4.1 World State

(3)

World State는 160비트의 주소와 계정 상태(RLP 데이터 구조로 직렬화된)의 매핑이다.

저장할 데이터가 너무 많기에 블록체인에 저장되지 않고, 변형된 머클 패트리샤 트리에 저장할 것이다.

(4) 왜 머클 패트리샤 트리일까?

첫번째로, State 루트는 내부의 모든 데이터에 암호화 방식을 의존하므로 해시 자체가 전체 시스템 상태의 보안 ID로 사용할 수 있습니다.

두번째로, 그림처럼 상태가 변한 값만 추적하기 때문에 불변 데이터 구조입니다. 또한, 모든 루트 해시를 블록체인에 저장하기 때문에 루트 해시를 변경하여 이전 상태로 간단하게 돌릴 수 있는 장점이 있습니다.

(넘김)

블록체인에 모든 데이터를 저장하면 너무 효율성이 없지 않냐? 라는 질문이 들겠지만, 실제로 검증이 이루어지지 않을 때 저장되는 공간은 우측 사진처럼 루트 해시값만 저장됩니다.)

(5)

앞서 설명한 World State는 키는 주소이고 값은 계정상태인 key-value 저장소이다. 계정 상태에는 4가지 구성요소가 있다. Nonce, balance, storageRoot, codeHash 이렇게 4가지가 있다.

(6)

식 8은 왼쪽 key값은 주소, value값은 계정 상태를 나타낸다.

먼저 주소를 살펴보면, 계정 생성할 때 무작위로 32바이트의 개인키가 생성됩니다. 32바이트 이하 임의의 문자열 k를 Keccak256 알고리즘을 통해 32바이트가 나오고, 256비트에서 하위 20바이트만 뺀 값이 World State의 주소 값이 됩니다.

두번째로 value에서 4가지 계정 상태 값들이 어떻게 1개의 value로 나왔는가? 질문이 듭니다.

식 11을 보시면 4가지 구성요소에 RLP 데이터 구조를 사용하여 직렬화 시켜서 하나의 value값이 됩니다.

(7)

StorageRoot은 physical 멤버가 아니며, 직렬화에 기여하지 않는다.

이 말에 의문이 들었는데, 물리적으로 구성원이 아니라는 말은 계정 주소와 계정 상태가 물리적인 구성요소지. 스토리지 루트의 값을 계정상태가 사용하여 나온 것이니까 논리적으로는 맞지만 물리적으로는 아니라고 판단되었습니다. 또한, 직렬화에 기여하지 않는다는 말은 블록 연결할 때, RLP 데이터 구조로 직렬화 시킬 때 스토리지 루트 값을 직접적으로 직렬화 하는 것이 아닌 스토리지 루트 값을 사용한 StateRoot 값을 사용한 헤더를 직접적으로 직렬화 하기 때문에 이러한 문장이 나온 것 같습니다.

(8)

Yellow paper에 빈 문자열의 해시값이 코드해시값으로 들어가는 non-contract계정이 있다고 하는데 이는 계정 종류중에 EOA에 속합니다. EOA는 CA와 다르게 코드가 없기 때문입니다.

(9)

World State & Account는 이렇게 구성되어 있습니다. 이러한 블록이 연결되어 블록체인이 됩니다. 블록체인에 모든 데이터가 저장되는 것이 아니라 블록 헤더에 루트들과, 계정 상태에 storageRoot 처럼 루트값만 저장됩니다.

4.2 Transaction

(10)

트랜잭션은 크게 2가지 type이 있는데 legacy, 2020년에 나온 EIP-2930 type이 존재합니다.

또한, 하위 타입으로는 message-call, ‘contract creation로 나뉘어져 있습니다.

(11)

먼저 Legacy와 EIP-2390을 비교해보자. Legacy는 우리가 본래 알고 있는 트랜잭션의 구조입니다.

근데, EIP-2930에는 새롭게 추가하는 요소들이 존재합니다.

(12)

Accesslist – 이전에는 모든 트랜잭션이 상태 루트를 변경할 때, 모든 계정을 순회하고 상태를 업데이트하였는데, 이는 비용 이 많이 드는 연산이었습니다. accessList를 사용하여 트랜잭션이 상태에 접근해야 하는 계정을 명시적으로 지정함으로써 업데이트해야 하는 계정을 줄일 수 있었습니다. 비용을 줄이고 효율적인 블록 생성을 가능하게 되었습니다. (다음 그림과 같이 베를린 업데이트 이후에 가스비가 올랐는데, 접근한 계정을 중복으로 접근할 때는 다음 표와 같이 소모되는 가스 비용을 줄일 수 있었습니다. )

ChainID – 같은 트랜잭션을 여러 네트워크에 전송하면 Replay Attack에 취약한 상태가 되는데, chainID를 트랜잭션에 포함하여 유효성을 특정 네트워크에 제한함으로써 Replay Attack을 방지할 수 있습니다. 과거에는 CHAINID라는 OPCODE를 사용했지만, 트랜잭션에 직접 포함시키는 방법을 사용하게 되었습니다.

yParity – 서명 y의 parity

(13)

Transaction은 subtype으로 Contract Creation과 MessageCall로 나뉩니다.

(14)

Contract Creation은 init이라는 추가 필드를 지니고 있는데, init은 계정 초기화 절차에 대한 EVM 코드를 지정하는 무제한 크기의 바이트 배열로, 계정 생성 시 한 번만 실행되고 그 후 즉시 삭제되는 생성자 같은 코드입니다.

말 그대로 Code를 사용할 수 있는 Contract Account를 생성하는 Transaction 입니다.

(15)

Message Call은 Message Call의 입력 데이터를 지정하는 무제한 크기의 바이트 배열인 Data라는 추가필드가 포함되어 있어 계정의 상태를 업데이트 할 수 있습니다.

EOA가 EOA나 CA를 호출하여 상태를 업데이트 하는 Transaction입니다.

(16)

기타사항 설명

4.3 블록의 전체적인 구조 설명

(17)

블록에는 블록 헤더 H와 블록 트랜잭션 T와 현재 블록의 부모의 부모와 동일한 블록을 갖는 것으로 알려진 ommers Block의 헤더를 U라고 한다. 이러한 구성요소를 합친 것을 Block이라고 하며

21 식으로 나타낼 수 있다.

(18)

블록은 이러한 구조로 되어있고, 아까 보여주었던 관련 요소들 처럼 되어있음

---- Contract Creation

OOG

OOG란 코드 실행시 가스가 모두 소모되어 잔여 가스가 부족할 때 코드 실행이 강제로 종료되는 예외 상태입니다. OOG 예외가 발생하여 코드의 실행이 강제로 종료되면 스마트 컨트랙트 상태는 변경되지 않으며 이전 상태를 유지합니다.

우리는 컨트랙트 생성 중에 OOG 예외가 발생하면 새로운 스마트 컨트랙트를 생성하는 작업이 무효화되고 블록체인은 생성 시도 이전 상태를 유지합니다.

만약, initialization 코드가 성공적으로 완료되면 생성된 contract 코드의 크기에 비례하는 생성된 최종 컨트랙트 생성 비용이 지불됩니다. 또한, 남은 가스는 발신자에게 환불되며 변경된 상태는 블록체인에 배포됩니다.

하지만, c가 잔여 가스인 g\*\*보다 크면 OOG예외를 선언하고 잔여가스는 0이 될 것 입니다. 컨트랙트 생성 트랜잭션이 수신되었다면, 예외적인 상황이라도 생성 비용은 지불됩니다. 하지만, 트랜잭션의 value인 보내는 값은 OOG로 인해 폐기된 컨트랙트의 주소에 전송되지 않고, contract의 코드도 저장되지 않습니다. (생성 비용은 지불, 같이 보냈던 값은 환불)