솔라나

백서 번역본

Solana: A new architecture for a high performance blockchain v0.8.13

Anatoly Yakovenko
anatoly@solana.io

본 번역본은 가상자산 투자자들을 위한 참고용입니다. 본 번역본은 투자 권유를 목적으로 만들어지지 않았으며 원문을 단순히 번역한 것으로, 내용의 정확성은 원문 작성 주체에 달려 있으며 크립토닷컴 코리아는 투자 결과에 대하여 어떠한 책임도 지지 않습니다. 본 번역본의 내용에 대한 자세한 문의는 백서 원문을 참고하시길 바랍니다.

개요

이 논문은 사건 간의 순서와 시간의 흐름을 검증하기 위한 증명인 Proof of History (PoH)를 기반으로 한 새로운 블록체인 아키텍처를 제안합니다. PoH는 시간의 흐름을 신뢰 없이 장부에 인코딩하는 데 사용되며, 장부는 추가 전용 데이터 구조입니다. PoH는 Proof of Work (PoW) 또는 Proof of Stake (PoS)와 같은 합의 알고리즘과 함께 사용될 때, 비잔틴 결함 허용 복제 상태 기계에서 메시징 오버헤드를 줄일 수 있으며, 이로 인해 초 단위의 최종성 시간(sub-second finality times)을 실현할 수 있습니다. 이 논문은 또한 PoH 장부의 시간 기록 속성을 활용하는 두 가지 알고리즘을 제안합니다. 첫 번째는 어떤 크기의 파티션에서도 복구할 수 있는 PoS 알고리즘이고, 두 번째는 효율적인 스트리밍 Proof of Replication (PoRep)입니다. PoRep과 PoH의 조합은 시간(순서)과 저장소에 대한 장부 위조를 방어하는 데 도움을 줍니다. 이 프로토콜은 1 gbps 네트워크에서 분석되었으며, 이 논문은 최신 하드웨어를 사용하여 초당 최대 710,000 거래의 처리량이 가능하다는 것을 보여줍니다.

1. 소개

블록체인은 결함 허용 복제 상태 기계의 구현입니다. 현재 공개적으로 이용 가능한 블록체인들은 시간에 의존하지 않거나 참가자들이 시간을 유지할 수 있는 능력에 대해 약한 가정을 합니다 [4, 5]. 네트워크의 각 노드는 일반적으로 다른 참가자들의 시계를 알지 못한 채 자신의 로컬 시계에

의존합니다. 신뢰할 수 있는 시간 소스의 부재는 메시지 타임스탬프가 메시지를 수락하거나 거부하는 데 사용될 때, 네트워크의 다른 모든 참가자들이 정확히 같은 선택을 할 것이라는 보장을 의미하지 않습니다. 여기서 제시된 PoH(Proof of History)는 검증 가능한 시간의 흐름을 생성하도록 설계되었습니다. 즉, 사건 간의 지속 시간과 메시지 순서를 기록하는 장부를 생성합니다. 네트워크의 모든 노드가 신뢰 없이 장부에 기록된 시간의 흐름을 신뢰할 수 있을 것으로 예상됩니다.

2. 아웃라인

이 문서의 나머지 부분은 다음과 같이 구성되어 있습니다. 전체 시스템 설계는 3장에서 설명됩니다. Proof of History의 자세한 설명은 4장에서 다룹니다. 제안된 Proof of Stake 합의 알고리즘의 자세한 설명은 5장에서 설명됩니다. 제안된 빠른 Proof of Replication의 자세한 설명은 6장에서 다루어집니다. 시스템 아키텍처와 성능 한계는 7장에서 분석됩니다. 고성능 GPU 친화적인 스마트 계약 엔진은 7.5장에서 설명됩니다.

3 네트워크 설계

그림 1에서 보듯이, 주어진 시점에서 시스템 노드는 Proof of History 시퀀스를 생성하기 위해 리더(Leader)로 지정됩니다. 이는 네트워크의 전역 읽기 일관성을 제공하고 검증 가능한 시간 경과를 보장합니다. 리더는 사용자 메시지를 시퀀싱하고 정렬하여 시스템의 다른 노드들이 효율적으로 처리할 수 있도록 하여 처리량을 극대화합니다. 리더는 현재 RAM에 저장된 상태에서 트랜잭션을 실행하고, 트랜잭션과 최종 상태의 서명을 복제 노드인 검증자(Verifiers)에게 게시합니다. 검증자들은 자신의 상태 복사본에서 동일한 트랜잭션을 실행하고, 계산된 상태 서명을 확인으로 게시합니다. 게시된 확인서는 합의 알고리즘에 대한 투표 역할을 합니다.

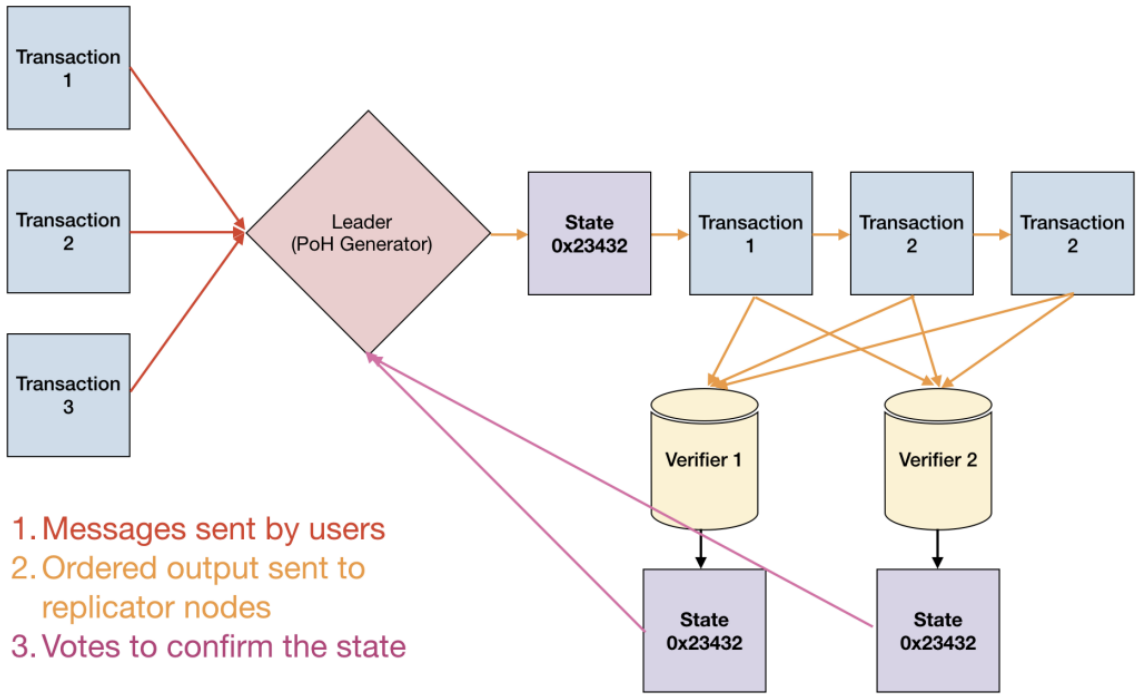


Figure 1: Transaction flow throughout the network.

비분할 상태에서는 주어진 시점에 네트워크에 하나의 리더(Leader)만 존재합니다. 각 검증자(Verifier) 노드는 리더와 동일한 하드웨어 성능을 가지고 있으며, PoS 기반 선거를 통해 리더로 선출될 수 있습니다. 제안된 PoS 알고리즘에 대한 선거 절차는 5.6절에서 자세히 설명됩니다.

CAP 정리에 관해서는, 분할 상황에서 일관성(Consistency)이 가용성(Availability)보다 거의 항상 우선시됩니다. 대규모 분할의 경우, 본 논문은 분할의 크기와 상관없이 네트워크의 제어를 복구하는 메커니즘을 제안합니다. 이 내용은 5.12절에서 자세히 다루어집니다.

4. 역사 증명 (Proof of History)

역사 증명(Proof of History, PoH)은 두 사건 간의 시간 경과를 암호학적으로 검증할 수 있는 계산의 연속입니다. 이 방식은 출력이 입력으로부터 예측될 수 없도록 설계된 암호학적으로 안전한 함수를 사용하며, 출력을 생성하기 위해 함수가 완전히 실행되어야 합니다.

함수는 단일 코어에서 순차적으로 실행되며, 이전 출력을 현재 입력으로 사용하고, 주기적으로 현재 출력을 기록하며, 호출된 횟수를 기록합니다. 이렇게 생성된 출력은 외부 컴퓨터가 각 시퀀스 세그먼트를 별도의 코어에서 확인하여 병렬로 재계산하고 검증할 수 있습니다.

데이터는 함수의 상태에 데이터를 추가하거나 데이터의 해시를 추가하여 이 시퀀스에 타임스탬프를 찍을 수 있습니다. 상태, 인덱스, 데이터가 시퀀스에 추가된 그대로 기록되면, 데이터가 시퀀스의 다음 해시가 생성되기 전에 생성되었음을 보장하는 타임스탬프를 제공합니다. 이 설계는 또한 수평 확장을 지원하며, 여러 생성기가 서로의 상태를 혼합하여 시퀀스를 동기화할 수 있습니다. 수평 확장에 대한 논의는 4.4절에서 자세히 설명됩니다.

4.1 설명

시스템은 다음과 같이 작동하도록 설계되었습니다. 예를 들어, sha256, ripemd 등과 같이 출력이 함수를 실행하지 않고는 예측할 수 없는 암호학적 해시 함수를 사용합니다. 이 함수를 무작위 시작 값으로 실행하고, 그 출력을 다시 같은 함수의 입력으로 전달합니다. 함수가 호출된 횟수와 각 호출 시의 출력을 기록합니다. 시작 값으로 선택된 무작위 값은 뉴욕 타임즈의 헤드라인과 같은 임의의 문자열이 될 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 `hashN`은 실제 해시 출력을 나타냅니다. 해시와 인덱스의 하위 집합만을 주기적으로 공개하는 것이 필요합니다.

예를 들어:

PoH Sequence

Index	Operation	Output Hash
1	sha256(”any random starting value”)	**hash1**
200	sha256(**hash199**)	**hash200**
300	sha256(**hash299**)	**hash300**

선택한 해시 함수가 충돌 저항성(collision resistance)을 가진다면, 이 해시 집합은 단일 컴퓨터 스레드에서 순차적으로만 계산할 수 있습니다. 이는 해시 값을 예를 들어 인덱스 300에서의 해시 값을 예측할 방법이 실제로 알고리즘을 300번 실행하지 않고는 알 수 없다는 사실에 기반합니다. 따라서 데이터 구조를 통해 인덱스 0과 인덱스 300 사이에 실제 시간이 경과했음을 추론할 수 있습니다.

그림 2의 예에서, 해시 `62f51643c1`은 카운트 510144806912에서 생성되었고, 해시 `c43d862d88`은 카운트 510146904064에서 생성되었습니다. 앞서 논의된 PoH 알고리즘의 속성을 따르면, 우리는 카운트 510144806912와 카운트 510146904064 사이에 실제 시간이 경과했음을 신뢰할 수 있습니다.

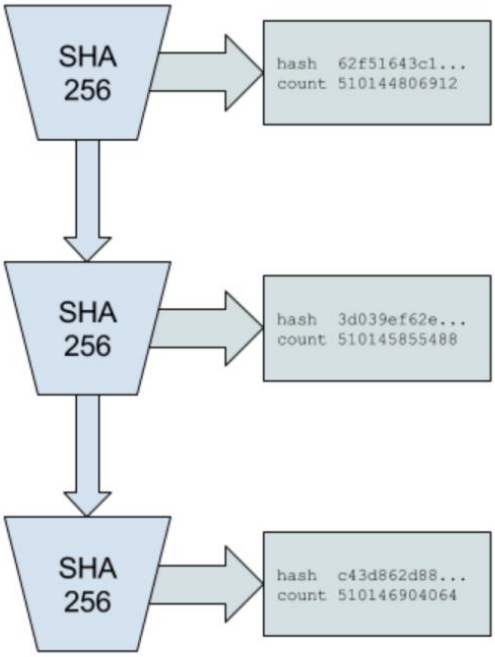


Figure 2: Proof of History sequence

4.2 이벤트 타임스탬프

이 해시 시퀀스는 특정 해시 인덱스가 생성되기 전에 어떤 데이터가 생성되었음을 기록하는 데 사용할 수 있습니다. 현재 인덱스의 현재 해시와 데이터를 결합하는 'combine' 함수를 사용하여 이를 수행합니다. 데이터는 임의의 이벤트 데이터의 암호학적으로 고유한 해시일 수 있습니다. 결합 함수는 데이터를 단순히 추가하거나 충돌 저항성(collision resistance)을 가지는 기타 연산일 수 있습니다. 다음에 생성된 해시는 데이터의 타임스탬프를 나타내며, 이는 해당 데이터 조각이 삽입된 후에만 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`

외부 이벤트가 발생하면, 예를 들어 사진이 찍히거나 임의의 디지털 데이터가 생성된 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**

해시336은 해시335의 이진 데이터와 사진의 SHA256 해시를 결합하여 계산됩니다. 인덱스와 사진의 SHA256 해시는 시퀀스 출력의 일부로 기록됩니다. 따라서 이 시퀀스를 검증하는 사람은 이 변경 사항을 재현할 수 있습니다. 검증은 여전히 병렬로 진행될 수 있으며, 이는 섹션 4.3에서 논의됩니다.

초기 과정이 여전히 순차적이기 때문에, 시퀀스에 들어간 것들은 미래의 해시된 값이 계산되기 전에 발생했음을 알 수 있습니다.

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

표 1에서 나타낸 시퀀스에서, photograph2는 해시600 이전에 생성되었고, photograph1은 해시336 이전에 생성되었습니다. 데이터를 해시 시퀀스에 삽입하면 시퀀스의 모든 이후 값이

변경됩니다. 사용된 해시 함수가 충돌 저항성이 있고 데이터가 추가된 경우, 이전 데이터에 대한 지식을 바탕으로 미래 시퀀스를 미리 계산하는 것은 계산적으로 불가능해야 합니다.

시퀀스에 혼합되는 데이터는 원본 데이터 자체일 수도 있고, 데이터와 함께 메타데이터가 포함된 해시일 수도 있습니다.

그림 3의 예에서, 입력 cfd40df8...이 Proof of History 시퀀스에 삽입되었습니다. 삽입된 카운트는 510145855488이며, 삽입된 상태는 3d039eef3입니다. 이 삽입으로 인해 모든 미래에 생성된 해시가 수정되며, 이 변경은 그림의 색상 변화로 표시됩니다.

이 시퀀스를 관찰하는 모든 노드는 모든 이벤트가 삽입된 순서를 결정하고 삽입 간의 실제 시간을 추정할 수 있습니다.

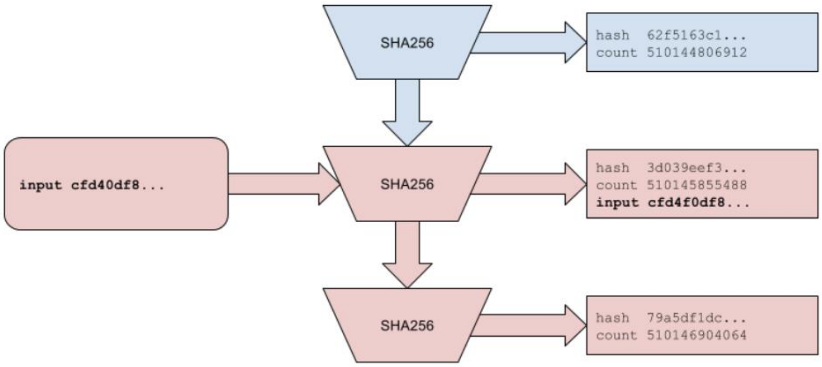


Figure 3: Inserting data into Proof of History

4.3 검증

시퀀스는 멀티코어 컴퓨터를 사용하여 생성하는 데 걸린 시간보다 훨씬 짧은 시간 안에 정확하게 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**

주어진 코어 수, 예를 들어 현대의 GPU가 4000코어를 갖고 있을 때, 검증자는 해시 시퀀스와 그 인덱스를 4000개의 슬라이스로 나누어 병렬로 처리하여 각 슬라이스가 시작 해시부터 마지막 해시까지 정확한지 확인할 수 있습니다. 시퀀스를 생성하는 데 예상되는 시간은 다음과 같습니다:

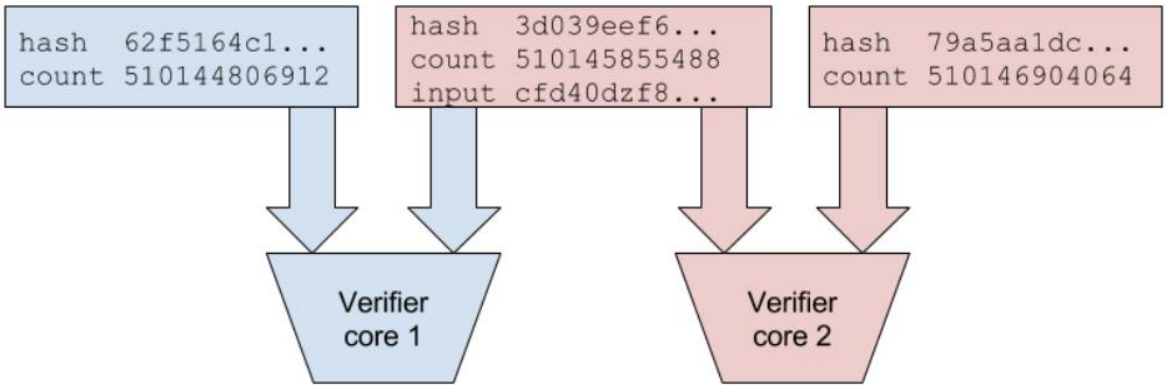


Figure 4: Verification using multiple cores

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

순서가 올바른지 확인하는 데 예상되는 시간은 다음과 같습니다:

$$\frac{\text{Total number of hashes}}{\text{(Hashes per second per core * Number of cores available to verify)}}$$

그림 4의 예에서, 각 코어는 시퀀스의 각 슬라이스를 병렬로 검증할 수 있습니다. 모든 입력 문자열은 출력에 기록되며, 이때 입력 문자열이 추가된 카운터와 상태도 기록됩니다. 검증자는 각 슬라이스를 병렬로 재현할 수 있습니다. 빨간색으로 표시된 해시는 데이터 삽입으로 인해 시퀀스가 수정되었음을 나타냅니다.

4.4 수평 확장

여러 개의 Proof of History 생성기를 동기화하는 것이 가능합니다. 각 생성기의 시퀀스 상태를 서로 다른 생성기로 혼합하여 Proof of History 생성기의 수평 확장을 달성할 수 있습니다. 이 확장은 샤딩 없이 수행됩니다. 시스템 내의 전체 이벤트 순서를 재구성하려면 두 생성기 모두의 출력이 필요합니다.

PoH Generator A		
Index	Hash	Data
1	hash1a	
2	hash2a	hash1b
3	hash3a	
4	hash4a	

PoH Generator B		
Index	Hash	Data
1	hash1b	
2	hash2b	hash1a
3	hash3b	
4	hash4b	

생성기 A와 B가 주어졌을 때, A는 B로부터 데이터 패킷(hash1b)을 수신합니다. 이 패킷에는 생성기 B의 마지막 상태와 생성기 B가 생성기 A로부터 관찰한 마지막 상태가 포함되어 있습니다. 그러면 생성기 A의 다음 상태 해시는 생성기 B의 상태에 의존하게 되며, 이로 인해 hash1b가 hash3a보다 먼저 발생했음을 유도할 수 있습니다. 이 속성은 전이적일 수 있으므로, 세 개의 생성기가 단일 공통 생성기를 통해 동기화된 경우(A ↔ B ↔ C), A와 C 간의 종속성을 추적할 수 있습니다. 비록 이들이 직접 동기화되지 않았더라도 말입니다.

생성기를 주기적으로 동기화함으로써, 각 생성기는 외부 트래픽의 일부를 처리할 수 있게 되며, 따라서 전체 시스템은 생성기 간의 네트워크 지연으로 인해 실제 시간 정확도의 비용을 감수하면서 더 많은 이벤트를 추적할 수 있습니다. 동기화 창 내의 이벤트를 순서대로 정렬하기 위해 해시 값과 같은 결정론적 함수를 선택하여 전역 순서를 여전히 달성할 수 있습니다.

그림 5에서는 두 생성기가 서로의 출력 상태를 삽입하고 작업을 기록합니다. 색상 변화는 동료의 데이터가 시퀀스를 수정했음을 나타냅니다. 각 스트림에 혼합된 생성된 해시가 굵게 강조 표시되어 있습니다.

동기화는 전이적입니다. A ↔ B ↔ C: B를 통해 A와 C 간의 이벤트 순서를 입증할 수 있습니다.

이러한 방식으로 확장하는 것은 가용성의 비용을 수반합니다. 0.999의 가용성을 가진 10 × 1 gbps 연결은 0.999^10 = 0.99의 가용성을 가지게 됩니다.

4.5 일관성

사용자들은 생성된 시퀀스의 일관성을 강제하고 공격에 대한 저항력을 높이기 위해 자신이 유효하다고 여기는 시퀀스의 마지막 관측 출력을 입력에 삽입할 수 있어야 합니다.

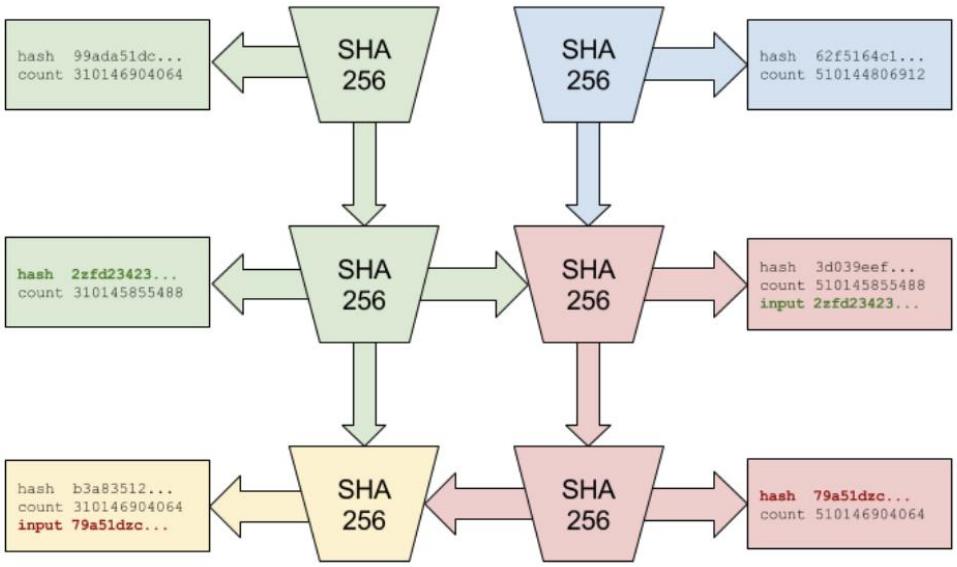


Figure 5: Two generators synchronizing

PoH Sequence A		
Index	Data	Output Hash
10		`hash10a`
20	Event1	`hash20a`
30	Event2	`hash30a`
40	Event3	`hash40a`

PoH Hidden Sequence B		
Index	Data	Output Hash
10		`hash10b`
20	Event3	`hash20b`
30	Event2	`hash30b`
40	Event1	`hash40b`

악의적인 PoH 생성자는 모든 이벤트를 한 번에 접근할 수 있거나 더 빠른 숨겨진 시퀀스를 생성할 수 있는 경우, 이벤트가 역순으로 나열된 두 번째 숨겨진 시퀀스를 생성할 수 있습니다.

이 공격을 방지하기 위해, 각 클라이언트가 생성한 이벤트는 자신이 유효하다고 여기는 시퀀스에서 마지막으로 관찰한 해시를 포함해야 합니다. 따라서 클라이언트가 "Event1" 데이터를 생성할 때, 그들은 자신이 관찰한 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 게시되면, Event3는 hash30a를 참조하게 되며, 이 해시가 Event 이전 시퀀스에 포함되어 있지 않으면 시퀀스 소비자들은 이 시퀀스가 유효하지 않다는 것을 알 수 있습니다. 부분적인 재배열 공격은 클라이언트가 이벤트를 관찰한 시점과 이벤트가 입력된 시점 사이에 생성된 해시의 수로 제한됩니다. 클라이언트는 마지막으로 관찰한 해시와 입력된 해시 사이의 짧은 해시 기간 동안 순서가 올바르다고 가정하지 않는 소프트웨어를 작성할 수 있어야 합니다.

악의적인 PoH 생성자가 클라이언트 이벤트 해시를 재작성하는 것을 방지하기 위해, 클라이언트는 데이터만 제출하는 대신 이벤트 데이터와 마지막으로 관찰한 해시의 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 데이터를 검증하기 위해서는 서명 검증과 이 해시 이전의 해시 시퀀스에서 해시를 조회해야 합니다. 검증 절차는 다음과 같습니다:

1. 서명 검증: 클라이언트가 제출한 서명이 올바른지 확인합니다. 서명은 이벤트 데이터와 마지막으로 관찰한 해시를 포함하고 있어야 하며, 이를 통해 데이터의 무결성과 출처를 확인할 수 있습니다.

2. 해시 조회: 서명이 유효하다고 확인된 후, 해당 해시가 이벤트 시퀀스 내에서 유효한지 확인합니다. 이는 해당 해시가 시퀀스에서 정확한 위치에 있는지를 검토하는 것을 포함합니다.

3. 시퀀스 검토: 해시가 시퀀스 내에서 예상된 위치에 존재하는지 확인합니다. 이 단계에서는 시퀀스의 정합성을 보장하기 위해 해시의 위치와 순서를 검토합니다.

4. 무결성 확인: 전체 시퀀스의 무결성을 검토하여 중간에 변조나 조작이 없었는지 확인합니다. 시퀀스의 모든 해시가 올바르게 연결되어 있는지 검토하고, 예기치 않은 변경이 없는지 확인합니다.

이러한 검증 절차를 통해 악의적인 PoH 생성자가 생성한 비정상적인 시퀀스나 조작된 데이터를 차단할 수 있습니다.

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

그림 6에서 사용자 제공 입력은 해시 0xdeadbeef...가 삽입되기 전의 생성된 시퀀스 내에 존재하는 것에 의존합니다.

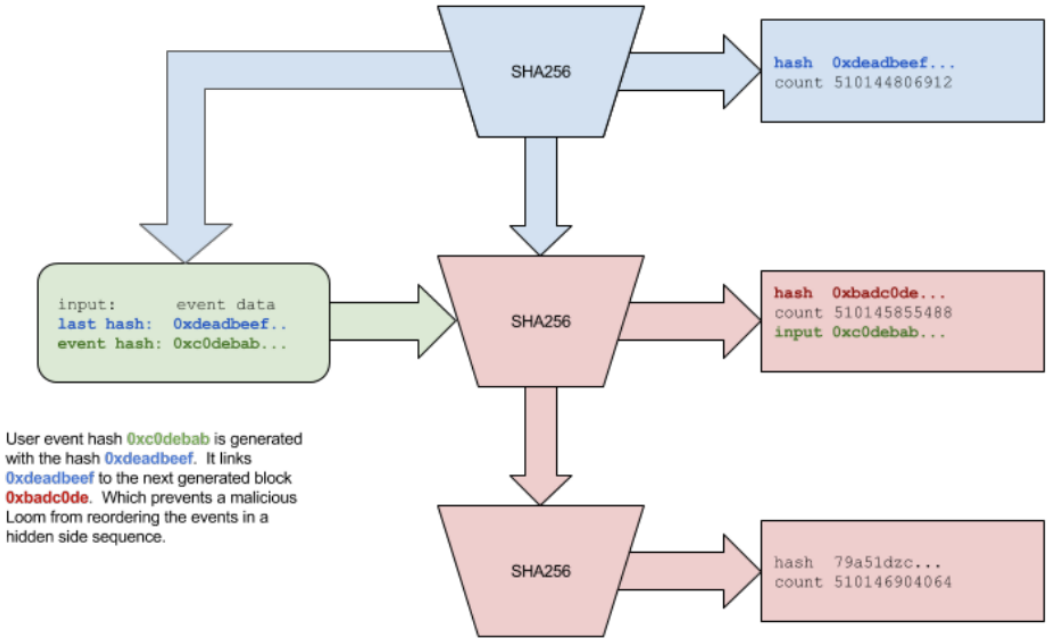


Figure 6: Input with a back reference.

4.6 오버헤드

파란색 왼쪽 상단 화살표는 클라이언트가 이전에 생성된 해시를 참조하고 있음을 나타냅니다. 클라이언트의 메시지는 해시 0xdeadbeef...가 포함된 시퀀스에서만 유효합니다. 시퀀스의 빨간색은 클라이언트의 데이터로 인해 시퀀스가 수정되었음을 나타냅니다.

4.7 공격

4.7.1 역순 생성

역순으로 해시를 생성하려면 공격자는 두 번째 이벤트 이후에 악의적인 시퀀스를 시작해야 합니다. 이 지연 시간 동안 악의적이지 않은 피어 투 피어 노드들이 원래 순서에 대해 소통할 수 있을 것입니다.

초당 4000개의 해시를 생성하면 추가로 160킬로바이트의 데이터가 발생하며, 이를 검증하려면 4000코어를 가진 GPU와 약 0.25~0.75밀리초의 시간이 필요합니다.

4.7.2 속도

여러 개의 생성기를 배치하면 공격에 대한 저항력을 높일 수 있습니다. 하나의 생성기는 높은 대역폭을 가지고 많은 이벤트를 시퀀스에 혼합할 수 있으며, 다른 생성기는 높은 속도와 낮은 대역폭을 가진 생성기로 주기적으로 높은 대역폭 생성기와 혼합됩니다.

높은 속도의 시퀀스는 데이터의 보조 시퀀스를 생성하며, 공격자는 이 시퀀스를 역으로 처리해야 합니다.

4.7.3 장기 범위 공격

장기 범위 공격은 오래된 폐기된 클라이언트 개인 키를 획득하여 위조된 원장을 생성하는 것을 포함합니다[10]. Proof of History는 장기 범위 공격에 대해 어느 정도의 보호를 제공합니다. 악의적인 사용자가 오래된 개인 키에 접근하게 되면, 그들은 위조하려는 원본 기록과 동일한 시간만큼 걸리는 역사적 기록을 재생성해야 합니다. 이는 네트워크에서 현재 사용 중인 프로세서보다 더 빠른 프로세서에 접근해야만 가능하며, 그렇지 않으면 공격자는 역사적 길이를 따라잡지 못할 것입니다.

추가로, 단일 시간 출처를 통해 더 간단한 Proof of Replication을 구축할 수 있습니다(자세한 내용은 6절에서 다룹니다). 네트워크는 모든 참가자가 단일 역사적 사건 기록을 신뢰하도록 설계되어 있기 때문입니다.

PoRep과 PoH는 함께 위조된 원장에 대한 공간과 시간의 방어를 제공해야 합니다.

5. Proof of Stake Consensus

5.1 설명

이 특정 Proof of Stake 인스턴스는 Proof of History 생성기가 생성한 현재 시퀀스의 신속한 확인, 다음 Proof of History 생성기 선택 및 투표, 그리고 잘못된 행동을 하는 검증자에 대한 처벌을 위해 설계되었습니다. 이 알고리즘은 메시지가 최종적으로 모든 참여 노드에 도착하는 것을 특정 시간 초과 내에 의존합니다.

5.2 용어

-채권(Bonds): 채권은 Proof of Work에서 자본 지출에 해당합니다. 채굴자는 하드웨어와 전력을 구매하고 이를 Proof of Work 블록체인의 단일 브랜치에 헌신합니다. 채권은 검증자가 거래를 검증하는 동안 담보로 제공하는 암호화폐입니다.

-슬래싱(Slashing): Proof of Stake 시스템의 "nothing at stake" 문제를 해결하기 위한 제안된 솔루션입니다. 만약 다른 브랜치에 대한 투표 증명이 공개되면, 그 브랜치는 검증자의 채권을 파기할 수 있습니다. 이는 검증자들이 여러 브랜치를 확인하지 않도록 유도하는 경제적 인센티브입니다.

-슈퍼 다수(Super Majority): 슈퍼 다수는 검증자의 채권에 의해 가중된 2/3의 검증자를 의미합니다. 슈퍼 다수의 투표는 네트워크가 합의에 도달했음을 나타내며, 최소한 1/3의 네트워크가 악의적으로 이 브랜치를 투표해야만 유효하지 않게 됩니다. 이는 공격의 경제적 비용을 암호화폐의 시가 총액의 1/3로 설정합니다.

5.3 채권 설정(Bonding)

채권 거래는 일정량의 암호화폐를 사용자의 신원 아래의 채권 계좌로 이동시킵니다. 채권 계좌의 암호화폐는 소비할 수 없으며 사용자가 이를 제거할 때까지 계좌에 남아 있어야 합니다.
사용자는 타임아웃이 지난 오래된 암호화폐만 제거할 수 있습니다. 채권은 현재의 이해관계자들이 슈퍼 다수로 시퀀스를 확인한 후 유효합니다.

5.4 투표(Voting)

Proof of History 생성기가 미리 정의된 기간에 상태의 서명을 게시할 수 있을 것으로 예상됩니다. 각 채권화된 신원은 자신의 서명된 상태 서명을 게시하여 해당 서명을 확인해야 합니다. 투표는 간단한 예 투표만을 포함하며, 아니오 투표는 없습니다.

채권화된 신원들 중 슈퍼 다수가 타임아웃 내에 투표를 완료하면, 해당 브랜치는 유효한 것으로 받아들여집니다.

5.5 언본딩(Unbonding)

N 수의 투표가 부족할 경우, 해당 암호화폐는 오래된 것으로 간주되며 더 이상 투표 자격이 없게 됩니다. 사용자는 이들을 제거하기 위해 언본딩 거래를 발행할 수 있습니다. N은 오래된 투표와

활성 투표의 비율에 따라 동적으로 조정됩니다. 오래된 투표의 수가 증가함에 따라 N도 증가합니다. 대규모 네트워크 파티션이 발생할 경우, 이는 큰 브랜치가 더 빨리 회복할 수 있도록 합니다.

5.6 선거(Elections)

새로운 PoH 생성기에 대한 선거는 PoH 생성기 실패가 감지되었을 때 발생합니다. 가장 많은 투표 권한을 가진 검증자, 또는 동점인 경우 가장 높은 공개 키 주소를 가진 검증자가 새로운 PoH 생성기로 선정됩니다.

새로운 시퀀스에 대한 슈퍼 다수의 확인이 필요합니다. 만약 새로운 리더가 슈퍼 다수의 확인을 받기 전에 실패할 경우, 다음 높은 순위의 검증자가 선택되고, 새로운 확인 세트가 필요합니다.

투표를 전환하려면, 검증자는 더 높은 PoH 시퀀스 카운터에서 투표해야 하며, 새로운 투표에는 전환하고자 하는 투표가 포함되어야 합니다. 그렇지 않으면 두 번째 투표는 슬래싱될 수 있습니다. 투표 전환은 슈퍼 다수가 없는 높이에서만 발생할 수 있도록 설계될 것으로 예상됩니다.

PoH 생성기가 설정된 후, 보조 생성자가 선출되어 거래 처리 업무를 인계받을 수 있습니다. 보조 생성자가 존재하는 경우, 주요 생성기가 실패할 때 다음 리더로 간주됩니다.

플랫폼은 예외가 감지되거나 미리 정의된 일정에 따라 보조 생성자가 주요 생성기로 승격되고, 낮은 순위의 생성자들이 승격되는 구조로 설계되어 있습니다.

5.7 선거 트리거(Election Triggers)

5.7.1 포크된 Proof of History 생성기

PoH 생성기는 생성된 시퀀스에 서명하는 신원으로 설계됩니다. PoH 생성기의 신원이 침해된 경우에만 포크가 발생할 수 있습니다. 포크는 두 개의 서로 다른 역사 기록이 동일한 PoH 신원에서 게시되었을 때 감지됩니다.

5.7.2 런타임 예외(Runtime Exceptions)

하드웨어 고장, 버그 또는 PoH 생성기에서의 의도적인 오류로 인해 생성기가 유효하지 않은 상태를 생성하고, 로컬 검증자의 결과와 일치하지 않는 상태 서명을 게시할 수 있습니다. 검증자들은 올바른 서명을 정보 전파를 통해 게시하며, 이 사건은 새로운 선거 라운드를 촉발합니다. 유효하지 않은 상태를 수용하는 검증자는 그들의 채권이 슬래싱됩니다.

5.7.3 네트워크 타임아웃(Network Timeouts)

네트워크 타임아웃은 선거를 촉발합니다.

5.8 슬래싱(Slashing)

슬래싱은 검증자가 두 개의 별개의 시퀀스에 투표할 때 발생합니다. 악의적인 투표의 증거가 제시되면, 채권이 유통에서 제거되며 채굴 풀에 추가됩니다.

경쟁 시퀀스에 대한 이전 투표를 포함하는 투표는 악의적인 투표의 증거로서 자격이 없습니다. 대신, 이 투표는 경쟁 시퀀스에서 현재 투표를 제거합니다.

슬래싱은 또한 PoH 생성기에 의해 생성된 무효한 해시를 위한 투표가 있을 때 발생합니다. 생성기가 무효한 상태를 무작위로 생성할 것으로 예상되며, 이는 Secondary로의 폴백을 촉발합니다.

5.9 보조 생성기 선거(Secondary Elections)

보조 및 하위 순위의 Proof of History 생성기는 제안되고 승인될 수 있습니다. 제안은 주요 생성기의 시퀀스에 제출됩니다. 제안에는 타임아웃이 포함되어 있으며, 타임아웃 전에 슈퍼 다수의 투표로 제안이 승인되면, 보조 생성기가 선출되어 예정대로 업무를 인계받습니다. 주요 생성기는 생성된 시퀀스에 메시지를 삽입하여 인계가 발생할 것임을 알리거나, 무효한 상태를 삽입하여 네트워크가 보조 생성기로 폴백하도록 강제할 수 있습니다.

보조 생성기가 선출되고 주요 생성기가 실패할 경우, 보조 생성기는 선거 중 첫 번째 폴백으로 간주됩니다.

5.10 가용성(Availability)

CAP 시스템은 파티션을 처리할 때 일관성(Consistency) 또는 가용성(Availability) 중 하나를 선택해야 합니다. 우리의 접근 방식은 궁극적으로 가용성을 선택하지만, 시간의 객관적 측정값이 있기 때문에 합리적인 인간 타임아웃 내에서 일관성을 선택할 수 있습니다.

Proof of Stake 검증자는 일부 코인을 스테이크에 잠가 특정 거래 세트에 투표할 수 있습니다. 코인을 잠가는 것은 PoH 스트림에 입력되는 거래와 동일하게 처리됩니다. 투표를 하려면 PoS 검증자는 PoH 원장 내에서 모든 거래를 처리한 후 계산된 상태의 해시를 서명해야 합니다. 이 투표 또한 PoH 스트림에 거래로 입력됩니다. PoH 원장을 살펴보면 각 투표 사이에 얼마나 많은 시간이 경과했는지, 그리고 파티션이 발생한 경우 각 검증자가 얼마나 오랫동안 사용 불가능했는지를 추론할 수 있습니다.

합리적인 인간 시간 프레임 내에서 파티션을 처리하기 위해, 우리는 사용 불가능한 검증자를 동적으로 언스테이킹하는 접근 방식을 제안합니다. 검증자의 수가 높고 2/3 이상일 때, 언스테이킹 프로세스는 빠를 수 있습니다. 사용 불가능한 검증자의 스테이크가 완전히 언스테이킹될 때까지 원장에 생성해야 하는 해시의 수가 적습니다. 검증자의 수가 2/3 미만이지만 1/2 이상일 때, 언스테이킹 타이머는 느려지며, 누락된 검증자가 언스테이킹되기 전에 생성해야 하는 해시의 수가 많아집니다. 검증자의 수가 1/2 이상 누락된 큰 파티션의 경우, 언스테이킹 프로세스는 매우 느립니다. 거래는 여전히 스트림에 입력될 수 있고, 검증자는 여전히 투표할 수 있지만, 전체 2/3의 합의는 많은 해시가 생성되고 사용 불가능한 검증자가 언스테이킹된 후에야 달성될 수 있습니다. 네트워크가 생존성을 회복하는 데 걸리는 시간 차이는 네트워크의 고객으로서 우리가 계속 사용할 파티션을 선택할 수 있게 합니다.

5.11 복구(Recovery)

우리가 제안하는 시스템에서는 원장에서 어떤 실패가 발생하더라도 완전히 복구할 수 있습니다. 즉, 전 세계의 누구나 원장에서 임의의 지점을 선택하고 새로 생성된 해시와 거래를 추가하여 유효한 포크를 생성할 수 있습니다. 만약 모든 검증자가 이 포크에서 누락된다면, 추가 채권이 유효해지고 이 가지가 2/3의 슈퍼 다수 합의를 달성하는 데에는 매우 오랜 시간이 걸릴 것입니다. 따라서, 사용할 수 있는 검증자가 전혀 없는 상태에서 완전한 복구를 하려면, 원장에 매우 많은 해시를 추가해야 하며, 사용 불가능한 검증자가 모두 언스테이킹된 후에야 새로운 채권이 원장을 검증할 수 있습니다.

5.12 최종성(Finality)

PoH는 네트워크의 검증자들이 과거에 발생한 사건들에 대해 일정 정도의 시간적 확실성을 가지고 관찰할 수 있게 해줍니다. PoH 생성기가 메시지 스트림을 생성하는 동안, 모든 검증자는 500ms 이내에 상태의 서명을 제출해야 합니다. 이 시간은 네트워크 조건에 따라 더 줄일 수 있습니다. 각 검증이 스트림에 입력되기 때문에, 네트워크의 모든 참여자는 실제로 투표를 직접 관찰하지 않고도 모든 검증자가 요구된 타임아웃 내에 자신의 투표를 제출했는지 검증할 수 있습니다.

5.13 공격(Attacks)

5.13.1 공유의 비극(Tragedy of Commons)

PoS 검증자는 PoH 생성기가 생성한 상태 해시를 단순히 확인합니다. 그들이 아무 작업도 하지 않고 생성된 상태 해시를 모두 승인하는 경제적 유인이 있습니다. 이 상황을 피하기 위해, PoH 생성기는 무효 해시를 무작위 간격으로 주입해야 합니다. 이 해시에 투표한 검증자는 슬래시(처벌)될 수 있습니다. 해시가 생성되면, 네트워크는 즉시 Secondary로 선출된 PoH 생성기를 승격시켜야 합니다.

각 검증자는 작은 타임아웃, 예를 들어 500ms 내에 응답해야 합니다. 타임아웃은 악의적인 검증자가 다른 검증자의 투표를 관찰하고 자신의 투표를 스트림에 빠르게 입력할 확률이 낮도록 충분히 짧게 설정해야 합니다.

5.13.2 PoH 생성기와의 공모(Collusion with the PoH generator)

PoH 생성기와 공모하는 검증자는 무효 해시가 언제 생성될지를 미리 알 수 있어 이를 투표하지 않을 수 있습니다. 이 시나리오는 PoH 신원이 더 큰 검증자 스테이크를 가진 것과 본질적으로 다르지 않습니다. PoH 생성기는 여전히 상태 해시를 생성하는 모든 작업을 수행해야 합니다.

5.13.3 검열(Censorship)

검열 또는 서비스 거부 공격은 1/3의 채권 보유자가 새로운 채권이 포함된 어떤 시퀀스도 검증하지 않을 때 발생할 수 있습니다. 이 형태의 공격에 대해 프로토콜은 채권이 오래된 것으로 간주되는 속도를 동적으로 조정하여 방어할 수 있습니다. 서비스 거부가 발생하면, 더 큰

파티션은 포크를 통해 비잔틴 채권 보유자들을 검열하도록 설계됩니다. 더 큰 네트워크는 비잔틴 채권이 시간이 지나면서 오래된 것으로 간주되면 회복될 것입니다. 더 작은 비잔틴 파티션은 더 오랜 기간 동안 진행되지 않을 것입니다.

알고리즘은 다음과 같이 작동합니다. 네트워크의 다수는 새로운 리더를 선출합니다. 리더는 비잔틴 채권 보유자들이 참여하지 못하도록 검열합니다. Proof of History 생성기는 충분한 비잔틴 채권이 오래된 것으로 간주될 때까지 시퀀스를 계속 생성하여 시간의 경과를 증명해야 합니다. 채권이 오래된 것으로 간주되는 속도는 채권의 활성 비율에 따라 동적으로 조정됩니다. 따라서 비잔틴 소수 파티션은 다수 파티션보다 슈퍼 다수결을 회복하는 데 훨씬 더 오랜 시간이 걸릴 것입니다. 슈퍼 다수결이 확립되면, 슬래싱을 사용하여 비잔틴 채권 보유자들을 영구적으로 처벌할 수 있습니다.

5.13.4 장기 공격(Long Range Attacks)

PoH는 장기 공격에 대한 자연적인 방어를 제공합니다. 과거의 어떤 시점에서 장부를 복구하려면 공격자는 PoH 생성기의 속도를 초과하여 유효한 장부를 능가해야 합니다.

합의 프로토콜은 두 번째 방어층을 제공하며, 공격자는 모든 유효한 검증자를 언스테이크하는 데 걸리는 시간보다 더 오랜 시간이 걸려야 합니다. 또한 장부의 역사에 가용성 격차를 생성합니다. 동일한 높이의 두 장부를 비교할 때, 최대 파티션이 가장 작은 장부를 객관적으로 유효한 것으로 간주할 수 있습니다.

5.13.5 ASIC 공격(ASIC Attacks)

이 프로토콜에는 두 가지 ASIC 공격 기회가 있습니다 - 파티션 중과 최종성에서의 시간 초과를 속이는 것입니다.

파티션 중 ASIC 공격에 대한 경우, 채권이 언스테이크되는 속도는 비선형적이며, 대규모 파티션이 있는 네트워크의 경우 이 속도는 ASIC 공격으로 기대되는 이득보다 수량적 차이가 크게 느리게 됩니다.

5.13.5 ASIC 공격(ASIC Attacks) (계속)

최종성에서 ASIC 공격의 경우, 취약점은 비잔틴 검증자들이 결합된 스테이크를 가지고 다른 노드로부터의 확인을 기다리며 협력하는 PoH 생성기와 함께 투표를 주입할 수 있도록 허용합니다. PoH 생성기는 더 빠른 ASIC을 사용하여 500ms 분량의 해시를 더 짧은 시간 내에 생성하고, PoH 생성기와 협력하는 노드 간의 네트워크 통신을 허용할 수 있습니다. 그러나 PoH 생성기도 비잔틴이라면, 비잔틴 생성기가 실패를 삽입할 때 정확한 카운터를 전달하지 않을 이유가 없습니다. 이 시나리오는 PoH 생성기와 모든 협력자들이 동일한 정체성을 공유하고 단일 결합된 스테이크를 가지며 단일 하드웨어 세트만 사용하는 것과 다르지 않습니다.

6. 스트리밍 복제 증명(Streaming Proof of Replication)

6.1 설명

Filecoin은 복제 증명(Proof of Replication)의 버전을 제안했습니다 [6]. 이 버전의 목표는 Proof of History에서 생성된 시퀀스의 시간 추적을 통해 복제 증명의 빠르고 스트리밍 가능한 검증을 가능하게 하는 것입니다. 복제는 합의 알고리즘으로 사용되지 않지만, 블록체인 역사 또는 상태를 높은 가용성으로 저장하는 비용을 계산하는 데 유용한 도구입니다.

6.2 알고리즘

그림 7에서 볼 수 있듯이, CBC 암호화는 데이터의 각 블록을 시퀀스에서 암호화하며, 이전에 암호화된 블록을 사용하여 입력 데이터를 XOR합니다.

각 복제 식별자는 Proof of History 시퀀스에서 생성된 해시를 서명하여 키를 생성합니다. 이는 키를 복제자의 정체성과 특정 Proof of History 시퀀스에 연결합니다. 특정 해시만 선택할 수 있습니다. (해시 선택에 대한 내용은 섹션 6.5 참조)

데이터 세트는 블록 단위로 완전히 암호화됩니다. 그런 다음 증명을 생성하기 위해, 키를 사용하여 의사 난수 생성기를 시드하여 각 블록에서 임의의 32 바이트를 선택합니다.

선택된 PoH 해시가 각 슬라이스에 선행하여 메르클 해시가 계산됩니다.

루트는 키와 함께 공개되며, 생성된 선택된 해시도 함께 발표됩니다. 복제 노드는 Proof of History 생성기에서 생성된 N개의 해시에서 추가적인 증명을 발표해야 합니다.

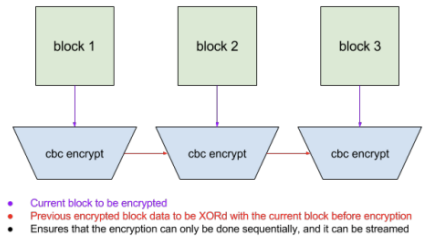


Figure 7: Sequential CBC encryption

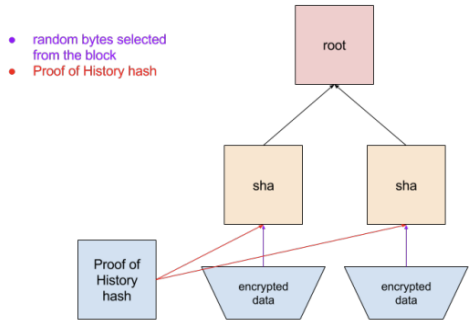


Figure 8: Fast Proof of Replication

여기서 N은 데이터를 암호화하는 데 걸리는 시간의 약 1/2입니다. Proof of History 생성기는 사전에 정의된 주기로 Proof of Replication을 위한 특정 해시를 발표합니다. 복제 노드는 증명을 생성하기 위해 다음에 발표된 해시를 선택해야 합니다. 해시는 서명되고, 무작위로 선택된 슬라이스가 블록에서 선택되어 메르클 루트가 생성됩니다.

N개의 증명 기간이 지난 후, 데이터는 새로운 CBC 키로 다시 암호화됩니다.

6.3 Verification

N개의 코어를 가진 경우, 각 코어는 각 아이덴티티에 대한 암호화를 스트리밍할 수 있습니다. 필요한 총 공간은 \(2 \times \text{블록} \times N_{\text{코어}}\)입니다. 이전에 암호화된 블록이

다음 블록을 생성하는 데 필요하기 때문입니다. 각 코어는 현재 암호화된 블록에서 파생된 모든 증명을 생성하는 데 사용할 수 있습니다.

증명을 검증하는 데 걸리는 총 시간은 암호화하는 데 걸리는 시간과 같을 것으로 예상됩니다. 증명 자체는 블록에서 몇 개의 무작위 바이트만 소비하므로, 해시할 데이터 양은 암호화된 블록 크기보다 상당히 적습니다. 동시에 검증할 수 있는 복제 아이덴티티의 수는 사용 가능한 코어의 수와 같습니다. 최신 GPU는 3500개 이상의 코어를 보유하고 있으며, CPU의 1/2에서 1/3 속도로 동작합니다.

6.4 Key Rotation

키 회전이 없으면 동일한 암호화된 복제가 여러 Proof of History 시퀀스에 대해 저렴한 증명을 생성할 수 있습니다. 키는 주기적으로 회전하며, 각 복제는 고유한 Proof of History 시퀀스에 연결된 새로운 키로 재암호화됩니다.

회전 주기는 GPU 하드웨어에서 복제 증명을 검증할 수 있을 정도로 충분히 느려야 합니다. 이는 GPU가 코어당 속도가 CPU보다 느리기 때문입니다.

6.5 Hash Selection

Proof of History 생성기는 전체 네트워크에서 Proof of Replication 암호화 및 빠른 증명의 바이트 선택을 위해 사용할 해시를 발표합니다. 해시는 데이터 세트를 암호화하는 데 걸리는 시간의 약 절반에 해당하는 주기적 카운터에서 발표됩니다. 각 복제 아이덴티티는 동일한 해시를 사용해야 하며, 해시의 서명 결과를 바이트 선택 또는 암호화 키의 시드로 사용해야 합니다.

각 복제자가 증명을 제공해야 하는 기간은 암호화 시간보다 짧아야 합니다. 그렇지 않으면 복제자는 암호화를 스트리밍하고 각 증명에 대해 삭제할 수 있습니다.

악의적인 생성자가 특정 해시를 생성하기 위해 이 해시 이전에 시퀀스에 데이터를 주입할 수 있습니다. 이 공격은 5.13.2에서 더 자세히 논의됩니다.

6.6 Proof Validation

Proof of History 노드는 제출된 Proof of Replication 증명을 검증할 것으로 기대되지 않습니다. 대신, Proof of History 노드는 복제기 아이덴티티에 의해 제출된 대기 중 및 검증된 증명의 수를

추적할 것으로 예상됩니다. 증명은 복제기가 네트워크의 슈퍼 다수의 검증자들에 의해 증명을 서명할 수 있을 때 검증될 것으로 기대됩니다.

검증은 복제기가 p2p gossip 네트워크를 통해 수집하고, 네트워크의 슈퍼 다수의 검증자를 포함하는 하나의 패킷으로 제출합니다. 이 패킷은 Proof of History 시퀀스에 의해 생성된 특정 해시 이전의 모든 증명을 검증하며, 동시에 여러 복제기 아이덴티티를 포함할 수 있습니다.

6.7 Attacks

6.7.1 Spam

악의적인 사용자가 많은 복제기 아이덴티티를 생성하고 나쁜 증명으로 네트워크를 스팸할 수 있습니다. 빠른 검증을 촉진하기 위해, 노드는 검증 요청 시 암호화된 데이터와 전체 머클 트리를 나머지 네트워크에 제공해야 합니다.

이 문서에서 설계된 Proof of Replication은 추가 증명의 검증이 저렴하게 이루어질 수 있도록 하며, 추가적인 공간을 필요로 하지 않습니다. 하지만 각 아이덴티티는 1개의 암호화 시간 코어를 소모하게 됩니다. 복제 목표는 readily available 코어의 최대 크기로 설정되어야 합니다. 최신 GPU는 3500개 이상의 코어를 탑재하고 있습니다.

6.7.2 Partial Erasure

복제기 노드가 전체 상태를 저장하지 않고 일부 데이터를 부분적으로 삭제하려고 할 수 있습니다. 증명의 수와 시드의 무작위성으로 인해 이 공격을 어렵게 만들 수 있습니다.

예를 들어, 1 테라바이트의 데이터를 저장하는 사용자가 각 1 메가바이트 블록에서 1바이트를 삭제한다고 가정합시다. 매 메가바이트에서 1바이트를 샘플링하는 단일 증명은 삭제된 바이트와의 충돌 가능성이 \(1 - (1 - 1/1,000,000)^{1,000,000} = 0.63\)입니다. 5개의 증명 후에는 충돌 가능성이 0.99로 증가합니다.

6.7 Attacks

6.7.3 Collusion with PoH Generator

서명된 해시는 샘플링을 위한 시드로 사용될 것으로 예상됩니다. 만약 복제기가 특정 해시를 사전에 선택할 수 있다면, 복제기는 샘플링되지 않을 바이트를 모두 삭제할 수 있습니다.

Proof of History 생성기와 공모하는 복제기 아이덴티티는 랜덤 바이트 선택을 위한 미리 정의된 해시가 생성되기 전에 시퀀스의 끝에 특정 트랜잭션을 주입할 수 있습니다. 충분한 코어를 가진 공격자는 복제기 아이덴티티에 유리한 해시를 생성할 수 있습니다. 이 공격은 단일 복제기 아이덴티티에만 이익을 줄 수 있습니다. 모든 아이덴티티는 ECDSA(또는 동등한 방식)로 암호화된 동일한 해시를 사용해야 하므로 결과 서명은 각 복제기 아이덴티티에 대해 고유하며 충돌 저항성이 있습니다. 단일 복제기 아이덴티티는 미미한 이득만을 얻을 수 있습니다.

6.7.4 Denial of Service

추가 복제기 아이덴티티를 추가하는 비용은 저장 비용과 같을 것으로 예상됩니다. 모든 복제기 아이덴티티를 검증하기 위해 추가적인 컴퓨팅 용량을 추가하는 비용은 복제기 아이덴티티당 CPU 또는 GPU 코어의 비용과 같을 것으로 예상됩니다.

이로 인해 많은 수의 유효한 복제기 아이덴티티를 생성하여 네트워크에 대한 서비스 거부 공격을 할 수 있는 기회가 생깁니다. 이 공격을 제한하기 위해 네트워크를 위한 합의 프로토콜은 복제 대상 목표를 선택하고, 네트워크의 가용성, 대역폭, 지리적 위치 등의 원하는 특성을 충족하는 복제 증명에 보상을 제공할 수 있습니다.

6.7.5 Tragedy of Commons

PoS 검증자는 작업을 수행하지 않고 PoRep만 확인할 수 있습니다. 경제적 인센티브는 PoS 검증자가 작업을 수행하도록 정렬되어야 하며, 예를 들어 채굴 보상을 PoS 검증자와 PoRep 복제기 노드 간에 나누는 방식이 될 수 있습니다.

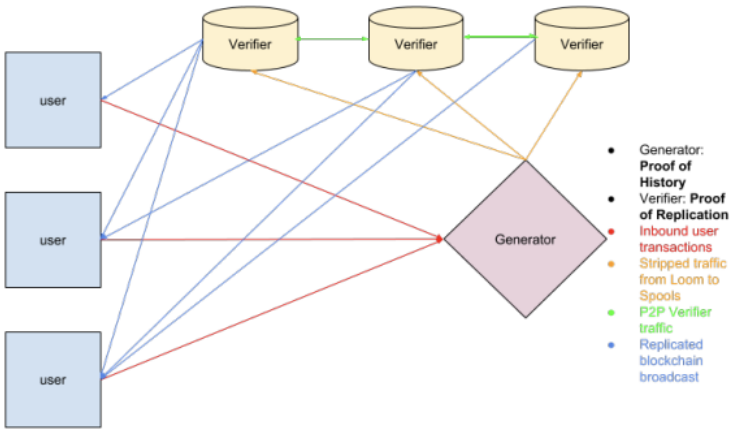


Figure 9: System Architecture

이 시나리오를 더욱 방지하기 위해, PoRep 검증자는 소량의 시간 동안 잘못된 증명을 제출할 수 있습니다. 잘못된 데이터를 생성한 함수 제공을 통해 증명이 잘못되었음을 증명할 수 있습니다. 잘못된 증명을 확인한 PoS 검증자는 벌금을 받을 것입니다.

7 System Architecture

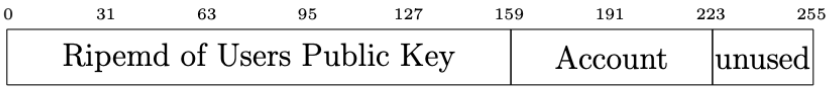
7.1 Components

7.1.1 Leader, Proof of History Generator

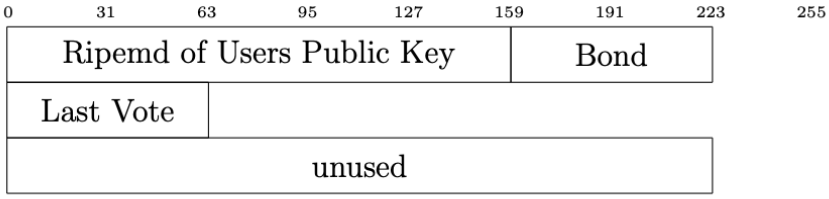
리더는 선출된 Proof of History 생성기입니다. 리더는 임의의 사용자 트랜잭션을 처리하고, 시스템 내에서 고유한 전역 순서를 보장하는 Proof of History 시퀀스를 출력합니다. 각 트랜잭션 배치 후에 리더는 해당 순서로 트랜잭션을 실행한 결과로서 상태의 서명을 출력합니다. 이 서명은 리더의 아이덴티티로 서명됩니다.

7.1.2 State

상태는 사용자 주소로 인덱싱된 단순 해시 테이블입니다. 각 셀에는 전체 사용자 주소와 이 계산에 필요한 메모리가 포함됩니다. 예를 들어 트랜잭션 테이블은 다음과 같습니다:



For a total of 32 bytes.
Proof of Stake bonds table contains:



For a total of 64 bytes.

7 System Architecture

7.1 Components

7.1.3 Verifier, State Replication

검증 노드는 블록체인 상태를 복제하고 블록체인 상태의 높은 가용성을 제공합니다. 복제 목표는 합의 알고리즘에 의해 선택되며, 합의 알고리즘의 검증자들이 오프체인에서 정의된 기준에 따라 승인된 Proof of Replication 노드를 선택하고 투표합니다. 네트워크는 최소 Proof of Stake 채권 크기와 각 채권당 하나의 복제기 아이덴티티를 요구하도록 구성될 수 있습니다.

7.1.4 Validators

이 노드는 검증자들로부터 대역폭을 소모합니다. 이들은 가상 노드이며, 검증자나 리더와 동일한 머신에서 실행될 수 있거나, 이 네트워크에 구성된 합의 알고리즘에 특화된 별도의 머신에서 실행될 수 있습니다.

7.2 Network Limits

리더는 수신된 사용자 패킷을 받아 가장 효율적인 방법으로 정렬하고, 이를 Proof of History 시퀀스로 순서화하여 하류 검증자들에게 공개합니다.

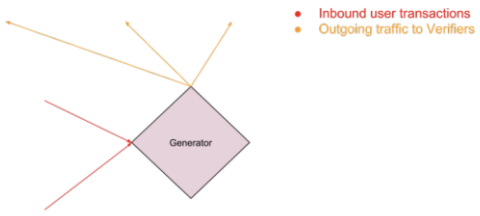
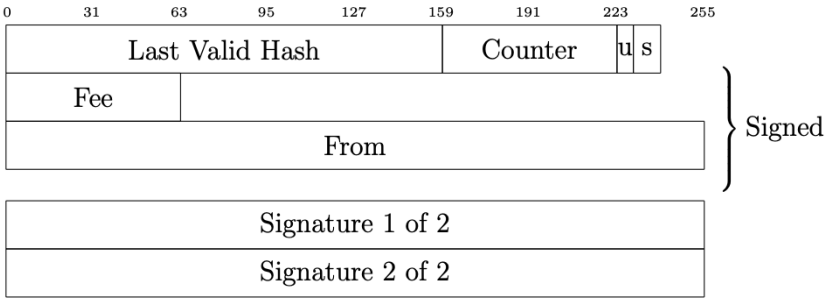


Figure 10: Generator network limits

효율성은 트랜잭션의 메모리 접근 패턴에 기반하므로, 트랜잭션은 결함을 최소화하고 프리패칭을 극대화하도록 정렬됩니다.

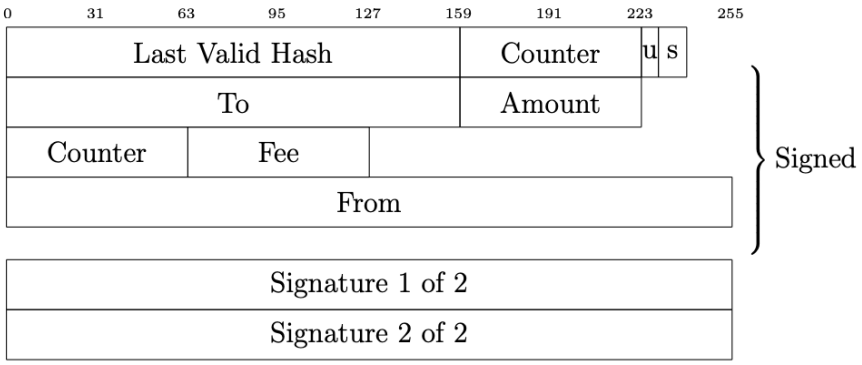
수신 패킷 형식:

Incoming packet format:



Size $20 + 8 + 16 + 8 + 32 + 3232 = 148$ bytes.

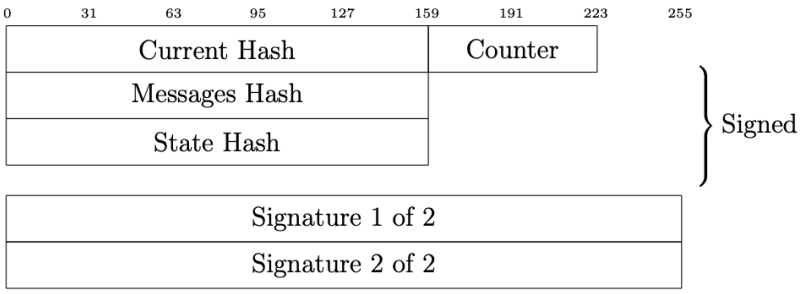
최소 지원 가능한 페이로드는 1개의 대상 계정이 될 수 있습니다. 페이로드가 다음과 같을 때:



With payload the minimum size: 176 bytes

Proof of History 시퀀스 패킷은 현재 해시, 카운터, PoH 시퀀스에 추가된 모든 새로운 메시지의 해시와 모든 메시지를 처리한 후의 상태 서명을 포함합니다. 이 패킷은 N개의 메시지가 방송될 때마다 한 번 전송됩니다.

Proof of History 패킷:



Minimum size of the output packet is: 132 bytes

7.2 네트워크 한계

1gbps 네트워크 연결에서는 최대 거래 수가 1기가비트/초 / 176바이트 = 최대 710k tps로 가능합니다. 이더넷 프레이밍으로 인해 1-4%의 손실이 예상됩니다. 네트워크의 목표량을 초과한 여유 용량은 Reed-Solomon 코드로 출력을 인코딩하고 이를 사용 가능한 다운스트림 검증자에게 스트라이핑하여 가용성을 증가시키는 데 사용할 수 있습니다.

7.3 컴퓨팅 한계

각 거래는 다이제스트 검증을 요구합니다. 이 작업은 거래 메시지 자체 외에 메모리를 사용하지 않으며 독립적으로 병렬화할 수 있습니다. 따라서 처리량은 시스템에서 사용할 수 있는 코어 수에 의해 제한될 것으로 예상됩니다. GPU 기반 ECDSA 검증 서버는 초당 90만 개의 작업을 수행한 실험 결과를 보였습니다 [9].

7.4 메모리 한계

상태를 32바이트 항목을 가진 50% 채워진 해시 테이블로 단순하게 구현하면, 이론적으로 640GB에 100억 개의 계정을 수용할 수 있습니다. 이 테이블에 대한 정기적인 무작위 접근은 초당 1.1 × 10^7번의 읽기 또는 쓰기로 측정됩니다. 거래당 2회의 읽기와 2회의 쓰기를 기준으로,

메모리 처리량은 초당 275만 개의 거래를 처리할 수 있습니다. 이는 Amazon Web Services의 1TB x1.16xlarge 인스턴스에서 측정된 값입니다.

7.5 고성능 스마트 계약

스마트 계약은 거래의 일반화된 형태입니다. 이들은 각 노드에서 실행되며 상태를 수정하는 프로그램입니다. 이 설계는 빠르고 분석하기 쉬운 확장된 Berkeley Packet Filter 바이트코드와 스마트 계약 언어로서 JIT 바이트코드를 활용합니다.
주요 장점 중 하나는 제로 비용 외부 함수 인터페이스입니다. 플랫폼에서 직접 구현된 내재 함수는 프로그램에서 호출할 수 있습니다. 내재 함수를 호출하면 프로그램이 중단되고 내재 함수가 고성능 서버에서 스케줄됩니다. 내재 함수는 GPU에서 병렬로 실행되도록 배치됩니다.

위의 예제에서 두 개의 서로 다른 사용자 프로그램이 동일한 내재 함수를 호출합니다. 각 프로그램은 내재 함수의 배치 실행이 완료될 때까지 일시 중단됩니다. 예를 들어 ECDSA 검증이 내재 함수 중 하나입니다. 이러한 호출을 배치하여 GPU에서 실행하면 처리량을 수천 배 증가시킬 수 있습니다.
이 트램폴린은 네이티브 운영 체제 스레드 컨텍스트 스위치를 필요로 하지 않으며, BPF 바이트코드는 사용하는 모든 메모리에 대해 잘 정의된 컨텍스트를 가지고 있습니다.

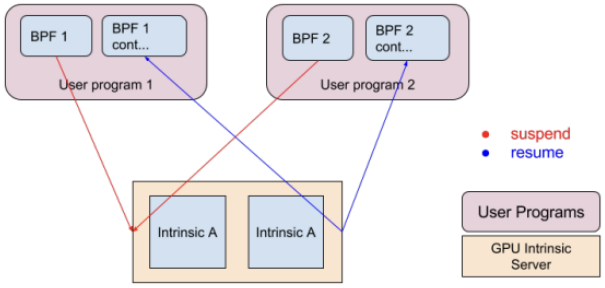


Figure 11: Executing BPF programs.

eBPF 백엔드는 2015년부터 LLVM에 포함되어 있으므로, 모든 LLVM 프론트엔드 언어를 사용하여 스마트 계약을 작성할 수 있습니다. 2015년부터 리눅스 커널에 포함되었으며, 바이트코드의 첫 번째 버전은 1992년부터 존재했습니다. 단일 패스를 통해 eBPF의 정확성을 확인하고, 실행 시간 및 메모리 요구 사항을 확인한 후, 이를 x86 명령어로 변환할 수 있습니다.