# FRAUD PROOFS

## MAXIMISING LIGHT CLIENT SECURITY AND SCALING BLOCKCHAINS WITH DISHONEST MAJORITIES

# REFERENCE


- Al-Bassam, Mustafa, Alberto Sonnino, and Vitalik Buterin. "Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities." arXiv preprint arXiv:1809.09044 (2018).

# ABSTRACT

# ABSTRACT

- 풀노드(full node)는 무조건 정직하지 않음
- 라이트 클라이언트(light client)
  - 단순 지불 증명(Simple Payment Verification, SPV)을 수행
  - 풀노드로부터 생성된 사기 증명(fraud proofs)을 받을 수 있는 상황
- 프로토콜 규칙을 훼손하는 블록이 존재하는 상황

# ABSTRACT

- 대다수가 정직하다는 가정(honest-majority assumption)을 무시하게 만듦
  - 대신에 적은 수의 정직한 노드를 상정하는 약한 가정을 상정하게 함

# ABSTRACT

- 사기 증명(fraud proofs)과 데이터 가용성 증명(data availability proofs)
  - 온체인 데이터는 가용 가능하고 유효하다는 강한 가정을 유지
  - 샤딩이나 더 큰 블록(bigger block) 솔루션 등
  - 온체인(on-chain) 확장성의 핵심 요소
- 본 논문
  - 우수한 사기 증명 및 데이터 가용성 증명을 제안, 구현, 평가

# INTRODUCTION

# INTRODUCTION

- 현존하는 블록체인의 확장성 제한 문제가 현실화
  - 비트코인 트랜잭션의 수수료가 $20를 넘어서거나
  - 이더리움 트랜잭션의 오랜 대기(pending)를 초래

# INTRODUCTION

- 온체인 용량(capacity) 제한을 높이는 것은 탈중앙화 정도 및 보안을 훼손
  - 용량이 증대하면 블록체인의 전체 다운로드에 필요한 용량 및
  - 네트워크 자원 요구사항 또한 커짐
  - 결국 소수의 사용자만이 풀노드를 운영할 수 있게 되기 때문

# INTRODUCTION

- 풀노드는
  - 네트워크의 다른 노드에 대한 의존성 없이 독립적으로 검증 가능한 노드
  - 합의를 유지하는 중요한 근간

# INTRODUCTION

- 일반적인 상황에서 라이트 클라이언트는 잘 작동
- 비정직한 채굴자 또는 블록 생산자가 대다수인 약한 가정 상황에서는 그렇지 못함

# INTRODUCTION

- 비정직한 노드가 대다수인 비트코인 또는 이더리움 네트워크를 상정
  - 블록에 대한 트랜잭션의 검열, 재배치, 심지어 유효하지 않은 트랜잭션의 포함
- 라이트 클라이언트는 (정직한) 풀노드의 보조 없이는 이를 검출할 수 없음
- 풀노드는 검출이 가능하다.
  - 즉, (라이트 클라이언트를 풀노드로 참여 가능하게끔 하는)
  - 확장성 문제의 해결책이 요구

# INTRODUCTION

- 다양한 확장성 해결책들이 제시
- 대표적인 오프체인(off-chain) 해결책
  - 지불 채널(payment channels) 혹은 상태 채널(state channel)
  - 채널을 열고 설정하는 과정은 온체인 트랜잭션을 필요로 함
  - 결국 지불 채널 또는 상태 채널의 확산을 위해서는 여전히 온체인 확장성이 필요

# INTRODUCTION

- 본 논문에서는 온체인 용량과 보안 사이의 트레이드 오프를 줄이고
  - 라이트 클라이언트가 풀노드로부터 만들어진
  - 유효하지 않은 블록의 사기 증명을 받고 검증 가능하도록 하는 방법을 제안

# INTRODUCTION

- 기꺼이 사기 증명을 제공하는 단 하나의 정직한 풀노드만이 존재하는 상황
  - 심지어 네트워크 지연(delay)이 최대인 상황에서도
  - 라이트 클라이언트가 유효하지 않은 블록을 거절할 수 있도록 함

# INTRODUCTION

- 사기 증명을 보완하는 데이터 가용성 증명을 디자인
  - 전체 디자인에 대한 보안 및 효율성을 평가
- 블록체인의 확장성에 중요한 요소
  - 사기 증명은 샤딩 시스템에서
  - 악의적인 샤드의 유효하지 않은 블록을 검출할 수 있도록 함

# ASSUMPTION

# ASSUMPTION

- 사기 증명 및 데이터 가용성 증명을 위한
  - 네트워크 및
  - 위협 모델

# NETWORK

- 네트워크가 두 종류의 노드로 구성되었다고 가정
- 풀노드
  - 전체 블록체인을 다운로드하고 검증하는 노드
  - 정직한 풀 노드는 유효한 블록들을 저장
  - 유효한 블록에 대한 블록 헤더를 라이트 클라이언트들에게 다시 브로드캐스트
  - 풀노드 중 일부는 블록을 생성하는 등 합의에 참여

# NETWORK

- 네트워크가 두 종류의 노드로 구성되었다고 가정
- 라이트 클라이언트
  - 라이트 클라이언트는 연산 성능이나 네트워크 대역폭 등이
  - 전체 블록체인을 다운로드하고 검증하기에는 너무 낮은 노드
  - 이들은 풀노드로부터 블록 헤더를 받고
  - 임의 트랜잭션이나 상태에 대한 머클 증명을 요청할 수 있음

# NETWORK

- 네트워크 토폴로지(topology)는 다음 그림과 같다고 가정

# NETWORK

- 네트워크 토폴로지(topology)는 다음 그림과 같다고 가정
  - 풀노드는 서로 통신
  - 라이트 클라이언트는 풀노드와 통신
  - 라이트 클라이언트는 서로 통신하지 않음

# NETWORK

- 최대 네트워크 지연은 $\delta$라 상정
  - 하나의 정직한 노드가 네트워크에 연결하고
  - 블록 등에 해당하는 임의의 데이터를 다운로드하는 시점 $T$에 대해
  - 다른 정직한 노드도 동일한 행동을 시점 $T' \leq T + \delta$에 행할 수 있음을 보장

# THREAT

- 위협 모델에 대해 다음과 같이 가정
- 블록과 합의
  - 블록 헤더는 공격자에 의해 생성될 수 있고
  - 그러므로 유효하지 않을 수 있음
  - 합의에 참여하는 노드들에 대해서도 정직한 다수를 상정하지 않음

# THREAT

- 위협 모델에 대해 다음과 같이 가정
- 풀노드가 비정직할 수 있음
  - 풀노드는 사기 증명과 같은 정보를 (검열함으로써) 전달하지 않거나
  - 유효하지 않은 블록을 전달할 수 있음
  - 네트워크에서 적어도 하나의 정직한 풀노드는 온라인 상태에 있음을 가정

# THREAT

- 위협 모델에 대해 다음과 같이 가정
  - 정직한 풀노드는 기꺼이 사기 증명을 생성하고 배포
  - 이클립스 공격(eclipse attack) 상황에 있지 않다고 가정

# THREAT

- 위협 모델에 대해 다음과 같이 가정
- 라이트 클라이언트
  - 각 라이트 클라이언트는 적어도 하나의 정직한 풀노드에 연결되어 있음을 가정
  - 데이터 가용성 증명에 대해, 블록이 재구축될 수 있기 위한
    최소 개수의 정직한 라이트 클라이언트를 가정
  - 이 특정 수치는 시스템의 매개변수에 달려있음

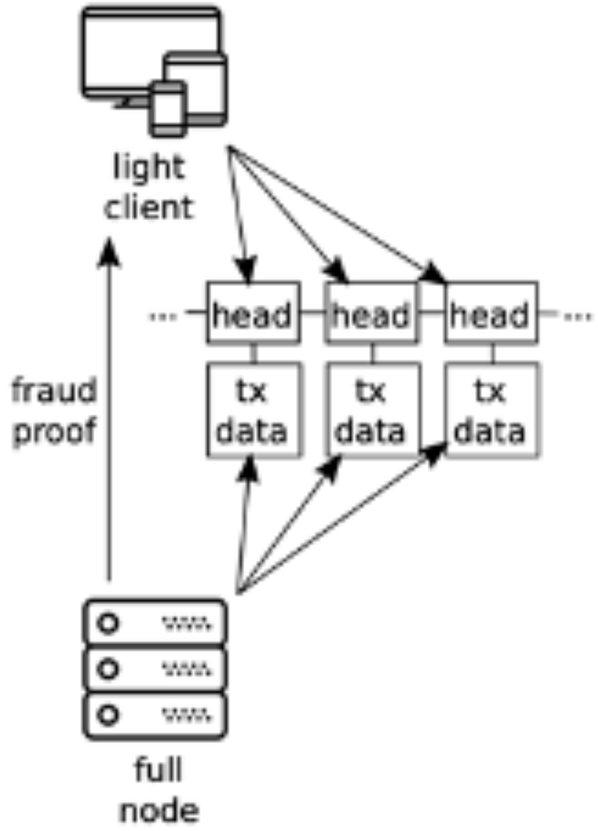
# THREAT

- 목표는 라이트 클라이언트가 비정직한 합의 참여 노드(풀노드)들이 대다수인 경우에도
  - 유효하지 않은 트랜잭션을 포함하는 블록을 수용하지 않음을 보장하는 것
- 비정직한 대다수가 체인을 분기하여 유효한 트랜잭션을 취소시키는
  - 이중 지불 문제와는 다르고, 본 논문에서도 초점 맞추지 않을 것
  - 이중 지불 문제를 막기 위해서는 여전히 정직한 다수가 존재하다는 가정이 필요

# FRAUD PROOFS

# FRAUD PROOFS

- 네트워크 레벨에서의 사기 증명 시스템 구조에 대한 개요
-



# FRAUD PROOFS

- 효율적인 사기 증명을 위해
  - 블록체인 데이터 구조가 사기 증명 생성을 지원하도록 설계하는 것이 필요

# FRAUD PROOFS

- 높이 $i$에서의 블록 헤더 $h_i$는 다음 요소들을 포함
  - $prevHash_i$는 체인의 이전 블록 헤더의 해시
  - $dataRoot_i$는 블록에 포함된 데이터(트랜잭션)의 머클 트리의 루트
  - $dataLength_i$는 $dataRoot_i$를 구성하는 트리의 리프 노드의 개수
  - $stateRoot_i$는 블록체인의 상태에 대한 (희소) 머클 트리의 루트
  - $additionalData_i$는 프로토콜에서 요구하는 추가적인 임의의 데이터
    - 가령 작업 증명(proof-of-work)에서 논스와 난이도 필드

# FRAUD PROOFS

- 각 블록 헤더의 해시 $blockHash_i = hash(h_i)$는
  - 라이트 클라이언트와 풀노드가 저장함

# TRANSITION

- 상태 전이
  - *rootTransition*을 정의할 것
  - 트랜잭션이 읽거나 수정할 수 있는 상태 트리에 대해
  - 전체 상태 트리 없이
  - 상태 루트와 머클 증명만으로 전이를 수행

# TRANSITION

- $rootTransition(stateRoot, t, w) \in \{stateRoot', err\}$
  - 트랜잭션 $t$
  - 머클 증명 (상태 증명) $w$
  - 새 상태 $stateRoot'$ 또는 $err$를 반환

# TRANSITION

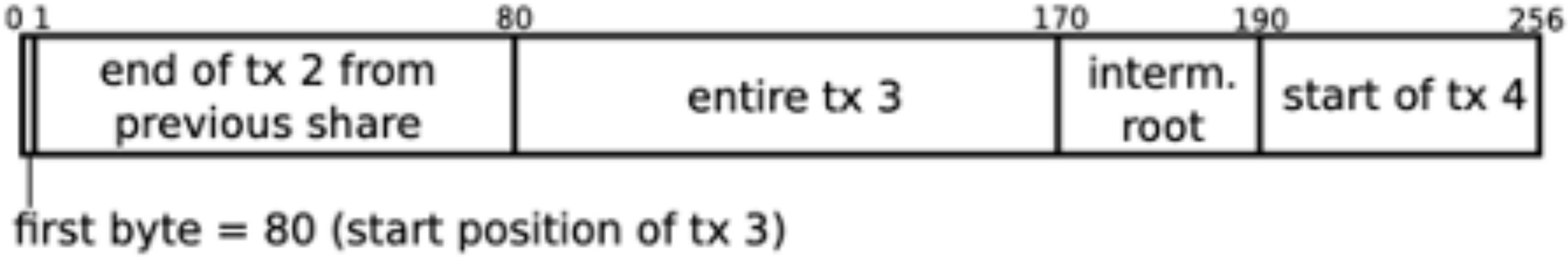
- $rootTransition(stateRoot, t, w) \in \{stateRoot', err\}$
- 키-값 쌍의 집합과 연관된 상태 트리의 (희소) 머클 증명으로 구성
  - $w = \{(k_1, v_1, \{k_1, v_1 \rightarrow stateRoot\}), (k_2, v_2, \{k_2, v_2 \rightarrow stateRoot\}),$
    $\ldots (k_n, v_n, \{k_n, v_n \rightarrow stateRoot\})\}$

# TRANSITION

- 중간 상태 루트 $interRoot_i^j = rootTransition(interRoot_i^{j-1}, t_i^j, w_i^j)$
  - $t_i^j$ : 블록 $i$의 $j$번째 트랜잭션
  - 기저 조건 $interRoot_i^{-1} = stateRoot_{i-1}$
  - $stateRoot_i = interRoot_i^n$

# SHARE

- 셰어(share)
  - 데이터 가용성 증명이 가능하도록 하기 위해
  - 데이터를 정해진 크기의 셰어로 정렬할 필요가 있음
-



# SHARE

- 셰어가 전체 트랜잭션들을 포함하지 않고 트랜잭션들의 일부만을 포함
- 각 셰어의 첫 번째 바이트를
  - 셰어에서 시작되는 첫 트랜잭션의 시작점으로써 미리 할당하거나(오프셋)
  - 만일 셰어에서 트랜잭션이 시작되지 않으면 0으로 할당
- 프로토콜 메시지 파서(parser)가 블록의 모든 트랜잭션이 없이도
  - 메시지 경계를 확립할 수 있도록 함

# SHARE

- 함수 *parseShares*
  - 셰어들로부터 $t$개의 메시지의 정렬된 리스트 $(m_0, m_1, \dots)$ 를 출력
  - 메시지는 트랜잭션이거나 중간 상태 루트(trace)
    - 중간 상태 루트를 적용할 빈도를 결정

# SHARE

- 함수 $parsePeriod$
  - 이전-중간 상태 루트, 이후-중간 상태 루트, 트랜잭션의 리스트를 반환
  - 이전-중간 상태 루트에 트랜잭션의 리스트를 적용하면 이후-상태 중간 루트가 됨
  - 메시지는 트랜잭션이거나 중간 상태 루트(trace)
  - $parsePeriod((m_0, m_1, \ldots, m_t)) \in \{(trace_i^x, trace_i^{x+1}, (t_i^g, t_i^{g+1}, \ldots, t_i^{g+h})), err\}$

# PROOF OF INVALID STATE TRANSITION

# PROOF OF INVALID STATE TRANSITION

- 악의적 혹은 결함이 있는 채굴자
  - 올바르지 않은 $stateRoot_i$를 제출할 수 있음
- 수행 자취(execution trace)를 사용해 일부가 유효하지 않음을 보일 수 있음
- 풀노드로부터 수신된 사기 증명을 검증하는 함수
  - $VerifyTransitionFraudProof$ 와 그 매개변수를 정의할 것

# PROOF OF INVALID STATE TRANSITION

- 사기 증명은
  - 올바르지 않은 상태 전이를 포함한 블록에서의 관계 있는 셰어들
  - 그 셰어들의 머클 증명
  - 그 셰어들에 포함된 트랜잭션들의 상태 증거로 구성

# PROOF OF INVALID STATE TRANSITION

- 사기 증명을 입력으로 받아
  - 블록 데이터의 기간 안의 트랜잭션들을 중간 이전-상태 루트에 적용
  - 그 결과가 블록 데이터에 명시된 중간 이후-상태 루트로 나오는지를 검사
- 만일 그렇지 않다면, 사기 증명이 유효한 것
  - 블록은 클라이언트에 의해 영구히 거절(reject)됨

# PROOF OF INVALID STATE TRANSITION

$$\text{VerifyTransitionFraudProof}\left(\begin{array}{c}\text{blockHash}_i,\\ \left(d_i^y, d_i^{y+1}, \ldots, d_i^{y+m}\right),\\ y,\\ \left(\left\{d_i^y \rightarrow \text{dataRoot}_i\right\}, \left\{d_i^{y+1} \rightarrow \text{dataRoot}_i\right\}, \ldots, \left\{d_i^{y+m} \rightarrow \text{dataRoot}_i\right\}\right),\\ \left(w_i^y, w_i^{y+1}, \ldots, w_i^{y+m}\right)\end{array}\right) \in \{\text{true}, \text{false}\}$$

- $d_i^j$ : 블록 $i$의 $j$번째 셰어
- 단순화를 위해 자취에서의 상태 증거가
  모든 개별 중간 상태 루트에 제공되는 모델을 가정

# PROOF OF INVALID STATE TRANSITION

- $VerifyTransitionFraudProof$ 는 다음 조건이 충족되면 참, 아니면 거짓을 반환
  - $blockHash_i$는 클라이언트가 다운로드해 저장한 블록 헤더 $h_i$에 상응
  - 증명에서의 각 셰어 $d_i^{y+a}$에 대해 다음이 참을 반환
    $VerifyMerkleProof(d_i^{y+a}, \{d_i^{y+a} \rightarrow dataRoot_i\}, dataRoot_i, dataLength_i, y+a)$
  - $parsePeriod(parseShares(d_i^y, d_i^{y+1}, \ldots, d_i^{y+m}))$
    $\in \{(trace_i^x, trace_i^{x+1}, (t_i^g, t_i^{g+1}, \ldots, t_i^{g+h})), err\}$ 의 결과는 $err$가 아니어야 함
  - $trace_i^x$에 대해 $(t_i^g, t_i^{g+1}, \ldots, t_i^{g+h})$를 적용해 $trace_i^{x+1}$이 나와야 함

# FRAUD PROOFS

## MAXIMISING LIGHT CLIENT SECURITY AND SCALING BLOCKCHAINS WITH DISHONEST MAJORITIES