

솔라나(Solana)

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.

솔라나: 고성능 블록체인을 위한 새로운 아키텍처 v0.8.14

Anatoly Yakovenko
anatoly@solana.io

법적 고지 본 백서의 어떠한 내용도 토큰 판매 제안 또는 구매 제안 권유를 구성하지 않습니다. 솔라나는 본 백서를 오로지 대중의 피드백과 의견을 수렴하기 위해 게시합니다. 솔라나가 향후 토큰(또는 미래 토큰에 대한 단순 계약)을 판매할 경우, 공개 문서 및 위험 요소를 포함한 확정적인 발행 문서를 통해 진행할 예정입니다. 해당 확정 문서에는 본 백서의 업데이트된 버전이 포함될 것으로 예상되며, 이는 현재 버전과 상당한 차이가 있을 수 있습니다. 솔라나가 미국 내에서 이러한 발행을 진행할 경우, 해당 발행은 공인 투자자에게만 제공될 가능성이 높습니다.

본 백서의 어떠한 내용도 솔라나 사업 또는 토큰의 발전 방향, 토큰의 유용성 또는 가치에 대한 보증이나 약속으로 간주되거나 해석되어서는 안 됩니다. 본 백서는 현재 계획을 개괄한 것으로, 솔라나의 재량에 따라 변경될 수 있으며, 그 성공 여부는 시장 기반 요인 및 데이터암호화폐 산업 내 요인 등 솔라나의 통제 범위를 벗어난 다양한 요인에 좌우될 것입니다. 미래 사건에 관한 모든 진술은 본 백서에 기술된 사안에 대한 솔라나의 분석에 전적으로 근거합니다. 해당 분석은 틀릴 수 있습니다.

초록

본 논문은 이벤트 간 순서와 시간 경과를 검증하는 증명인 역사 증명(Proof of History, PoH)을 기반으로 한 새로운 블록체인 아키텍처를 제안한다. PoH는 신뢰 없는 시간 경과를 부록 전용 데이터 구조인 원장(ledger)에 인코딩하는 데 사용된다. 작업 증명(PoW)이나 지분 증명(PoS)과 같은 합의 알고리즘과 함께 사용될 때, PoH는 비잔틴 장애 허용 복제 상태 기계에서 메시징 오버헤드를 줄여 1초 미만의 최종성 시간을

달성할 수 있다. 본 논문은 또한 PoH 원장의 시간 유지 특성을 활용하는 두 가지 알고리즘을 제안합니다. 이는 모든 규모의 분할에서 복구 가능한 PoS 알고리즘과 효율적인 스트리밍 복제 증명(PoRep)입니다. PoRep과 PoH의 조합은 시간(순서) 및 저장과 관련하여 원장의 위조에 대한 방어 기능을 제공합니다. 이 프로토콜은 1Gbps 네트워크에서 분석되었으며, 본 논문은 현재 하드웨어로 초당 최대 710k 트랜잭션의 처리량이 가능함을 보여줍니다.

1 소개

블록체인은 내결함성 복제 상태 기계의 구현체입니다. 현재 공개적으로 이용 가능한 블록체인은 시간에 의존하지 않거나, 참여자의 시간 유지 능력에 대해 약한 가정을 합니다[4, 5]. 네트워크 내 각 노드는 일반적으로 네트워크 내 다른 참여자의 시계에 대한 정보 없이 자체 로컬 시계에 의존합니다. 신뢰할 수 있는 시간 기준의 부재는 메시지 타임스탬프를 사용하여 메시지를 수락하거나 거부할 때 네트워크 내 다른 모든 참여자가 정확히 동일한 선택을 할 것이라는 보장이 없음을 의미합니다. 본 문서에서 제시하는 PoH(Proof of History)는 검증 가능한 시간 경과, 즉 이벤트 간 지속 시간과 메시지 순서를 기록하는 원장을 생성하도록 설계되었습니다. 네트워크 내 모든 노드가 신뢰 없이 원장에 기록된 시간 경과를 의존할 수 있을 것으로 예상됩니다.

2 개요

본 문서의 나머지 부분은 다음과 같이 구성됩니다. 전체 시스템 설계는 3장에서 설명합니다. 역사 증명(Proof of History)에 대한 심층 설명은 4장에서 다룹니다. 제안된 지분 증명(Proof of Stake) 합의 알고리즘에 대한 심층 설명은 5장에서 설명합니다. 제안된 빠른 복제 증명(Proof of Replication)에 대한 심층 설명은 6장에서 설명합니다. 시스템 아키텍처와 성능 한계는 7장에서 분석합니다. 고성능 GPU 친화적 스마트 계약 엔진은 7.5장에서 설명합니다.

3 네트워크 설계

그림 1에서 볼 수 있듯이, 특정 시점에 시스템 노드 중 하나가 리더(Leader)로 지정되어 역사 증명(Proof of History) 시퀀스를 생성함으로써 네트워크에 전역 읽기 일관성과 검증 가능한 시간 경과를 제공합니다. 리더는 사용자 메시지를 시퀀싱하고 순서를 정렬하여 시스템 내 다른 노드에서 효율적으로 처리될 수 있도록 하여 처리량을 극대화합니다. 리더는 RAM에 저장된 현재 상태에서 트랜잭션을 실행하고, 트랜잭션과 최종 상태의 서명을 검증자(Verifier)라고 불리는 복제 노드에 게시합니다. 검증자는 자신의 상태 복사본에서 동일한 트랜잭션을 실행하고, 계산된 상태 서명을 확인(confirmation)으로 게시합니다. 게시된 확인은 합의 알고리즘을 위한 투표 역할을 합니다.

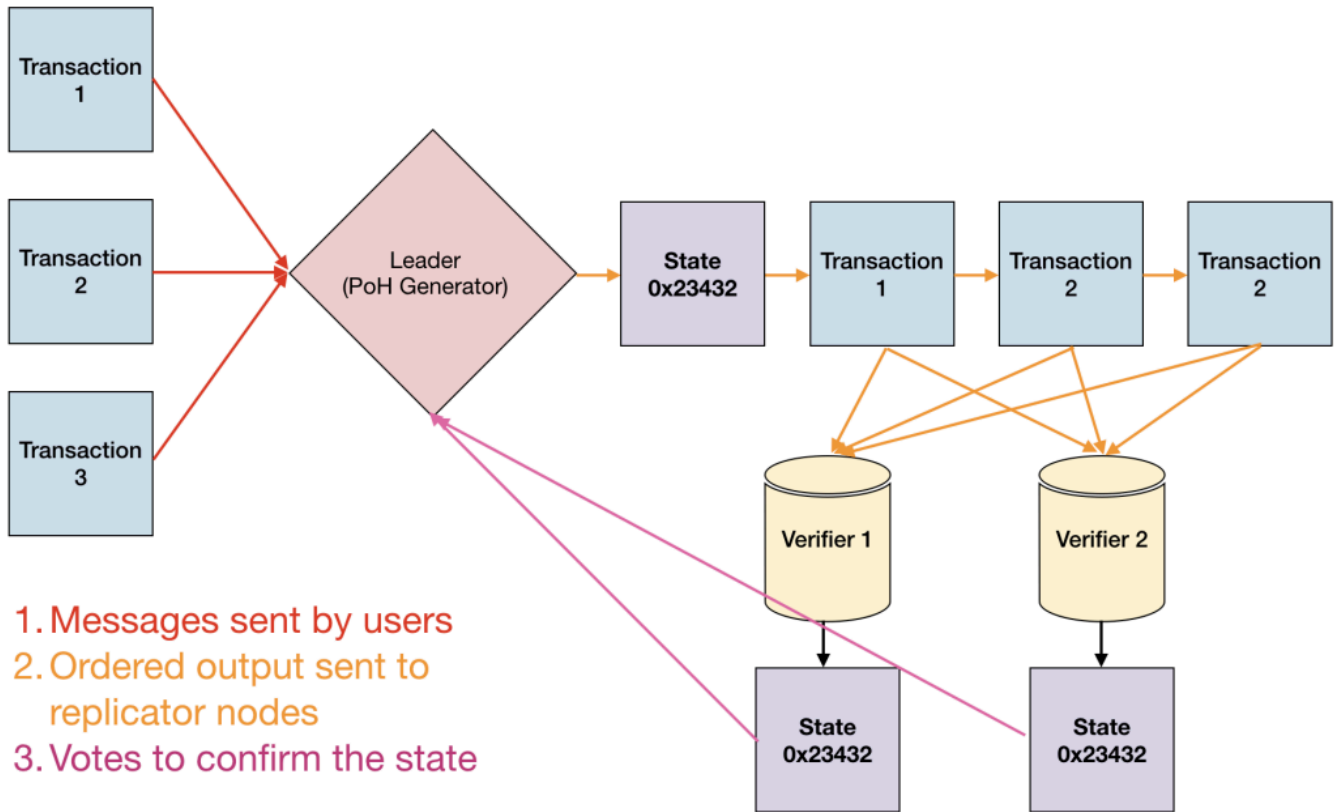


Figure 1: Transaction flow throughout the network.

비분할 상태에서는 언제든지 네트워크에 하나의 리더가 존재합니다. 각 검증자 노드는 리더와 동일한 하드웨어 성능을 가지며, PoS 기반 선거를 통해 리더로 선출될 수 있습니다. 제안된 PoS 알고리즘의 선거 방식은 5.6절에서 상세히 다루고 있습니다. CAP 정리에 따르면, 분할 발생 시 가용성보다 일관성이 거의 항상 우선 선택됩니다. 대규모 분할 발생 시 본 논문은 모든 규모의 분할로부터 네트워크 통제권을 회복하는 메커니즘을 제안합니다. 이는 5.12절에서 상세히 설명됩니다.

4 역사 증명(Proof of History)

역사 증명은 두 사건 사이의 시간 경과를 암호학적으로 검증할 수 있는 계산 순서입니다. 입력으로부터 출력을 예측할 수 없도록 작성된 암호학적으로 안전한 함수를 사용하며, 출력을 생성하기 위해 반드시 완전히 실행되어야 합니다. 이 함수는 단일 코어에서 순차적으로 실행되며, 이전 출력을 현재 입력으로 사용합니다. 현재 출력과 호출 횟수를 주기적으로 기록합니다. 이후 외부 컴퓨터들은 각 시퀀스 세그먼트를 별도의 코어에서 병렬로 재계산하여 검증할 수 있습니다. 데이터(또는 일부 데이터의 해시)를 함수 상태에 추가함으로써 이 시퀀스에 타임스탬프를 찍을 수 있습니다. 시퀀스에 추가된 상태, 인덱스 및 데이터의 기록은 시퀀스에서 다음 해시가 생성되기 전에 데이터가 생성되었음을 보장할 수 있는 타임스탬프를 제공합니다. 이 설계는 또한 여러 생성기가 서로의 상태를 서로의 시퀀스에 혼합하여 동기화할 수 있으므로 수평적 확장을 지원합니다. 수평적 확장은 4.4절에서 자세히 논의됩니다.

4.1 설명

본 시스템은 다음과 같이 작동하도록 설계되었습니다. 함수를 실행하지 않고는 출력을 예측할 수 없는 암호화 해시 함수(예: sha256, ripemd 등)를 사용하여, 임의의 시작 값으로부터 함수를 실행하고 그 출력을 다시 동일한 함수의 입력으로 전달합니다. 함수가 호출된 횟수와 각 호출 시의 출력을 기록합니다. 선택된 시작 무작위 값은 당일 뉴욕 타임즈의 헤드라인과 같은 임의의 문자열일 수 있습니다.

예를 들어:

PoH Sequence		
Index	Operation	Output Hash
1	sha256("any random starting value")	hash1
2	sha256(hash1)	hash2
3	sha256(hash2)	hash3

여기서 hashN은 실제 해시 출력을 나타냅니다.

일정 간격으로 해시와 인덱스의 일부만 공개하면 됩니다.

예를 들어:

PoH Sequence		
Index	Operation	Output Hash
1	sha256("any random starting value")	hash1
200	sha256(hash199)	hash200
300	sha256(hash299)	hash300

선택된 해시 함수가 충돌 저항성을 갖는 한, 이 해시 집합은 단일 컴퓨터 스레드에 의해 순차적으로만 계산될 수 있습니다. 이는 시작값부터 300번 알고리즘을 실행하지 않고서는 인덱스 300의 해시 값이 무엇일지 예측할 방법이 없기 때문입니다. 따라서 데이터 구조로부터 인덱스 0과 인덱스 300 사이에는 실제 시간이 경과했음을 추론할 수 있습니다. 그림 2의 예시에서 해시 62f51643c1은 카운트 510144806912에서 생성되었고, 해시 c43d862d88은 카운트 510146904064에서 생성되었습니다. 앞서 논의한 PoH 알고리즘의 특성에 따라, 카운트 510144806912와 카운트 510146904064 사이에는 실제 시간이 경과했다고 신뢰할 수 있습니다.

4.2 이벤트 타임스탬프

이 해시 시퀀스는 특정 해시 인덱스가 생성되기 전에 어떤 데이터가 생성되었음을 기록하는 데에도 사용될 수 있습니다. ‘결합(combine)’ 함수를 사용하여 해당 데이터를 현재 인덱스의 현재 해시와 결합합니다. 데이터는 단순히 임의의 이벤트 데이터에 대한 암호학적으로 고유한 해시일 수 있습니다. 결합 함수는 단순한 데이터 추가나 충돌 저항성을 가진 어떤 연산이 될 수 있습니다. 이후 생성된 해시는 해당 데이터의 타임스탬프를 나타내는데, 이는 특정 데이터가 삽입된 후에야 생성될 수 있기 때문입니다.

예를 들어:

PoH Sequence		
Index	Operation	Output Hash
1	sha256("any random starting value")	hash1
200	sha256(hash199)	hash200
300	sha256(hash299)	hash300

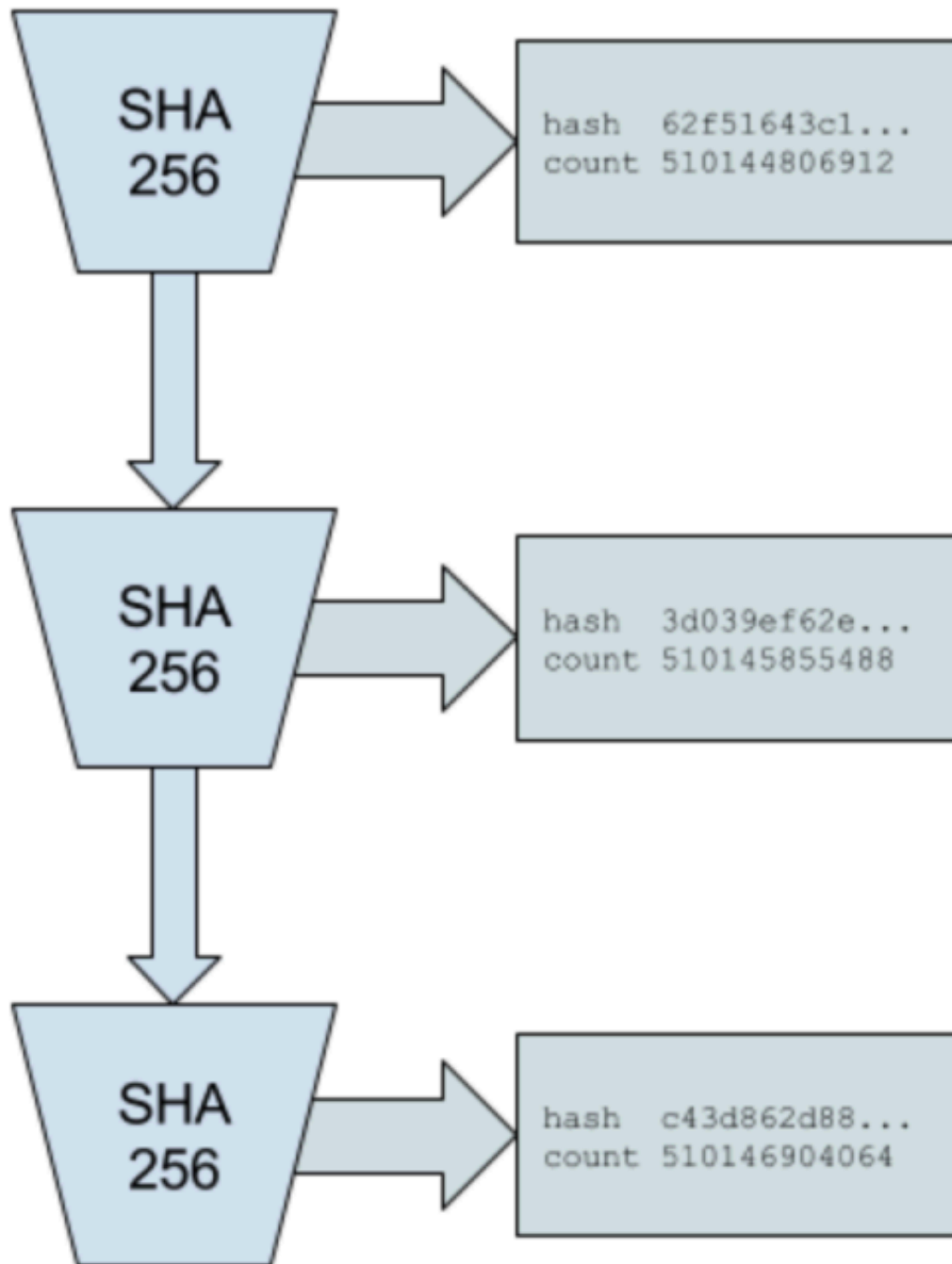


Figure 2: Proof of History sequence

어떤 외부 사건이 발생합니다. 예를 들어 사진이 촬영되거나 임의의 디지털 데이터가 생성되는 경우:

PoH Sequence With Data

Index	Operation	Output Hash
1	sha256("any random starting value")	hash1
200	sha256(hash199)	hash200
300	sha256(hash299)	hash300
336	sha256(append(hash335, photograph_sha256))	hash336

Hash336은 hash335에 추가된 이전 데이터와 사진의 sha256 해시값으로부터 계산됩니다. 인덱스와 사진의 sha256 해시값은 시퀀스 출력의 일부로 기록됩니다. 따라서 이 시퀀스를 검증하는 사람은 누구나 시퀀스에 대한 이 변경 사항을 재현할 수 있습니다. 검증은 여전히 병렬로 수행될 수 있으며, 이는 4.3절에서 논의됩니다.

POH Sequence

Index	Operation	Output Hash
1	sha256("any random starting value")	hash1
200	sha256(hash199)	hash200
300	sha256(hash299)	hash300
336	sha256(append(hash335, photograph1_sha256))	hash336
400	sha256(hash399)	hash400
500	sha256(hash499)	hash500
600	sha256(append(hash599, photograph2_sha256))	hash600
700	sha256(hash699)	hash700

Table 1: PoH Sequence With 2 Events

초기 프로세스가 여전히 순차적이므로, 시퀀스에 입력된 항목들은 미래 해시 값이 계산되기 이전에 발생했음을 알 수 있습니다.

표 1이 나타내는 순서에서 photograph2는 hash600보다 먼저 생성되었으며, photograph1은 hash336보다 먼저 생성되었습니다. 데이터를 해시 순서에 삽입하면 순서 내 모든 후속 값이 변경됩니다. 사용된 해시 함수가 충돌 저항성을 가지며 데이터가 추가된 경우, 순서에 통합될 데이터에 대한 사전 지식을 바탕으로 향후 순서를 미리 계산하는 것은 계산상 불가능해야 합니다.

시퀀스에 혼합되는 데이터는 원시 데이터 자체일 수도 있고, 메타데이터와 함께 제공되는 데이터의 해시값일 수도 있습니다.

그림 3의 예시에서 입력값 cfd40df8...이 역사 증명(Proof of History) 시퀀스에 삽입되었습니다. 삽입된 카운트는 510145855488이며, 삽입된 상태는 3d039eef3입니다. 이 시퀀스의 변경 사항은 향후 생성되는 모든 해시에 반영되며, 그림에서 색상 변화로 표시됩니다.

이 시퀀스를 관측하는 모든 노드는 모든 이벤트가 삽입된 순서를 파악하고 삽입 간 실제 시간을 추정할 수 있습니다.

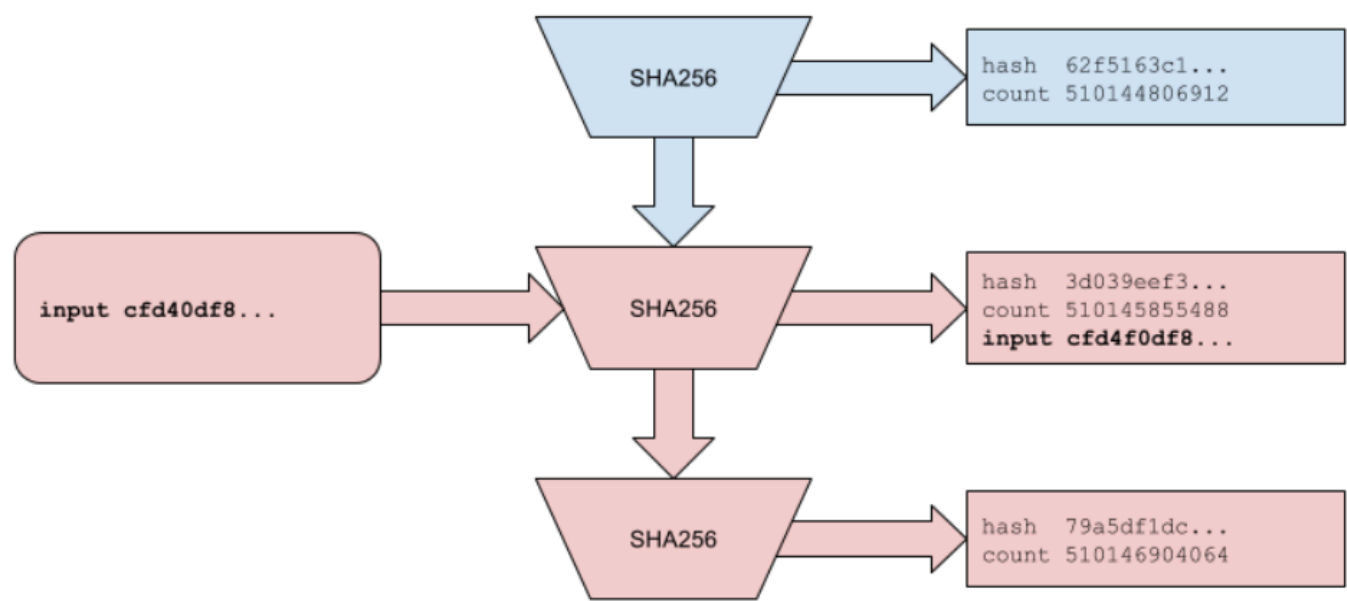


Figure 3: Inserting data into Proof of History

4.3 검증

해당 시퀀스는 생성하는 데 소요된 시간보다 훨씬 짧은 시간 내에 멀티코어 컴퓨터로 정확성을 검증할 수 있습니다.

예를 들어:

Core 1		
Index	Data	Output Hash
200	sha256(hash199)	hash200
300	sha256(hash299)	hash300
Core 2		
Index	Data	Output Hash
300	sha256(hash299)	hash300
400	sha256(hash399)	hash400

코어 수가 주어지면, 예를 들어 4000개의 코어를 가진 현대 GPU의 경우, 검증기는 해시 시퀀스와 해당 인덱스를 4000개의 슬라이스로 분할할 수 있으며, 각 슬라이스 내에서 시작 해시부터 마지막 해시까지의 정확성을 병렬로 확인할 수 있습니다. 시퀀스 생성에 소요될 것으로 예상되는 시간이 다음과 같을 경우:

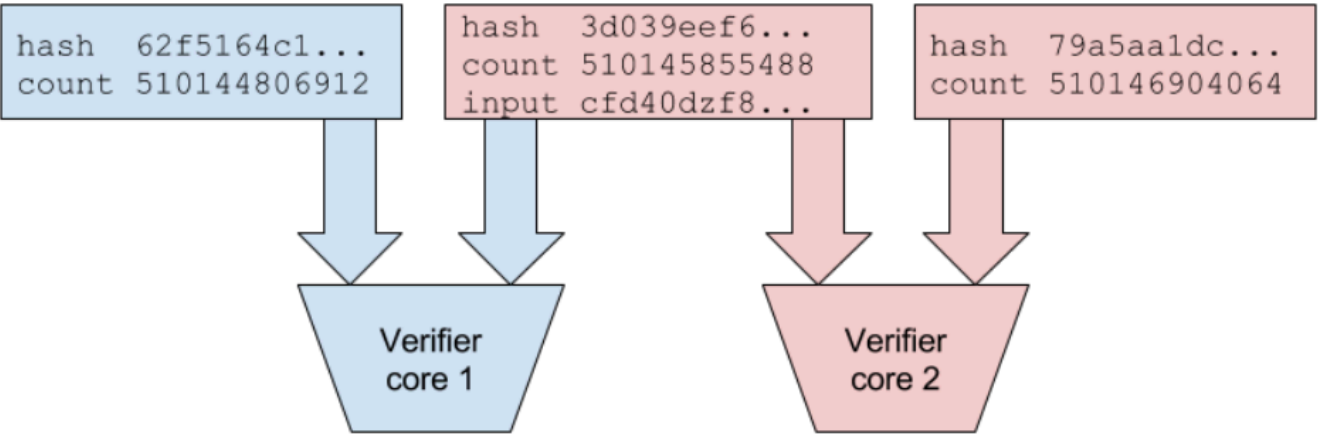


Figure 4: Verification using multiple cores

$$\frac{\text{Total number of hashes}}{\text{Hashes per second for 1 core}} \tag{1}$$

다중 코어를 이용한 병렬 검증 시 예상 소요 시간은 다음과 같습니다:

$$\frac{\text{Total number of hashes}}{\text{Hashes per second per core} \times \text{Number of cores available to verify}} \tag{2}$$

그림 4의 예시에서 각 코어는 시퀀스의 각 슬라이스를 병렬로 검증할 수 있습니다. 모든 입력 문자열이 해당 문자열이 추가된 카운터 및 상태와 함께 출력에 기록되므로 검증기는 각 슬라이스를 병렬로 복제할 수 있습니다. 빨간색으로 표시된 해시는 시퀀스가 데이터 삽입으로 수정되었음을 나타냅니다.

4.4 수평 확장

여러 증명 기록 생성기를 동기화하는 것은 각 생성기의 시퀀스 상태를 서로 혼합함으로써 가능하며, 이를 통해 증명 기록 생성기의 수평 확장을 달성할 수 있습니다. 이 확장은 샤딩 없이 이루어집니다. 시스템 내 이벤트의 완전한 순서를 재구성하기 위해서는 두 생성기의 출력이 모두 필요합니다.

PoH Generator A			PoH Generator B		
Index	Hash	Data	Index	Hash	Data
1	hash1a		1	hash1b	
2	hash2a	hash1b	2	hash2b	hash1a
3	hash3a		3	hash3b	
4	hash4a		4	hash4b	

생성기 A와 B가 주어졌을 때, A는 B로부터 데이터 패킷(hash1b)을 수신한다. 이 패킷에는 생성기 B의 마지막 상태와 생성기 A로부터 관측된 생성기 B의 마지막 상태가 포함된다. 생성기 A의 다음 상태 해시는 생성기 B의 상태에 의존하므로, hash1b가 hash3a보다 이전 시점에 발생했음을 추론할 수 있다. 이 속성은 전이적일 수 있으므로, 세 개의 생성기가 단일 공통 생성기 $A \leftrightarrow B \leftrightarrow C$ 를 통해 동기화될 경우, A와 C가 직접 동기화되지 않았더라도 이 둘 사이의 의존성을 추적할 수 있습니다.

발전기를 주기적으로 동기화함으로써 각 발전기는 외부 트래픽의 일부를 처리할 수 있게 되며, 이로 인해 전체 시스템은 발전기 간 네트워크 지연으로 인한 실시간 정확도 저하를 감수하는 대가로 추적해야 할 더 많은 이벤트를 처리할 수 있습니다. 동기화 창 내에 있는 모든 이벤트를 정렬하기 위해 해시 값 자체와 같은 결정론적 함수를 선택함으로써 여전히 전역적 순서를 달성할 수 있습니다.

그림 5에서 두 생성기는 서로의 출력 상태를 삽입하고 해당 작업을 기록합니다. 색상 변화는 상대방의 데이터가 시퀀스를 수정했음을 나타냅니다. 각 스트림에 혼합된 생성된 해시는 굵은 글씨로 강조 표시됩니다.

동기화는 전이적이다. $A \leftrightarrow B \leftrightarrow C$ 를 통해 A와 C 사이에는 증명 가능한 사건의 순서가 존재한다.

이러한 방식으로 확장하면 가용성이 저하됩니다. 가용성 0.999인 10×1 Gbps 연결은 $0.999^{10} = 0.99$ 의 가용성을 가집니다.

4.5 일관성

사용자는 생성된 시퀀스의 일관성을 강제할 수 있어야 하며, 유효하다고 판단하는 시퀀스의 마지막 관측된 출력을 입력에 삽입함으로써 공격에 대한 내성을 확보할 수 있어야 합니다.

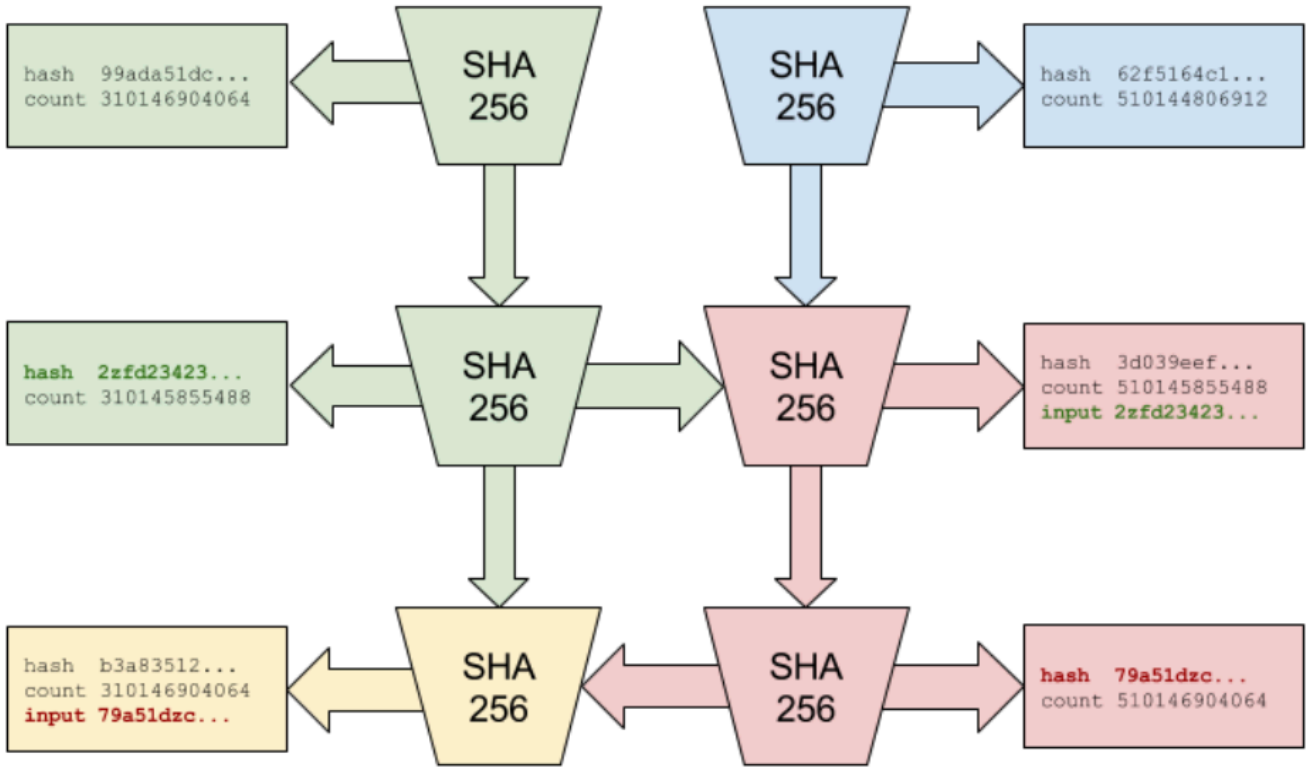


Figure 5: Two generators synchronizing

PoH Sequence A			PoH Hidden Sequence B		
Index	Data	Output Hash	Index	Data	Output Hash
10		hash10a	10		hash10b
20	Event1	hash20a	20	Event3	hash20b
30	Event2	hash30a	30	Event2	hash30b
40	Event3	hash40a	40	Event1	hash40b

악의적인 PoH 생성기는 모든 이벤트에 동시에 접근할 수 있거나 더 빠른 숨겨진 시퀀스를 생성할 수 있다면, 이벤트를 역순으로 배열한 두 번째 숨겨진 시퀀스를 생성할 수 있다.

이러한 공격을 방지하기 위해, 클라이언트가 생성한 각 이벤트에는 클라이언트가 유효한 시퀀스로 간주하는 것에서 관측한 최신 해시가 포함되어야 합니다. 따라서 클라이언트가 “이벤트1” 데이터를 생성할 때, 관측한 마지막 해시를 추가해야 합니다.

PoH Sequence A

Index	Data	Output Hash
10		hash10a
20	Event1 = append(event1 data, hash10a)	hash20a
30	Event2 = append(event2 data, hash20a)	hash30a
40	Event3 = append(event3 data, hash30a)	hash40a

시퀀스가 공개될 때 Event3는 hash30a를 참조하게 되며, 이 이벤트 이전 시퀀스에 해당 해시가 존재하지 않으면 시퀀스 사용자는 시퀀스가 무효임을 인지합니다. 부분 재정렬 공격은 클라이언트가 이벤트를 관측한 시점부터 해당 이벤트가 입력될 때까지 생성된 해시 수로 제한됩니다. 클라이언트는 마지막으로 관측된 해시와 삽입된 해시 사이의 짧은 기간 동안 순서가 정확하다고 가정하지 않는 소프트웨어를 작성할 수 있어야 합니다.

악의적인 PoH 생성자가 클라이언트 이벤트 해시를 재작성하는 것을 방지하기 위해, 클라이언트는 단순히 데이터만 제출하는 대신 이벤트 데이터와 마지막으로 관측된 해시에 대한 서명을 제출할 수 있습니다.

PoH Sequence A

Index	Data	Output Hash
10		hash10a
20	Event1 = sign(append(event1 data, hash10a), Client Private Key)	hash20a
30	Event2 = sign(append(event2 data, hash20a), Client Private Key)	hash30a
40	Event3 = sign(append(event3 data, hash30a), Client Private Key)	hash40a

이 데이터의 검증에는 서명 검증이 필요하며, 이 해시 이전의 해시 시퀀스에서 해당 해시를 조회해야 합니다. 검증:

(Signature, PublicKey, hash30a, event3 data) = Event3 Verify(Signature, PublicKey, Event3) Lookup(hash30a, PoHSequence)

그림 6에서 사용자가 제공한 입력은 생성된 시퀀스에 삽입되기 전에 해시 oxdeadbeef...가 존재해야 합니다. 왼쪽 상단의 파란색 화살표는 클라이언트가 이전에 생성된 해시를 참조하고 있음을 나타냅니다. 클라이언트의 메시지는 해시 oxdeadbeef...를 포함하는 시퀀스에서만 유효합니다. 시퀀스 내의 빨간색은 클라이언트의 데이터로 인해 시퀀스가 수정되었음을 나타냅니다.

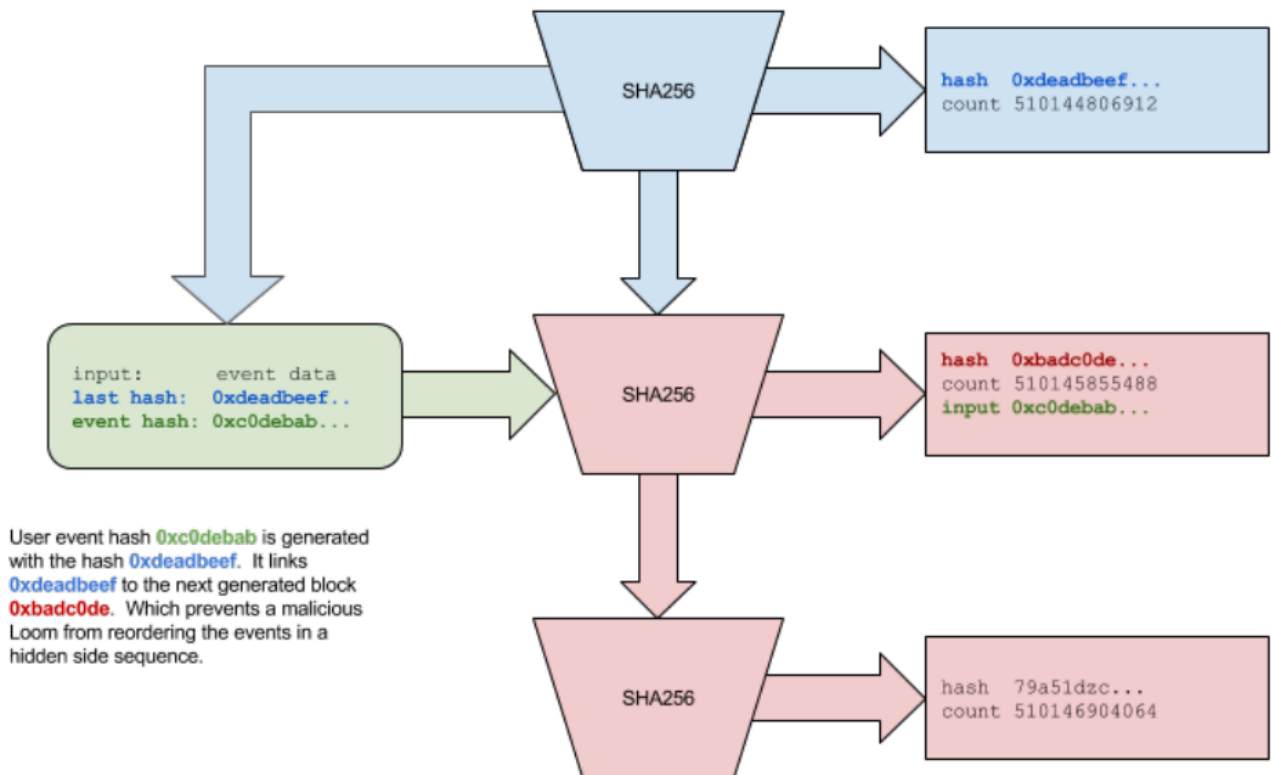


Figure 6: Input with a back reference.

4.6 오버헤드

초당 4000개의 해시는 추가로 160킬로바이트의 데이터를 생성하며, 검증에는 4000개의 코어를 가진 GPU 접근과 약 0.25~0.75밀리초의 시간이 필요합니다.

4.7 공격

4.7.1 역순 생성

역순을 생성하려면 공격자가 두 번째 이벤트 이후에 악의적인 시퀀스를 시작해야 합니다. 이 지연 시간은 악의적이지 않은 피어 투 피어 노드들이 원래 순서에 대해 통신할 수 있도록 허용해야 합니다.

4.7.2 속도

여러 생성기를 사용하면 배포가 공격에 더 강해질 수 있습니다. 한 생성기는 대역폭이 높아 많은 이벤트를 수신하여 시퀀스에 혼합할 수 있고, 다른 생성기는 고속 저대역폭으로 주기적으로 고속 생성기와 혼합할 수 있습니다.

고속 시퀀스는 공격자가 역순으로 만들어야 하는 보조 데이터 시퀀스를 생성합니다.

4.7.3 장거리 공격

장거리 공격은 오래전에 폐기된 클라이언트 개인 키를 획득하고 위조된 원장을 생성하는 것을 포함합니다 [10]. 역사 증명(Proof of History)은 장거리 공격에 대한 일부 보호 기능을 제공합니다. 오래된 개인 키에 접근한 악의적인 사용자는 위조하려는 원본 기록과 동일한 시간만큼 소요되는 역사적 기록을 재구성해야 합니다. 이는 네트워크가 현재 사용하는 것보다 더 빠른 프로세서에 접근해야 가능하며, 그렇지 않으면 공격자는 역사 길이에서 절대 따라잡을 수 없습니다.

또한 단일 시간 기준원은 더 단순한 복제 증명(Proof of Replication) 구축을 가능하게 합니다(자세한 내용은 6절 참조). 네트워크는 모든 참여자가 단일 사건 기록에 의존하도록 설계되었기 때문입니다.

PoRep과 PoH는 위조된 원장에 대해 공간과 시간 양측의 방어 기능을 함께 제공해야 합니다.

5 지분 증명 합의

5.1 설명

이 특정 지분 증명 방식은 역사 증명 생성기가 생성한 현재 시퀀스의 신속한 확인, 다음 역사 증명 생성기의 투표 및 선정, 그리고 잘못된 행동을 하는 검증자에 대한 처벌을 위해 설계되었습니다. 이 알고리즘은 특정 시간 제한 내에 모든 참여 노드에 메시지가 최종적으로 도달하는 것에 의존합니다.

5.2 용어 설명

- 보증금(bonds) : 보증금은 작업 증명(Proof of Work)에서의 자본 지출과 동일합니다. 채굴자는 하드웨어와 전기를 구매하여 작업 증명 블록체인의 단일 분기에 이를 투입합니다. 보증금은 검증자가 거래를 검증하는 동안 담보로 투입하는 코인입니다.
- 슬래싱(slashing) : 지분 증명(Proof of Stake) 시스템에서 '무관심 문제(nothing at stake problem)'를 해결하기 위한 제안된 방안[7]. 다른 분기에 대한 투표 증명이 게시되면 해당 분기는 검증자의 보증금을 소멸시킬 수 있습니다. 이는 검증자가 여러 분기를 확인하는 것을 억제하기 위해 고안된 경제적 인센티브입니다.
- 초다수결(super majority) : 보증금 가중치로 계산된 검증자의 2/3를 의미합니다. 초과 다수결 투표는 네트워크가 합의를 달성했음을 나타내며, 해당 분기가 무효화되려면 네트워크의 최소 1/3이 악의적으로 투표해야 합니다. 이는 공격의 경제적 비용을 해당 코인 시가총액의 1/3로 설정합니다.

5.3 보증금 예치

보증금 예치 거래는 일정량의 코인을 사용자의 신원 하에 보증금 예치 계정으로 이동시킵니다. 보증금 예치 계정의 코인은 사용자가 인출할 때까지 계정에 보관되어야 하며 사용 불가합니다. 사용자는 시간 초과된 오래된 코인(stale coins)만 인출할 수 있습니다. 보증금은 현재 이해관계자의 초다수(super majority)가 시퀀스를 확인한 후 유효해집니다.

5.4 투표

역사 증명 생성자는 미리 정의된 주기에 상태 서명을 게시할 수 있을 것으로 예상됩니다. 각 보증된 신원은 자체 서명된 상태 서명을 게시하여 해당 서명을 확인해야 합니다. 투표는 반대 없이 단순 찬성 투표입니다. 시간 제한 내에 보증된 신원의 과반수가 투표하면 해당 분기가 유효한 것으로 인정됩니다.

5.5 보증 해제

N개 미달 투표는 코인을 유효하지 않은 상태로 표시하며 더 이상 투표 자격이 없습니다. 사용자는 보증 해제 트랜잭션을 발행하여 이를 제거할 수 있습니다. N은 유효 투표 대비 유효하지 않은 투표의 비율에 기반한 동적 값입니다.

유효하지 않은 투표 수가 증가함에 따라 N은 증가합니다. 대규모 네트워크 분할 발생 시, 이는 더 큰 분기가 더 작은 분기보다 빠르게 복구할 수 있도록 합니다.

5.6 선거

새로운 PoH 생성자 선거는 PoH 생성자 장애가 감지될 때 발생합니다. 가장 큰 투표권을 가진 검증자, 또는 동점 시 가장 높은 공개 키 주소를 가진 검증자가 새로운 PoH 생성자로 선정됩니다.

새로운 시퀀스에 대해서는 초다수 확인이 필요합니다. 초다수 확인이 확보되기 전에 새 리더가 실패할 경우, 다음으로 높은 검증자가 선택되며 새로운 확인 세트가 요구됩니다.

투표를 전환하려면 검증자는 더 높은 PoH 시퀀스 카운터에서 투표해야 하며, 새 투표에는 전환하려는 투표가 포함되어야 합니다. 그렇지 않으면 두 번째 투표는 슬래시 대상이 됩니다. 투표 전환은 초다수 확증이 없는 높이에서만 발생하도록 설계될 것으로 예상됩니다.

PoH 생성자가 확립되면, 거래 처리 업무를 인수할 보조 생성자를 선출할 수 있습니다. 보조 생성자가 존재할 경우, 주 생성자 장애 시 차기 리더로 간주됩니다.

플랫폼은 예외 감지 시 또는 사전 정의된 일정에 따라 보조 생성자가 주 생성자로 승격되고 하위 등급 생성자가 승진하도록 설계되었습니다.

5.7 선거 트리거

5.7.1 포크된 역사 증명 생성기

PoH 생성기는 생성된 시퀀스에 서명하는 신원 정보로 설계됩니다. 포크는 PoH 생성기의 신원 정보가 손상된 경우에만 발생할 수 있습니다. 동일한 PoH 신원 정보에 두 개의 서로 다른 역사 기록이 게시되면 포크가 감지됩니다.

5.7.2 런타임 예외

하드웨어 장애, 버그 또는 PoH 생성기의 의도적 오류로 인해 무효한 상태가 생성되고, 로컬 검증자의 결과와 일치하지 않는 상태 서명이 게시될 수 있습니다. 검증자들은 가십을 통해 올바른 서명을 게시하며, 이 사건은 새로운 선거 라운드를 트리거합니다. 무효한 상태를 수락한 검증자는 보증금이 몰수됩니다.

5.7.3 네트워크 시간 초과

네트워크 시간 초과 시 선거가 발동됩니다.

5.8 슬래싱

검증자가 두 개의 서로 다른 시퀀스에 투표할 경우 슬래싱이 발생합니다. 악의적 투표에 대한 증거가 있을 경우, 보증된 코인은 유통에서 제거되어 채굴 풀에 추가됩니다.

경쟁 시퀀스에 대한 이전 투표를 포함하는 투표는 악의적 투표의 증거로 인정되지 않습니다. 보증금을 슬래싱하는 대신, 이 투표는 경쟁 시퀀스에 현재 행사된 투표를 제거합니다.

PoH 생성기가 생성한 무효 해시에 대한 투표가 행사된 경우에도 슬래싱이 발생합니다. 생성기는 무효 상태를 무작위로 생성하도록 설계되어 있으며, 이는 세컨더리로의 복귀를 유발합니다.

5.9 보조 선거

보조 및 하위 등급의 역사 증명 생성기를 제안하고 승인할 수 있습니다. 제안은 주 생성기의 시퀀스에 투표로 제출됩니다. 제안에는 시간 제한이 포함되며, 시간 제한 전에 투표의 과반수 이상이 동의하면 보조 생성기가 당선된 것으로 간주되어 예정된 일정에 따라 업무를 인수합니다. 주 생성자는 생성된 시퀀스에 인계 발생을 알리는 메시지를 삽입하거나, 무효 상태를 삽입하여 네트워크가 보조 생성자로 강제 전환되도록 함으로써 보조 생성자에게 소프트 인계를 수행할 수 있습니다.

보조 생성자가 선출된 상태에서 주 생성자가 실패할 경우, 해당 보조 생성자는 선거 시 첫 번째 대체 후보로 간주됩니다.

5.10 가용성

분할을 다루는 CAP 시스템은 일관성(Consistency)과 가용성(Availability) 중 하나를 선택해야 합니다. 우리의 접근 방식은 궁극적으로 가용성을 선택하지만, 객관적인 시간 측정 기준이 존재하기 때문에 합리적인 인간 시간 제한 내에서 일관성을 선택합니다.

지분 증명 검증자는 특정 거래 집합에 투표할 수 있도록 일정량의 코인을 “지분”으로 잠급니다. 코인 잠금은 다른 거래와 마찬가지로 PoH 스트림에 입력되는 거래입니다. 투표하기 위해 PoS 검증자는 PoH 원장의 특정 위치까지 모든 거래를 처리한 후 계산된 상태의 해시에 서명해야 합니다. 이 투표 역시 거래로 PoH 스트림에 입력됩니다. PoH 원장을 살펴보면 각 투표 사이의 경과 시간과, 분할이 발생했을 경우 각 검증자가 얼마나 오랫동안 접근 불가능했는지 추론할 수 있습니다.

합리적인 인간적 시간 범위 내에서 파티션을 처리하기 위해, 우리는 이용할 수 없는 검증자를 “언스테이킹(unstake)”하는 동적 접근 방식을 제안한다. 검증자 수가 많고 2/3 이상일 때, “언스테이킹” 과정은 빠르게 진행될 수 있다. 이 경우, 이용할 수 없는 검증자의 스테이크가 완전히 언스테이킹되어 더 이상 합의에 포함되지 않기 전까지 원장에 기록되어야 하는 해시의 수는 적다. 검증자 수가 2/3 미만이지만 1/2 이상일 때, 언스테이킹 타이머는 더 느리게 작동하여, 결측된 검증자가 언스테이킹되기 전에 더 많은 수의 해시가 생성되어야 한다. 1/2 이상의 검증자가 결측된 대규모 파티션의 경우, 언스테이킹 과정은 매우 매우 느리게 진행된다. 이 경우에도 트랜잭션은 여전히 스트림에 입력될 수 있고, 검증자들도 여전히 투표할 수 있지만, 매우 많은 수의 해시가 생성되어 이용할 수 없는 검증자가 언스테이킹되기 전까지는 완전한 2/3 합의에 도달할 수 없다. 네트워크가 생동성을 회복하는 데 걸리는 시간 차이는, 네트워크의 고객으로서 우리가 계속 사용하고자 하는 파티션을 인간적인 시간 범위 내에서 선

5.11 복구

우리가 제안하는 시스템에서 원장은 어떤 실패로부터도 완전히 복구될 수 있다. 이는 전 세계 누구라도 원장에서 임의의 지점을 선택하여 새로 생성된 해시와 트랜잭션을 덧붙임으로써 유효한 포크를 만들 수 있음을 의미한다. 만약 이 포크에서 모든 검증자가 결측된 경우, 추가적인 채권(bond)이 유효해지고 이 분기가 2/3 초과 다수 합의에 도달하기까지는 매우 매우 오랜 시간이 걸릴 것이다. 따라서 사용 가능한 검증자가 전혀 없는 상태에서의 완전한 복구는 원장에 매우 많은 수의 해시를 추가해야만 가능하며, 모든 이용할 수 없는 검증자가 언스테이킹된 이후에야 새로운 채권이 원장을 검증할 수 있게 될 것이다.

5.12 파이널리티

PoH는 네트워크의 검증자들이 과거에 일어난 사건들의 시간을 일정 수준의 확실성을 가지고 관찰할 수 있도록 한다. PoH 생성기가 메시지 스트림을 생성하는 동안, 모든 검증자는 500ms 이내에 상태에 대한 서명을 제출해야 한다. 이 수치는 네트워크 조건에 따라 더 줄어들 수 있다. 각 검증은 스트림에 기록되므로, 네트워크의 모든 참여자는 실제 투표 과정을 직접 관찰하지 않고도 모든 검증자가 요구된 타임아웃 내에 투표를 제출했음을 검증할 수 있다.

5.13 공격

5.13.1 공유지의 비극

PoS 검증자들은 단순히 PoH 생성기가 생성한 상태 해시를 확인한다. 그들에게는 아무런 작업을 수행하지 않고 단순히 생성된 모든 상태 해시를 승인할 경제적 유인이 존재한다. 이 상황을 피하기 위해, PoH 생성기는 무작위 간격으로 잘못된 해시를 삽입해야 한다. 이 해시에 투표하는 모든 검증자는 슬래싱(slashed)되어야 한다. 해시가 생성되면, 네트워크는 즉시 보조로 선출된 PoH 생성기를 승격시켜야 한다. 각 검증자는 짧은 타임아웃 내에 응답해야 하는데, 예를 들어 500ms가 될 수 있다. 이 타임아웃은 악의적인 검증자가 다른 검증자의 투표를 관찰하고 자신의 투표를 스트림에 충분히 빠르게 반영할 확률이 낮아지도록 충분히 낮게 설정되

어야 한다.

5.13.2 PoH 생성기와와의 담합

PoH 생성기와 담합한 검증자는 잘못된 해시가 언제 생성될지를 사전에 알 수 있으며, 그 해시에 투표하지 않을 것이다. 이 시나리오는 사실상 PoH ID가 더 큰 검증자 스테이크를 보유한 것과 다르지 않다. PoH 생성기는 여전히 상태 해시를 생성하기 위해 모든 작업을 수행해야 한다.

5.13.3 검열

검열 또는 서비스 거부는 전체 채권자(bond holders)의 1/3이 새로운 채권이 포함된 어떤 시퀀스도 검증하기를 거부할 때 발생할 수 있다. 이러한 형태의 공격에 대해 프로토콜은 채권이 오래되어 무효(stale)화되는 속도를 동적으로 조정함으로써 방어할 수 있다. 서비스 거부 발생하면, 더 큰 파티션이 포크를 형성하여 비잔틴 채권자를 검열하도록 설계된다. 비잔틴 채권은 시간이 지남에 따라 무효화되므로, 더 큰 네트워크는 회복할 수 있다. 작은 비잔틴 파티션은 더 오랜 기간 동안 전진하지 못한다.

알고리즘은 다음과 같이 작동한다. 네트워크의 다수는 새로운 리더를 선출한다. 리더는 이후 비잔틴 채권자가 참여하지 못하도록 검열한다. 역사 증명(Proof of History) 생성기는 시간이 흐름을 증명하기 위해 시퀀스를 계속 생성해야 하며, 충분한 수의 비잔틴 채권이 무효화되어 더 큰 네트워크가 초과 다수(super majority)를 가지게 될 때까지 이를 수행해야 한다. 채권이 무효화되는 속도는 활성화된 채권의 비율에 따라 동적으로 결정된다. 따라서 네트워크의 비잔틴 소수 포크는 다수 포크가 초과 다수를 회복하기 위해 기다려야 하는 것보다 훨씬 더 오래 기다려야 한다. 일단 초과 다수가 확립되면, 슬래싱(slaing)을 사용하여 비잔틴 채권자를 영구적으로 처벌할 수 있다.

5.13.4 롱레인지 공격

PoH는 롱레인지 공격에 대해 자연스러운 방어를 제공한다. 과거의 어느 시점에서든 원장을 복구하려면, 공격자는 PoH 생성기의 속도를 능가하여 유효한 원장을 시간적으로 추월해야 한다. 합의 프로토콜은 두 번째 방어 계층을 제공하는데, 어떤 공격도 모든 유효한 검증자의 언스테이킹에 걸리는 시간보다 더 오래 지속되어야 하기 때문이다. 또한 이는 원장 이력에 가용성 “간극(gap)”을 만든다. 동일한 높이의 두 원장을 비교할 때, 최대 파티션 크기가 더 작은 원장이 객관적으로 유효한 것으로 간주될 수 있다.

5.13.5 ASIC 공격

이 프로토콜에는 ASIC 공격의 기회가 두 가지 존재한다 — 파티션 동안과 파이널리티에서의 타임아웃 속임수이다. 파티션 동안의 ASIC 공격의 경우, 채권(bond)이 언스테이킹되는 속도는 비선형적이며, 대규모 파티션이 있는 네트워크에서는 이 속도가 ASIC 공격에서 예상되는 이익보다 몇 자릿수 이상 더 느리다. 파이널리티 동안의 ASIC 공격의 경우, 취약점은 채권을 보유한 비잔틴 검증자들이 다른 노드들의 확인을 기다린 뒤, 협력하는 PoH 생성기와 함께 자신의 투표를 주입할 수 있도록 허용한다. 그 후 PoH 생성기는 더 빠른 ASIC을 사용하여 500ms 분량의 해시를 더 짧은 시간에 생성하고, PoH 생성기와 협력 노드 간의 네트워크 통신을 가능하게 할 수 있다. 그러나 PoH 생성기 역시 비잔틴이라면, 그 비잔틴 생성기가 실패를 삽입하려는 정확한 시점의 카운터를 협력자들에게 전달하지 않을 이유가 없다. 이 시나리오는 PoH 생성기와 모든 협력자가 동일한 ID를 공유하는 경우와, 단일 결합된 스테이크를 가지고 단 하나의 하드웨어 세트를 사용하

는 경우와 다르지 않다.

6 스트리밍 복제 증명

6.1 설명

Filecoin은 복제 증명의 한 버전을 제안하였다 [6]. 이 버전의 목표는 복제 증명을 빠르고 스트리밍 방식으로 검증할 수 있도록 하는 것이며, 이는 역사 증명(Proof of History)으로 생성된 시퀀스에서 시간을 추적함으로써 가능해진다. 복제는 합의 알고리즘으로 사용되지는 않지만, 블록체인 이력이나 상태를 높은 가용성으로 저장하는 비용을 산정하는 데 유용한 도구이다.

6.2 알고리즘

그림 7에서 보듯이 CBC 암호화는 순차적으로 데이터 블록을 암호화하며, 이전에 암호화된 블록을 사용하여 입력 데이터와 XOR 연산을 수행합니다.

각 복제자 신원은 증명 기록(Proof of History) 시퀀스를 통해 생성된 해시에 서명하여 키를 생성합니다. 이는 키를 복제자의 신원과 특정 증명 기록 시퀀스에 연결합니다. 특정 해시만 선택될 수 있습니다. (해시 선택에 관한 내용은 6.5절 참조)

데이터 세트는 블록 단위로 완전히 암호화됩니다. 그런 다음 증명을 생성하기 위해 키를 사용하여 의사 난수 생성기의 시드를 설정하고, 각 블록에서 무작위 32바이트를 선택합니다.

선택된 PoH 해시를 각 슬라이스 앞에 추가하여 머클 해시를 계산합니다.

루트 해시, 생성된 키, 선택된 해시가 함께 공개됩니다. 복제 노드는 Proof of History 생성기에 의해 생성되는 N개의 해시에 대해 추가 증명을 공개해야 하며, 여기서 N은 데이터 암호화 소요 시간의 약 1/2입니다.

Proof of History 생성기는 사전 정의된 주기로 Proof of Replication용 특정 해시를 공개합니다. 복제 노드는 증명을 생성하기 위해 다음 공개된 해시를 선택해야 합니다. 다시 해시에 서명하고, 블록에서 무작위 슬라이스를 선택하여 머클 루트를 생성합니다.

N개의 증명 주기가 지나면 데이터는 새로운 CBC 키로 재암호화됩니다.

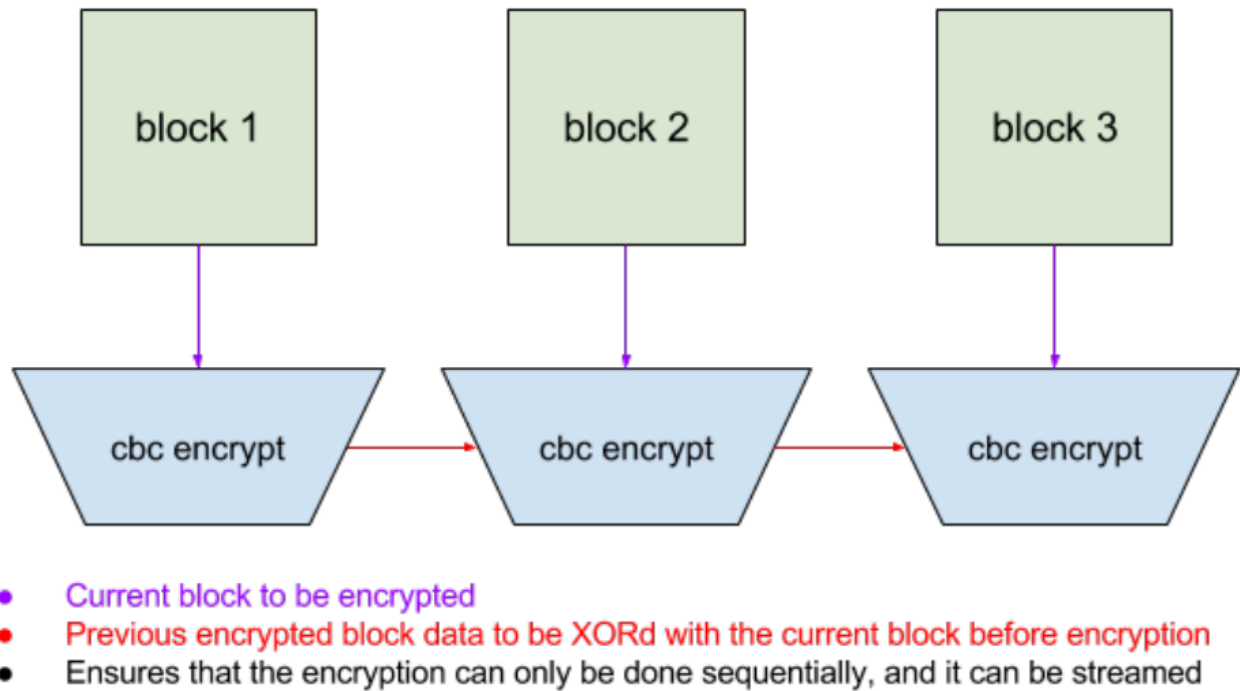


Figure 7: Sequential CBC encryption

6.3 검증

N개의 코어가 있으면, 각 코어는 각 개별 ID에 대해 암호화를 스트리밍할 수 있다. 필요한 총 공간은 $2 \times \text{blocks} \times N_{\text{cores}}$ 이며, 이는 이전에 암호화된 블록이 다음 블록을 생성하는 데 필요하기 때문이다. 각 코어는 이후 현재 암호화된 블록에서 파생된 모든 증명을 생성하는 데 사용할 수 있다. 증명을 검증하는 총 시간은 암호화에 걸리는 시간과 동일할 것으로 예상된다. 증명 자체는 블록에서 몇 바이트의 무작위 데이터를 소비하므로, 해시해야 하는 데이터의 양은 암호화된 블록 크기보다 훨씬 적다. 동시에 검증할 수 있는 복제 ID(replication identities)의 수는 사용 가능한 코어의 수와 같다. 최신 GPU는 3500개 이상의 코어를 제공하지만, CPU 클럭 속도의 $1/2 \sim 1/3$ 수준에서 동작한다.

- random bytes selected from the block
- Proof of History hash

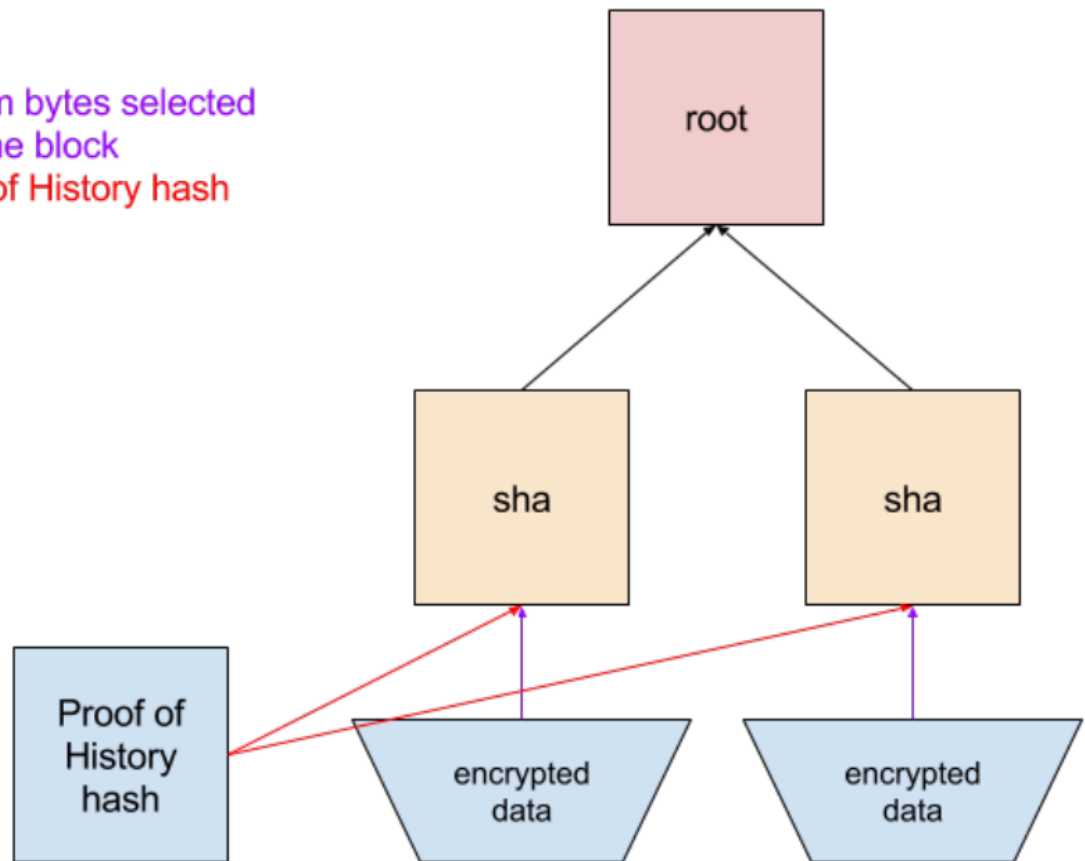


Figure 8: Fast Proof of Replication

6.4 키 로테이션

키 로테이션이 없으면 동일한 암호화된 복제본이 여러 증명 기록 시퀀스에 대해 저렴한 증명을 생성할 수 있습니다. 키는 주기적으로 로테이션되며, 각 복제본은 고유한 증명 기록 시퀀스에 연결된 새 키로 재암호화됩니다. 로테이션 속도는 GPU 하드웨어에서 복제본 증명을 검증하는 것이 실용적일 정도로 충분히 느려야 합니다. GPU는 코어당 처리 속도가 CPU보다 느리기 때문입니다.

6.5 해시 선택

역사 증명 생성기는 복제 증명 암호화에 사용되고, 빠른 증명에서 바이트 선택을 위한 의사 난수 생성기로 활용될 해시를 전체 네트워크에 공개합니다.

해시는 데이터 세트 암호화 소요 시간의 약 1/2에 해당하는 주기 카운터에서 공개됩니다. 각 복제 식별자는 동일한 해시를 사용해야 하며, 해시의 서명 결과를 바이트 선택 시드 또는 암호화 키로 사용해야 합니다. 각 복제자가 증명을 제공해야 하는 주기는 암호화 시간보다 짧아야 합니다. 그렇지 않으면 복제자가 암호화 과정을 스트리밍하고 각 증명마다 이를 삭제할 수 있습니다.

악의적인 생성자는 특정 해시를 생성하기 위해 이 해시 이전 시퀀스에 데이터를 삽입할 수 있습니다. 이 공격은 5.13.2절에서 더 자세히 논의됩니다.

6.6 증명 검증

역사 증명 노드는 제출된 복제 증명(Proof of Replication)을 검증할 필요가 없습니다. 복제자 신원이 제출한 대기 중 및 검증 완료 증명의 수를 추적하기만 하면 됩니다. 증거는 복제자가 네트워크 내 검증자의 초다수(super majority)에 의해 서명할 수 있을 때 검증된 것으로 간주됩니다.

검증 결과는 복제자가 P2P 가십 네트워크를 통해 수집하며, 네트워크 내 검증자의 초다수를 포함하는 하나의 패킷으로 제출됩니다. 이 패킷은 증명 시퀀스에 의해 생성된 특정 해시 이전의 모든 증명을 검증하며, 한번에 여러 복제자 신원을 포함할 수 있습니다.

6.7 공격

6.7.1 스팸

악의적인 사용자가 다수의 복제자 신원을 생성하여 잘못된 증명으로 네트워크에 스팸을 유포할 수 있습니다. 더 빠른 검증을 위해 노드는 검증 요청 시 암호화된 데이터와 전체 머클 트리를 네트워크에 제공해야 합니다. 본 논문에서 설계된 복제 증명(Proof of Replication)은 추가 공간을 차지하지 않으므로 모든 추가 증명을 저렴하게 검증할 수 있습니다. 그러나 각 신원(identity)은 1개의 암호화 코어 시간을 소모합니다. 복제 대상은 즉시 사용 가능한 코어의 최대 크기로 설정해야 합니다. 현대 GPU는 3500개 이상의 코어를 탑재하고 있습니다.

6.7.2 부분 삭제

복제 노드는 전체 상태 저장을 피하기 위해 일부 데이터를 부분적으로 삭제할 수 있습니다. 증명 수와 시드의 무작위성은 이러한 공격을 어렵게 만들어야 합니다.

증명 샘플링 충돌 가능성

예를 들어, 1테라바이트 데이터를 저장하는 사용자가 1메가바이트 블록마다 단일 바이트를 삭제한다고 가정합니다. 매 메가바이트마다 1바이트를 샘플링하는 단일 증명은 삭제된 바이트와 충돌할 확률이 다음과 같습니다:

$$1 - \left(1 - \frac{1}{1,000,000}\right)^{1,000,000} = 0.63 \quad (3)$$

5번의 증명 후 확률은 0.99입니다.

6.7.3 PoH 생성자와의 공모

서명된 해시는 샘플 시드에 사용될 것으로 예상됩니다. 복제자가 사전에 특정 해시를 선택할 수 있다면, 복제자는 샘플링되지 않을 모든 바이트를 삭제할 수 있습니다.

Proof of History 생성자와 공모하는 복제자 신원은 임의 바이트 선택을 위한 사전 정의된 해시가 생성되기 전 시퀀스 끝에 특정 트랜잭션을 삽입할 수 있습니다. 충분한 코어를 보유한 공격자는 복제자 신원보다 유

리한 해시를 생성할 수 있습니다.

이 공격은 단일 복제자 신원에게만 이익이 됩니다. 모든 신원이 ECDSA(또는 동등한 방식)로 암호화 서명된 동일한 해시를 사용해야 하므로, 결과 서명은 각 복제자 신원마다 고유하며 충돌 저항성을 가집니다. 단일 복제자 신원은 미미한 이득만 얻을 수 있습니다.

6.7.4 서비스 거부 공격

추가 복제자 신원 추가 비용은 저장 비용과 동일할 것으로 예상됩니다. 모든 복제자 신원을 검증하기 위한 추가 계산 능력 추가 비용은 복제 신원당 CPU 또는 GPU 코어 비용과 동일할 것으로 예상됩니다.

이는 다수의 유효한 복제자 신원을 생성함으로써 네트워크에 대한 서비스 거부 공격의 기회를 창출합니다. 이러한 공격을 제한하기 위해 네트워크에 선택된 합의 프로토콜은 복제 대상을 선정하고, 네트워크 가용성, 대역폭, 지리적 위치 등과 같은 원하는 특성을 충족하는 복제 증명을 보상할 수 있습니다.

6.7.5 공유지의 비극

PoS 검증자는 아무 작업도 하지 않고 PoRep을 단순히 확인할 수 있습니다. PoS 검증자가 작업을 수행하도록 경제적 인센티브를 조정해야 합니다. 즉, 채굴 보상을 PoS 검증자와 PoRep 복제 노드 간에 분배하는 방식입니다.

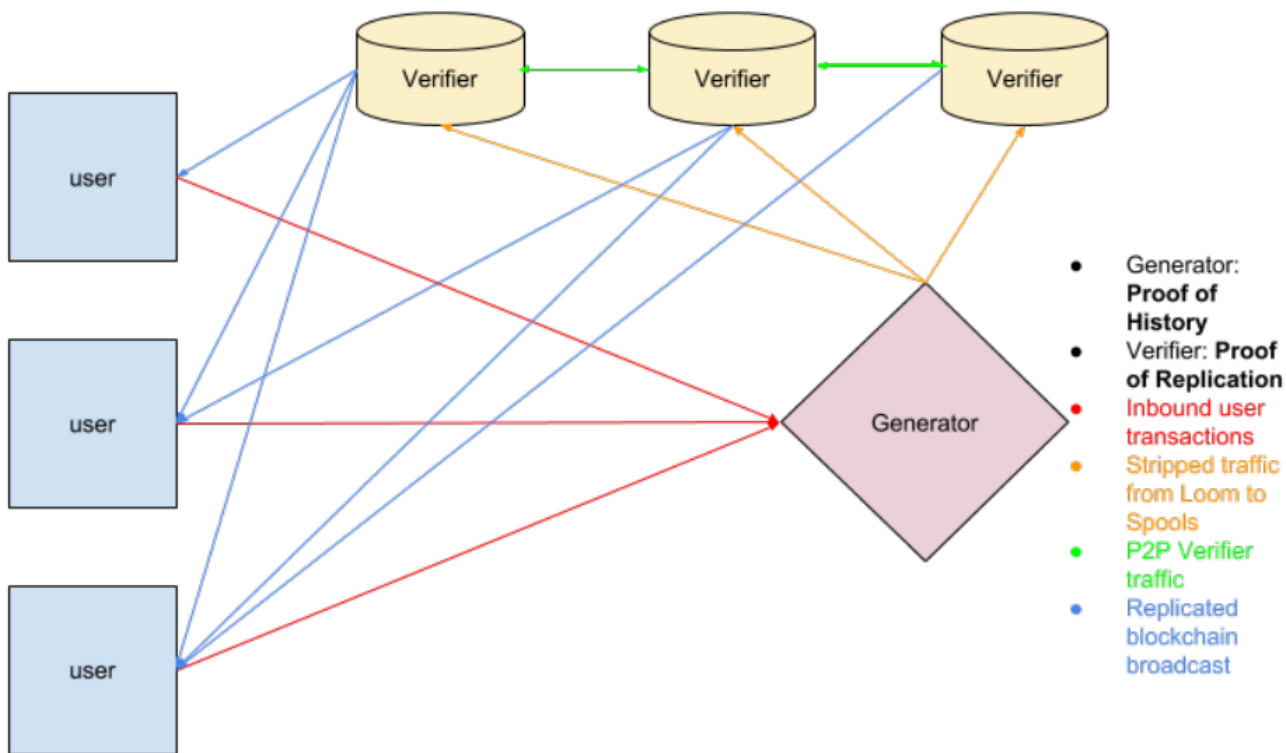


Figure 9: System Architecture

이러한 상황을 더욱 방지하기 위해 PoRep 검증자들은 소량의 비율로 허위 증명을 제출할 수 있습니다. 그들은 허위 데이터를 생성한 함수를 제공함으로써 해당 증명이 허위임을 증명할 수 있습니다. 허위 증명을 확인한 PoS 검증자는 슬래시 처분을 받게 됩니다.

7 시스템 아키텍처

7.1 구성 요소

7.1.1 리더, 역사 증명 생성자

리더는 선출된 역사 증명 생성자입니다. 리더는 임의의 사용자 트랜잭션을 처리하여 시스템 내에서 유일한 전역 순서를 보장하는 모든 트랜잭션의 역사 증명 시퀀스를 출력합니다. 각 트랜잭션 배치 처리 후 리더는 해당 순서로 트랜잭션을 실행한 결과인 상태에 대한 서명을 출력합니다. 이 서명은 리더의 신원 정보로 서명됩니다.

7.1.2 상태

상태는 사용자 주소로 인덱싱된 단순 해시 테이블입니다. 각 셀에는 전체 사용자 주소와 이 계산에 필요한 메모리가 포함됩니다. 예를 들어, 트랜잭션 테이블에는 다음이 포함됩니다:

0	31	63	95	127	159	191	223	255
Ripemd of Users Public Key						Account		unused

총 32바이트입니다.

지분 증명 채권의 테이블에는 다음이 포함됩니다:

0	31	63	95	127	159	191	223	255
Ripemd of Users Public Key						Bond		
Last Vote								
unused								

총 64바이트입니다.

7.1.3 검증자, 상태 복제

검증자 노드는 블록체인 상태를 복제하고 고가용성을 제공합니다. 복제 대상은 합의 알고리즘에 의해 선택되며, 합의 알고리즘 내 검증자들은 오프체인에서 정의된 기준에 따라 승인하는 복제 증명 노드를 선출하고 투표합니다.

네트워크는 최소 지분 증명 보증금 규모와 보증금당 단일 복제자 신원 요구 사항으로 구성될 수 있습니다.

7.1.4 검증자

이 노드들은 검증자로부터 대역폭을 소비합니다. 이들은 가상 노드이며, 검증자나 리더와 동일한 머신에서 실행되거나, 해당 네트워크에 구성된 합의 알고리즘 전용 별도 머신에서 실행될 수 있습니다.

7.2 네트워크 한계

리더는 수신 사용자 패킷을 받아 가능한 가장 효율적인 방식으로 정렬하고, 이를 하류 검증자에게 게시되는 증명 역사 시퀀스로 순서화할 수 있어야 합니다. 효율성은 트랜잭션의 메모리 접근 패턴에 기반하므로, 트랜잭션은 오류 발생을 최소화하고 프리페칭을 극대화하도록 정렬됩니다.

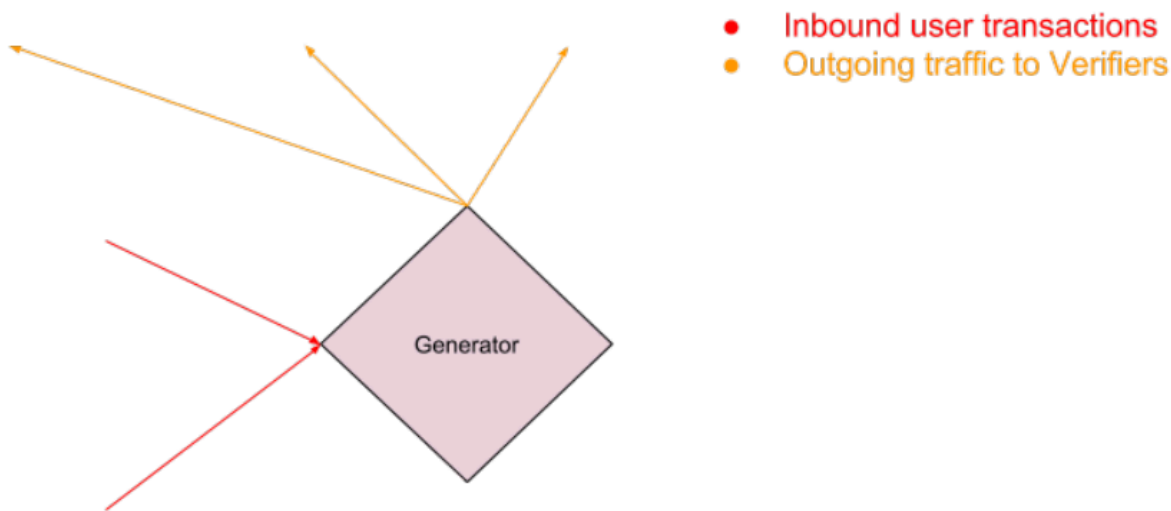
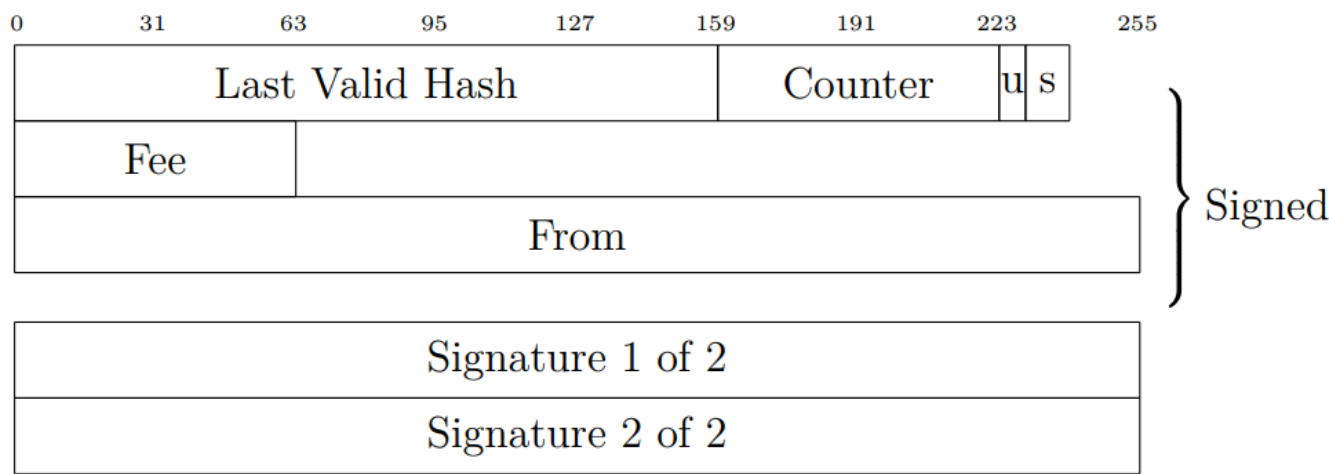


Figure 10: Generator network limits

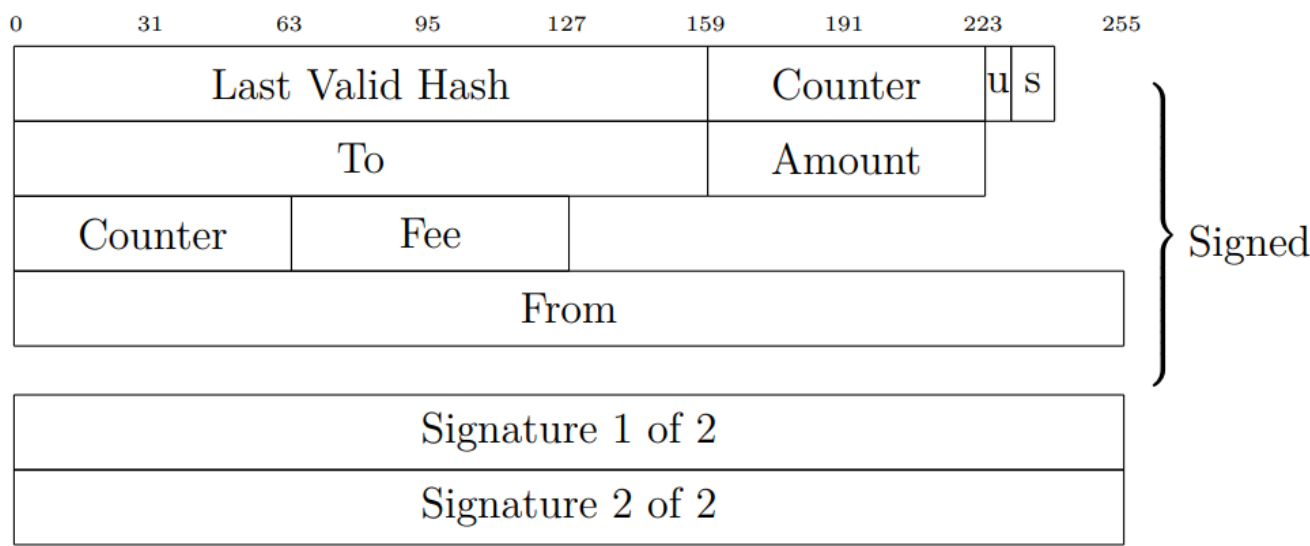
Incoming packet format:



Size $20 + 8 + 16 + 8 + 32 + 32 + 32 = 148$ bytes.

지원 가능한 최소 페이로드는 최소 크기 176바이트의 대상 계정 1개입니다.

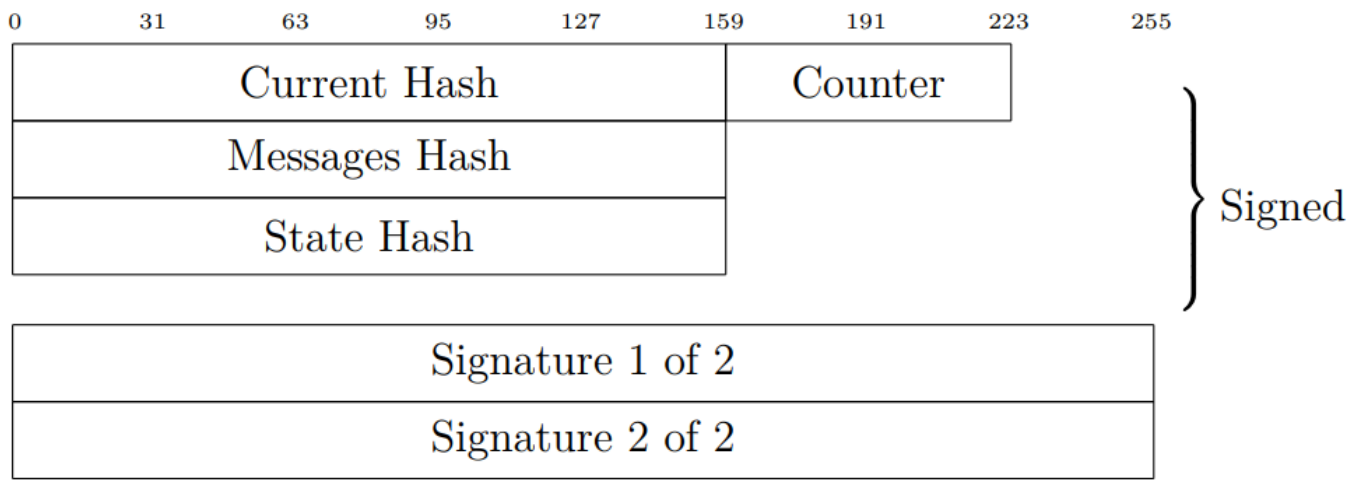
페이로드가 있는 경우:



페이로드 최소 크기: 176 바이트

Proof of History 시퀀스 패킷은 현재 해시, 카운터, PoH 시퀀스에 추가된 모든 새 메시지의 해시, 그리고 모든 메시지 처리 후의 상태 서명을 포함합니다. 이 패킷은 N개의 메시지가 브로드캐스트될 때마다 한 번 전송됩니다.

Proof of History 패킷:



출력 패킷의 최소 크기는: 132바이트

1Gbps 네트워크 연결에서 가능한 최대 트랜잭션 수는 1초당 1기가비트 / 176바이트 = 최대 710k TPS입니다. 이더넷 프레임으로 인해 약간의 손실(1~4%)이 예상됩니다. 네트워크의 목표 용량을 초과하는 여유 용량은 리드-솔로몬 코드로 출력을 코딩하고 사용 가능한 다운스트림 검증자에게 스트라이핑하여 가용성을 높이는 데 활용할 수 있습니다.

7.3 계산적 한계

각 거래는 다이제스트 검증을 필요로 합니다. 이 작업은 거래 메시지 자체 외부의 메모리를 전혀 사용하지 않으며 독립적으로 병렬화될 수 있습니다. 따라서 처리량은 시스템에서 사용 가능한 코어 수에 의해 제한될 것으로 예상됩니다.

GPU 기반 ECDSA 검증 서버는 초당 90만 건의 연산 속도를 실험적으로 달성한 바 있습니다[9].

7.4 메모리 한계

각 계정에 32바이트 항목을 할당하는 50% 채움 해시테이블로 상태를 단순 구현할 경우, 이론적으로 640GB에 100억 개의 계정을 수용할 수 있습니다. 이 테이블에 대한 정상 상태 무작위 접근은 초당 1.1×10^7 회의 쓰기 또는 읽기로 측정됩니다. 트랜잭션당 2회의 읽기와 2회의 쓰기를 기준으로 할 때, 메모리 처리량은 초당 275만 건의 트랜잭션을 처리할 수 있습니다. 이는 Amazon Web Services의 1TB x1.16xlarge 인스턴스에서 측정된 결과입니다.

7.5 고성능 스마트 계약

스마트 계약은 거래의 일반화된 형태입니다. 이는 각 노드에서 실행되어 상태를 수정하는 프로그램입니다. 이 설계는 빠르고 분석이 용이한 확장 버클리 패킷 필터 바이트코드와 JIT 바이트코드를 스마트 계약 언어로 활용합니다.

주요 장점 중 하나는 제로 비용의 외부 함수 인터페이스(FFI)입니다. 플랫폼에 직접 구현된 내장 함수(intrinsics)는 프로그램에서 호출 가능합니다. 내장 함수 호출 시 해당 프로그램은 일시 중지되고, 내장 함수는 고성능 서버에서 스케줄링됩니다. 내장 함수는 GPU에서 병렬 실행되도록 배치 처리됩니다. 위 예시에서 두 개의 서로 다른 사용자 프로그램이 동일한 내장 함수를 호출합니다. 각 프로그램은 내장 함수의 배치 실행이 완료될 때까지 일시 중지됩니다. 내장 함수의 예로는 ECDSA 검증(전자 서명 검증)이 있습니다. 이러한 호출을 GPU에서 배치 처리하여 실행하면 처리량이 수천 배 증가할 수 있습니다.

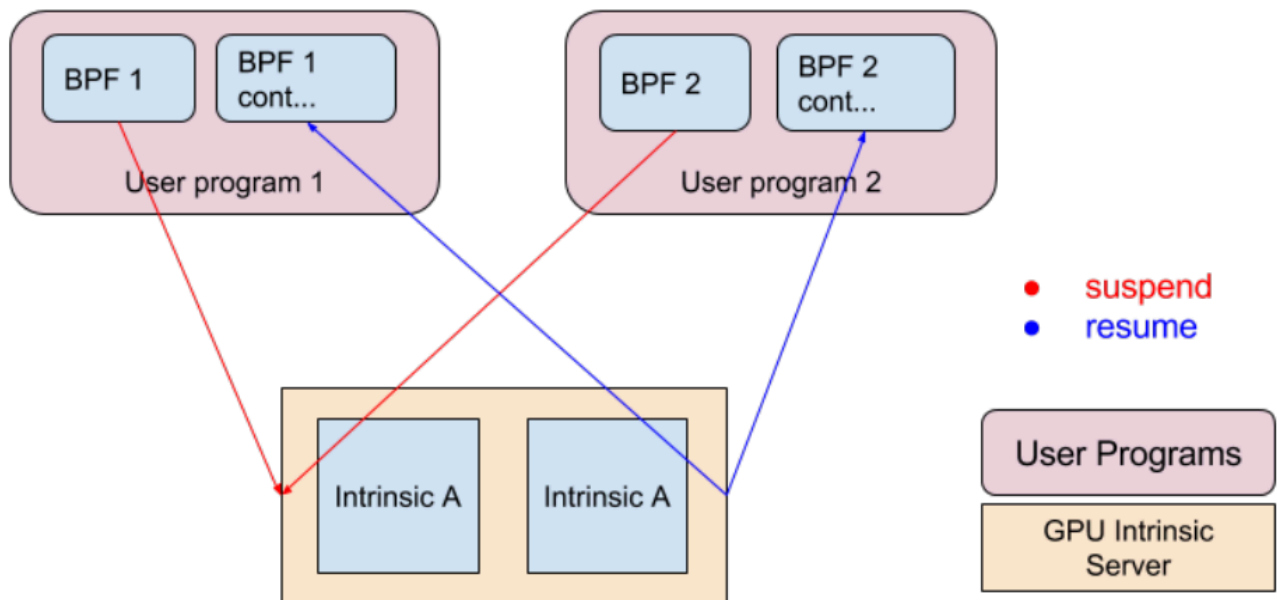


Figure 11: Executing BPF programs.

이 트램폴린은 BPF 바이트코드가 사용하는 모든 메모리에 대해 명확히 정의된 컨텍스트를 가지므로, 네이티브 운영체제 스레드 컨텍스트 전환이 필요하지 않습니다.

eBPF 백엔드는 2015년부터 LLVM에 포함되어 있으므로, 모든 LLVM 프론트엔드 언어로 스마트 계약을 작성할 수 있습니다. 2015년부터 리눅스 커널에 포함되었으며, 바이트코드의 초기 버전은 1992년부터 존재해 왔습니다. 단일 패스로 eBPF의 정확성을 검증하고, 런타임 및 메모리 요구 사항을 확인하며, 이를 x86 명령어로 변환할 수 있습니다.

References

- [1] Liskov, Practical use of Clocks
<http://www.dainf.cefetpr.br/~tacla/SDII/PracticalUseOfClocks.pdf>
- [2] Google Spanner TrueTime consistency
<https://cloud.google.com/spanner/docs/true-time-external-consistency>
- [3] Solving Agreement with Ordering Oracles
<http://www.inf.usi.ch/faculty/pedone/Paper/2002/2002EDCCb.pdf>
- [4] Tendermint: Consensus without Mining
<https://tendermint.com/static/docs/tendermint.pdf>
- [5] Hedera: A Governing Council & Public Hashgraph Network
<https://s3.amazonaws.com/hedera-hashgraph/hh-whitepaperv1.0-180313.pdf>
- [6] Filecoin, proof of replication,
<https://filecoin.io/proof-of-replication.pdf>
- [7] Slasher, A punitive Proof of Stake algorithm

<https://blog.ethereum.org/2014/01/15/slasher-a-punitiveproof-of-stake-algorithm/>

[8] BitShares Delegated Proof of Stake

<https://github.com/BitShares/bitshares/wiki/Delegated-Proofof-Stake>

[9] An Efficient Elliptic Curve Cryptography Signature Server With GPU Acceleration

<http://ieeexplore.ieee.org/document/7555336/>

[10] Casper the Friendly Finality Gadget

<https://arxiv.org/pdf/1710.09437.pdf>