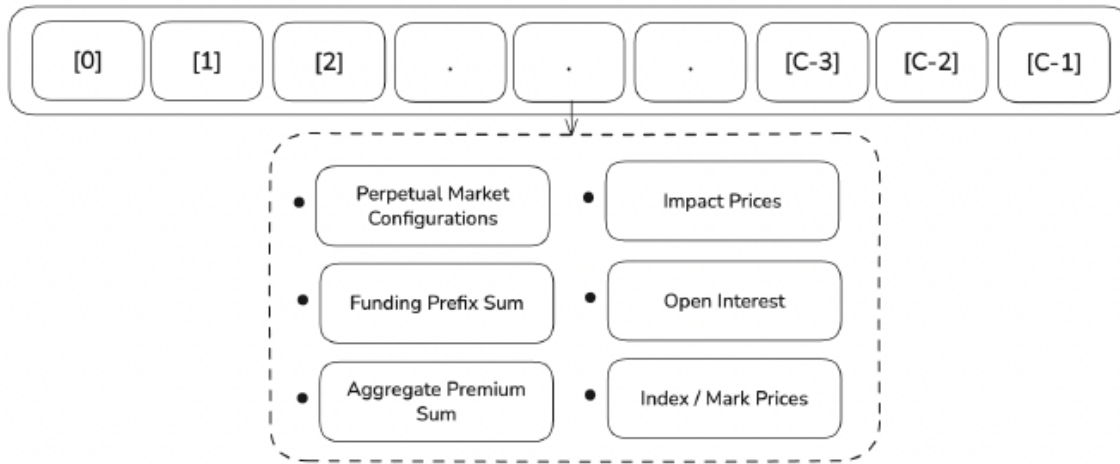


그림 5: 무기한 시장 세부 정보

무기한 시장 세부 정보(Perpetual Market Details)는 무기한 시장의 구성 설정과 함께, 펀딩 비율, 지수 가격(index price), 마크 가격(mark price), 프리미엄과 같은 시장의 전역 상태를 정의한다. 여기에는 다음과 같은 데이터가 저장된다.

무기한 시장 설정(Perpetual Market Configurations): 미결제약정(open interest) 한도, 증거금 요건, 펀딩 설정을 포함한 무기한 시장의 구성 파라미터를 정의한다.

펀딩 프리픽스 합(Funding Prefix Sum): 시장에 적용된 모든 펀딩 라운드에 대해, 펀딩 비율과 마크 가격의 곱을 누적한 합을 나타낸다.

집계 프리미엄 합(Aggregate Premium Sum): 마지막 펀딩 라운드 이후 수집된 모든 프리미엄 샘플을 누적한다. 이는 다음 펀딩 업데이트의 파라미터를 계산할 때 사용된다.

임팩트 가격(Impact Prices): 특정 시점에서 해당 오더북의 임팩트 가격을 저장한다. 프리미엄 값을 계산할 때 사용된다.

미결제약정(Open Interest): 시장의 양방향(two-sided) 총 미결제약정을 나타낸다. 이는 시장 정산 과정과 거래 시 미결제약정 한도를 강제하는 데 사용된다.

지수/마크 가격(Index / Mark Prices): 무기한 계약의 기초 자산에 대한 공정 가격(지수 가격)과 무기한 계약 자체의 공정 가격(마크 가격)을 저장한다.

3.4. 퍼블릭 무기한 시장 세부 정보(Public Perpetual Market Details)

퍼블릭 무기한 시장 세부 정보(Public Perpetual Market Details)는 퍼블릭 계정 트리와 유사한 역할을 수행하며, 이더리움에 게시된 블록 데이터만으로도 완전히 재구성할 수 있다. 이 구조에는 개별 계정의 미결 포지션 가치를 계산하는 데 사용되는 최신 마크 가격과, 미실현 펀딩 지급액을 계산하는 데 필요한 전역 펀딩 데이터가 포함된다. 퍼블릭 무기한 시장 세부 정보와 퍼블릭 계정 트리를 결합함으로써, 사용자는 자신의 전체 자산 가치를 재구성하고 온체인에서 이를 독립적으로 검증할 수 있으며, 필요 시 이스케이프 해치를 통해 안전하게 계정을 이탈할 수 있다.

3.5. 마켓 트리(Market Tree)

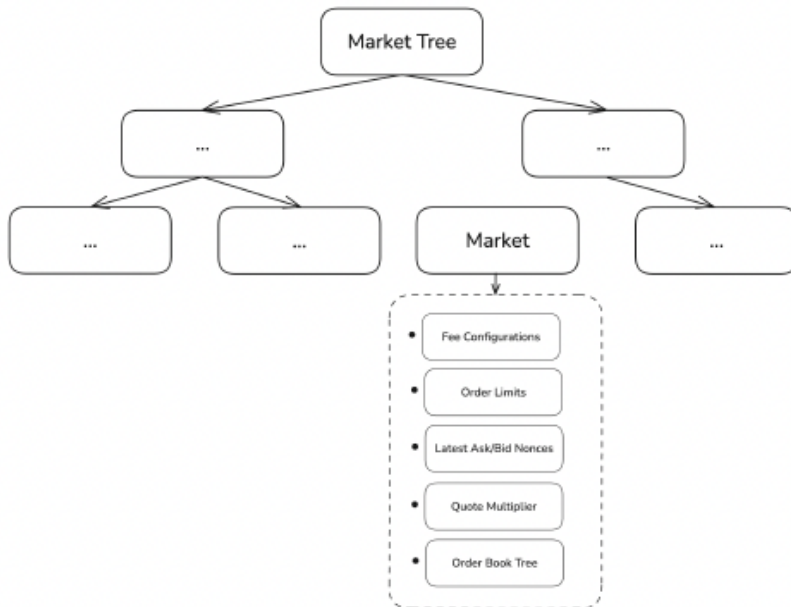


그림 6: 마켓 트리

마켓 트리(Market Tree)는 라이터 프로토콜 내의 모든 오더북과 그에 대응하는 설정 정보를 캡슐화한다. 마켓 트리의 리프에는 다음과 같은 데이터가 포함되며, 이에 한정되지는 않는다.

수수료 설정(Fee Configurations): 메이커와 테이커 거래에 적용되는 수수료율과, 청산과 관련된 수수료를 정의한다.

주문 한도(Order Limits): 허용되는 주문 크기의 최소값과 최대값을 정의한다.

최신 매도/매수 논스(Latest Ask/Bid Nonces): 해당 시장에서 가장 최근의 매도 및 매수 주문 논스를 나타내는 식별자이다.

쿼트 승수(Quote Multiplier): 각 시장은 가격과 주문 수량 모두에 대해 이산적인 단계 크기를 정의한다. 주문을 저장할 때 상태는 원시 값 대신 가격 단계 수와 수량 단계 수를 기록하며, 쿼트 승수는 이 단계 값들의 곱을 실제 쿼트 값으로 변환하는 데 사용된다.

오더북 트리(Order Book Tree): 활성 주문을 저장하고 가격-시간 우선순위(price-time-priority)에 따른 효율적인 주문 매칭 연산을 수행하기 위해 사용되는 특수 데이터 구조이다.

3.6. 인스트렉션 스택(Instruction Stack)

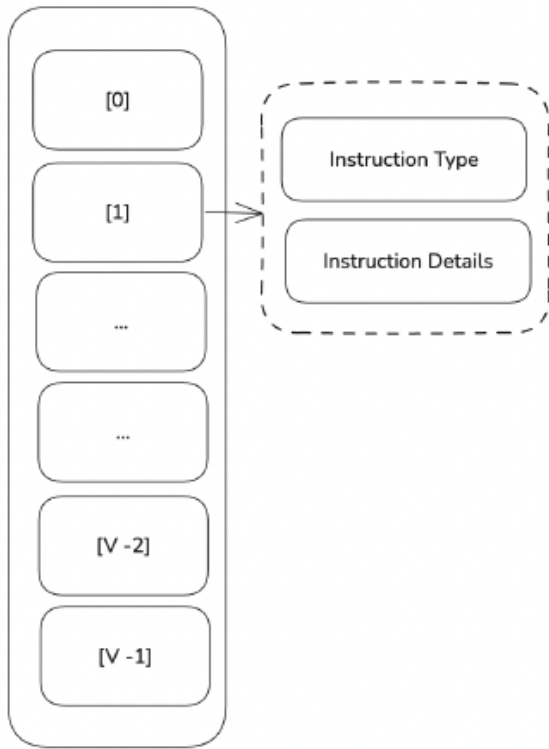


그림 7: 인스트럭션 스택

인스트럭션 스택(Instruction Stack)은 하나의 배치(batch) 동안 실행된 연산을 기록하기 위해 사용되는 순차적 데이터 구조이다. 인스트럭션 스택의 각 엔트리는 인스트럭션 유형과 이에 대응하는 인스트럭션 세부 정보를 포함한다. 이 스택은 연산의 정확한 실행 순서를 보존하여, 프로버 회로(Prover circuits) 내에서 결정론적인 재실행과 검증을 가능하게 한다.

라이트 가상 머신(Lighter Virtual Machine)에서 실행되는 대부분의 트랜잭션은 단일 실행 사이클에 대응된다. 그러나 일부 연산은 여러 실행 사이클에 걸쳐 수행될 수 있다. 예를 들어, 다수의 체결을 발생시키는 테이커 주문, 계정 내 모든 주문을 취소하는 요청, 또는 그룹화된 주문을 제출하는 경우가 이에 해당한다.

또한 거래소에 의해 트리거되는 이벤트도 존재한다. 예를 들어, 예약된 “전체 주문 취소(cancel all orders)” 요청에 따라 모든 사용자 주문을 취소하는 경우나, 거래 중 계정의 포지션 부호가 변경될 때 포지션에 연동된 주문(예: 리듀스 온리 주문, reduce-only orders)을 취소하는 경우가 있다.

이러한 이벤트를 처리하기 위해, 라이트 가상 머신은 인스트럭션 스택을 구현한다. 인스트럭션 스택은 현재 실행 사이클에서 수행되어야 할 동작을 정의하며, 스택이 비워질 때에만 다음 사용자 서명 트랜잭션이 실행된다.

개념적으로 인스트럭션 스택은 다음과 같은 데이터를 포함한다.

인스트럭션 유형(Instruction Type): 현재 실행 사이클에서 수행될 동작을 고유하게 정의하며, 주문 삽입, 주문 집합 취소, 또는 부모 주문에 연동된 자식 주문 트리거(예: SL/TP 주문 트리거 이벤트)와 같은 동작을 포함한다.

인스트럭션 세부 정보(Instruction Details): 인스트럭션의 파라미터를 정의하고, 연속된 실행 사이클 간에 데이터를 전달하기 위한 범용 필드를 구현한다. 저장되는 정보에는 관련 주문 세부 정보, 계정 인덱스, 시장 인덱스, 그리고 연산의 연속을 위해 필요한 기타 컨텍스트가 포함될 수 있다.

3.7. 오더북 트리(Order Book Tree)

오더북 트리(Order Book Tree)는 라이트 프로버를 위해 개발된 고유하고 특수한 데이터 구조이다. 이는 활성 주문을 위한 동적 저장 구조로 작동하며, 매칭 엔진의 핵심 데이터 구조를 구성하여 다음과 같은 연산을 지원한다.

1. 매도(ask) 또는 매수(bid) 지정가 주문을 오더북에 삽입
2. 활성 주문을 오더북에서 제거
3. 오더북으로부터 호가(quote) 조회

4. 가격-시간 우선순위(price-time priority)에 따른 대기 거래 실행

3.7.1. 주문 데이터(Order Data)

가격-시간 우선순위 매칭 엔진에서는 각 주문의 생성 시점을 나타내는 지표가 필요하다. 실제 타임스탬프를 저장하는 대신, 주문 간의 상대적 생성 순서를 비교하기 위해 논스(nonce)라 불리는 정수 값이 사용된다. 매도 주문과 매수 주문 모두 논스 값은 0부터 시작하며, 동일한 시장 내 동일한 방향(side)에서 새로운 주문이 생성될 때마다 이전 주문 대비 1씩 증가한다.

활성 주문은 오더북 트리의 리프 노드에 저장된다. 활성 주문의 리프 인덱스는 가격을 나타내는 P 비트와 논스를 나타내는 O 비트에 의해 결정된다. 가격 p 와 논스 o 를 갖는 활성 매도 지정가 주문의 리프 인덱스 I 는 다음과 같이 계산된다.

$$I = p \times 2^O + o.$$

매수 주문의 경우, 논스의 역값(inverse of the nonce)⁴이 계산에 사용되며, 이에 따라 다음과 같은 식이 된다.

$$I = p \times 2^O + (2^O - 1 - o).$$

이에 따라 트리의 높이 H 는 P 와 O 의 함수로 정의되며, 다음과 같다.

$$H = O + P.$$

두 개의 매도 주문 $order_i$ 와 $order_j$ 가 각각 $index_{order_i}$ 와 $index_{order_j}$ 라는 인덱스를 가진다고 하자. 각 매도 주문은 고유한 논스를 가지므로, $index_{order_i} \neq index_{order_j}$ 임이 보장된다. 인덱스 계산 방식에 의해 다음이 성립한다.

$$index_{order_i} < index_{order_j}$$

이는 다음 조건과 동치이다.

$$price_{order_i} < price_{order_j} \quad \text{또는} \quad (price_{order_i} = price_{order_j} \text{ 이고 } nonce_{order_i} < nonce_{order_j}).$$

매도 주문의 경우, 더 낮은 가격이 우선되며, 더 작은 논스는 더 이른 생성 시점을 의미한다. 따라서 매도 주문의 인덱스는 가격-시간 우선순위를 결정하며, 인덱스가 작을수록 매칭 시 더 높은 우선순위를 갖는다.

유사하게, 두 개의 서로 다른 매수 주문 $order_i$ 와 $order_j$ 가 각각 $index_{order_i}$ 와 $index_{order_j}$ 를 가진다고 하자. 각 매수 주문 역시 고유한 논스를 가지므로, $index_{order_i} \neq index_{order_j}$ 가 보장된다. 인덱스 계산 방식에 의해 다음이 성립한다.

$$index_{order_i} > index_{order_j}$$

이는 다음 조건과 동치이다.

$$price_{order_i} > price_{order_j} \quad \text{또는} \quad (price_{order_i} = price_{order_j} \text{ 이고 } nonce_{order_i} < nonce_{order_j}).$$

매수 주문의 경우, 더 높은 가격이 우선되며, 더 작은 논스는 더 이른 생성 시점을 의미한다. 따라서 매수 주문의 인덱스는 가격-시간 우선순위를 결정하며, 인덱스가 클수록 매칭 시 더 높은 우선순위를 갖는다.

또한 O 는 매도 주문의 논스 값이 항상 역변환된 매수 주문 논스보다 작도록 보장하고, 전체 논스 범위가 $2^O - 1$ 을 초과하지 않도록 충분히 큰 값으로 선택된다. 이 데이터 구조는 이러한 논스 조건이 만족되는 경우에만 주문 삽입을 허용한다.

모든 활성 지정가 주문에 대해 다음 조건이 항상 성립함이 보장된다.

$$\max\{price_{order_i} \mid order_i \text{가 매수 주문}\} < \min\{price_{order_j} \mid order_j \text{가 매도 주문}\}$$

그렇지 않다면, $price_{order_i} \leq price_{order_j}$ 를 만족하는 매도 주문 $order_j$ 와 매수 주문 $order_i$ 가 존재하게 되며, 이러한 주문은 이미 매칭되었어야 하므로 오더북에 동시에 존재할 수 없다.

이에 따라 오더북 트리는 매수 주문이 좌측에 그룹화되고, 해당 그룹의 가장 오른쪽 리프에 최우선 매수 주문(best bid)이 위치하도록 구성되며, 매도 주문은 우측에 그룹화되고, 해당 그룹의 가장 왼쪽 리프에 최우선 매도 주문(best ask)이 위치하도록 구조화된다.

가격	논스	수량	방향
1	0	2	매수
2	1	2	매수
3	0	2	매도
3	1	5	매도

표 1: 활성 주문 예시.

$H = 5, P = 2, O = 3$ 인 오더북 트리에 대해, 표 1에 나열된 주문을 고려하면, 트리의 리프는 그림 8과 같이 구성된다.

3.7.2. 내부 노드 데이터(Internal Node Data)

효율적인 호가 조회와 주문 매칭을 가능하게 하기 위해, 오더북 트리(Order Book Tree)의 각 내부 노드는 해당 서브트리에 포함된 활성 주문들에 대한 집계 정보를 저장한다. 구체적으로 다음과 같은 구조를 따른다.

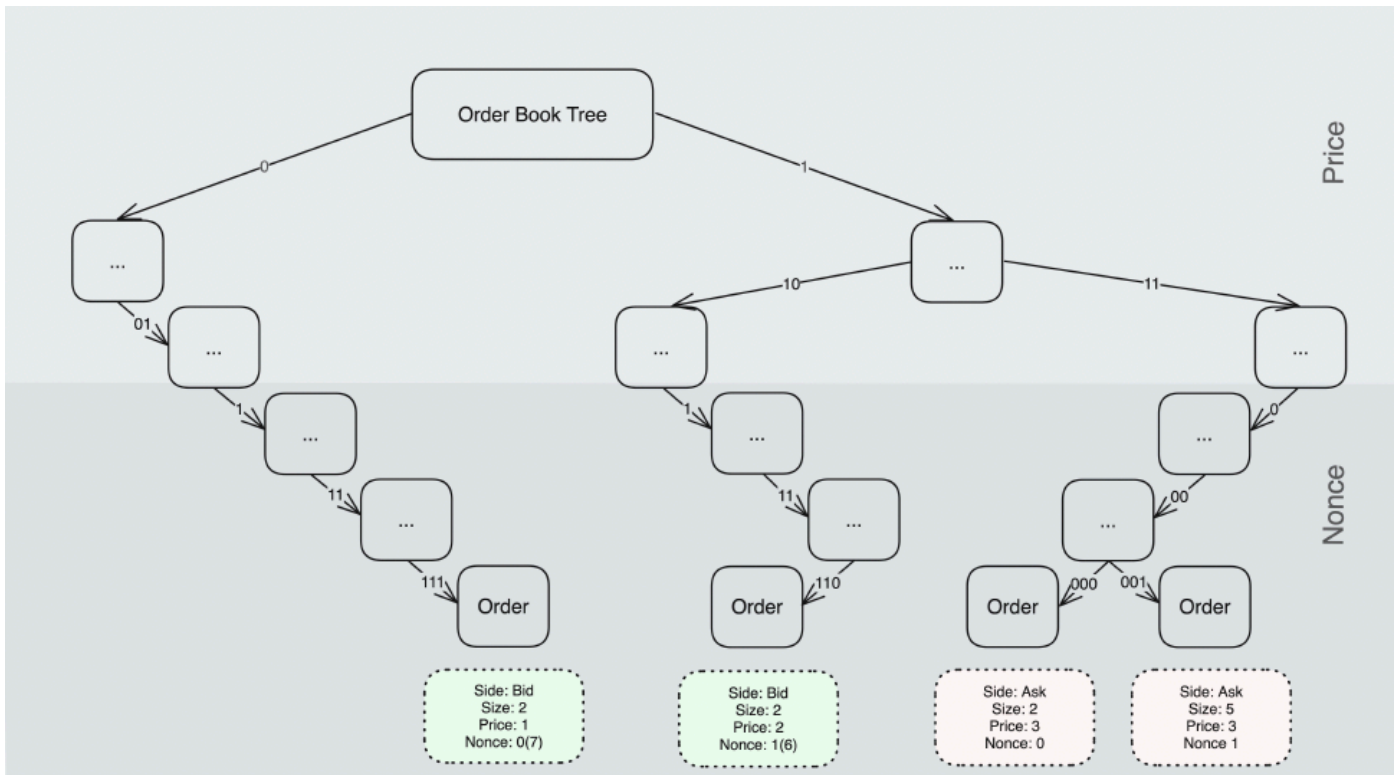


그림 8: 오더북 트리 리프 구성 예시(비어 있는 노드는 생략).

내부 노드 i 에 대해, 데이터 구조는 네 가지 상이한 값을 유지한다: $AskSizeSum_i$, $BidSizeSum_i$, $AskQuoteSum_i$, $BidQuoteSum_i$.

오더북 트리에서 내부 노드 i 에 저장되는 데이터는, 해당 노드의 서브트리에 포함된 주문 리프들을 기준으로 다음과 같이 정의된다.

$$AskSizeSum_i = \sum_{order_j \in SubTree_i} size_{order_j} \times isAsk_{order_j} \quad (1)$$

$$BidSizeSum_i = \sum_{order_j \in SubTree_i} size_{order_j} \times (1 - isAsk_{order_j}) \quad (2)$$

$$AskQuoteSum_i = \sum_{order_j \in SubTree_i} size_{order_j} \times price_{order_j} \times isAsk_{order_j} \quad (3)$$

$$BidQuoteSum_i = \sum_{order_j \in SubTree_i} size_{order_j} \times price_{order_j} \times (1 - isAsk_{order_j}) \quad (4)$$

내부 노드의 데이터를 효율적으로 계산하기 위해, 라이터 프로버(Lighter Prover)는 내부 노드 i 에 대해 다음과 같은 재귀적 관계식을 적용한다. 여기서 $left_i$ 와 $right_i$ 는 각각 해당 노드의 좌측 자식과 우측 자식을 의미한다.

$$AskSizeSum_i = AskSizeSum_{left_i} + AskSizeSum_{right_i} \quad (5)$$

$$BidSizeSum_i = BidSizeSum_{left_i} + BidSizeSum_{right_i} \quad (6)$$

$$AskQuoteSum_i = AskQuoteSum_{left_i} + AskQuoteSum_{right_i} \quad (7)$$

$$BidQuoteSum_i = BidQuoteSum_{left_i} + BidQuoteSum_{right_i} \quad (8)$$

리프 노드 i 와 이에 대응하는 주문 $order_i$ 의 경우, 각 합계 값은 다음과 같이 정의된다.

$$AskSizeSum_i = size_{order_i} \times isAsk_{order_i} \quad (9)$$

$$BidSizeSum_i = size_{order_i} \times (1 - isAsk_{order_i}) \quad (10)$$

$$AskQuoteSum_i = size_{order_i} \times price_{order_i} \times isAsk_{order_i} \quad (11)$$

$$BidQuoteSum_i = size_{order_i} \times price_{order_i} \times (1 - isAsk_{order_i}) \quad (12)$$

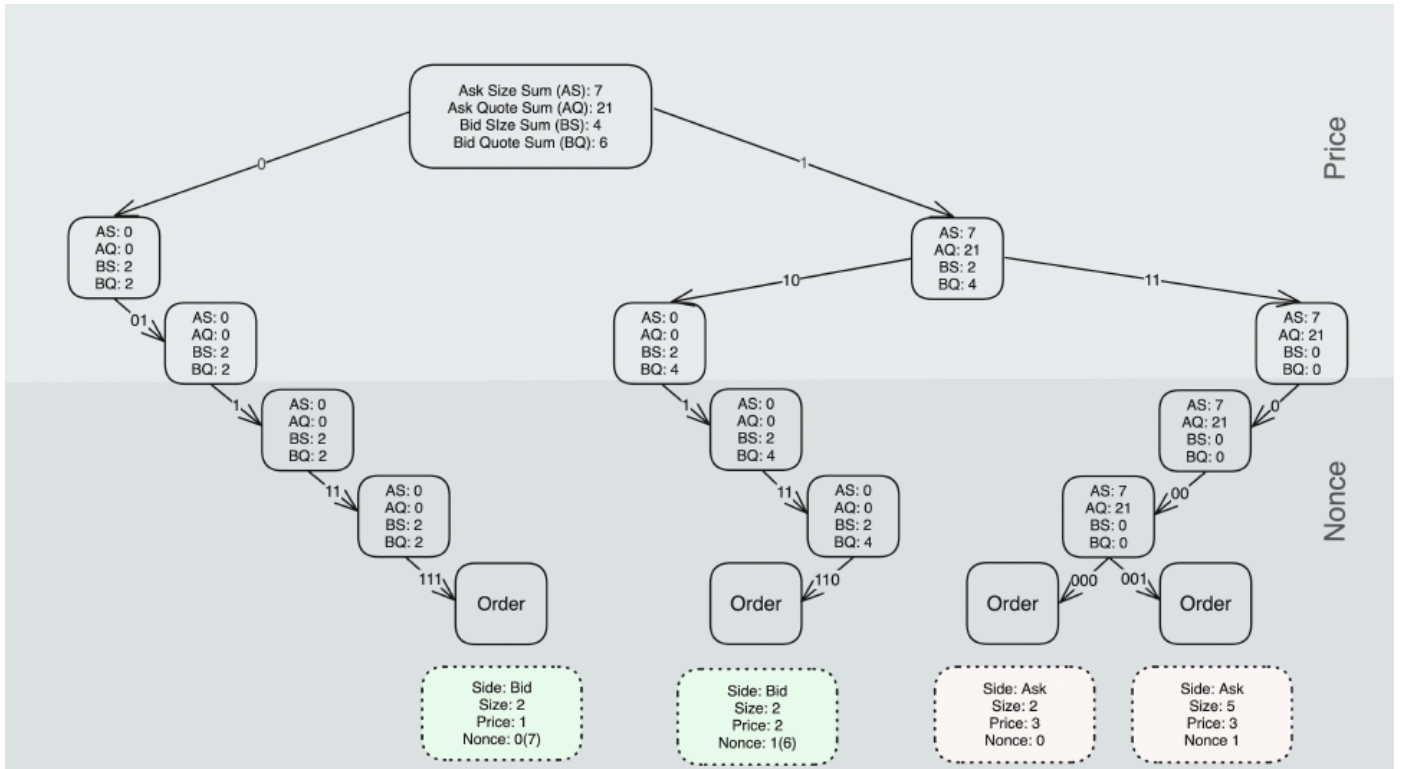


그림 9: 오더북 트리 내부 노드 구성 예시(비어 있는 노드는 생략).

$H = 5, P = 2, O = 3$ 인 오더북 트리에 대해, 표 1에 나열된 주문을 고려하면, 트리의 내부 노드 데이터는 그림 9와 같이 구성될 수 있다.

정의상, 리프 값의 어떠한 변경(예: 주문 삽입 또는 제거)은 루트에서 해당 리프까지의 경로 상에 있는 노드들에만 영향을 미친다. 따라서 내부 노드 값은 리프 데이터의 변경에 대응하여 효율적으로 갱신할 수 있다.

라이터 프로버는 주문 트랜잭션 실행에 대한 증명을 생성할 때, 오더북 트리로부터 해당 경로 데이터를 위트니스 입력으로 수신한다. 이후 회로(circuits)는 저장된 상태 루트에 대해 이 경로의 유효성을 검증한다. 이러한 검증을 가능하게 하기 위해, 오더북 트리는 머클 트리와 유사한 메커니즘을 사용하며, 머클 증명(Merkle proofs)과 유사한 해싱 패턴을 활용한다. 다만 핵심적인 차이점은, 오더북 트리가 내부 노드에도 데이터를 저장한다는 점이다. 트리 내 노드 i 의 해시는 다음과 같이 정의된다.

$$Hash_i = \begin{cases} Hash(Order_i), & \text{노드 } i \text{가 리프 노드인 경우,} \\ Hash(Hash_{left_i}, Hash_{right_i}, InternalData_i), & \text{그 외의 경우.} \end{cases} \quad (13)$$

경로의 유효성이 검증된 이후, 라이터 프로버의 회로는 리프 수준에서 발생한 변경 사항을 반영하여 리프 주문과 내부 노드 값을 갱신한다. 이후 영향을 받은 노드들은 다시 해싱되며, 회로는 실행 사이클 이후의 상태를 나타내는 새로운 상태 루트를 계산한다.

3.7.3. 대안적 데이터 구조에 대한 고려 사항(Alternative Data Structure Considerations)

매칭 엔진 내 데이터 구조는 주문 삽입 및 제거, 최우선 매도-매수 주문 조회, 그리고 지정된 퀴트 수량에 기반한 평균 체결 가격 계산 등 여러 핵심 연산을 효율적으로 처리하도록 설계되어 있다. 이러한 계산은 임팩트 가격(impact price)을 산출하는 데 사용되며, 이는 다시 무기한 선물 프리미엄 계산에 활용된다.

라이터 프로버(Lighter Prover)와 그 회로(circuits)는 무상태(stateless)로 동작한다. 즉, 상태를 유지하지 않고 제공된 위트니스(witness)의 일관성을 검증하는 증명을 생성한다. 따라서 라이터 상태(Lighter State)는 주문 위트니스를 상태 루트 입력값에 대해 효율적으로 검증할 수 있도록 지원해야 한다.

데이터 구조가 최대 N 개의 활성 주문을 지원할 수 있다고 가정하면, 이론적으로는 주문을 우선순위 내림차순으로 정렬한 연결 리스트(linked list)를 구현할 수 있다. 그러나 회로 내에서는 주어진 주문 위트니스가 상태 루트에 존재함을 검증할 수 있어야 하며, 전체 데이터 구조를 단일 해시로 표현해야 한다. 연결 리스트의 경우, 이는 모든 노드의 합성 해시를 계산해야 함을 의미하며, 각 주문 연산마다 $\Theta(N)$ 회의 해시 연산이 필요하게 된다.

이를 개선한 접근 방식으로, 연결 리스트를 머클 트리(Merkle Tree) 구조에 내장하는 방법이 있다. 이 설계에서는 각 비어 있지 않은 리프에 주문 정보와 함께 연결 리스트에서의 이전 노드와 다음 노드의 인덱스를 저장한다. 존재 증명은 표준 머클 트리와 유사하게 $\Theta(\log_2 N)$ 회의 해시 연산으로 수행할 수 있다. 주문 삽입 및 제거 시에는 영향을 받는 리프를 갱신해야 하며, 하나의 리프 변경이 인접 노드로 전파되기 때문에 각 업데이트마다 세 개의 리프-루트 경로가 위트니스로 필요하다. 또한 이 방식은 효율적인 호가 조회를 지원하지 못하는데, 모든 활성 주문의 존재를 증명해야 하므로 $\mathcal{O}(N \log_2 N)$ 회의 해시 연산이 요구된다.

이러한 한계를 극복하기 위해, 라이터 프로버는 주문 우선순위를 리프 인덱스에 직접 인코딩하는 접두 트리(prefix tree) 구조를 구현한다. 이 구조는 모든 주문을 리프 레벨에서 본질적으로 정렬하며, 최우선 매도 주문(best ask)은 가장 왼쪽 리프에, 최우선 매수 주문(best bid)은 가장 오른쪽 리프에 위치한다. 또한 내부 노드는 좌측 및 우측 서브트리에 매도 주문 리프가 존재하는지를 나타내는 정보를 저장하며, 이는 매도 주문 리프의 최좌측 위치를 검증하는 데 필수적이다.

이 구조는 존재 증명, 최우선 매도-매수 주문 식별, 주문 삽입, 주문 제거를 모두 단일 트리 경로를 따라 $\Theta(\log_2 N)$ 회의 해시 연산으로 수행할 수 있도록 지원한다. 더 나아가, 서브트리 단위 주문 집계 데이터를 저장함으로써 효율적인 호가 조회도 가능하게 하여, $\Theta(\log_2 N)$ 복잡도의 호가 계산을 지원한다.

이러한 설계는 라이터 오더북 트리(Order Book Tree)의 기반을 이루며, 매칭 엔진에 요구되는 모든 기능을 제공하는 동시에 모든 연산에 대해 최적의 해시 복잡도를 유지한다.

접근 방식	존재 증명	호가 조회	실행 사이클당 총 해시 수
연결 리스트	$\Theta(N)$	$\Theta(N)$	$\Theta(N)$
내장 연결 리스트	$\Theta(\log_2 N)$	$\mathcal{O}(N \log_2 N)$	$\mathcal{O}(N \log_2 N)$
오더북 트리	$\Theta(\log_2 N)$	$\Theta(\log_2 N)$	$\Theta(\log_2 N)$

표 2: 주문 관리를 위한 대안적 데이터 구조 비교 ⁵

4. 라이터 블록 회로(Lighter Block Circuits)

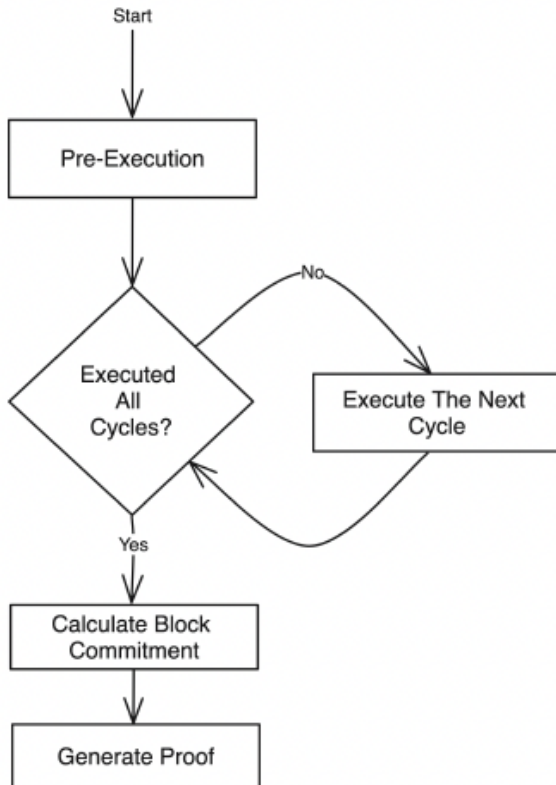


그림 10: 블록 회로

라이터 프로버는 블록 실행에 대한 간결한 증명을 생성하는 핵심 거래소 회로를 구현한다. 각 블록은 사용자 또는 거래소에 의해 개시된 트랜잭션을 포함하는 일련의 **실행 사이클(execution cycles)**로 구성되며, 사전 실행(pre-execution) 및 사후 실행(post-execution) 단계를 포함한다. 트랜잭션은 단일 사이클 내에서 실행될 수도 있고, 인스트럭션 스택(Instruction Stack)을 통해 여러 사이클로 확장되어 실행될 수도 있다.

모든 실행 사이클이 처리된 이후, 산술 회로는 이전 상태 루트와 새로운 상태 루트, 그리고 블록 및 트랜잭션 데이터를 결합한 합성 해시인 **블록 커밋먼트(block commitment)**를 계산한다. 이후 프로버는 이 커밋먼트와 보조 데이터를 공개 입력(public input)으로 사용하여, 집계 레이어에서 검증 가능한 블록에 대한 간결한 증명을 생성한다.

4.1. 사전 실행(Pre-Execution)

그림 11: 사전 실행 회로

블록 회로의 사전 실행 단계는 무기한 선물 시장 상태를 계산하고 갱신한다. 이 단계에서는 다음과 같은 작업이 수행된다.

펀딩 적용(Applying Funding): 블록 타임스탬프를 확인하여 무기한 시장에 펀딩을 적용해야 하는지를 판단하고, **무기한 시장 세부 정보(Perpetual Market Details)**를 갱신한다. 이 과정에서 펀딩 비율과 마크 가격의 곱이 펀딩 합계 값에 누적된다.

지수 및 마크 가격 갱신(Index and Mark Price Updates): 필요 시 해당 시장의 지수 가격과 마크 가격을 갱신하고, 이를 **무기한 시장 세부 정보**에 저장한다.

프리미엄 계산(Premium Calculation): 적용 가능한 경우 시장 프리미엄을 계산하여 **무기한 시장 세부 정보**에 기록한다.

마크 가격 계산(Mark Price Calculation): **무기한 시장 세부 정보**에 저장된 데이터를 사용하여 마크 가격을 재계산하고 갱신한다.

4.2. 실행 사이클(Execution Cycle)

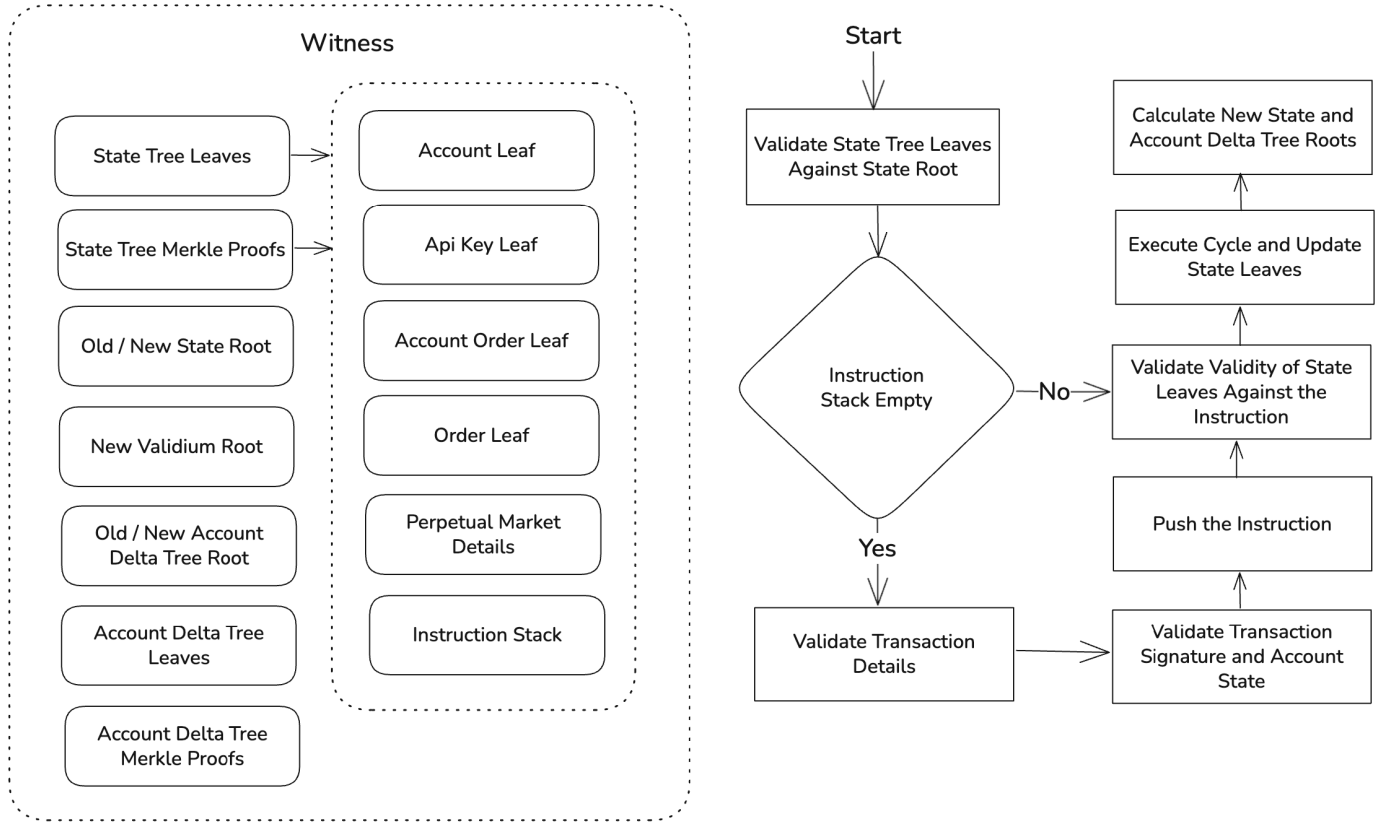


그림 12: 실행 사이클

사전 실행(pre-execution) 단계가 완료되면, 프로버(Prover)는 일련의 실행 사이클(execution cycles)을 수행한다. 각 사이클에서 회로는 인스트럭션 스택(Instruction Stack)에 저장된 인스트럭션에 따라 새로운 트랜잭션을 개시하거나, 스택에 남아 있는 인스트럭션의 처리를 계속한다. 새로운 트랜잭션을 실행할 경우, 회로는 트랜잭션 데이터의 유효성을 검증하고, 위트니스로 제공된 서명이 트랜잭션 개시자와 최신 API 키 nonce(nonce)에 정확히 대응하는지를 확인한다.

이후 회로는 제공된 머클 경로(Merkle paths)를 따라 각 노드를 상태 루트(State Root) 및 계정 델타 트리(Account Delta Tree)와 대조함으로써 위트니스 데이터의 유효성을 검증한다. 이 과정에서 실행에 필요한 모든 리프 데이터가 올바르게 제공되었는지를 확인한다. 예를 들어, 계정 및 서브 계정 리프가 트랜잭션 개시자와 일치하는지, 주문 취소 시 제공된 주문 경로가 정확한 리프를 가리키는지를 검증한다. 또한 실행 전후에 계정이 충분한 증거금을 유지하는지를 확인하기 위한 리스크 체크도 수행된다.

모든 검증이 통과되면, 회로는 관련 리프 데이터를 갱신하고, 그 결과로 생성된 상태를 계산한 뒤 새로운 루트를 도출함으로써 현재 실행 사이클을 완료한다.

4.3. 주문 삽입(Order Insertion)

주문은 반대편에 교차 주문(crossing orders)이 존재하지 않는 경우에만 오더북에 삽입된다. 새로운 매도 지정가 주문 $order_{new}$ 를 예로 들자. 삽입에 앞서, 회로는 오더북 트리(Order Book Tree)에 저장된 내부 노드 값을 활용하여 교차 주문이 존재하지 않음을 검증한다. $order_{new}$ 의 인덱스를 $index_{order_{new}}$ 라고 할 때, 교차 주문 집합은 다음과 같이 정의된다.

$$\{order_i \mid order_i \text{ 는 매수 주문이며 } index_{order_i} > index_{order_{new}}\}.$$

$L = \{L_i \mid L_i \text{ 는 루트에서 인덱스 } index_{order_{new}} \text{ 를 갖는 리프까지의 경로 상에 있고, 높이 } i \text{ 에 위치한 노드}\}$ 라고 하자. 또한

$$index_{order_{new}} = I = I_0 I_1 I_2 \dots I_{H-1}$$

로 두며, 여기서 I_i 는 I 의 i 번째 비트를 의미한다. $I_i = 0$ 이면 L_i 는 L_{i+1} 의 좌측 자식이고, $I_i = 1$ 이면 우측 자식이다.

교차 주문의 총 수량은 다음과 같이 계산된다.

$$S = \{order_i \mid order_i \text{ 는 매수 주문이고 } index_{order_i} > index_{order_{new}}\} \quad (14)$$

$$crossingSize = \sum_{order_i \in S} size_{order_i} \quad (15)$$

$$= \sum_{L_i \in L, 0 < i \leq H} (BidSizeSum_{L_i} - BidSizeSum_{L_{i-1}}) \times (1 - I_{i-1}) \quad (16)$$

즉, 루트에서 대상 리프까지의 경로를 따라 내려가면서, 다음 경로 노드가 좌측 자식인 경우 해당 노드의 우측 서브트리에 포함된 매수 주문 수량을 합산함으로써 전체 교차 수량을 계산한다.

유사하게, 매수 지정가 주문의 경우 교차 수량은 다음과 같이 정의된다.

$$S = \{ order_i \mid order_i \text{ 는 매도 주문이고 } index_{order_i} < index_{order_{new}} \} \quad (17)$$

$$crossingSize = \sum_{order_i \in S} size_{order_i} \quad (18)$$

$$= \sum_{L_i \in L, 0 < i \leq H} (AskSizeSum_{L_i} - AskSizeSum_{L_{i-1}}) \times I_{i-1} \quad (19)$$

회로는 경로 L 과 해당 내부 노드 값을 이용해 교차 수량을 계산하고, 그 결과가 0임을 확인함으로써 교차 주문이 존재하지 않음을 검증한 뒤에만 새로운 주문을 삽입한다.

4.4. 대기 거래 실행(Pending Trade Execution)

교차 주문이 존재하는 경우, 회로는 거래를 실행하기 전에 제공된 주문 리프가 반대편의 교차 주문 중 가장 높은 우선순위를 가진 주문임을 기대한다. 제공된 주문 리프가 실제로 반대편의 교차 주문에 해당함을 확인한 뒤, 해당 주문이 모든 교차 주문 중 최우선 순위임을 추가로 검증한다.

taker 매도 주문 $order_{new}$ 와 이에 대응하는 교차 메이커 주문 $order_{cross}$ 를 가정하자. I 를 최고 우선순위를 갖는 매수 주문인 $order_{cross}$ 의 인덱스라고 하자. 앞선 경우와 마찬가지로, 회로는 현재 실행 사이클의 입력으로 루트에서 인덱스 I 를 갖는 리프까지의 경로 L 을 수신한다. $order_{cross}$ 의 유효성은 이 경로 L 을 사용하여 $crossingSize$ 를 다시 계산하고, $crossingSize = 0$ 임을 확인함으로써 검증된다.

이 조건이 만족되면, 회로는 주문 만료 여부, 리듀스 온리(reduce-only) 제약, 증거금 충분성 등과 같은 검증을 수행하여 대기 거래를 검증한다. 검증이 성공하면 거래가 실행되며, 인스트럭션 스택에 저장된 인스트럭션이 그에 맞게 갱신된다.

5. 다층 집계 및 블롭 회로(Multi-Layered Aggregation and Blob Circuits)

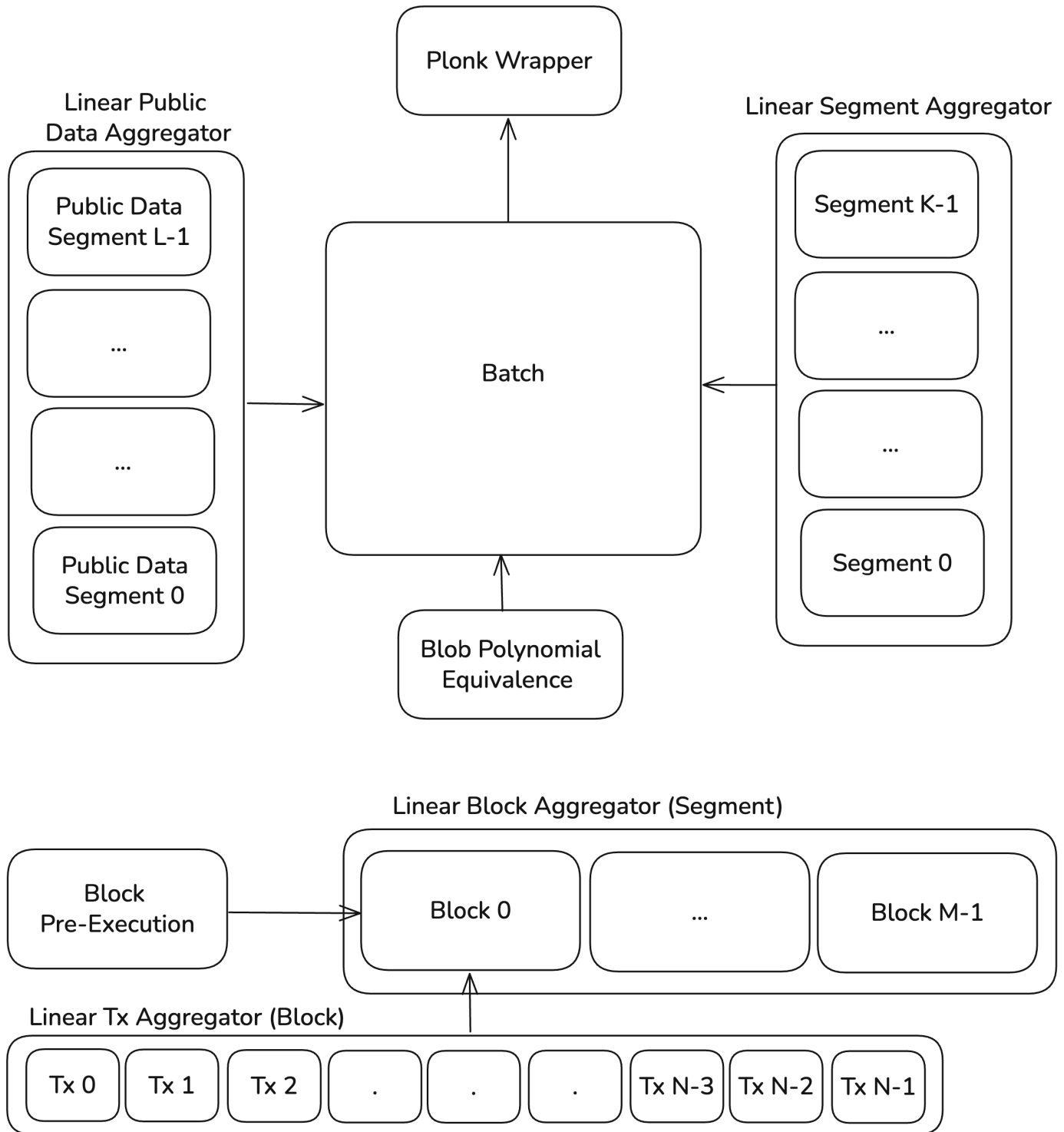


그림 13: 계층적 집계 회로

하나의 배치(batch)는 연속된 세그먼트(segment)들로 구성되며, 각 세그먼트는 다시 연속된 블록(block)들로 이루어진다. 세그먼트는 병렬화 계층으로 작동하여, 개별 블록에 대한 증명은 독립적으로 생성될 수 있고, 이후 선형적으로 집계되어 해당 세그먼트의 증명을 형성한다. 그 다음, 모든 세그먼트 증명은 다시 선형적으로 집계되어, 배치 내의 모든 실행 사이클을 대표하는 단일 증명이 생성된다.

실행의 정확성을 증명하는 것 외에도, 라이터 프로버(Lighter Prover)는 이더리움(Ethereum)에 게시된 블록(blob) 데이터가 동일한 배치에서 생성된 실행 퍼블릭 데이터와 정확히 일치함을 검증해야 한다. 이 검증은 두 단계로 나뉜다.

1. **게시된 블록-청구된 블록 동등성(Posted Blob-Claimed Blob Equivalence):** 이더리움은 게시된 블록 데이터에 대해 다항식 평가를 수행하는 프리컴파일(precompile)을 제공한다. 라이터 프로토콜이 스마트 컨트랙트에 배치를 커밋할 때, 블록 다항식을 임의의 점에서 평가하고, 해당 평가 지점과 결과를 최종 증명 커밋먼트의 공개 입력(public input)으로 포함한다. **블록 다항식 동등성 회로(Blob Polynomial Equivalence Circuit)**는 동일한 임의의 지점에서 청구된 블록 다항식을 평가하고,

두 평가 결과가 일치함을 검증함으로써, 이더리움에 게시된 ब्ल록 데이터가 증명 시스템 내에서 청구된 ब्ल록 데이터와 동일함을 확인한다.

2. **실행 퍼블릭 데이터 – 청구된 ब्ल록 동등성(Execution Public Data – Claimed Blob Equivalence):** 라이터 프로버는 *계정 델타 순회 회로(Account Delta Traversal Circuits)*를 사용하여 *계정 델타 트리 루트(Account Delta Tree Root)*를 값 스트림으로 직렬화하고, 이 스트림으로부터 구성된 다항식을 임의의 점에서 평가한다. 동시에, 배치 회로는 청구된 ब्ल록 바이트 데이터를 역직렬화하여 대응되는 값 스트림을 생성하고, 동일한 방식으로 다항식 평가를 수행한다. 이후 *선형 퍼블릭 데이터 집계기(Linear Public Data Aggregator)*의 증명과 해당 평가 결과가 검증되면, 배치 회로는 두 평가 결과를 비교하여 실행 퍼블릭 데이터와 청구된 ब्ल록 데이터가 동일함을 확인한다.

두 동등성 검증에서 사용되는 임의의 평가 지점은 *피아트-샤미르 휴리스틱(Fiat-Shamir heuristic)*을 통해 도출되며, 이를 통해 두 다항식의 모든 입력이 난수 생성 과정에 포함되도록 보장한다.

모든 배치 수준의 검증이 완료되면, 집계된 데이터로부터 단일 퍼블릭 커밋먼트가 계산되고, 이에 대한 증명이 생성된다. 이 증명은 이후 PLONK 증명으로 래핑되어, 온체인 검증을 위해 이더리움에 제출된다.

6. 이스케이프 해치(Escape Hatch)

라이터 프로토콜의 정상 실행 모드에서, 사용자는 자산을 인출하기 위해 두 가지 방법을 사용할 수 있다. 하나는 시퀀서를 통해 롤업 트랜잭션을 게시하는 것이고, 다른 하나는 스마트 컨트랙트를 통해 우선 트랜잭션을 직접 큐에 적재하는 것이다. 두 경우 모두 시퀀서는 이러한 트랜잭션을 모니터링하여 향후 배치에 포함시키며, 스마트 컨트랙트는 우선 트랜잭션이 사전에 정의된 기한 내에 반드시 포함되도록 강제한다.

만약 시퀀서가 이 기한 내에 어떠한 우선 트랜잭션도 배치에 포함시키지 못할 경우, 시스템은 허가 없이(permissionlessly) *이스케이프 해치(Escape Hatch)* 모드로 전환된다. 이스케이프 해치는 결코 트리거되지 않도록 설계된 강력한 안전장치로서, 시스템의 무결성을 보존하는 동시에 시퀀서의 인센티브를 올바른 운영 행위와 정렬시키는 역할을 수행한다.

이스케이프 해치가 활성화되면, 추가적인 배치 커밋과 검증이 모두 비활성화되어 프로토콜 상태가 동결되며, 결과적으로 시퀀서의 동작이 중단된다. 이 동결 상태에서 스마트 컨트랙트는 자산 회수를 보장하기 위한 사용자 출구 연산만을 지원한다.⁶

이더리움에 게시된 ब्ल록 데이터에는 전체 *퍼블릭 계정 트리(Public Account Tree)*와 *퍼블릭 무기한 시장 세부 정보(Public Perpetual Market Details)*를 재구성하는 데 필요한 모든 정보가 포함되어 있기 때문에, 사용자는 자신의 자산 소유권, 미결제 포지션의 손익(PnL), 그리고 퍼블릭 풀 지분 가치를 검증하는 증명을 독립적으로 생성할 수 있다. 이러한 증명이 제출되면, 스마트 컨트랙트는 이를 동결된 상태 루트와 대조하여 검증하고, 검증이 성공할 경우 해당 자산을 정당한 소유자에게 반환한다.

7. 결론(Conclusion)

라이터 프로토콜은 블록체인 인프라 상에서 효율적인 거래를 가능하게 하는 새로운 아키텍처를 제시한다. zk-SNARK 기술과 확장 가능한 증명 시스템의 발전을 활용함으로써, 정교한 트레이더의 요구와 현재 블록체인 네트워크의 역량 사이의 간극을 해소한다.

기존의 오더북 거래를 위한 롤업 또는 레이어 1 솔루션과 달리, 라이터는 완전한 매칭 및 청산 엔진을 *라이터 프로버(Lighter Prover)*에 직접 통합하고, 이를 목적 특화 데이터 구조로 지원한다. 이러한 설계는 엄격한 가격-시간 우선순위를 강제하고, 공정한 청산을 보장하며, 오더북 내에서의 검열 리스크를 최소화한다. 그 결과, 라이터는 비수탁적이고, 검증 가능하며, 저지연·저슬리피지 거래 환경을 제공함으로써 시장의 무결성과 효율성을 동시에 향상시킨다.

8. 향후 과제(Future Work)

라이터 프로토콜의 핵심은 광범위한 금융 연산을 지원하기 위한 인프라를 제공하는 데 있다. 여기에는 현물 거래, 예측 시장, 무기한 선물뿐만 아니라, 대출 및 기타 복잡한 금융 프리미티브도 포함될 수 있다. 라이터의 비전은 투명성, 검증 가능성, 그리고 강력한 보안 보장을 전통적인 금융 인프라에 내재화하는 것이다. 초기에는 디지털 자산의 무기한 거래에 중점을 두고 있지만, 이 아키텍처는 다양한 자산 클래스와 시장 유형 전반으로 일반화될 수 있도록 설계되어 있어, 신뢰 최소화 금융의 최전선에 라이터를 위치시킨다.

연산 역량을 확장하기 위해, 라이터 프로토콜은 범용 연산을 위한 사이드카 가상 머신(VM)을 도입할 계획이다. 재귀적 증명(recursive proving)을 통해, 라이터 프로버는 핵심 금융 연산에 대해서는 높은 성능을 유지하면서도, 더 느린 범용 증명 환경을 통합할 수 있다. 이러한 설계는 고효율 금융 프리미티브와 유연한 연산 확장을 결합한 폭넓은 생태계를 가능하게 한다.

라이터 팀은 프로토콜 내에서 최대 추출 가능 가치(Maximal Extractable Value, MEV)의 잠재적 원천을 최소화하는 데 지속적으로 헌신하고 있다. 현재 공정한 시퀀싱 기법과 트랜잭션 암호화, 사전 커밋(pre-commitment) 스킴과 같은 암호학적 방법을 포함한 연구가 진행 중이다. 이러한 노력은 증명 검증이 완료되기 이전 단계에서도 운영자가 즉각적인 최종성을 제공하고 모든 프로토콜 연산의 무결성을 유지하도록 강력한 인센티브를 부여하는 것을 목표로 한다.

9. 감사의 말(Acknowledgements)

우리는 이더리움 생태계 내 다수의 전문가들과, 특히 영지식 증명, 프로토콜 아키텍처, 탈중앙화 금융을 중심으로 매우 생산적인 논의를 진행해 왔다. 그들의 질문과 피드백은 본 연구와 개발, 그리고 본 문서의 완성도에 큰 영향을 미쳤다. 특히 다음 분들께 감사를 표하고자 한다: Scott Wu, Jacob Willemsma, D.C. Posch, Nalin Bhardwaj, Brian Gu, Yi Sun, Uma Roy, Anthony Rose, Liam Horne, Ye Zhang, Brian Redford, John Adler, Gabriel Blaut, Kydo Ox, Dmitriy Berenzon, Benji Funk, Scott Kominers, Gautam Botrel, Joseph Lubin.

각주(Footnotes)

-
1. 위트니스(Witness)에 대한 상세한 설명은 2.4절에서 다룬다. ↩
 2. 라이터 프로토콜은 현재 프라이버시 중심 롤업으로 운영되고 있지 않기 때문에 zk-SNARK의 영지식(zero-knowledge) 속성을 완전히 활용하지는 않는다. 대신, 라이터 내에서 SNARK의 주요 역할은 트랜잭션 실행과 상태 전이의 정확성을 효율적으로 검증하는 데 있다. ↩
 3. 머클화에는 Poseidon2 해시 함수가 사용된다. ↩
 4. 논스의 역값은 O 비트로 표현된 논스의 모든 비트를 반전시켜 얻은 값으로 정의된다. ↩
 5. 실행 사이클은 라이터 프로버의 단위 연산을 의미하며, 이에 대한 상세한 설명은 4장에서 다룬다. ↩
 6. 이스케이프 해치가 트리거되면, 모든 미결 포지션은 최신 마크 가격에서 정산된 것으로 간주된다. 활성화 시점에 미결 포지션을 보유한 계정의 경우, 사용자는 공개적으로 이용 가능한 블록 데이터를 사용하여 해당 포지션의 세부 정보와 손익(PnL)을 증명할 수 있다. 이후 미실현 손익, 퍼블릭 풀 지분 가치, 그리고 잔여 자산을 인출할 수 있다. ↩