ETHEREUM: 안전한 분산형 일반 트랜잭션 원장

베를린 버전 beacfbd – 2022-10-24

DR. GAVIN WOOD 설립자,
이더리움 및 패리티 GAVIN@PARITY.IO

추상적인. 암호화 보안 트랜잭션과 결합된 블록체인 패러다임 은 여러 프로젝트를 통해 그 유용성을 입증했으며 , 비트코인은 가장 주목할만한 프로젝트 중 하나입니다. 이러한 각 프로젝트는 분산형이지만 싱글톤 컴퓨팅 리소스에 대한 간단한 애플리케이션 으로 볼 수 있습니다 . 우리는 이 패러다임을 공유 상태를 가진 트랜잭션 싱글톤 기계라고 부를 수 있습니다 .

이더리움은 일반화된 방식으로 이 패러다임을 구현합니다. 또한 각각 별개의 상태와 운영 코드를 가지고 있지만 메시지 전달 프레임워크를 통해 다른 사람들과 상호 작용할 수 있는 복수의 리소스를 제공합니다 .
우리는 디자인, 구현 문제, 제공하는 기회 및 우리가 예상하는 미래의 장애물에 대해 논의합니다.

1. 소개

세계 대부분의 장소에서 유비쿼터스 인터넷 연결로 인해 글로벌 정보 전송은 엄청나게 저렴해졌습니다. 비트 코인과 같은 기술에 뿌리를 둔 움직임은 기본, 합의 메커니즘 및 사회적 계약의 자발적 존중을 통해 인터넷을 사용하여 공유할 수 있는 분산된 가치 전송 시스템을 만드는 것이 가능하다는 것을 보여주었습니다. 사실상 무료로 사용할 수 있습니다. 이 시스템은 암호학적으로 안전한 트랜잭션 기반 상태 머신의 매우 특수화된 버전이라고 할 수 있습니다 .

Namecoin과 같은 후속 시스템은 이 기술의 원래 "통화 응용 프로그램"을 다른 응용 프로그램에 적용했지만 다소 단순합니다.

이더리움은 일반화된 기술 구축을 시도하는 프로젝트입니다 . 모든 트랜잭션 기반 상태 머신 개념이 구축될 수 있는 기술. 또한 최종 개발자에게 지금까지 주류에서 탐색되지 않은 컴퓨팅 패러다임인 신뢰할 수 있는 개체 메시징 컴퓨팅 프레임워크 에 소프트웨어를 구축하기 위해 긴밀하게 통합된 엔드 투 엔드 시스템을 제공하는 것을 목표로 합니다.

1.1. 추진 요인.

이 프로젝트에는 많은 목표가 있습니다 . 한 가지 주요 목표는 서로를 신뢰할 수단이 없는 합의된 개인 간의 거래를 촉진하는 것입니다 . 이는 지리적 분리, 인터페이스 어려움 또는 기존 법률 시스템의 비호환성 , 무능력, 내키지 않음, 비용, 불확실성, 불편 또는 부패 때문일 수 있습니다 . 풍부하고 모호하지 않은 언어를 통해 상태 변경 시스템을 지정 하고 더 나아가 계약이 자율적으로 시행될 것이라고 합리적으로 기대할 수 있는 시스템을 설계함으로써 이를 위한 수단을 제공할 수 있습니다 .

이 제안된 시스템에서의 거래는 현실 세계에서 흔히 볼 수 없는 몇 가지 속성을 가질 것입니다 . 종종 찾기 어려운 부패하지 않는 판단력은 무관심한 알고리즘 해석기에서 자연스럽게 나옵니다. 투명성, 즉 트랜잭션 로그와 규칙 또는 교육 코드를 통해 상태나 판단이 어떻게 발생했는지 정확히 볼 수 있는 것은 자연어가 반드시 모호하기 때문에 인간 기반 시스템에서는 완벽하게 발생하지 않습니다.

종종 부족하고 평범하고 오래된 편견을 흔들기 어렵습니다.

전반적으로 우리는 사용자가 상호 작용 하는 다른 개인 , 시스템 또는 조직에 관계 없이 가능한 결과 와 그 결과가 어떻게 나올지 에 대해 절대적인 확신을 가지고 그렇게 할 수 있도록 보장할 수 있는 시스템을 제공하고자 합니다 .

1.2. 이전 작업.

Buterin[2013a]은 2013년 11월 말에 이 작업의 핵심을 처음 제안했습니다 . 지금은 여러 가지 방식으로 발전했지만 Turing-complete 언어와 실질적으로 무제한인 트랜잭션 간 저장 기능을 갖춘 블록 체인의 핵심 기능은 변경되지 않았습니다.

Dwork와 Naor[1992]는 인터넷을 통해 가치 신호를 전송하는 수단으로 계산 지출의 암호화 증명 ("작업 증명")을 사용하는 첫 번째 작업을 제공했습니다 . 여기서 가치 신호는 통화가 아닌 스팸 억제 메커니즘으로 활용 되었지만 기본 데이터 채널이 강력한 경제적 신호를 전달할 수 있는 잠재력을 비판적으로 보여 주어 수신자가 신뢰에 의존하지 않고도 물리적 주장을 할 수 있도록 합니다. . Back [2002]은 나중에 비슷한 맥락의 시스템을 제작했습니다.

통화를 확보하기 위한 강력한 경제적 신호 로 작업 증명을 사용한 첫 번째 예는 Vish numurthy et al. [2003]. 이 경우 토큰은 P2P 파일 거래를 확인하는 데 사용되어 "소비자"가 서비스에 대해 "공급자" 에게 소액 결제를 할 수 있는 기능을 제공합니다. 작업 증명이 제공하는 보안 모델은 과거 기록이 손상되지 않고 악의적인 행위자가 지불을 속이거나 서비스 제공에 대해 부당하게 불평할 수 없도록 하기 위해 디지털 서명과 원장 으로 강화되었습니다 . 5년 후, Nakamoto [2008]는 범위가 다소 넓은 또 다른 작업 증명 보안 가치 토큰을 소개했습니다 . 이 프로젝트의 결실인 비트코인은 최초로 널리 채택된 글로벌 분산 트랜잭션 원장이 되었습니다.

비트코인의 성공을 기반으로 한 다른 프로젝트들; 알트 코인은 프로토콜 변경을 통해 수많은 다른 통화를 도입했습니다 . 가장 잘 알려진 것 중 일부는 Sprankel [2013]이 논의한 Litecoin과 Primecoin입니다. 다른 프로젝트에서는 프로토콜의 핵심 가치 콘텐츠 메커니즘을 가져와 용도를 변경하려고 했습니다. Aron [2012]은 예를 들어 다음과 같이 논의합니다.

분산된 이름 확인 시스템을 제공하는 것을 목표로 하는 Namecoin 프로젝트 .

다른 프로젝트는 여전히 비트코인 네트워크 자체를 기반으로 시스템 에 배치된 많은 양의 가치 와 합의 메커니즘에 들어가는 방대한 양의 계산을 활용하는 것을 목표로 합니다. Willett[2013]이 처음 제안한 Mastercoin 프로젝트는 핵심 프로토콜에 대한 여러 보조 부품을 활용하여 비트코인 프로토콜 위에 많은 추가 고급 기능을 포함하는 보다 풍부한 프로토콜을 구축하는 것을 목표로 합니다. Rosenfeld et al.이 제안한 Colored Coins 프로젝트. [2012]는 비트코인의 기본 통화의 대체 가능성을 깨고 특별한 "크로마 지갑" 프로토콜 인식 조각을 통해 토큰의 생성 및 추적을 허용하기 위해 거래 규칙을 꾸미는 유사하지만 더 단순화된 전략을 취합니다. 소프트웨어.

탈 중앙화 기반을 폐기하는 영역에서 추가 작업이 수행되었습니다 . Boutellier와 Heinzen[2014] 이 논의한 Ripple은 통화 교환을 위한 "fed erated" 시스템을 만들어 효과적으로 새로운 재정 청산 시스템을 만들 려고 했습니다 . 분산화 전제를 버리면 고효율 이득을 얻을 수 있음을 입증했습니다 .

스마트 계약에 대한 초기 작업은 Szabo[1997]와 Miller[1997]가 수행했습니다. 1990년대 즈음 알고리즘에 의한 계약 집행이 인간 협력에 중요한 힘이 될 수 있다는 것이 분명해졌습니다. 그러한 시스템을 구현하기 위해 특정 시스템이 제안되지는 않았지만 그러한 시스템이 법의 미래에 큰 영향을 미칠 것이라고 제안되었습니다 . 이러한 관점에서 이더리움은 이러한 암호법 시스템의 일반적인 구현 으로 볼 수 있습니다 .

이 백서에 사용된 용어 목록은 부록 A를 참조하십시오.

2. 블록체인 패러다임

이더리움은 전체적으로 트랜잭션 기반 상태 머신으로 볼 수 있습니다. 우리는 초기 상태에서 시작하여 점진적으로 트랜잭션을 실행하여 이를 일부 현재 상태로 변형합니다. 우리가 하는 것은 바로 이 현재 상태입니다.

이더리움 세계의 정식 "버전"으로 받아들입니다.
상태는 계정 잔액, 평판, 신탁 계약, 물리적 세계의 정보와 관련된 데이터와 같은 정보를 포함할 수 있습니다. 요컨대, 현재 컴퓨터로 표현할 수 있는 모든 것이 허용됩니다. 따라서 트랜잭션은 두 상태 사이의 유효한 아크를 나타냅니다 . '유효한' 부분이 중요합니다. 유효한 상태 변경보다 유효하지 않은 상태 변경이 훨씬 더 많이 존재합니다 . 유효하지 않은 상태 변경은 예를 들어 다른 곳에서 동등하고 반대되는 증가 없이 계정 잔액을 줄이는 것과 같은 것일 수 있습니다 .

유효한 상태 전환은 트랜잭션을 통해 발생하는 전환입니다. 공식적으로:

(1) $\sigma_{t+1} \equiv Y(\sigma_t, T)$

여기서 Y는 이더리움 상태 전환 함수입니다. 이더리움 에서 Y는 σ 와 함께 기존의 비교 가능한 시스템보다 훨씬 더 강력합니다 . Y는 구성 요소가 임의의 계산을 수행하도록 허용하는 반면 σ는 구성 요소가 트랜잭션 사이에 임의의 상태를 저장할 수 있도록 합니다 .

트랜잭션은 블록으로 수집됩니다. 블록은 참조 수단으로 암호화 해시를 사용하여 함께 연결됩니다 . 블록은 이전 블록 및 최종 상태에 대한 식별자와 함께 일련의 트랜잭션을 기록하는 저널 역할을 합니다 (최종 상태를 저장하지는 않음).

너무 클 것입니다). 그들은 또한 채굴할 노드에 대한 인센티브로 트랜잭션 시리즈를 강조합니다. 이 인센티브는 지정된 계정에 가치를 추가하는 상태 전환 기능으로 발생합니다 .

마이닝은 다른 잠재적 경쟁자 블록 보다 하나의 일련의 트랜잭션(블록)을 강화 하기 위해 노력(작동)을 쏟는 과정입니다 . 암호학적으로 안전한 증명 덕분에 가능합니다 . 이 체계는 작업 증명으로 알려져 있으며 섹션 11.5에서 자세히 설명합니다.

공식적으로 다음으로 확장합니다.

(2) $\sigma_{t+1} \equiv \Pi(\sigma_t, B)$

(3) $B \equiv (\ldots,(T0, T1, \ldots), \ldots)$ $\Pi(\sigma, B) \equiv$

(4) $\Omega(B, Y(Y(\sigma, T0), T1)\ldots)$

여기서 Ω은 블록 종료 상태 전환 기능입니다.
(지명된 당사자에게 보상하는 기능); B 는 다른 구성 요소 간의 일련의 트랜잭션을 포함하는 이 블록입니다. Π는 블록 수준 상태 전이 함수입니다.

이것은 이더리움뿐만 아니라 지금까지의 모든 탈중앙화 합의 기반 트랜잭션 시스템 의 중추를 형성하는 모델인 블록체인 패러다임의 기초입니다 .

2.1. 값. 네트워크 내에서 계산을 장려하려면 값을 전송하기 위한 합의된 방법이 필요합니다. 이 문제를 해결하기 위해 이더리움에는 고유 통화인 Ether가 있습니다. 이더는 ETH 라고도 하며 때로는 고대 영어 ¯D로 불립니다. Ether 의 가장 작은 단위, 따라서 통화의 모든 정수 값이 계산되는 단위 는 Wei입니다. 1Ether는 1018 Wei 로 정의됩니다 . Ether 의 다른 하위 단위가 있습니다 .

승수 이름
10^0 위
10^{12} 싸보
10^{15} 피니
10^{18} 에테르

현재 작업 전반에 걸쳐 Ether, 통화, 잔액 또는 지불의 맥락에서 가치에 대한 모든 언급은 Wei로 계산되는 것으로 가정해야 합니다.

2.2. 어떤 역사? 시스템이 분산되어 있고 모든 당사자가 일부 오래된 기존 블록에 새 블록을 생성할 수 있는 기회가 있기 때문에 결과 구조는 반드시 블록 트리입니다. 블록체인이라고 하는 이 트리 구조를 통해 루트(제네시스 블록)에서 리프(가장 최근 트랜잭션이 포함된 블록)까지 어떤 경로 에 대한 합의를 형성하기 위해서는 합의된 체계가 있어야 합니다. 블록 트리 아래의 루트-리프 경로가 '최상의' 블록체인인지에 대해 노드 간에 불일치가 있는 경우 포크가 발생합니다.

이는 특정 시점(블록)을 지나면 시스템의 여러 상태가 공존할 수 있음을 의미합니다. 일부 노드는 정식 트랜잭션을 포함하기 위해 하나의 블록에 놓여 있고, 다른 노드는 다른 블록이 정식이라고 믿으며 잠재적으로 근본적으로 다르거나 호환되지 않는 블록을 포함할 수 있습니다. 업무. 불확실성이 전체 시스템에 대한 모든 신뢰를 죽일 가능성이 있기 때문에 이것은 어떤 대가를 치르더라도 피해야 합니다 .

합의를 생성하기 위해 우리가 사용하는 체계는 Sompolinsky와 Zohar[2013] 가 도입한 GHOST 프로토콜의 단순화된 버전 입니다 . 이 프로세스는 섹션 10에 자세히 설명되어 있습니다.

때때로 경로는 특정 높이(블록 번호)에서 새로운 프로토콜을 따릅니다.
이 문서에서는 설명합니다.
프로토콜의 한 버전, 즉 베를린 버전
Beiko et al.에 의해 정의됨. [2021b]. 뒤를 따르기 위해서는
경로의 기록, 여러 버전의 경로를 참조해야 합니다.
이 문서. 다음은 프로토콜의 블록 번호입니다.
Ethereum 메인 네트워크 업데이트:

이름	첫 번째 블록 번호
F홈스테드	1150000
FTangerineWhistle	2463000
FSpuriousDragon	2675000
비잔틴의	4370000
F콘스탄티노플	7280000
F피터스버그	7280000
이스탄불	9069000
FMuirGlacier	9200000
베를린	12244000
런던	12965000
FArrowGlacier	13773000
F화색빙하	15050000

때때로 액터는 프로토콜 변경에 동의하지 않습니다.
영구적인 분기가 발생합니다. 분기된 블록체인 사이를 구별하기 위해
Buterin [2016b]의 EIP-155
우리가 β로 표시하는 체인 ID의 개념을 도입했습니다 .
Ethereum 메인 네트워크의 경우

(5) β = 1

3. 협약

우리는 다음을 위해 다양한 표기법을 사용합니다.
공식 표기법 중 일부는
현재 작업:
고도로 구조화된 두 세트의 '최상위' 상태 값은 굵은 그리스 소문자로 표
시됩니다. 그들
σ (또는 a
이에 대한 변형) 및 기계 상태의 변형, μ.
고도로 구조화된 값에서 작동하는 함수는
그리스 대문자로 표시, 예: Y,
Ethereum 상태 전환 기능.
대부분의 함수에는 대문자(예: C)가 사용됩니다.
일반적인 비용 함수. 이들은 아래 첨자일 수 있습니다.
예를 들어 CSSTORE, SSTORE 작업 의 비용 함수와 같은 특수 변형을 나
타냅니다 . 전문적이고 가능한 경우
외부적으로 정의된 함수, 우리는 타자기로 포맷할 수 있습니다.
텍스트, 예: Keccak-256 해시 함수(버전별)
Bertoni의 SHA-3 콘테스트 우승작 중 3개
외. [2011]은 최종 SHA-3 사양이 아니라
KEC (및 일반적으로 일반 Keccak라고 함) 로 표시됩니다 .
또한 KEC512는 Keccak-512 해시 함수를 나타냅니다.
튜플은 일반적으로 대문자로 표시됩니다.
예를 들어 T는 Ethereum 트랜잭션을 나타내는 데 사용됩니다. 이것
기호는 그에 따라 정의된 경우 아래 첨자로 참조할 수 있습니다.
예를 들어 Tn 과 같은 개별 구성 요소에 대해 nonce를 나타냅니다.
해당 거래의. 첨자의 형태는 다음과 같이 사용됩니다.
유형을 표시하십시오. 예: 대문자 아래 첨자는 튜플을 나타냅니다.
첨자 가능한 구성요소 포함.
스칼라 및 고정 크기 바이트 시퀀스(또는 동의어로,
배열)은 일반적인 소문자로 표시됩니다.
n 은 트랜잭션 논스를 나타내기 위해 문서에서 사용됩니다.
특히 특별한 의미를 가진 것들은 그리스,
예를 들어 , δ는 스택에 필요한 항목의 수입니다.
주어진 작업.
임의 길이 시퀀스는 일반적으로 다음과 같이 표시됩니다.
굵은 소문자, 예: o 는 바이트를 나타내는 데 사용됩니다.
메시지 호출의 출력 데이터로 주어진 시퀀스. 을 위한
특히 중요한 값인 경우 굵은 대문자를 사용할 수 있습니다.
사용할 수 있습니다.
전체적으로 우리는 스칼라가 음수가 아닌 정수라고 가정 하고 따
라서 집합 N에 속합니다. 모든 바이트 집합
시퀀스는 부록 B에 공식적으로 정의된 B 입니다 . 그러한 경우
일련의 시퀀스는 특정 시퀀스로 제한됩니다.
길이, 아래 첨자로 표시되므로 모든 집합
길이가 32인 바이트 시퀀스는 B32 로 명명되고
2256 보다 작은 모든 음수가 아닌 정수 의 이름은 N256입니다 .
이는 섹션 4.3에서 공식적으로 정의됩니다.
대괄호는 색인 및 참조에 사용됩니다.
개별 구성 요소 또는 시퀀스의 하위 시퀀스, 예.
μs [0]은 시스템 스택의 첫 번째 항목을 나타냅니다. 을 위한
하위 시퀀스, 타원은 의도한 것을 지정하는 데 사용됩니다.
두 한계에 있는 요소를 포함하는 범위, 예: μm[0..31]
시스템 메모리의 처음 32개 항목을 나타냅니다.
글로벌 상태 σ의 경우 , 이는 다음의 시퀀스입니다.
계정, 자체 튜플, 대괄호가 사용됨
개인 계정을 참조합니다.
기존 값의 변형을 고려할 때 다음을 따릅니다.
주어진 정의 범위 내에서 우리가
수정되지 않은 '입력' 값은 다음과 같이 표시된다고 가정합니다.
자리 표시자는 수정되고 사용 가능한 값입니다.
&c 로 표시됨 , 중간 값은 ,
니다 . 매우 특별한 경우에 최대화하기 위해
가독성과 의미가 모호하지 않은 경우에만
영숫자 아래 첨자를 사용하여 중간 값을 나타냅니다.
특히 주목할만한 것.
기존 기능의 사용을 고려할 때, 주어진
함수 f, 함수 f 유사한 요소를 나타냅니다.
시퀀스 사이 대신 함수 매핑의 버전.
4.3절에 공식적으로 정의되어 있습니다.
전체적으로 여러 가지 유용한 기능을 정의합니다. 하나
더 일반적인 것은 마지막 항목으로 평가되는 것입니다.
주어진 순서대로:

(6) (x) ≡ x[x 1]

4. 블록, 상태 및 트랜잭션

이더리움의 기본 개념을 소개한 후,
트랜잭션, 블록 및
상태를 더 자세히 설명합니다.

4.1. 세계 상태.

세계 상태(상태)는 주소(160비트 식별자)와 계정 간의 매핑
핑입니다.
상태(RLP로 직렬화된 데이터 구조, 부록 B 참조).
블록체인에 저장되지는 않지만,
구현은 수정 된 Merkle Patricia 트리(트리, 부록 D 참조) 에서 이 매
핑을 유지합니다 . 트라이
바이트 배열과 바이트 배열의 매핑을 유지 관리하는 간단한 데이터베이스 백엔
드가 필요합니다 . 우리는 이것을 기본
데이터베이스 상태 데이터베이스. 여기에는 여러 가지 이점이 있습니다.
먼저 이 구조의 루트 노드는 암호학적으로
모든 내부 데이터에 의존하므로 해시는
전체 시스템 상태에 대한 보안 ID로 사용됩니다.
둘째, 변경할 수 없는 데이터 구조이기 때문에 모든 것을 허용합니다.
리콜할 이전 상태(루트 해시가 알려짐)
그에 따라 루트 해시를 변경하기만 하면 됩니다. 우리 이후로
이러한 모든 루트 해시를 블록체인에 저장하면
간단하게 이전 상태로 되돌립니다.

계정 상태 σ[a]는 다음 4개의 필드로 구성됩니다.

nonce: 이 주소에서 전송된 거래 수 또는 관련 코드가 있는 계정의 경우 이 계정에서 생성된 계약 생성 수와 동일한 스칼라 값입니다 . 상태 σ에 있는 주소 a 의 계정 수에 대해 이것은 σ[a]n 으로 표시된 것입니다 . balance: 이 주소가 소유한 Wei의 수와 동일한 스칼라 값 . 공식적으로는 σ[a]b로 표시됩니다 . storageRoot: 계정의 스토리지 콘텐츠(256비트 정수 값 간의 매핑)를 인코딩하는 Merkle
Patricia 트리 루트 노드의 256비트 해시로, Keccak 256비트 해시의 매핑으로 트리에 인코딩됩니다. 256비트 정수 키를 RLP 인코딩된 256비트 정수 값으로 변환합니다.

해시는 공식적으로 σ[a]s로 표시됩니다. codeHash:
이 항목의 EVM 코드 해시
계정—이 주소가 메시지 호출을 받을 경우 실행되는 코드입니다 . 이러한 모든 코드 조각은 나중에 검색할 수 있도록 해당 해시 아래 상태 데이터베이스에 포함됩니다 . 이 해시는 공식적으로 σ[a]c 로 표시 되므로 KEC(b) = σ[a]c 인 경우 코드를 b 로 표시할 수 있습니다.

우리는 일반적으로 트리의 루트 해시가 아니라 내부에 저장된 키/값 쌍의 기본 세트를 참조하기를 원하므로 편리한 동등성을 정의합니다.

(7) 정렬 L 나 (σ[a]s) ≡ σ[a]s

트리 의 키/값 쌍 집합에 대한 축소 함수 L은 다음과 같이 기본 함수 LI 의 요소별 변환으로 정의됩니다 .

(8) LI (k, v) ≡ KEC(k), RLP(v)

어디:

(9) k ∈ B32 ∧ v ∈ N

σ[a]s는 계정의 '물리적' 구성원이 아니며 나중에 계정에 기여하지 않는다는 점을 이해해야 합니다.
연재.
codeHash 필드가 Keccak-256 해시인 경우
빈 문자열, 즉 σ[a]c = KEC () 이면 노드는 "비계약" 계정이라고도 하는 단순한 계정을 나타냅니다.

따라서 우리는 세계 상태 붕괴 함수 LS를 정의할 수 있습니다.

(10) LS(σ) ≡ {p(a) : σ[a] = ∅}

어디

(11) p(a) ≡ KEC(a), RLP (σ[a]n, σ[a]b, σ[a]s, σ[a]c)

이 함수 LS는 세계 상태의 짧은 ID(해시)를 제공하기 위해 trie 함수와 함께 사용됩니다 . 우리
추정하다:

(12) ∀a : σ[a] = ∅ ∨ (a ∈ B20 ∧ v(σ[a]))

여기서 v는 계정 유효성 함수입니다.

(13) v(x) ≡ xn ∈ N256∧xb ∈ N256∧xs ∈ B32∧xc ∈ B32

코드가 없고 논스가 0 이고 잔액이 0인 계정은 비어 있습니다. (14)

EMPTY(σ, a) ≡ σ[a]c = KEC () ∧σ[a]n = 0∧σ[a]b = 0

호출 가능한 미리 컴파일된 계약도 빈 계정 상태를 가질 수 있습니다. 이는 계정 상태가
일반적으로 동작을 설명하는 코드를 포함합니다.
계정 상태가 존재하지 않으면 계정이 죽은 것입니다.
또는 비어 있음:

(15) DEAD(σ, a) ≡ σ[a] = ∅ ∨ EMPTY(σ, a)

4.2. 트랜잭션. 트랜잭션(공식적으로 T)은 이더리움 범위 외부의 행위자가 구성한 암호화 서명된 단일 명령 입니다. 거래 발신자는 계약일 수 없습니다. 궁극적인 외부 행위자는 본질적으로 인간이 될 것이라고 가정하지만 소프트웨어 도구는 구성 및 보급 에 사용될 것입니다 1 . Zoltu [2020]의 EIP-2718은 다양한 트랜잭션 유형의 개념을 도입했습니다 . 프로토콜 의 베를린 버전에는 0(레거시) 및 1(Buterin 및 Swende [2020b]의 EIP-2930)의 두 가지 트랜잭션 유형이 있습니다 . 또한 트랜잭션에는 두 가지 하위 유형이 있습니다. 메시지 호출을 초래하는 하위 유형과 관련 코드가 있는 새 계정을 생성하는 하위 유형 (비공식적으로 '계약 생성'이라고 함)이 있습니다. 모든 트랜잭션 유형은 여러 공통 필드를 지정합니다.

유형: EIP-2718 거래 유형; 공식적으로 Tx. nonce: 발신자가 보낸 트랜잭션 수와 동일한 스칼라 값입니다 . 공식적으로 Tn. gasPrice: 이 실행 결과로 발생하는 모든 계산 비용에 대해 가스 단위당 지불해야 하는
Wei 수와 동일한 스칼라 값

거래; 공식적으로 Tp. gasLimit: 이
트랜잭션을 실행하는 데 사용해야 하는 가스의 최대 양과 같은 스칼라 값입니다 . 이것은 계산이 완료되기 전에 선불로 지불되며 나중에 증가할 수 없습니다 . 공식적으로 Tg. to: 메시지 호출 수신자의 160비트 주소 또는 계약 생성 트랜잭션의 경우 ∅, 여기에서 B0 의 유일한 구성원을 나타내기 위해 사용됨 ; 공식적으로 Tt.

값: 메시지 호출 수신자에게 전송될 Wei 수와 동일한 스칼라 값 또는 계약 생성의 경우 새로 생성된 계정에 대한 기부금으로 전송됩니다. 정식으로 티비. r, s: 트랜잭션의 서명에 해당하는 값으로 발신자를 결정하는 데 사용됩니다.

거래; 공식적으로 Tr 및 Ts. 이것은 부록 F에 확장되어 있습니다.

EIP-2930(유형 1) 트랜잭션에는 다음도 있습니다.

accessList: 워밍업할 액세스 항목 목록입니다. 말리 TA. 각 액세스 목록 항목 E는 계정 주소와 스토리지 키 목록 E ≡ (Ea, Es) 의 튜플입니다 . chainId: 체인 ID; 공식적으로 Tc. 네트워크 체인 ID β 와 같아야 합니다 . yParity: 서명 Y 패리티; 공식적
으로 타이.

1 특히, 그러한 '도구' 는 궁극적으로 인간에 기반한 시작에서 너무 인과적으로 제거될 수 있습니다 . 예를 들어 계약은 실행을 시작하기 위해 전송된 트랜잭션에 대해 사람 에게 포상금을 제공할 수 있습니다 .

레거시 트랜잭션에는 accessList (TA = ()) 가 없지만 레거시 트랜잭션의 chainId 및 yParity 는 단일 값으로 결합됩니다.

w: Y 패리티 및 가능한 체인 ID를 인코딩하는 스칼라 값; 공식적으로 Tw. Tw = 27 + Ty 또는 Tw = 2β+35+Ty (Buterin [2016b] 의 EIP-155 참조).

또한 계약 생성 트랜잭션(
레거시 또는 EIP-2930)에는 다음이 포함됩니다. init: 계

정 초기화 절차(공식적으로는 Ti) 에 대한 EVM 코드를 지정하는 무제한 크기의 바이트 배열입니다 . init 는 EVM 코드 조각입니다. 계정이 메시지 호출을 받을 때마다 실행

되는 두 번째 코드 조각인 본문을 반환합니다 (트랜잭션을 통해 또는 코드의 내부 실행으로 인해). init은 계정 생성 시 한 번만 실행되며 그 직후 폐기 됩니다 .

반대로 메시지 호출 트랜잭션에는 다음이 포함됩니다.

데이터: 메시지 호출의 입력 데이터(형식적으로 Td)를 지정하는 무제한 크기의 바이트 배열 .

부록 F 는 트랜잭션을 발신자에게 매핑하고 SECP-256k1 곡선의 ECDSA를 통해 발생하는 함수 S를 지정하며 트랜잭션의 해시 (후자의 세 서명 필드 제외)를 서명할 데이터로 사용합니다. 현재 우리는 주어진 트랜잭션 T의 발신자를 S(T)로 나타낼 수 있다고 간단히 주장합니다.

(16)

$$LT(T) \equiv \begin{cases} (Tn, Tp, Tg, Tt, Tv, p, Tw, Tr, Ts) & Tx = 0 \text{ 인 경우} \\ (Tc, Tn, Tp, Tg, Tt, Tv, p, TA, Ty, Tr, Ts) & Tx = 1 \text{ 인 경우} \end{cases}$$

어디

(17)

$$\text{피} \equiv \begin{cases} Ti \text{ if } Tt = \varnothing \\ \text{그렇지 않으면 } Td \end{cases}$$

여기서는 액세스 목록 TA 와 임의 길이 바이트 배열 Ti 및 Td 를 제외한 모든 구성 요소가 RLP에 의해 정수 값으로 해석된다고 가정합니다 .

(18) $Tx \in \{0, 1\} \wedge Tc = \beta \wedge Tn \in N256 \wedge Tp \in N256 \wedge Tg \in N256 \wedge Tv \in N256 \wedge Tw \in N256 \wedge Tr \in N256 \wedge Ts \in N256 \wedge \text{타이} \in N256 \wedge \text{티디} \in B \wedge \text{티} \in B$

어디

(19) $$Nn = \{P : P \in N \wedge P < 2^{N}\}$$

주소 해시 Tt 는 약간 다릅니다. 20바이트 주소 해시이거나 계약 생성 트랜잭션인 경우 (따라서 공식적으로 ∅와 같음) RLP 빈 바이트 시퀀스이므로 B0 의 구성원 입니다. :

(20)

$$TT \in \_ \begin{cases} Tt = \varnothing \text{ 인 경우 } B20 \\ \text{그렇지 않으면 } B0 \end{cases}$$

4.3. 블록. Ethereum의 블록은 수집품입니다.

관련 정보(블록 헤더로 알려짐) H, 구성된 트랜잭션에 해당하는 정보 T 및 현재 블록의 부모의 부모 와 동일한 부모를 갖는 것으로 알려진 다른 블록 헤더 U 집합 (이러한 블록은 ommers2) 로 알려져 있습니다 . 블록 헤더에는 다음과 같은 여러 정보가 포함되어 있습니다.

parentHash: 부모 블록 헤더 전체 의 Keccak 256비트 해시입니다 . 공식적으로 Hp. ommersHash: om의 Keccak 256비트 해시입니다.

이 블록의 mers 목록 부분; 공식적으로 호. 수혜자: 이 블록의 성
공적인 채굴에서 수집된 모든 수수료가 전송되는 160비트 주소입니다 . 공식적으로 Hc.

stateRoot: 루트의 Keccak 256비트 해시

모든 트랜잭션이 실행되고 종료가 적용된 후 상태 트리의 노드 ; 공식적으로 Hr. transactionRoot: 트랜잭션 의 Keccak 256비트 해시

블록 의 트랜잭션 목록 부분에 있는 각 트랜잭션으로 채워진 트리 구조의 루트 노드 ; 공식적으로 Ht. receiptsRoot: 블록의 트랜잭션 목록 부분에 있는 각 트랜잭션의 영수증
으로 채워진 트리 구조의 루트 노드 의 Keccak 256비트 해시입니다 . 공식적으로 그는. logsBloom: 트랜잭션 목록의 각 트랜잭션 수신에서 각 로그 항목에 포함된 인덱스 가능한 정보(로거 주소 및 로그 주제)로 구성된 Bloom 필터 . 공식적으로 Hb.

난이도: 이 블록의 난이도 에 해당하는 스칼라 값입니다 . 이것은 이전 블록의 난이도와 타임스탬프에서 계산할 수 있습니다 . 공식적으로 Hd.

number: 세스터 블록 의 개수와 같은 스칼라 값 . 제네시스 블록의 숫자는 0 입니다. 형식적으로 안녕하세요. gasLimit: 블록당 가스 지출의 현재 제한과 동일한 스칼라 값 . 공식
적으로 Hl. gasUsed: 이 블록의 트랜잭션에 사용된 총 가스와 같은 스칼라 값 입니다 . 공식적으로 Hg. 타임스탬프: 이 블록이 시작될 때 Unix
의 time()의 합당한 출력과 같은 스칼라 값입니다 . 공식적으로 Hs. extraData: 이 블록과 관련된 데이터를 포함하는 임의의 바이트 배열입니다 .
32바이트 이하여야 합니다 . 공식적으로 Hx. mixHash: nonce 와 결합된 256 비트 해시로 이 블록에서 충분한 양의 계산이 수행되었음을 증명합니다. 공식적으로 흠.

nonce: 혼합 해시와 결합된 64비트 값으로 이 블록에서 충분한 양의 계산이 수행되었음을 증명합니다. 공식적으로 Hn.

블록의 다른 두 구성 요소는 단순히 ommer 블록 헤더(위와 동일한 형식)의 목록인 BU 와 일련의 트랜잭션인 BT입니다. 공식적으로 블록 B를 참조할 수 있습니다.

(21) $B \equiv (BH, BT, BU)$

2ommer는 "부모의 형제자매"를 의미하는 성별 중립적 용어입니다. https://nonbinary.miraheze.org/wiki/Gender_neutral_language_in_English#Aunt/Uncle 참조

4.3.1. 거래 영수증. 영지식 증명 또는 색인 및 검색을 형성하는 데 유용할 수 있는 트랜잭션에 대한 정보를 인코딩하기 위해 실행에서 특정 정보를 포함하는 각 트랜잭션의 영수증을 인코딩합니다 . i번째 트랜잭션 에 대해 BR[i]로 표시된 각 영수증은 인덱스 키 트리에 배치되고 루트는 헤더에 He로 기록됩니다.

거래 영수증 R은 거래 유형 Rx, 거래 상태 코드 Rz, 거래가 발생한 직후 거래 영수증을 포함하는 블록에서 사용된 누적 가스로 구성된 5개 항목의 튜플입니다. 발생, Ru, 트랜잭션 실행을 통해 생성된 로그 집합 R1 및 해당 로그의 정보로 구성된 Bloom 필터 Rb:

(22) $$R \equiv (Rx, Rz, Ru, Rb, Rl)$$

Rx 는 해당 트랜잭션의 유형과 동일합니다.
LR 함수는 거래 영수증을 준비합니다.
RLP 직렬화 바이트 배열로 변환:

(23) $$LR(R) \equiv (Rz, Ru, Rb, R1)$$

상태 코드 Rz가 음수가 아닌 정수임을 확인합니다 .

(24) $$Rz \in N_{-}$$

우리는 사용된 누적 가스 Ru가 음수 가 아닌 정수이고 로그 Bloom Rb가 2048비트(256바이트) 크기 의 해시라고 주장합니다 .

(25) $$루 \in N \wedge Rb \in B256$$

시퀀스 R1 은 일련의 로그 항목 (O0, O1, ...)입니다.
로그 항목 O는 로거 주소 Oa, 비어 있을 수 있는 32바이트 로그 항목 시리즈 Ot 및 몇 바이트의 데이터 Od의 튜플입니다.

(26) $$O \equiv (Oa,(Ot0, Ot1, ...), Od)$$

(27) $$Oa \in B20 \wedge \forall x \in Ot : x \in B32 \wedge Od \in B$$

로그 항목을 단일 256바이트 해시로 줄이기 위해 Bloom 필터 함수 M을 정의합니다 .

(28) $$M(O) \equiv \quad x \in \{Oa\} \cup Ot \; M3{:}2048(x)$$

여기서 M3:2048은 임의의 바이트 시퀀스가 주어지면 2048에서 3비트를 설정하는 특수 블룸 필터입니다 . 이것은 바이트 시퀀스 의 Keccak-256 해시에서 처음 세 바이트 쌍 각각의 하위 11비트를 취함으로써 수행됩니다 .3 공식적으로:

(29) $$M3{:}2048(x : x \in B) \equiv y : y \in B256$$ 여기서: (30) $$y = (0, 0, ..., 0)$$ 제외: (31) $$\forall i \in \{0, 2, 4\} : B2047 \quad m(x,i)(y) = 1$$ (32)

$$m(x, i) \equiv KEC(x)[i, i + 1] \bmod 2048$$

여기서 B 는 Bj(x)가 바이트 배열 x 에서 인덱스 j 의 비트 (0부터 인덱싱됨) 와 같은 비트 참조 함수입니다 .
특히 x를 big-endian으로 취급합니다 (중요 비트가 많을수록 인덱스가 작아짐).

4.3.2. 전체적인 타당성. 블록이 몇 가지 조건을 충족하는 경우에만 블록의 유효성을 주장할 수 있습니다 . 기본 상태 σ에서 순서대로 실행될 때 ommer 및 트랜잭션 블록 해시와 주어진 트랜잭션 BT (섹션 11에 지정된 대로)와 내부적으로 일관성이 있어야 합니다 . (부모 블록의 최종 상태에서 파생됨), ID Hr의 새로운 상태가 됩니다.

(33)
$$\text{시간} \equiv TRIE(LS(\Pi(\sigma, B))) \quad \wedge$$
$$Ho \equiv KEC(RLP(L \quad H(BU))) \quad \wedge$$
$$Ht \equiv TRIE(\{\forall i < BT, i \in N : pT(i, BT[i])\}) \quad \wedge$$
$$He \equiv TRIE(\{\forall i < BR, i \in N : pR(i, BR[i])\}) \quad \wedge$$
$$Hb \equiv \quad r \in BR \; {}^{rb}$$

여기서 pT(k, v) 및 pR(k, v)는 쌍별 RLP 변환 이지만 EIP-2718 트랜잭션에 대한 특별 처리가 있습니다 .

(34)
$$pT(k, T) \equiv RLP(k), \quad \begin{cases} Tx = 0 \text{ 인 경우 } RLP(LT(T)) \\ (Tx) \cdot RLP(LT(T)) \text{ 그렇지 않은 경우} \end{cases}$$

그리고

(35)
$$pR(k, R) \equiv RLP(k), \quad \begin{cases} Rx = 0 \text{ 인 경우 } RLP(LR(R)) \\ (Rx) \cdot RLP(LR(R)) \text{ 그렇지 않은 경우} \end{cases}$$

(·는 바이트 배열의 연결입니다).
뿐만 아니라:

(36) $$TRIE(LS(\sigma)) = P(BH)\text{시간}$$

따라서 TRIE(LS(σ)) 는 RLP를 사용하여 인코딩된 값과 상태 σ 의 키-값 쌍을 포함하는 Merkle Patricia 트리 구조의 루트 노드 해시 이고 P(BH) 는 직접 정의된 B의 부모 블록 입니다 .

트랜잭션의 계산 , 특히 트랜잭션 영수증 BR 과 트랜잭션의 상태 누적 함수 Π를 통해 정의된 값은 나중에 섹션 11.4에서 공식화됩니다.

4.3.3. 직렬화. 함수 LB 및 LH는 각각 블록 및 블록 헤더에 대한 준비 함수 입니다 .
RLP 변환이 필요한 경우 구조의 유형과 순서를 주장합니다 .

(37) $$LH(H) \equiv (Hp, Ho, Hc, Hr, Ht, He, Hb, Hd, \text{안녕}, Hl, Hg, Hs, Hx, Hm, Hn)$$

(38) $$LB(B) \equiv LH(BH), L^* T(BT), L^* H(BU)$$

여기서 LT는 EIP-2718 트랜잭션을 특별히 관리합니다.

(39) $$LT(\text{티}) = \begin{cases} Tx = 0 \text{ 인 경우 } LT(T) \\ (Tx) \cdot RLP(LT(T)) \text{ 그렇지 않은 경우} \end{cases}$$

L* 포함 티 그리고 패 H 는 요소별 시퀀스 변환입니다.
따라서: (40)
(x0,
예쁘 x1, ...) ≡ f(x0), f(x1), ... 모든 함수 f에 대해

32048 = 211(11비트), 하위 11비트는 피연산자의 모듈로 2048이며, 이 경우 "바이트 시퀀스의 Keccak-256 해시에서 처음 세 바이트 쌍 각각"입니다.

구성요소 유형은 다음과 같이 정의됩니다.

(41) 마력 ∈ B32 ∧ 호 ∈ B32 ∧ HC ∈ B20 ∧
시 ∈ B32 ∧ Ht ∈ B32 ∧ 헤 ∈ B32 ∧
Hb ∈ B256 ∧ Hd ∈ N ∧ 하이 ∈ N ∧
Hl ∈ N ∧ Hg ∈ N ∧ Hs ∈ N256 ∧
Hx ∈ B ∧ Hm ∈ B32 ∧ Hn ∈ B8

어디

(42) Bn = {B : B ∈ B ∧ B = n}

우리는 이제 공식적인 블록 구조의 구성에 대한 엄격한 사양을 갖게 되었습니다 . RLP 함수 RLP
(부록 B 참조)는 이 구조를 유선으로 전송하거나 로컬에 저장할 준비가 된 일련의 바이트로 변환하기 위한 정식 방법을 제공합니다 .

4.3.4. 블록 헤더 유효성. 우리는 P(BH)를 B의 상위 블록 으로 정의합니다 .

(43) P(H) ≡ B : KEC(RLP(BH)) = Hp

블록 번호는 1씩 증가한 상위 블록 번호입니다 .

(44) 하이 ≡ P(H)하이 + 1

헤더 H 블록의 정규 난이도는 D(H)로 정의됩니다 : (45)

$$\text{디}(H) \equiv \begin{cases} 2^{34} & \text{안녕} = 0 \text{ 인 경우} \\ \text{최대 Dmin, P(H)Hd} + x \times 2 + \text{그렇지 않은 경우} \end{cases}$$

어디:

(46) 분 2 17

(47) $x \equiv \frac{P(H)Hd}{2048}$

(48) 2 ≡ 최대 y $\frac{Hs - P(H)Hs\ 9}{}$, -99

(49) $\text{그리고} \equiv \begin{cases} P(H)U = 0 \text{ 인 경우 } 1 \\ \text{그렇지 않으면 } 2 \end{cases}$

(50) $\equiv 2^{\,H \div 100000 \quad 2\,i}$

i ≡ 최대(Hi - κ, 0)(51)

(52)

$$k \equiv \begin{cases} \text{FByzantium Hi} < \text{FConstantinople 인 경우 } 3000000 \\ \text{FConstantinople Hi} < \text{FMuirGlacier 인 경우 } 5000000 \\ \text{FMuirGlacier Hi} < \text{FLondon 인 경우 } 9000000 \\ \text{FLondon Hi} < \text{FArrowGlacier 인 경우 } 9700000 \\ \text{FArrowGlacier Hi} < \text{FGrayGlacier 인 경우 } 10700000 \\ 11400000 \text{ 하이 FGrayGlacier 인 경우} \end{cases}$$

홈스테드 난이도 매개변수 2는 Buterin [2015]의 EIP-2에서 구현된 것처럼 아래에서 설명하는 것처럼 블록 사이의 시간이 변하기 때문에 블록 사이의 시간의 동적 항상성에 영향을 미치는 데 사용됩니다 . 홈스테드 릴리스에서 지수 적 난이도 기호는 난이도를 기하급수적으로 천천히 증가(100,000 블록마다)하여 블록 시간 차이를 증가시키고 지분 증명으로 전환하는 데 시간 압박을 가합니다.

"어려움 폭탄" 또는 "아이스 에이지"로 알려진 이 효과는 Schoedon과 Buterin [2017]이 EIP-649에서 설명했으며 EIP-2에서 더 일찍 지연되고 구현되었습니다. 2는 또한 EIP-100에서 위의 조정 계수인 x와 분모 9를 사용하여 수정되었습니다.

Buterin [2016a]의 삼촌 블록을 포함한 평균 블록 시간 . Byzantium 릴리스에서는 EIP-649를 사용하여 실제 블록 번호에서 3백만을 뺀 가짜 블록 번호 H 를 생성하여 빙하기를 지연시켰습니다. 지분 증명을 개발하고 네트워크가 "정지"되는 것을 방지하는 데 더 많은 시간을 허용합니다 . 그 후, Schoedon [2018]의 EIP-1234 , Conner [2019]의 EIP-2384, Hancock [2021] 의 EIP-3554 , Beiko et al.의 EIP-4345. [2021a] 및 Stanczak 등의 EIP-5133. 서브트라헨드 κ를 증가시켰다.

헤더 H 블록의 캐노니컬 가스 한계 Hl은 다음 과 같아야 합니다.
관계를 이행하십시오:

(53) $H1 < P(H)H1 + \frac{P(H)H1}{1024} \quad \wedge$

$Hl > P(H)Hl - \frac{P(H)Hl}{1024} \quad \wedge$

5000 힐

Hs 는 블록 H 의 타임스탬프(Unix의 time()에서) 이며 다음 관계를 충족해야 합니다.

(54) Hs > P(H)Hs

이 메커니즘은 다음과 같은 측면에서 항상성을 시행합니다.
블록 사이의 시간; 마지막 두 블록 사이의 기간이 짧으면 난이도가 높아져 추가 계산이 필요하므로 다음 기간이 길어집니다. 반대로 기간이 너무 길면 난이도와 다음 블록까지의 예상 시간이 줄어듭니다.

nonce Hn은 다음 관계를 충족해야 합니다.

(55) $N \quad \frac{2^{256}}{\text{고화질}} \quad \wedge\ m = \text{흠}$

(n, m) = PoW(Hn, Hn, d).

여기서 Hn 은 새 블록의 헤더 H이지만 nonce 및 혼합 해시 구성 요소가 없는 경우 d 는 혼합 해시를 계산하는 데 필요한 대규모 데이터 세트인 현재 DAG이고 PoW 는 작업 증명 기능입니다(섹션 참조). 11.5): 첫 번째 항목이 혼합 해시인 배열로 평가되어 올바른 DAG가 사용되었음을 증명하고 두 번째 항목은 H 및 d에 암호화된 의사 난수입니다 . [0, 2 64) 범위의 대략 균일한 분포가 주어 지면 솔루션을 찾는 데 걸리는 예상 시간은 난이도 Hd에 비례합니다.

이것이 블록체인 보안의 기초 이며 악의적인 노드가 기록을 덮어쓰는("재작성") 새로 생성된 블록을 전파 할 수 없는 근본적인 이유입니다 . nonce는 이 요구 사항을 충족해야 하고 그 만족은 블록의 내용과 구성된 트랜잭션에 따라 달라지기 때문에 새롭고 유효한 블록을 생성하기 어렵고,

시간이 지남에 따라 마이닝 피어의 신뢰할 수 있는 부분의 대략적인 총 컴퓨팅 성능이 필요합니다 .

따라서 블록 헤더 유효성 함수 V(H)를 정의할 수 있습니다 .

$$
\begin{aligned}
(56)\quad V(H) \equiv\ & n \quad \frac{2^{256}}{\text{고화질}} \quad \wedge m = \text{흠} \wedge \\
& Hd = \text{디}(H) \wedge \\
& HG \leq HL \wedge \\
& H1 < P(H)H1 + \frac{P(H)Hl}{1024} \quad \wedge \\
& Hl > P(H)Hl - \frac{P(H)H1}{1024} \quad \wedge \\
& 5000 \wedge \\
& Hs > P(H)Hs \wedge \\
& \text{하이} = P(H)\text{하이} + 1 \wedge \\
& \text{높이} \leq 32
\end{aligned}
$$

여기서 (n, m) = PoW(Hn, Hn, d)

추가로 extraData는 최대 32바이트여야 합니다.

5. 가스 및 결제

네트워크 남용 문제를 피하고 Turing 완전성에서 비롯된 불가피한 질문을 피하기 위해 Ethereum의 모든 프로그래밍 가능한 계산에는 수수료가 부과됩니다. 요금표는 가스 단위로 지정됩니다(다양한 계산과 관련된 요금은 부록 G 참조).

따라서 프로그래밍 가능한 계산의 특정 부분(계약 생성, 메시지 호출, 계정 스토리지 활용 및 액세스, 가상 머신에서 작업 실행 포함)은 가스 측면에서 보편적으로 합의된 비용을 갖습니다.

모든 트랜잭션에는 관련된 특정 양의 가스 (gasLimit)가 있습니다. 발신자의 계정 잔액에서 암묵적으로 구매한 가스의 양입니다 .

구매는 트랜잭션에 지정된 해당 gasPrice 에서 발생합니다. 계정 잔액이 그러한 구매를 지원할 수 없는 경우 거래가 유효하지 않은 것으로 간주됩니다. 트랜잭션이 끝날 때 사용하지 않은 가스가 발신인의 계정으로 환불(동일한 구매 비율로)되기 때문에 이름이 gasLimit 입니다 . 가스는 거래 실행 외에는 존재하지 않습니다. 따라서

신뢰할 수 있는 코드가 연결된 계정에서는 상대적으로 높은 가스 한도를 설정하고 그대로 둘 수 있습니다.

일반적으로 가스 구매에 사용된 이더는 환불되지 않고 수취인 주소로 전달되며, 일반적으로 채굴자가 관리하는 계정의 주소입니다 .

거래자는 원하는 가스 가격을 자유롭게 지정할 수 있지만 채굴자는 거래를 무시할 수 있습니다 . 따라서 트랜잭션의 가스 가격이 높을수록 발신자에게 Ether 측면에서 더 많은 비용이 들고 광부에게 더 큰 가치를 제공하므로 더 많은 광부가 포함하도록 선택될 가능성이 더 큽니다. 일반적으로 채굴자는 거래를 실행할 최소 가스 가격을 광고하기로 선택 하고 거래자는 제공 할 가스 가격을 결정할 때 이러한 가격을 자유롭게 캔버스에 표시할 수 있습니다 . 허용 가능한 최소 가스 가격의 (가중) 분배가 있기 때문에 거래자는 가스 가격을 낮추는 것과 거래가 적시에 채굴 될 가능성을 최대화하는 것 사이에서 반드시 절충해야 합니다.

6. 거래 실행

트랜잭션 실행은 이더리움 프로토콜에서 가장 복잡한 부분입니다. 상태 전환 함수 Y를 정의합니다. 먼저 실행된 트랜잭션은 내재적 유효성의 초기 테스트를 통과한다고 가정합니다 . 여기에는 다음이 포함됩니다.

(1) 트랜잭션이 후행 바이트가 추가되지 않은 올바른 형식의 RLP입니다. (2) 트랜잭션 서명이 유효합니다. (3)
거래 임시값이 유효합니다(송신자 계정의 현재 임시값과
동일). (4) 발신자 계정에 배포된 계약 코드가 없습니다 (Feist et al. [2021]의 EIP-3607 참조). (5) 가스 한도는 거래에 사용
된 고유 가스 g0 보다 작지 않습니다. (6) 발신자 계정 잔액에는 적어도 선불 결제에 필요한 비용 v0이 포함됩니다.

공식적으로 우리는 T 가 a인 함수 Y를 고려합니다.
트랜잭션 및 σ 상태:

$$(57)\qquad \sigma = Y(\sigma, T)$$

따라서 σ 는 트랜잭션 후 상태입니다. 또한 Yg 는 트랜잭션 실행에 사용된 가스의 양으로 평가하고 , Yl은 트랜잭션의 누적된 로그 항목으로 평가하고, Yz 는 트랜잭션 결과 상태 코드로 평가합니다 . 이들은 나중에 공식적으로 정의될 것입니다.

6.1. 하위 상태. 트랜잭션 실행을 통해 트랜잭션 직후에 실행되는 특정 정보가 발생합니다 . 이를 누적 트랜잭션 하위 상태 또는 줄여서 누적 하위 상태라고 하며 튜플인 A로 나타냅니다.

$$(58)\qquad A \equiv (As, Al, At, Ar, Aa, AK)$$

튜플 내용에는 트랜잭션 완료 후 폐기될 계정 집합인 자폭 집합인 As가 포함됩니다 . Al 은 로그 시리즈입니다. 이것은 이더리움 세계 외부의 구경꾼(예: 분산형 애플리케이션 프런트 엔드)이 계약 호출을 쉽게 추적할 수 있도록 하는 VM 코드 실행에서 일련의 보관 및 인덱싱 가능한 '체크포인트'입니다. At 는 거래가 끝나면 빈 계정이 삭제되는 터치된 계정 집합입니다. Ar 은 계약 스토리지를 0이 아닌 값에서 0 으로 재설정하기 위해 SSTORE 명령을 사용하여 증가한 환불 잔액입니다 . 즉시 환급되지는 않지만 전체 실행 비용의 일부를 상쇄할 수 있습니다 . 마지막으로 Buterin과 Swende[2020a]의 EIP-2929는 액세스된 계정 주소 집합인 Aa 와 액세스된 스토리지 키 집합인 AK를 도입했습니다 (더 정확하게는 AK 의 각 요소는 20바이트 계정 주소의 튜플이며 32바이트 스토리지 슬롯).

우리는 빈 발생 하위 상태 A^0를 자체 파괴, 로그 없음 , 손대지 않은 계정 없음, 환불 잔액 없음 , 액세스된 주소의 모든 사전 컴파일된 계약 및 액세스된 스토리지 없음으로 정의합니다.

$$(59)\qquad \vdash^0 \equiv (\varnothing, (), \varnothing, 0, \pi, \varnothing)$$

여기서 π는 미리 컴파일된 모든 주소의 집합입니다.

6.2. 실행. 거래가 실행되기 전에 지불해야 하는 가스 의 양인 고유 가스 g0를 다음과 같이 정의합니다 .

(60)

g0 ≡ Gtxdateevery 내가 = 0인 경우
i∈Ti ,Td 그렇지 않으면 Gtxdatanonzero
+ Gtxcreate if Tt = ∅
0 그렇지 않으면
+ 지트랜잭션
TA-1
+ Gaccesslistaddress + TA[j]sGaccessliststorage
j=0

여기서 Ti, Td는 트랜잭션이 계약 생성용인지 메시지 호출용인지에 따라 트랜잭션 관련 데이터 및 초기화 EVM 코드 의 일련의 바이트를 의미합니다 . Gtxcreate 는 거래가 계약 생성인 경우 추가되지만 EVM 코드의 결과인 경우에는 추가되지 않습니다. Gaccesslistaddress 및 Gaccessliststorage 는 각각 계정 워밍업 비용 과 스토리지 액세스 비용입니다. G는 부록 G 에 완전히 정의되어 있습니다.

선행 비용 v0은 다음과 같이 계산됩니다.

(61) v0 ≡ TgTp + TV

유효성은 다음과 같이 결정됩니다.

(62) S(T) = ∅ ∧ σ[S(T)]c = KEC() ∧
Tn = σ[S(T)]n ∧
g0 Tg ∧
v0 σ[S(T)]b ∧
TG BH1 - (BR)u

최종 조건에 유의하십시오. 트랜잭션의 가스 한도 Tg 와 (BR)u 로 주어진 이전 블록에서 사용된 가스의 합은 블록의 gasLimit BHl 보다 크지 않아야 합니다 . 또한 표기법을 약간 남용하여 σ[S(T)]c = KEC ( ) , σ[S(T)]n = 0, σ[S(T)]b = 0 S(T)] = ∅.

유효한 트랜잭션의 실행은 취소할 수 없는 상태 변경으로 시작됩니다. 보낸 사람 계정의 nonce인 S(T) 는 1씩 증가하고 잔액은 선결제 비용인 TgTp의 일부만큼 줄어듭니다. 진행 계산에 사용할 수 있는 가스 g 는 Tg g0 로 정의됩니다 . 계약 생성이든 메시지 호출이든 상관없이 계산 결과 최종 상태(법적으로 현재 상태와 동일할 수 있음), 결정적이며 절대 유효하지 않은 변경이 발생합니다 . 이 시점부터 유효하지 않은 트랜잭션이 있을 수 없습니다 .

체크포인트 상태 σ0을 정의합니다 .

(63) σ0 ≡ σ 제외:
(64) σ0[S(T)]b ≡ σ[S(T)]b TgTp σ0[S(T)]n ≡
(65) σ[S(T)]n + 1

σ0 에서 σP를 평가하는 것은 트랜잭션 유형에 따라 다릅니다. 계약 생성 또는 메시지 호출; 실행 후 임시 상태 σP, 나머지 가스 g 의 튜플을 정의합니다 .

발생한 하위 상태 A 및 상태 코드 z: (66)

(σP, g , A, z) ≡ Λ4(σ0, A , S(T), S(T), g, Tp, Tv, Ti, 0, ∅,) Tt = ∅ 인 경우
Θ4(σ0, A* , S(T), S(T), Tt, Tt, g, Tp, Tv, Tv, Td, 0,) 그렇지 않은 경우

어디

제외(67) (68) ⊦ 0 ≡ A
⊦ ⊦ 0 ≡ A ⊦ ∪ {S(T)} ∪E∈TA {Ea} ∀i < Es, i

(69) ⊦ K ≡ ∈ N : (Ea, Es[i])
E∈TA

g 는 트랜잭션 존재에 대한 지불에 필요한 기본 금액을 공제한 후 남은 가스의 양입니다 .

(70) g ≡ Tg - g0

Θ4 및 Λ4를 사용하여 함수 값의 처음 4개 구성 요소 만 취한다는 사실을 나타냅니다 . 최종은 메시지 호출의 출력 값(바이트 배열)을 나타내며 트랜잭션 평가 컨텍스트에서 사용되지 않습니다 .

메시지 호출 또는 계약 생성이 처리된 후 해당 계정에 대한 환불 카운터를 증가시켜야 합니다.
호출 내내 자폭했습니다.

(71) ⊦ ≡ + 와 함께 Rselfdestruct
i∈As

그런 다음 상태 는 남은 가스 g에서 환불 카운터의 약간의 허용량을 원래 비율로 발신자에게 환불할 금액을 결정하여 확정됩니다 .

(72) g ≡ g + 분 Tg -g2 , ⊦

환불 가능한 총 금액은 Ar 에 추가된 합법적으로 남아 있는 가스 g 이며 후자의 구성 요소는 사용된 총량 Tg - g 의 최대 절반(내림)으로 제한됩니다 . 따라서 g 는 트랜잭션이 실행된 후 남아 있는 총 가스입니다 .

가스에 대한 Ether는 현재 블록 B의 수취인으로 주소가 지정된 채굴자에게 제공됩니다. 따라서 임시 상태 σP 로 사전 최종 상태 σ를 정의합니다 .

(73) 피 ≡ σP 제외
(74) 피 [S(T)]b ≡ σP[S(T)]b + g 도시
(75) 피 [m]b ≡ σP[m]b + (Tg - gm ≡ BHc)Tp
(76)

최종 상태 σ , self-destruct 세트에 나타나거나 터치되고 비어 있는 모든 계정을 삭제한 후에 도달합니다 .

(77) σ ≡ σ 제외하고
(78) As : σ ∀i ∈ [i] = ∅ ∀i ∈
(79) At : σ [i] = ∅ if DEAD(σ , 나)

그리고 마지막으로, 우리는 이 트랜잭션에 사용된 총 가스 Yl 이 트랜잭션에 의해 생성된 로그를 지정하고

와이'', 이 트랜잭션의 상태 코드:

(80) Yg (σ, T) ≡ Tg - g

(81) 내가ㅏ (σ, T) ≡ Al (σ, T) ≡

(82) 와이'' z

거래 영수증을 정의하는 데 사용됩니다.
또한 나중에 상태 및 nonce 유효성 검사에도 사용됩니다.

7. 계약 생성

계정을 생성할 때 사용되는 여러 가지 고유 매개변수가 있습니다 : 송신자 (s), 원래 거래자4 (o), 사용 가능한 가스 (g), 가스 가격 (p), 기부금 (v) 및 임의의 길이 바이트 배열 i, 초기화 EVM 코드, 메시지 호출/계약 생성 스택 의 현재 깊이 (e), 새 계정 주소의 솔트 (ζ) , 마지막으로 상태를 수정할 수 있는 권한 (w). 소금 ζ가 누락되었을 수 있습니다(ζ = ∅). 공식적으로

(83) ζ ∈ B32 ∪ B0

생성이 CREATE2 에 의해 발생한 경우 ζ = ∅입니다.

우리는 공식적으로 생성 함수를 함수 Λ로 정의합니다 . 이 값은 상태 σ 및 발생한 하위 상태 A 와 함께 이러한 값에서 새 상태, 남은 가스, 새로 발생한 하위 상태, 상태 코드 및 출력 (σ , g , , z, o):

ㅏ

(84) (σ , g , A, z, o) ≡ Λ(σ, A, s, o, g, p, v, i, e, ζ, w)

새 계정의 주소는 보낸 사람과 계정 난스 만 포함하는 구조의 RLP 인코딩의 Keccak-256 해시의 가장 오른쪽 160비트로 정의됩니다 . CREATE2 의 경우 규칙이 다르며 Buterin [2018]의 EIP-1014에 설명되어 있습니다. 두 가지 경우를 결합하여 새 항목에 대한 결과 주소를 정의합니다.

계정 a:

a ≡ ADDR(s, σ[s]n - 1, ζ, i)(85)

(86) ADDR(s, n, ζ, i) ≡ B96..255 KEC LA(s, n, ζ, i)

(87) LA(s, n, ζ, i) ≡ RLP (s, n) if ζ = ∅ (255) · s · ζ · KEC(i) 아니면

여기서 · 는 바이트 배열의 연결이고, Ba..b(X)는 이진 데이터 X 의 [a, b] 범위 에 있는 인덱스의 비트를 포함하는 이진 값으로 평가되며 , σ[x] 는 주소 상태 입니다. x 의 , 또는 존재하지 않는 경우 ∅ . 보낸 사람의 nonce 값 보다 하나 적은 값을 사용합니다 . 우리는 이 호출 이전에 발신자 계정의 nonce를 증가시켰다고 주장하므로 사용된 값은 책임 있는 트랜잭션 또는 VM 작업 시작 시 발신자의 nonce입니다 .

새 계정의 주소가 세트에 추가됩니다.
액세스한 계정:

(88) ㅏ ≡ A 제외 A ㅏ ≡ 아 ∪ {a}

계정의 nonce는 처음에 1로 정의되고 잔액은 전달된 값으로, 저장소는 비어 있고 코드 해시는 빈 문자열의 Keccak 256비트 해시로 정의됩니다. 보낸 사람의 잔액도 전달된 값만큼 줄어듭니다.

따라서 돌연변이 상태는 σ :

(89) 피 ≡ σ 제외:

(90) 피 [a] = 1, v + v , TRIE(∅), KEC() ∅ if

(91) 피 [s] = σ[s] = ∅ ∧ v = 0
ㄱ 그렇지 않으면

(92) ㄱ ≡ (σ[s]n, σ[s]b - v, σ[s]s, σ[s]c)

여기서 v 는 계정이 이전에 존재했던 경우의 기존 값입니다 .

(93) ≡에서 0 σ[a] = ∅
σ[a]b 이면 그렇지 않으면

마지막으로 실행 모델에 따라 초기화 EVM 코드 i 의 실행을 통해 계정이 초기화됩니다 (섹션 9 참조). 코드 실행은 실행 상태 내부에 있지 않은 여러 이벤트에 영향을 미칠 수 있습니다.

계정의 저장소를 변경할 수 있고 추가 계정을 만들 수 있으며 추가 메시지 호출을 할 수 있습니다. 따라서 코드 실행 함수 Ξ는 결과 상태 σ , 남은 가용 가스 g , 결과로 발생한 하위 상태 A 및 본체 코드의 튜플로 평가됩니다.

계정 오.

(94) (피 , g , A , o) ≡ Ξ(p , g, A* , 나)

여기서 I는 실행 환경의 매개변수를 포함합니다 .

(95) ≡ 로
(96) 나 ≡ 오
(97) IP ≡ 피
(98) 아이디 ≡ ()
(99) is ≡ s
(100) IV ≡ v
(101) Ib ≡ i
(102) 즉 ≡ e
(103) Iw ≡ w

이 호출에 대한 입력 데이터가 없으므로 Id는 빈 튜플로 평가됩니다. IH는 특별한 처리가 없으며 블록체인에서 결정됩니다.

코드 실행은 가스를 고갈시키고 가스는 0 아래로 떨어지지 않을 수 있으므로 코드가 자연적으로 정지 상태가 되기 전에 실행이 종료될 수 있습니다. 이(및 기타 여러) 예외적인 경우에서 우리는 out-of-gas(OOG) 예외가 발생했다고 말합니다. 평가된 상태는 빈 세트, ∅로 정의되며 전체 생성 작업은 상태에 영향을 미치지 않아야 합니다. , 생성을 시도하기 직전 의 상태로 효과적으로 유지합니다 .

초기화 코드가 성공적으로 완료되면 생성된 계약 코드의 크기에 비례하는 코드 보증금 비용 c인 최종 계약 생성 비용이 지불됩니다.

(104) c ≡ Gcodedeposit × o

이를 지불할 만큼 남은 가스가 충분하지 않은 경우(즉, < c) 가스 부족 예외도 선언
g 합니다.
이러한 예외적인 조건에서 남은 가스는 0이 됩니다 . 즉, 생성이 트랜잭션 수신으로 수행된 경우 이는 계약 생성의 내재 비용 지불에 영향을 미치지 않습니다. 관계없이 지급됩니다.

단, 거래대금은 양도되지 않습니다.

4트랜잭션에 의해 직접 트리거되지 않고 들어오는 메시지 호출 또는 계약 생성의 경우 발신자와 다를 수 있습니다.
EVM 코드 실행에서

가스가 없을 때 중단된 계약의 주소로 보내 므로 계약의 코드가 저장되지 않습니다.

이러한 예외가 발생하지 않으면 나머지 가스는 생성자에게 환불되고 현재 변경된 상태가 지속되도록 허용됩니다. 따라서 공식적으로 결과 상태, 가스, 발생한 하위 상태 및 상태 코드를 (σ , g , A , z) 로 지정할 수 있습니다 .

(105)

$$g \equiv \begin{cases} 0 & \text{만약 } F \\ g & \text{c 그렇지 않으면} \end{cases}$$

(106)

$$\sigma \equiv \begin{cases} \text{피} & F \vee p \text{ 인 경우} \quad = \varnothing \\ \text{피} & \text{제외: } \sigma \\ & [a] = \varnothing \text{ 제외:} \quad \text{죽은 경우}(\sigma \quad , a) \\ & p \quad \sigma\,[a]c = \\ & \text{KEC(o) 그렇지 않은 경우} \end{cases}$$

(107)

$$A \equiv \begin{cases} \text{ㅏ} & F \vee p \text{ 인 경우} \quad = \varnothing \\ \text{ㅏ} & \text{그렇지 않으면} \end{cases}$$

(108)

$$\equiv \text{포함} \begin{cases} F \vee p \text{ 인 경우 } 0 & = \varnothing \\ \text{그렇지 않으면 } 1 \end{cases}$$

어디

(109)

$$F \equiv \sigma[a] = \varnothing \wedge \sigma[a]c = KEC\,() \vee \sigma[a]n = 0 \vee (\text{피} \quad = \varnothing \wedge \text{또는} = \varnothing) \vee g \quad < c \vee o > 24576$$

σ 결정의 예외는 초기화 코드 실행의 결과 바이트 시퀀스 인 o 가 새로 생성된 계정의 최종 본문 코드를 지정한다는 것을 나타냅니다.

의도는 결과가 성공이거나

기부금이 포함된 새로운 계약을 끊임없이 생성하거나 가치 이전 없이 새로운 계약을 맺지 않습니다. 또한 초기화 코드의 실행이 되돌리면 (σ = ∅ ∧ o = ∅) 결과 가스 g 는 고갈되지 않지만(다른 예외가 없는 경우) 새 계정이 생성되지 않습니다.

7.1. 미묘함. 초기화 코드가

실행 중이면 새로 생성된 주소가 존재하지만 고유 본문 코드는 없습니다5 . 따라서 이 시간 동안 받은 메시지 호출로 인해 코드가 실행되지 않습니다. 초기화 실행이 SELFDESTRUCT 로 끝나는 경우

지시에 따라 거래가 완료되기 전에 계정이 삭제되기 때문에 문제는 논쟁의 여지가 있습니다 . 정상적인 STOP 코드의 경우 또는 반환된 코드가 비어 있는 경우 상태는 좀비 계정으로 남고 나머지 잔액은 계정에 영원히 잠깁니다.

8. 메시지 호출

메시지 호출을 실행하는 경우 발신자 (s), 트랜잭션 발신자 (o), 수신자 (r), 코드를 실행할 계정 (c, 일반적으로 수신자와 동일) 등 여러 매개 변수가 필요합니다. 가스 (g), 값 (v) 및 가스 가격 (p) 과 함께 임의 길이의 바이트 배열, d, 호출의 입력 데이터, 메시지 호출/계약 생성 스택 의 현재 깊이 (e) 및 마지막으로 상태(w)를 수정할 수 있는 권한 .

새로운 상태 및 누적된 트랜잭션 하위 상태로 평가하는 것 외에도 메시지 호출 에는 바이트 배열 o로 표시되는 출력 데이터라는 추가 구성 요소가 있습니다 .

이것은 트랜잭션을 실행할 때 무시되지만 VM 코드 실행으로 인해 메시지 호출이 시작될 수 있으며 이 경우 이 정보가 사용됩니다. (110) (σ , g, A

$$, z, o) \equiv \Theta(\sigma, A, s, o, r, c, g, p, v, v, \sim d, e, w)$$

다음 값을 구별해야 합니다.

DELEGATECALL 명령 에 대한 실행 컨텍스트 v~ 에서 명백한 값 v 에서 전송됩니다 .

우리는 첫 번째 전환 상태인 σ1을 원래 상태로 정의 하지만 보낸 사람에서 전송된 값으로

받는 사람:

(111) $\sigma 1[r]b \equiv \sigma[r]b + v \wedge \sigma 1[s]b \equiv \sigma[s]b \quad v$

s = r이 아니면.

본 작업 전반에 걸쳐 σ1[r]이 원래 정의되지 않은 경우 코드 또는 상태가 없고 잔고 및 논스가 0인 계정으로 생성될 것이라고 가정합니다 . 따라서 이전 방정식은 다음을 의미하는 것으로 간주되어야 합니다.

(112) $\sigma 1 \equiv \sigma\,1$ 제외:

(113) $$\sigma 1[s] \equiv \varnothing \text{ if } \sigma\,1[s] = \varnothing \wedge v = 0 \text{ a1 아니면}$$

(114) $a1 \equiv \sigma\,1[s]n, \sigma\,1[s]b \quad v, \sigma\,1[s]s, \sigma\,1[s]c$

(115) 및 $\sigma\,1 \equiv \sigma$ 제외:

(116)

$$\sigma\,1[r] \equiv \begin{cases} (0, v, TRIE(\varnothing), KEC(())) & \text{if } \sigma[r] = \varnothing \wedge v = 0 \\ \varnothing & \text{if } \sigma[r] = \varnothing \wedge v = 0 \\ a\,1 & \text{아니면} \end{cases}$$

(117) $a\,1 \equiv (\sigma[r]n, \sigma[r]b + v, \sigma[r]s, \sigma[r]c)$

계정의 관련 코드(Keccak-256 해시가 σ[c]c인 조각 으로 식별됨)는 실행 모델에 따라 실행됩니다 (섹션 9 참조). 컨트랙트 생성과 마찬가지로 실행이 예외적인 방식으로 중단되면(예: 소진된 가스 공급, 스택 언더 플로우 , 유효하지 않은 점프 대상 또는 유효하지 않은 명령으로 인해) 호출자에게 가스가 환불되지 않고 상태가 원래 상태로 되돌아갑니다. 잔액 이체 직전 시점(즉, σ).

5초기화 코드 실행 중에 주소의 EXTCODESIZE는 계정 코드의 길이인 0을 반환해야 합니다. CODESIZE는 초기화 코드의 길이를 반환해야 합니다(H.2에 정의된 대로).

$$\sigma \equiv \begin{cases} \varnothing & \text{만약 에 } \_ = \varnothing \\ \sigma_p & \text{그렇지 않으면} \end{cases} \tag{118}$$

$$g \equiv \begin{cases} 0 & \text{만약 피} = \varnothing \wedge o = \varnothing \\ g & \text{그렇지 않으면} \end{cases} \tag{119}$$

$$A \equiv \begin{cases} \varnothing & \text{피 라면 A} = \varnothing \\ \dagger & \text{그렇지 않으면} \end{cases} \tag{120}$$

$$\equiv \text{포함} \begin{cases} 0 & \text{이면 p} = \varnothing \\ 1 & \text{그렇지 않으면} \end{cases} \tag{121}$$

$$(\text{피}, g, A, o) \equiv \Xi \tag{122}$$

$$\text{그것} \equiv r \tag{123}$$

$$\text{나} \equiv o \tag{124}$$

$$IP \equiv \text{피} \tag{125}$$

$$ID \equiv d \tag{126}$$

$$is \equiv s \tag{127}$$

$$IV \equiv v \sim \tag{128}$$

$$\text{즉} \equiv e \tag{129}$$

$$Iw \equiv w \tag{130}$$

어디

$$\Xi \equiv \begin{cases} \Xi_{ECREC}(\sigma 1, g, A, I) & c = 1\text{인 경우} \\ \Xi_{SHA256}(\sigma 1, g, A, I) & c = 2\text{인 경우} \\ \Xi_{RIP160}(\sigma 1, g, A, I) & c = 3\text{인 경우} \\ \Xi_{ID}(\sigma 1, g, A, I) & c = 4\text{인 경우} \\ \Xi_{EXPMOD}(\sigma 1, g, A, I) & c = 5\text{인 경우} \\ \Xi_{BN\_ADD}(\sigma 1, g, A, I) & c = 6\text{인 경우} \\ \Xi_{BN\_MUL}(\sigma 1, g, A, I) & c = 7\text{인 경우} \\ \Xi_{SNARKV}(\sigma 1, g, A, I) & c = 8\text{인 경우} \\ \Xi_{BLAKE2\_F}(\sigma 1, g, A, I) & \text{if } c = 9 \\ \xi(\sigma 1, g, A, I) & \text{그렇지 않으면} \end{cases} \tag{131}$$

그리고

$$KEC(Ib) = \sigma[c]c \tag{132}$$

클라이언트가 쌍을 저장했다고 가정합니다.
(KEC(Ib), Ib)
가능한 Ib 의 결정 .

보시다시피 사용에는 9가지 예외가 있습니다.
일반 실행 프레임워크의 평가를 위한 Ξ
메시지 호출: 소위 '사전 컴파일된' 계약입니다.
아키텍처의 예비 조각으로 의미 나중에
기본 확장이 됩니다. 주소 1에서 계약
9 타원 곡선 공개 키 복구 기능을 실행하고,
SHA2 256비트 해시 체계, RIPEMD 160비트 해시
체계, 항등 함수, 임의 정밀 모듈
지수화, 타원 곡선 덧셈, 타원 곡선 스칼라
곱셈, 타원 곡선 페어링 확인 및
BLAKE2 압축 함수 F 각각. 그들의 전체
공식적인 정의는 부록 E에 있습니다.
π로 미리 컴파일된 계약의 주소:

$$\pi \equiv \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \tag{133}$$

9. 실행 모델

실행 모델은 시스템 상태를 지정합니다.
일련의 바이트코드 명령과 작은
환경 데이터의 튜플. 이것은 다음을 통해 지정됩니다.
가상 상태 머신의 공식 모델,
이더리움 가상 머신(EVM). 준 튜링 완전 기계입니다. 준 자격은

계산이 본질적으로 다음을 통해 제한된다는 사실
수행된 총 계산량을 제한하는 매개변수 가스 .

9.1. 기초. EVM은 단순한 스택 기반 아키텍처입니다 . 기계의 단어 크기(따라서 스택의 크기)
항목)은 256비트입니다. 이것은 Keccak 256 해시 체계와 타원 곡선
계산을 용이하게 하기 위해 선택되었습니다. 그만큼
메모리 모델은 간단한 워드 주소 바이트 배열입니다. 그만큼
스택의 최대 크기는 1024입니다.
독립적인 저장 모델; 이것은 개념적으로 비슷합니다
메모리에 저장하지만 바이트 배열이 아니라 워드 주소 지정이 가능한
워드 배열입니다. 휘발성인 메모리와 달리
스토리지는 비휘발성이며
시스템 상태. 저장소와 메모리의 모든 위치
처음에는 0으로 잘 정의됩니다.
기계는 표준 폰 노이만 아키텍처를 따르지 않습니다. 프로그램 코드를 저장하는 대신
일반적으로 액세스 가능한 메모리 또는 저장소에 저장됩니다.
통해서만 상호 작용할 수 있는 가상 ROM에서 별도로
전문 교육.
기계는 여러 가지에 대해 예외적인 실행을 할 수 있습니다.
스택 언더플로 및 유효하지 않은 명령을 포함한 이유. out-of-gas 예외처럼 그들은 떠나지 않는다.
그대로 상태가 변경됩니다. 오히려 시스템이 즉시 중지되고 문제를 실행 에이전트(또는
트랜잭션 프로세서 또는 재귀적으로 생성
실행 환경)에서 별도로 처리합니다.

9.2. 수수료 개요. 요금(가스로 표시)은 다음과 같습니다.
세 가지 별개의 상황에서 청구되며 세 가지 모두 다음과 같습니다.
작업 실행의 전제 조건. 첫번째
가장 일반적인 것은 계산에 내재된 수수료입니다.
(부록 G 참조). 두 번째로 가스는
부하에 대한 지불을 형성하기 위해 공제
메시지 호출 또는 계약 생성 이것은
CREATE, CREATE2, CALL 및 CALLCODE 에 대한 지불 .
마지막으로 사용량 증가로 인해 가스를 지불할 수 있습니다.
기억의.
계정을 실행하는 동안 지불해야 하는 메모리 사용에 대한 총 요금
은 32의 최소 배수에 비례합니다.
모든 메모리 인덱스에 필요한 바이트
(읽기 또는 쓰기 여부) 범위에 포함됩니다. 이것
적시에 지급됩니다. 이와 같이 참조
이전에 인덱싱된 메모리보다 최소 32바이트 더 큰 메모리 영역은 확실히 추가

메모리 사용료. 이 수수료로 인해 가능성이 매우 낮습니다.
주소는 32비트 경계를 초과합니다. 즉,
구현은 이러한 상황을 관리할 수 있어야 합니다.
스토리지 비용은 약간 미묘한 차이가 있습니다 .

항목을 지우는 작업에 대한 실행 수수료
보관이 면제될 뿐만 아니라 적격 환불이 제공됩니다.
실제로 이 환불은
스토리지 위치의 초기 사용 비용은 훨씬 더 비쌉니다.
정상적인 사용보다.
EVM의 엄격한 정의는 부록 H를 참조하십시오.
가스 비용.

9.3. 실행 환경. 시스템 상태 σ, 계산을 위한 나머지 가스 g 및 발생한 하위 상태 A 외에도 실행 환경에서 사용되는 몇 가지 중요한 정보가 있습니다 .

집행 대리인은 다음을 제공해야 합니다. 이들은 튜플 I에 포함되어 있습니다 .

- Ia, 코드를 소유한 계정의 주소 실행하는 것입니다.
- Io, 이 실행을 시작한 트랜잭션의 발신자 주소 .
- Ip, 이 실행을 시작한 트랜잭션의 가스 가격 .
- Id, 이 실행에 대한 입력 데이터인 바이트 배열 . 실행 에이전트가 트랜잭션인 경우 트랜잭션 데이터가 됩니다.
- Is는 코드를 실행하게 한 계정의 주소입니다. 실행 에이전트가 트랜잭션인 경우 트랜잭션 발신자가 됩니다. • Iv, Wei의 값은 실행과 동일한 절차의 일부로 이 계정에 전달됩니다. 실행 에이전트가 트랜잭션 인 경우 트랜잭션 값 이 됩니다 .
- Ib, 처리할 기계어 코드인 바이트 배열 실행.
- IH, 현재 블록의 블록 헤더. • 즉, 현재 메시지 호출 또는 계약 생성의 깊이(즉, 현재 실행 중인 CALL 또는 CREATE(2) 의 수). • Iw, 수정 권한 상태.

실행 모델은 결과 상태 σ , 나머지 가스 g , 결과 발생 하위 상태 A 및 결과 출력 o 를 계산할 수 있는 함수 Ξ를 정의합니다 . 현재 상황에서는 다음과 같이 정의합니다.

(134) (s , g , A , o) ≡ Ξ(σ, g, A, I)

여기에서 발생한 하위 상태인 A가 섹션 6.1에 정의되어 있음을 기억할 것입니다 .

9.4. 실행 개요. 이제 Ξ 함수를 정의해야 합니다. 대부분의 실제 구현에서 이는 전체 시스템 상태 σ 와 기계 상태 μ 로 구성된 쌍의 반복적인 진행으로 모델링 됩니다 . mally의 경우 함수 X를 사용하여 재귀적으로 정의합니다. 이것은 현재 상태가 기계의 예외적인 정지 상태 인지 결정하는 함수 Z 와 함께 반복자 함수 O (상태 기계의 단일 주기의 결과를 정의함)를 사용합니다. 및 H, 현재 상태가 기계의 정상적인 정지 상태 인 경우에만 명령의 출력 데이터를 지정합니다 .

()로 표시된 빈 시퀀스는 ∅ 로 표시된 빈 집합과 같지 않습니다 . 이것은 실행이 계속될 때 ∅ 로 평가 되지만 실행될 때 시리즈(잠재적으로 비어 있음)로 평가되는 H 의 출력을 해석할 때 중요합니다.

중단해야합니다.

(135) ξ(σ, g, A, I) ≡ (σ , μ g, ├ , o)

(136) (σ , μ , A , ..., o) ≡ X (σ, μ, A, I)

(137) μg ≡ g μpc ≡

(138) 0 μm ≡ (0, 0, ...)

(139) μi ≡ 0 μs ≡ () μo ≡ ()

(140)

(141)

(142)

(143)

$$X(\sigma, \mu, A, I) \equiv \begin{cases} \varnothing, \mu, A, I, \varnothing & Z(\sigma, \mu, A, I) \text{ 인 경우} \\ \varnothing, \mu, A, I, o & w = \text{되돌리기 인 경우} \\ O(\sigma, \mu, A, I) \cdot o \text{ if } o = \varnothing & \\ X\ O(\sigma, \mu, A, I) & \text{그렇지 않은 경우} \end{cases}$$

어디

(144) o ≡ H(μ, I) (a, b, c,

(145) d) · e ≡ (a, b, c, d, e) ≡ μ 제외: ≡ μg

(146) 중 C(σ, μ, A, I)

(147) 지 _

Ξ를 평가할 때 Ξ에서 네 번째를 떨어뜨립니다. 요소 I 및 추출 나머지 가스 μ 결과 기계 상태 μ . g

따라서 X 는 현재 상태가 예외적이며 기계가 정지되고 모든 변경 사항을 폐기해야 함을 나타내는 Z가 참이 될 때 까지 또는 H 가 시리즈(빈 세트 가 아닌) 기계가 제어된 정지에 도달했음을 나타냅니다.

9.4.1. 기계 상태. 기계 상태 μ는 사용 가능한 가스인 튜플 (g, pc, m, i, s), 프로그램 카운터 pc ∈ N256 메모리 내용, 메모리의 활성 단어 수(위치 0부터 계속 카운트) 로 정의됩니다 . , 및 스택 내용. 메모리 내용 μm은 크기가 2256 인 일련의 0입니다 .

읽기 쉽도록 작은 대문자(예: ADD) 로 작성된 명령어 니모닉은 해당 숫자 로 해석해야 합니다 . 지침의 전체 표와 세부 사항은 부록 H에 나와 있습니다.

Z, H 및 O를 정의하기 위해 w를 실행할 현재 작업으로 정의합니다 .

(148) ≡에서 μpc < Ib 인 경우 Ib[μpc]
그렇지 않으면 중지

우리는 또한 δ 와 α 의 고정된 양을 가정하고 , 제거되고 추가된 스택 항목을 지정하며, 명령어와 명령어 비용 함수 C eval 에 모두 첨자 가능합니다.

주어진 명령을 실행하는 데 드는 전체 비용(가스)을 계산합니다 .

9.4.2. 뛰어난 정지. 예외적인 정지 기능 Z는 다음과 같이 정의됩니다. (149)

$$Z(\sigma, \mu, A, I) \equiv \mu_g < C(\sigma, \mu, A, I) \vee \delta_w = \varnothing \vee \mu_s < \delta_w \vee (w = \text{JUMP} \wedge \mu_s[0] \in/\ D(I_b)) \vee (w = \text{JUMPI} \wedge \mu_s[1] = 0 \wedge \mu_s[0] \in/\ D(I_b)) \vee (w = \text{RETURNDATACOPY} \wedge \mu_s[1] + \mu_s[2] > \mu_o) \vee \mu_s \delta_w + \alpha_w > 1024 \vee (\neg I_w \wedge W(w, \mu)) \vee (w = \text{SSTORE} \wedge \mu_g \text{Gcallstipend})$$

어디

(150) $W(w, \mu) \equiv w \in \{\text{CREATE, CREATE2, SSTORE, 자폭}\} \vee \text{LOG0} \leq w \wedge w \leq \text{LOG4} \vee w = \text{호출} \wedge \mu_s[2] = 0$

이는 가스가 충분하지 않거나, 명령이 유효하지 않거나(따라서 δ 첨자가 정의되지 않음), 스택 항목이 충분하지 않거나, JUMP/JUMPI 대상이 유효하지 않은 경우 실행이 예외적인 정지 상태에 있음을 나타냅니다. 새 스택 크기가 1024보다 크거나 정적 호출 중에 상태 수정이 시도됩니다. 기민한 독자라면 어떤 명령도 실행을 통해 예외적인 중단을 일으킬 수 없다는 것을 의미한다는 것을 알 수 있을 것입니다 .

또한 SSTORE 명령을 실행하기 전에 남은 가스가 호출 stipend Gcallstipend 보다 작 거나 같으면 실행이 예외적인 정지 상태에 있습니다 . 자세한 내용은 Tang [2019]의 EIP-2200을 참조하십시오 .

9.4.3. 점프 대상 유효성. 이전에는 실행 중인 코드가 주어지면 유효한 점프 목적지 세트를 결정하는 함수로 D를 사용했습니다 . 우리는 이것을 JUMPDEST 명령 이 차지하는 코드의 모든 위치로 정의합니다 .

이러한 모든 위치는 PUSH 작업 의 데이터 부분에 있는 것이 아니라 유효한 명령어 경계에 있어야 하며 코드의 명시적으로 정의된 부분 내에 나타나야 합니다(암시적으로 정의된 STOP 작업이 아닌).

공식적으로:

(151) $D(c) \equiv DJ(c, 0)$

어디:

(152)

$$DJ(c, i) \equiv \begin{cases} \{\} & \text{만약 내가 } \text{씨} \\ \{i\} \cup DJ(c, N(i, c[i])) & \text{if } c[i] = \text{JUMPDEST} \\ DJ(c, N(i, c[i])) & \text{그렇지 않으면} \end{cases}$$

여기서 N은 다음으로 유효한 명령 위치입니다.
코드, PUSH 명령 의 데이터 건너뛰기 (있는 경우): (153)

$$N(i, w) \equiv \begin{cases} i + w - w \in [\text{PUSH1}, & \\ \quad \text{PUSH32}] \text{ 인 경우 PUSH1} + 2 & \\ i + 1 & \text{그렇지 않으면} \end{cases}$$

9.4.4. 정상적인 정지. 일반 정지 함수 H는 다음과 같이 정의 됩니다 . (154)

$$H(\mu, I) \equiv \begin{cases} \text{HRETURN}(\mu) & \text{if } w \in \{\text{RETURN, REVERT}\} \\ () & \text{if } w \in \{\text{STOP, SELFDESTRUCT}\} \\ \varnothing & \text{그렇지 않으면} \end{cases}$$

데이터 반환 중단 작업인 RETURN 및 REVERT에는 특수 기능 HRETURN이 있습니다. 여기에서 논의된 바와 같이 빈 시퀀스와 빈 세트의 차이점 도 참고하십시오 .

9.5. 실행 주기. 스택 항목은 시리즈 의 맨 왼쪽, 색인이 낮은 부분에서 추가되거나 제거됩니다 . 다른 모든 항목은 변경되지 않습니다.

(155) $O(\sigma, \mu, A, I) \equiv (\sigma', \mu', A', 나)$

(156) $\Delta \equiv \alpha_w \quad \delta_w$

(157) $중_{\text{새}} \equiv \mu_s + \Delta[x] \equiv$

(158) $\forall x \in [\alpha_w, \mu_s') : \mu'_{\text{새}} \mu_s[x \quad \Delta]$

가스는 명령의 가스 비용에 의해 감소되며 대부분의 명령에 대해 프로그램 카운터는 매 사이클마다 증가합니다. 세 가지 예외에 대해 함수를 가정합니다.
J는 다음 두 명령어 중 하나에 의해 아래 첨자로 표시되며 해당 값으로 평가됩니다.

(159) $지_{\_} \equiv \mu_g \quad C(\sigma, \mu, A, I)$

(160) $중_{pc} \equiv \begin{cases} \text{JUMP}(\mu) & \text{if } w = \text{JUMP} \\ \text{JUMPI}(\mu) & \text{if } w = \text{JUMPI} \\ N(\mu_{pc}, w) & \text{그렇지 않으면} \end{cases}$ (w = JUMP 인 경우 JJUMP(μ))

일반적으로 메모리, 누적된 하위 상태 및 시스템 상태가 변경되지 않는다고 가정합니다.

(161) $중_{총} \equiv \mu_m$

(162) $나는_{\_} 나는_{\_}$

(163) $\vdash \equiv A$

(164) $피 \equiv 피$

그러나 명령은 일반적으로 이러한 값의 하나 또는 여러 구성 요소를 변경합니다. 지침 에 따라 나열된 변경된 구성 요소는 부록 H에 α 및 δ 값과 가스 요구 사항에 대한 공식적인 설명과 함께 표시됩니다.

10. 블록트리에서 블록체인으로

정식 블록체인은 전체 블록 트리를 통해 루트에서 리프까지의 경로입니다 . 어떤 경로인지에 대한 합의를 얻기 위해 개념적으로 가장 많은 계산이 수행된 경로 또는 가장 무거운 경로를 식별합니다. 분명히 가장 무거운 경로를 결정하는 데 도움이 되는 한 가지 요소는 경로에서 채굴되지 않은 제네시스 블록을 계산하지 않고 블록 수와 동일한 리프의 블록 번호입니다 . 경로가 길수록 리프에 도달하기 위해 수행해야 하는 총 채굴 노력이 커집니다. 이는 비트코인에서 파생된 프로토콜에 사용된 것과 같은 기존 체계와 유사합니다 .

블록 헤더에는 난이도가 포함되어 있기 때문에 헤더 만으로도 완료된 계산을 검증하기에 충분합니다. 모든 블록은 체인의 총 계산 또는 총 난이도에 기여합니다 .

따라서 블록 B 의 총 난이도를 재귀적으로 정의합니다.
처럼:

(165) Bt ≡ B 티 + Bd

(166) 비 ≡ 피(BH)

블록 B가 주어지면 Bt 는 전체 난이도이고 B 는 상위 블록이고 Bd 는 난이도입니다.

11. 블록 마무리

블록을 마무리하는 프로세스에는 다음과 같은 4단계가 포함됩니다. (2) 트랜잭션을 검증(또는 채굴하는 경우 결정)합니다. (3) 보상을 적용합니다. (4) 상태를 확인(또는 채굴하는 경우 유효한 계산)하고 난스를 차단합니다.

11.1. 오메르 검증. ommer 헤더의 유효성 검사는 각 ommer 헤더가 유효한 헤더인지 확인하고 N ≤ 6 인 현재 블록에 대한 N세대 ommer의 관계를 만족하는지 확인하는 것 이상을 의미하지 않습니다 .

최대 ommer 헤더는 2개입니다. 공식적으로:

(167) BU 2 V(U) ∧ k(U, P(BH)H, 6)
유∈부

여기서 k는 "친족" 속성을 나타냅니다: (168)

k(U, H, n) ≡ 거짓 n = 0인 경우
s(U, H)
∨ k(U, P(H)H, n 1) 그렇지 않으면

그리고 s는 "형제" 속성을 나타냅니다: (169) s(U, H) ≡
(P(H)
= P(U) ∧ H = U ∧ U /∈ B(H)U) 여기서 B(H) 및 P(H) 는 각각 해당 헤더 H 의 블록 및 부모 블록입니다 .

11.2. 트랜잭션 유효성 검사. 주어진 gasUsed는 나열된 트랜잭션과 충실히 일치해야 합니다. 블록에서 사용된 총 가스인 BHg는 최종 트랜잭션에 따라 사용된 축적된 가스와 동일해야 합니다.

(170) BHg = (R)u

11.3. 보상 신청. 블록에 보상을 적용하는 것은 블록의 수취인 주소와 각 ommer의 계정 잔액을 일정 금액만큼 올리는 것입니다. 우리는 Rblock 에 의해 블록의 수혜자 계정을 올립니다 . 각 ommer에 대해 추가 블록 보상으로 블록 의 수혜자를 높이고 ommer의 수혜자는 블록 번호에 따라 보상을 받습니다. 공식적으로 함수 Ω을 정의합니다.

1 32

(171)
Ω(B, σ) ≡ σ : σ = σ 제외:

(172) σ [BHc]b = σ[BHc]b + 1 + 이것 / 32 R블록

(173)
∀U ∈ 부 :

σ [Uc] = ∅ σ[Uc] = ∅ ∧ R = 0 인 경우
그렇지 않으면

(174) ≡ (σ[Uc]n, σ[Uc]b + R, σ[Uc]s, σ[Uc]c)

(175) R ≡ 1 + 1/8 (Ui - BHi) R블록

ommer와 블록 사이에 beneficiary address가 충돌하는 경우 (즉, 현재 블록과 beneficiary address가 같은 두 ommer 또는 beneficiary address가 같은 ommer) 추가가 누적되어 적용됩니다.

Wei에서 블록 보상을 정의합니다.

(176)

R블록 = 1018 × Hi < F비잔티움 인 경우 5
3 if FByzantium Hi < FConstantinople 2 if Hi
FConstantinople

11.4. 상태 및 논스 유효성 검사. 이제 블록 B 를 시작 상태로 매핑하는 함수 Γ를 정의할 수 있습니다. (177)

C(B) ≡ p0 P(BH) = ∅ σi 인 경우
우 : TRIE(LS(σi)) = P(BH)Hr 그렇지 않은 경우

여기서 TRIE(LS(σi))는 트리 상태 σi 의 루트 노드의 해시를 의미합니다 . 구현 시 이를 상태 데이터베이스에 저장 한다고 가정합니다 . 트리는 본질적으로 불변 데이터 구조이기 때문에 사소하고 효율적입니다.

마지막으로 불완전한 블록 B 를 완전한 블록으로 매핑하는 블록 전환 함수인 Φ를 정의합니다.
비 :

(178) Φ(B) ≡ B : B = B 제외하고:

(179) 비 N = 엔 : 엑스 $\frac{2^{256}}{Hd}$

(180) 비 ㅎ = m with (x, m) = PoW(B n, n, d) = r(Π(Γ(B)', B))

(181) 비 ≡ B 제외: B

d는 부록 J에 지정된 데이터 세트입니다.

현재 작업의 시작 부분에 명시된 바와 같이 Π는 상태 전이 함수이며 블록 종료 함수인 Ω과 트랜잭션 평가 함수인 Υ로 정의되며 둘 다 잘 정의되어 있습니다.

앞에서 설명한 것처럼 R[n]z, R[n]l 및 R[n]u는 n번째 해당 상태 코드, 로그 및 각 트랜잭션 후에 사용되는 누적 가스 입니다 (R[n]b, 튜플의 네 번째 구성 요소 , 이미 로그 측면에서 정의되었습니다).

우리는 또한 n번째 상태 σ[n]을 정의합니다. 이는 단순히 이전 트랜잭션의 결과 상태 (또는 첫 번째 트랜잭션 의 경우 블록의 초기 상태) 에 해당 트랜잭션을 적용한 상태로 정의됩니다.

(182) σ[n] = Γ(B) n < 0인 경우
Υ(σ[n - 1], BT[n]) 그렇지 않은 경우

BR[n]u 의 경우 , 각 항목을 이전 항목과 합산된 해당 트랜잭션을 평가하는 데 사용되는 가스(또는 첫 번째 항목인 경우 0) 로 정의하는 유사한 접근 방식을 취하여 누적 합계를 제공합니다. :

(183) R[n]u = 0 n < 0인 경우
Yg (σ[n - 1], BT[n])
+R[n - 1]유 그렇지 않으면

R[n]l 의 경우 우리는 컨버팅하는 Yl 함수를 사용합니다.
트랜잭션 실행 함수에 정의되어 있습니다.

(184) R[n]l = Yl (σ[n - 1], BT[n])

비슷한 방식으로 R[n]z를 정의합니다 .

(185) R[n]z = Yz (σ[n - 1], BT[n])

마지막으로 Π를 최종 트랜잭션의 결과 상태 (σ) 에 적용된 블록 보상 함수 Ω이 주어진 새로운 상태로 정의합니다 . (186)

$$\Pi(\sigma, B) \equiv \Omega(B, (\sigma))$$

따라서 완전한 블록 전환 메커니즘은 de
작업 증명 기능인 PoW를 제외하고 벌금이 부과됩니다.

11.5. 마이닝 작업 증명. 마이닝 작업 증명(PoW)은 일부 토큰 값 n을 결정하는 데 특정 양의 계산이 소비되었음을 합리적인 의심의 여지 없이 증명하는 암호학적으로 안전한 논스로 존재합니다. 시행하는데 활용됩니다.

난이도 개념에 의미와 신뢰성을 부여하여 블록체인 보안을 강화합니다(더 나아가 전체 난이도). 그러나 새로운 블록을 채굴하면 보상이 따르기 때문에 작업 증명은 블록체인 이 미래에도 정식으로 남을 것이라는 신뢰를 확보하는 방법일 뿐만 아니라 부의 분배 메커니즘으로도 기능합니다.

두 가지 이유로 작업 증명 기능 에는 두 가지 중요한 목표가 있습니다 . 첫째, 가능한 한 많은 사람들이 접근할 수 있어야 합니다 . 특수하고 일반적이지 않은 하드웨어에 대한 요구 사항 또는 보상을 최소화해야 합니다. 이것은 배포 모델을 가능한 한 개방적으로 만들고 , 이상적으로는 전 세계 누구에게나 거의 동일한 비율로 전기에서 Ether로 간단한 스왑을 마이닝하는 행위를 만듭니다 .

둘째, 초 선형 이익을 만드는 것이 가능하지 않아야 하며, 특히 초기 장벽이 높으면 더욱 그렇습니다.
이러한 메커니즘을 통해 자금이 풍부한 적이 네트워크의 총 채굴 능력 중 골치아픈 양을 얻을 수 있으므로 네트워크 보안이 감소할 뿐만 아니라 매우 선형적인 보상(따라서 분포가 왜곡됨)을 제공합니다.

Bitcoin 세계의 전염병 중 하나는 ASIC입니다. 이들은 단일 작업을 수행하기 위해서만 존재하는 특수 컴퓨팅 하드웨어 입니다(Smith [1997]). 비트코인의 경우 작업은 SHA256 해시 함수입니다(Courtois et al. [2014]). 작업 증명 기능을 위해 ASIC이 존재 하지만 두 가지 목표 모두 위험에 처해 있습니다. 이 때문에 ASIC 내성이 있는 작업 증명 기능 (즉, 특수 컴퓨팅 하드웨어에서 구현하기 어렵거나 경제적으로 비효율적임)이 묘책으로 식별되었습니다.

ASIC 저항에는 두 가지 방향이 있습니다. 먼저 순차 메모리 하드 (sequential memory-hard)로 만듭니다 . 즉 , nonce를 결정하는 데 많은 메모리 와 대역폭이 필요하여 동시에 여러 nonce를 발견하기 위해 메모리를 병렬로 사용할 수 없도록 기능을 설계합니다 . 두 번째는 범용 수행 에 필요한 계산 유형을 만드는 것입니다 . 범용 작업 세트에 대한 "특수 하드웨어"의 의미는 당연히 범용 하드웨어이며 그러한 범용 데스크탑 컴퓨터는 작업을 위한 "특수 하드웨어"에 매우 가깝습니다. Ethereum 1.0의 경우 첫 번째 경로를 선택했습니다.

보다 공식적으로 작업 증명 기능은 PoW 형식을 취합니다.

(187)

$$m = 흠 \wedge n \quad \frac{2^{256}}{고화질} \quad (m, n) = \text{PoW}(Hn, Hn, d) \quad \nearrow$$

여기서 Hn 은 새 블록의 헤더이지만 nonce 및 혼합 해시 구성 요소가 없습니다. Hn 은 헤더 의 nonce입니다 . d 는 혼합 해시를 계산하는 데 필요한 큰 데이터 세트 이고 Hd 는 새 블록의 난이도 값(즉, 섹션 10의 블록 난이도). PoW 는 첫 번째 항목이 mixHash이고 두 번째 항목이 H 와 d 에 암호화된 의사 난수인 배열로 평가되는 작업 증명 기능입니다 .

기본 알고리즘은 Ethash라고 하며 아래에 설명되어 있습니다.

11.5.1. 에타쉬. Ethash는 Ethereum 1.0의 PoW 알고리즘입니다 . Buterin [2013b]과 Dryja [2014]가 소개한 Dagger-Hashimoto의 최신 버전이지만 두 알고리즘의 원래 기능 중 많은 부분이 2015년 2월부터 2015년 5월 4일 (Jentzsch [2015]). 알고리즘이 취하는 일반적인 경로는 다음과 같습니다.

해당 지점까지 블록 헤더를 스캔하여 각 블록에 대해 계산할 수 있는 시드가 있습니다 . 시드에서 의사 난수 캐시, 초기 크기의 Jcacheinit 바이트를 계산할 수 있습니다. 라이트 클라이언트는 캐시를 저장합니다. 캐시에서 데이터 세트의 각 항목이 캐시의 적은 수의 항목에만 의존한다는 속성을 사용하여 초기 크기의 Jdatasetinit 바이트 데이터 세트를 생성할 수 있습니다. 전체 클라이언트와 광부는

데이터 세트. 데이터 세트는 시간이 지남에 따라 선형적으로 증가합니다.

마이닝에는 데이터 세트의 임의 조각을 가져와 함께 해싱하는 작업이 포함됩니다. 캐시를 사용하여 필요한 데이터 세트의 특정 부분을 재생성함으로써 낮은 메모리 로 검증을 수행할 수 있으므로 캐시만 저장하면 됩니다. 대규모 데이터 세트는 Jepoch 블록마다 한 번씩 업데이트되므로 광부 작업의 대부분은 데이터 세트를 변경하는 것이 아니라 데이터 세트를 읽는 것입니다. 언급된 매개 변수 와 알고리즘은 부록 J에 자세히 설명되어 있습니다.

12. 계약 이행

특정 유용한 동작을 허용하는 여러 계약 공학 패턴이 있습니다 . 간략하게 논의 할 두 가지는 데이터 피드와 난수입니다.

12.1. 데이터 피드. 데이터 피드 계약은 단일 서비스를 제공하는 것입니다: 이더리움 내 외부 세계의 정보에 대한 액세스를 제공합니다. 이 정보의 정확성과 적시성은 보장되지 않으며 단일 데이터 피드를 얼마나 신뢰할 수 있는지 결정하는 것은 데이터 피드를 활용하는 계약인 2차 계약 작성자의 임무 입니다.

일반적인 패턴은 이더리움 내의 단일 계약과 관련이 있으며 , 메시지 호출이 주어 지면 외부 현상에 관한 시기적절한 정보 로 응답합니다 . 뉴욕시 의 현지 온도를 예로 들 수 있습니다 . 이것은 저장소에서 알려진 일부 지점의 값을 반환하는 계약으로 구현됩니다 . 물론 저장소의 이 지점은 올바른 온도로 유지되어야 하므로 패턴 의 두 번째 부분은 외부 서버가 Ethereum 노드를 실행하고 새 블록을 발견하는 즉시 새로운 유효한 트랜잭션을 생성하는 것입니다. 계약으로 전송되어 스토리지에서 해당 값을 업데이트합니다. 계약의 코드는 해당 서버에 포함된 ID에서만 이러한 업데이트를 수락합니다.

12.2. 난수. 결정론적 시스템 내에서 난수를 제공하는 것은 당연히 불가능한 작업입니다. 그러나 일반적으로 알 수 없는 데이터를 활용하여 의사 난수로 근사할 수 있습니다.

거래시. 이러한 데이터 에는 블록의 해시, 블록의 타임스탬프 및 블록의 수혜자 주소가 포함될 수 있습니다 . 악의적인 채굴자가 이러한 값을 제어하기 어렵게 하려면 이전 256개 블록의 해시를 의사 난수로 사용하기 위해 BLOCKHASH 작업을 사용해야 합니다 . 이러한 일련의 숫자에 대해 간단한 해결책은 일정한 양을 추가하는 것입니다.

결과를 해싱합니다.

13. 향후 방향

상태 데이터베이스는 모든 과거 상태 트리 구조를 미래에 유지하도록 강제되지 않습니다 . 각 노드에 대한 수명을 유지하고 최종적으로 충분히 최근이지도 체크포인트도 아닌 노드를 폐기해야 합니다. 체크포인트 또는 특정 블록의 상태 트리를 통과할 수 있는 데이터베이스의 노드 집합을 사용하여 블록체인 전체에서 상태를 검색하는 데 필요한 계산량을 최대로 제한할 수 있습니다 .

블록체인 통합은 다시 하기 위해 사용될 수 있습니다.
클라이언트가 다운로드해야 하는 블록의 양을 줄입니다.
완전한 마이닝 노드 역할을 합니다. 주어진 시점에서 트리 구조의 압축된 아카이브(아마도 10,000번째 블록마다 하나)가 피어 네트워크에 의해 유지 관리되어 제네시스 블록을 효과적으로 재구성할 수 있습니다. 이렇게 하면 단일 아카이브에 다운로드되는 양과 최대 블록 제한이 줄어듭니다 .

마지막으로, 블록체인 압축이 수행될 수 있습니다 . 일정한 양의 블록에서 트랜잭션을 보내거나 받지 않은 상태 트리의 노드는

Ether 누출과 상태 데이터베이스의 성장을 모두 줄입니다 .

13.1. 확장성. 확장성은 영원한 관심사로 남아 있습니다.

일반화된 상태 전이 기능을 사용하면 분할 정복 전략을 적용하기 위해 트랜잭션을 분할하고 병렬화하는 것이 어려워집니다 . 주소가 지정되지 않으면 시스템의 동적 가치 범위는 본질적으로 고정된 상태로 유지되며 평균 거래 가치가 증가함에 따라 가치가 낮은 항목은 무시되어 주 원장에 포함하는 것이 경제적으로 무의미합니다. 그러나 훨씬 더 확장 가능한 프로토콜을 제공하기 위해 잠재적으로 악용될 수 있는 몇 가지 전략이 있습니다 .

더 작은 경량 체인을 메인 블록으로 통합하거나 증분 조합을 통해 메인 블록을 구축하고 더 작은 트랜잭션 세트의 부착(작업 증명을 통해)을 통해 달성되는 계층 구조의 일부 형태는 트랜잭션 조합 의 병렬화를 허용할 수 있습니다. 그리고 블록 빌딩. 병렬성은 각 블록을 통합하고 그에 따라 중복되거나 유효하지 않은 트랜잭션을 폐기하는 병렬 블록체인의 우선 순위 집합에서 나올 수도 있습니다.

마지막으로 검증 가능한 계산이 일반적으로 사용 가능하고 충분히 효율적이라면 작업 증명이 최종 상태를 검증할 수 있는 경로를 제공할 수 있습니다 .

14. 결론

우리는 이더리움의 프로토콜을 소개하고 토론하고 공식적으로 정의했습니다 . 이 프로토콜을 통해 독자는 이더리움 네트워크에 노드를 구현하고 분산형 보안 소셜 운영 체제에서 다른 사람들과 합류할 수 있습니다.

상호 작용 규칙을 알고리즘적으로 지정하고 자율적으로 시행하기 위해 계약을 작성할 수 있습니다 .

15. 감사의 말

홈스테드 수정을 저작한 Aeron Buchanan, Ethash 알고리즘을 저작한 Christoph Jentzsch, 대부분의 EIP-150 변경을 수행한 Yoichi Hirai 에게 감사드립니다 . Gustav Simonsson, Pawel Bylica, Jutta Steiner, Nick Savers, Viktor Tr´on, Marko Simovic , Giacomo Tazzari 및 , 물론 Vitalik Buterin.

16. 가용성

이 문서의 출처는 https: 자동 //github.com/ethereum/yellowpaper/. https://ethereum.github 에 있습니다 . io/ 생성된 PDF는 yellowpaper/paper.pdf.

참조

제이콥 아론. BitCoin 소프트웨어는 새로운 삶을 찾습니다. 뉴 사이언티스트, 213(2847):20, 2012. URL http://www.sciencedirect. com/science/article/pii/S0262407912601055.

아담 백. Hashcash - 상각 가능한 공개적으로 감사 가능한 비용 함수, 2002. URL http://www.hashcash.org/papers/amortizable.pdf .

팀 베이코, 제임스 핸콕, 토마스 제이 러시. EIP 4345: 난이도 폭탄이 2022년 6월, 2021년 10월로 연기됩니다.
URL https://eips.ethereum.org/EIPS/eip-4345.

Tim Beikoet al. 베를린 네트워크 업그레이드 사양, 2021b. URL https://github.com/ethereum/eth1.0-specs/blob/master/network-upgrades/mainnet-upgrades/berlin.md . _

Guido Bertoni, Joan Daemen, Michal Peeters, Gilles Van Assche. KECCAK SHA-3 제출 임무, 2011. URL https://keccak.team/files/Keccak-submission-3.pdf.

로만 부텔리에와 마레이케 하인젠. 해적, 피오니어, 혁신가, 모방가. 혁신을 통한 성장, 85-96페이지. 스프링거, 2014. URL https: //www.springer.com/gb/book/9783319040158.

비탈릭 부테린. 이더리움: 차세대 스마트
계약 및 분산 응용 프로그램 플랫폼, 2013a. URL https://github.com/ethereum/wiki/wiki/ 백서.

비탈릭 부테린. Dagger: A Memory-Hard to Compute, Memory-Easy-Easy Scrypt Alternative, 2013b. URL http://www.hashcash.org/papers/dagger.html.

비탈릭 부테린. EIP-2: 홈스테드 하드포크 변경 사항, 2015. URL https://eips.ethereum.org/EIPS/eip-2.

비탈릭 부테린. EIP-100: 삼촌을 포함한 목표 평균 블록 시간으로 난이도 조정 변경, 2016년 4월a.
URL https://eips.ethereum.org/EIPS/eip-100.

비탈릭 부테린. EIP-155: 단순 재생 공격 보호, 2016년 10월b. URL https://eips.ethereum. org/EIPS/eip-155.

비탈릭 부테린. EIP-1014: 스키니 CREATE2, 2018년 4월.
URL https://eips.ethereum.org/EIPS/eip-1014.

비탈릭 부테린과 마틴 스웬데. EIP-2929: 주 액세스 opcode에 대한 가스 비용 증가, 2020년 9월a. URL https://eips.ethereum.org/EIPS/eip-2929.

비탈릭 부테린과 마틴 스웬데. EIP-2930: 선택적 액세스 목록, 2020년 8월b.
URL https://eips.ethereum. org/EIPS/eip-2930.

에릭 코너. EIP-2384: Muir Glacier 난이도 폭탄 지연, 2019년 11월. URL https://eips.ethereum.org/EIPS/eip-2384.

Nicolas T. Courtois, Marek Grajek, Rahul Naik. Bitcoin Mining에서 SHA256 최적화, 131-144페이지. Springer Berlin Heidelberg, Berlin, Heidel berg, 2014. ISBN 978-3-662-44893-9. 도이: 10. 1007/978-3-662-44893-9 12. URL https://doi.org/10.1007/978-3-662-44893-9_12.

BA Davey 및 HA Priestley. 격자 및 순서 소개. 2판. 캠브리지: Cambridge University Press, 2nd ed. 에디션, 2002. ISBN 0-521-78451-4/pbk.

타데우스 드라이야. Hashimoto: I/O 작업 증명 , 2014. URL http://diyhpl.us/~bryan/papers2/ bitcoin/meh/hashimoto.pdf.

신시아 드워크와 모니 나오어. 정크 메일 처리 또는 방지를 통한 가격 책정. In 12th An nual International Cryptology Conference, 페이지 139–147 , 1992. URL http://www.wisdom.weizmann.ac.il/ ~naor/PAPERS/pvp.pdf.

Dankrad Feist, Dmitry Khovratovich, Marius van der Wijden. EIP-3607: 코드가 배포된 발신자의 트랜잭션 거부, 2021년 6월. URL https://eips.ethereum.org/EIPS/eip-3607.

퐁 보 글렌 파울러, 랜던 커트 놀 FowlerNol lVo 해시 함수, 1991. URL http://www.isthe.com/chongo/tech/comp/fnv/index.html .

Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, Sheueling Chang Shantz. 8비트 CPU에서 타원 곡선 암호화와 RSA 비교 . 암호화 하드웨어 및 임베디드 시스템-CHES 2004, 119-132페이지. Springer, 2004. URL https://www.iacr.org/archive/ches2004/31560117/31560117.pdf .

제임스 핸콕. EIP-3554: 난이도 폭탄이 2021년 12월, 2021년 5월로 연기됩니다. URL https://eips.ethereum. org/EIPS/eip-3554.

Tjaden Hess, Matt Luongo, Piotr Dyraga 및 James Hancock. EIP-152: BLAKE2 압축 기능 'F' 프리컴파일 추가 , 2016년 10월. URL https://eips.ethereum.org/EIPS/eip-152.

크리스토프 젠츠쉬. ethash 커밋 날짜, 2015. URL https://github.com/ethereum/yellowpaper/commit/77a8cf2428ce245bf6e2c39c5e652ba58a278666#commitcomment-24644869.

Don Johnson, Alfred Menezes 및 Scott Van 타원 곡선 디지털 서명 알고리즘 결석. (ECDSA), 2001. URL https: //web.archive.org/web/20170921160141/http:// cs.ucsb.edu/~koc/ ccs130h/notes/ecdsa-cert.pdf.

2017년 9월 21일에 액세스했지만 2017년 10월 19일에는 원래 링크에 액세스할 수 없었습니다 . ECDSAPUBKEY 에 대해서는 섹션 6.2를 참조 하고 ECDSASIGN 및 ECDSARECOVER에 대해서는 섹션 7을 참조하십시오.

세르지오 데미안 러너. 엄격한 메모리 하드 해싱 기능 , 2014. URL http://www.hashcash.org/papers/메모해시.pdf.

마크 밀러. 법의 미래. 1997년 8월 9일 Extro 3 컨퍼런스에서 논문 발표. URL https://drive.google.com/file/d/ 0Bw0VXJKBgYPMS0J2VGIyWWlocms/edit?usp=sharing.

나카모토 사토시. 비트코인: P2P 전자 현금 시스템, 2008. URL http://www.bitcoin.org/bitcoin. pdf.pdf

Meni Rosenfeld, Yoni Assia, Vitalik Buterin, m li orhakiLior, Oded Leiba, Assaf Shomer, Eli ran Zach. 컬러 코인 프로토콜 사양, 2012. URL https://github.com/Colored-Coins/ Colored-Coins-Protocol-Specification.

Markku-Juhani Saarinen과 Jean-Philippe Aumasson. RFC 7693: BLAKE2 암호화 해시 및 메시지 인증 코드(MAC), 2015년 11월. URL https://tools.ietf.org/html/rfc7693.

아프리 쇼던. EIP-1234: 콘스탄티노플 난이도 폭탄 지연 및 블록 보상 조정, 2018. URL https: //eips.ethereum.org/EIPS/eip-1234.

Afri Schoedon과 Vitalik Buterin. EIP-649: Metropo lis 난이도 폭탄 지연 및 블록 보상 감소, 2017년 6월. URL https://eips.ethereum.org/EIPS/eip-649 .

마이클 존 세바스찬 스미스. 애플리케이션별 집적 회로. Addison-Wesley, 1997. ISBN 0201500221.

요나탄 솜폴린스키와 아비브 조하르. 비트 코인의 트랜잭션 처리 가속화. 빠른 돈은 사슬이 아니라 나무에서 자랍니다, 2013. URL https://eprint.iacr.org/ 2013/881.

사이먼 스프랭클. 디지털 통화의 기술 기반, 2013. URL http://www.coderblog.de/wp-content/uploads/technical-basis-of-digital-currencies . pdf.pdf

Tomasz Kajetan Stanczak, Eric Marti Haynes, Josh Klopfenstein, Abhimanyu Nag. EIP-5133: 난이도 폭탄을 2022년 9월 중순, 2022년 6월로 연기합니다. URL https://eips.ethereum.org/EIPS/eip-5133.

닉 사보. 공식화 및 관계 확보 첫째 월요일, 2(9), 공용 네트워크에 배송됩니다. 1997. URL http://firstmonday.org/ojs/index.php/fm/article/view/548 .

웨이 탕. EIP-2200: 순 가스 계량에 대한 구조화된 정의, 2019. URL https://eips.ethereum.org/EIPS/eip-2200 .

Vivek Vishnumurthy, Sangeeth Chandrakumar, Emin Gn Sirer. KARMA: P2P 리소스 공유를 위한 안전한 경제 프레임워크 , 2003. URL https://www.cs.cornell.edu/people/egs/papers/karma.pdf .

JR 윌렛. MasterCoin 전체 사양, 2013. URL https://github.com/mastercoin-MSC/spec.

미카 졸투. EIP-2718: 입력된 트랜잭션 봉투, 2020년 6월. URL https://eips.ethereum.org/EIPS/eip-2718 .

부록 A. 용어

외부 행위자(External Actor): 이더리움 노드와 인터페이스할 수 있지만 이더리움 세계 외부에 있는 사람 또는 기타 엔티티 . 서명된 트랜잭션을 입금하고 블록체인 및 관련 상태를 검사하여 이더리움과 상호 작용할 수 있습니다. 하나 이상의 고유 계정이 있습니다.

주소: 계정 식별에 사용되는 160비트 코드입니다.

계정: 계정에는 이더리움 상태의 일부로 유지되는 본질적인 잔액과 트랜잭션 수가 있습니다. 또한 일부(비어 있을 수 있음) EVM 코드와 연관된(비어 있을 수 있음) 스토리지 상태가 있습니다.

동질적이긴 하지만 두 가지 실제 유형의 계정, 즉 비어 있는 연결된 EVM 코드가 있는 계정(따라서 계정 잔액은 일부 외부 엔터티에 의해 제어됨)과 비어 있지 않은 연결된 EVM 코드가 있는 계정(따라서 계정은 자율 객체를 나타냅니다). 각 계정에는 이를 식별하는 단일 주소가 있습니다.

트랜잭션: 외부 행위자가 서명한 데이터 조각입니다. 메시지 또는 새로운 자율을 나타냅니다.
물체. 트랜잭션은 블록체인의 각 블록에 기록됩니다.

Autonomous Object: 이더리움의 가상 상태에만 존재하는 개념적 객체. 고유 주소가 있으므로 연결된 계정이 있습니다. 계정에는 비어 있지 않은 연결된 EVM 코드가 있습니다. 해당 계정의 스토리지 상태로만 통합됩니다 .

저장소 상태: 특정 계정에 대한 특정 정보로, 해당 계정이
계정에 연결된 EVM 코드가 실행됩니다.

메시지: 자율 개체의 결정론적 작업 또는 트랜잭션의 암호화된 보안 서명을 통해 두 계정 간에 전달되는 데이터(바이트 집합) 및 값(Ether로 지정됨) .

메시지 호출: 한 계정에서 다른 계정으로 메시지를 전달하는 행위. 대상 계정이 비어 있지 않은 EVM 코드와 연결되어 있으면 해당 개체 및 메시지의 상태로 VM이 시작됩니다 . 메시지 발신자가 자율 객체인 경우 호출은 VM 작업에서 반환된 모든 데이터를 전달합니다.

가스: 기본 네트워크 비용 단위입니다. 필요에 따라 가스 로 또는 가스에서 자유롭게 변환되는 Ether(PoC-4 기준)에 의해 독점적으로 지불됩니다 . 가스는 내부 이더리움 계산 엔진 외부에 존재하지 않습니다. 그 가격은 트랜잭션에 의해 결정되며 광부는 가스 가격이 너무 낮은 트랜잭션을 무시할 수 있습니다.

계약: 계정과 연결될 수 있는 EVM 코드 조각 또는
자율 객체.

개체: 자율 개체의 동의어입니다.

앱: Ethereum 브라우저에서 호스팅되는 최종 사용자가 볼 수 있는 애플리케이션입니다.

Ethereum 브라우저: (Ethereum 참조 클라이언트라고도 함) 백엔드가 순수하게 Ethereum 프로토콜에 있는 샌드박스 응용 프로그램을 호스팅할 수 있는 단순화된 브라우저(Chrome) 와 유사한 인터페이스의 교차 플랫폼 GUI입니다 .

Ethereum Virtual Machine: (일명 EVM) 실행 모델의 핵심 부분을 형성하는 가상 머신
계정과 관련된 EVM 코드의 경우.

Ethereum Runtime Environment: (일명 ERE) EVM에서 실행되는 자율 객체에 제공되는 환경입니다 . EVM을 포함하지만 EVM이 CALL 및 CREATE를 포함한 특정 I/O 명령에 의존하는 세계 상태의 구조도 포함합니다 .

EVM 코드: EVM이 기본적으로 실행할 수 있는 바이트코드입니다. 공식적으로 의미를 지정하는 데 사용되며
메시지가 계정에 미치는 영향.

EVM 어셈블리: 사람이 읽을 수 있는 EVM 코드 형식입니다.

LLL: Lisp와 유사한 저수준 언어로, 간단한 계약 및 일반 문서 작성에 사용되는 사람이 작성할 수 있는 언어입니다.
트랜스 컴파일을 위한 저수준 언어 툴킷.

부록 B. 재귀 길이 접두사

임의로 구조화된 이진 데이터(바이트 배열)를 인코딩하기 위한 직렬화 방법입니다.
가능한 구조 집합 T를 정의합니다.

$$T \equiv LB \tag{188}$$

$$L \equiv \{t : t = (t[0], t[1], ...) \wedge \forall n < t : t[n] \in T\} \tag{189}$$

$$B \equiv \{b : b = (b[0], b[1], ...) \wedge \forall n < b : b[n] \in O\} \tag{190}$$

여기서 O는 (8비트) 바이트 집합입니다. 따라서 B 는 바이트의 모든 시퀀스 집합(그렇지 않으면 바이트 배열이라고도 하며 트리로 상상할 경우 잎)이고, L 은 단일 잎(가지 트리로 상상되는 경우 노드) T는 모든 바이트 배열 및 이러한 구조적 시퀀스의 집합입니다. 디스조인트 합집합은 빈 바이트 배열() ∈ B 와 빈 리스트() ∈ L을 구별하기 위해서만 필요하며, 이는 아래에 정의된 바와 같이 다르게 인코딩됩니다. 일반적 으로 우리는 표기법을 남용하고 분리된 합집합 인덱스를 암묵적으로 남겨두고 문맥에서 추론할 수 있습니다.

두 개의 하위 함수를 통해 RLP 함수를 RLP 로 정의합니다. 첫 번째 함수는 값이 바이트일 때 인스턴스를 처리합니다.
배열, 추가 값의 시퀀스인 경우 두 번째:

$$RLP(x) \equiv \begin{cases} Rb(x) & x \in B\text{인 경우} \\ R1(x) & \text{그렇지 않으면} \end{cases} \tag{191}$$

직렬화할 값이 바이트 배열인 경우 RLP 직렬화는 다음 세 가지 형식 중 하나를 사용합니다.

- 바이트 배열에 단일 바이트만 포함되고 해당 단일 바이트가 128보다 작은 경우 입력은 정확히 동일합니다.
  출력에.
- 바이트 배열이 56바이트 미만을 포함하는 경우 출력은 다음과 같은 바이트가 접두사로 붙은 입력과 같습니다.
  바이트 배열의 길이에 128을 더한 값입니다.

- 그렇지 않은 경우 출력은 264 바이트 미만을 포함 하고 빅엔디안 정수로 해석될 때 입력 바이트 배열의 길이와 동일한 최소 길이 바이트 배열이 접두사로 붙는 경우 입력과 동일합니다. 이 길이 값에 183을 더한 값을 충실하게 인코딩하는 데 필요한 바이트 수가 앞에 붙습니다.

264 바이트 이상을 포함하는 바이트 배열은 인코딩할 수 없습니다. 이 제한은 인코딩의 첫 번째 바이트가
바이트 배열의 는 항상 192 미만이므로 L의 시퀀스 인코딩과 쉽게 구분할 수 있습니다.

공식적으로 Rb를 정의합니다 .

$$\text{Rb}(x) \equiv \begin{cases} x & x = 1 \wedge x[0] < 128 \text{인 경우 } x < 56 \text{인 경우} \\ (128 + x) \cdot x & \\ 183 + \text{BE}(x) \cdot \text{BE}(x) \cdot x & \text{가 아닌 경우 } x < 2^{64} \\ \varnothing & \text{그렇지 않으면} \end{cases} \tag{192}$$

$$\text{BE}(x) \equiv (b0, b1, ...) : b0 = 0 \wedge x = \sum_{n=0}^{b-1} bn \cdot 256^{b-1-n} \tag{193}$$

$$(x1, ..., xn) \cdot (y1, ..., ym) = (x1, ..., xn, y1, ..., ym) \tag{194}$$

따라서 BE 는 음수가 아닌 정수 값을 최소 길이의 빅 엔디안 바이트 배열로 확장하는 함수이며
도트 연산자는 시퀀스 연결을 수행합니다.

대신 직렬화할 값이 다른 항목의 시퀀스인 경우 RLP 직렬화는 다음 두 가지 형식 중 하나를 취합니다.

- 포함된 각 항목의 연결된 직렬화가 길이가 56바이트 미만인 경우 출력은 동일합니다.
  이 바이트 배열의 길이에 192를 더한 것과 같은 바이트가 접두사로 붙은 연결에.
- 그렇지 않으면 출력은 264 바이트 미만을 포함하고 빅 엔디안 정수로 해석될 때 연결된 직렬화 바이트 배열의 길이와 동일한 최소 길이 바이트 배열이 접두사로 붙는 경우 연결된 직렬화와 동일합니다. 이 길이 값에 247을 더한 값을 충실하게 인코딩하는 데 필요한 바이트 수가 앞에 붙습니다 .

연결된 직렬화 항목이 264 바이트 이상인 시퀀스는 인코딩할 수 없습니다. 이 제한은 인코딩의 첫 번째 바이트가 255를 초과하지 않도록 합니다 (그렇지 않으면 바이트가 아닙니다).

따라서 공식적으로 R1을 정의하여 마무리합니다 .

$$\text{R1}(x) \equiv \begin{cases} (192 + \text{s}(x)) \cdot \text{s}(x) & \text{if } \text{s}(x) = \varnothing \wedge \text{s}(x) < 56^{64} \\ 247 + \text{BE}(\text{s}(x)) \cdot \text{BE}(\text{s}(x)) \cdot \text{s}(x) & \text{else if } \text{s}(x) = \varnothing \wedge \text{s}(x) < 2^{64} \\ \varnothing & \text{그렇지 않으면} \end{cases} \tag{195}$$

$$\text{s}(x) \equiv \begin{cases} \text{RLP}(x[0]) \cdot \text{RLP}(x[1]) \cdot ... & \text{if } \forall i : \text{RLP}(x[i]) = \varnothing \text{ 아니면} \\ \varnothing & \end{cases} \tag{196}$$

RLP가 음이 아닌 정수(N 또는 임의의 x에 대한 Nx)로만 정의된 스칼라를 인코딩하는 데 사용되는 경우 빅 엔디안 해석이 스칼라인 최단 바이트 배열로 인코딩되어야 합니다 . 따라서 음이 아닌 정수 i 의 RLP는 다음과 같이 정의 됩니다 .

$$\text{RLP}(i : i \in N) \equiv \text{RLP}(\text{BE}(i)) \tag{197}$$

RLP 데이터를 해석할 때 예상되는 조각이 스칼라로 디코딩되고 바이트 시퀀스에서 앞에 오는 0이 발견되면 클라이언트는 이를 비표준으로 간주하고 잘못된 RLP 데이터와 동일한 방식으로 취급하여 완전히 무시해야 합니다.

부호 있는 또는 부동 소수점 값에 대한 특정 표준 인코딩 형식은 없습니다.

부록 C. Hex-Prefix 인코딩

16진수 접두사 인코딩은 임의의 수의 니블을 바이트 배열로 인코딩하는 효율적인 방법입니다. 트리의 컨텍스트(사용되는 유일한 컨텍스트)에서 사용될 때 노드 유형 사이를 명확하게 하는 추가 플래그를 저장할 수 있습니다 .

부울 값 과 함께 일련의 니블(세트 Y로 표시됨)에서 일련 의 바이트(세트 B로 표시됨)로 매핑하는 함수 HP 로 정의됩니다 .

$$\text{HP}(x,t) : x \in Y \equiv \begin{cases} (16f(t), 16x[0] + x[1], 16x[2] + x[3], ...) & (16(f(t) + 1) \\ + x[0], 16x[1] + x[2], 16x[3] + x[4], ...) & \text{그렇지 않은 경우} \end{cases} \quad \text{x가 짝수인 경우} \tag{198}$$

$$\text{에프}(\text{티}) \equiv \begin{cases} t = 0 \text{인 경우 } 2 \\ \text{그렇지 않으면 } 0 \end{cases} \tag{199}$$

따라서 첫 번째 바이트의 상위 니블에는 두 개의 플래그가 포함됩니다. 가장 낮은 비트는 길이의 기이함을 인코딩하고 두 번째로 낮은 비트는 플래그 t를 인코딩합니다. 첫 번째 바이트의 하위 니블은 니블 수가 짝수일 경우 0이고 홀수일 경우 첫 번째 니블입니다. 나머지 모든 니블(이제 짝수)은 나머지 바이트에 적절하게 맞습니다.

부록 D. 수정된 Merkle Patricia 트리

수정된 Merkle Patricia 트리(trie)는 임의 길이의 이진 데이터(바이트 배열) 간에 매핑하기 위한 영구 데이터 구조를 제공합니다 . 일반적으로 데이터베이스로 구현되는 임의 길이의 이진 데이터 와 256비트 이진 조각 사이를 매핑하는 가변 데이터 구조로 정의됩니다 . 트리의 핵심 및 프로토콜 사양 측면에서 유일한 요구 사항은 주어진 키-값 쌍 세트를 식별하는 단일 값을 제공하는 것입니다. 이 값은 32바이트 시퀀스 또는 빈 바이트 시퀀스일 수 있습니다. 프로토콜의 효과적이고 효율적인 구현을 허용하는 방식으로 트리의 구조를 저장하고 유지하는 것은 구현 고려 사항으로 남겨둡니다 .

공식적으로 고유 키가 있는 바이트 시퀀스 쌍을 포함하는 집합인 입력 값 I를 가정합니다. (200)

$$I = \{(k0 \in B, v0 \in B),(k1 \in B, v1 \in B), ...\}$$

이러한 시퀀스를 고려할 때 튜플의 키 또는 값을 참조하기 위해 일반적인 숫자 아래 첨자 표기법을 사용하므로 다음과 같습니다.

(201) $$\forall I \in I : I \equiv (I0, I1)$$

일련의 바이트는 엔디안 특정 표기법이 주어지면 일련의 니블로 볼 수도 있습니다. 여기서 우리는 빅 엔디안을 가정합니다. 따라서: (202)

$$y(I) = \{(k\ 0 \in Y, v0 \in B),(k\ 1 \in Y, v1 \in B), ...\}$$

(203) $$\forall n : \forall i < 2kn : kn\ [i] \equiv \begin{cases} kn[i \div 2] \div 16 & \text{내가 짝수라면} \\ kn[i \div 2] \bmod 16 & \text{그렇지 않으면} \end{cases}$$

우리는 함수 TRIE를 정의합니다. TRIE 는 이 집합으로 인코딩될 때 이 집합을 나타내는 트리의 루트로 평가됩니다.
구조:

(204) $$TRIE(I) \equiv KEC\ RLP(c(I, 0))$$

또한 트리의 노드 캡 함수인 함수 n을 가정합니다 . 노드를 구성할 때 RLP를 사용하여 구조를 인코딩합니다. 스토리지 복잡성을 줄이기 위한 수단으로 구성된 RLP가 32바이트 미만인 노드를 직접 저장합니다. 더 큰 경우에는 Keccak-256 해시가 참조로 평가되는 바이트 배열의 예지력을 주장합니다. 따라서 우리는 노드 구성 함수인 c로 정의합니다.

(205) $$n(I, i) \equiv \begin{cases} () \in 비 & \text{만약 내가 = } \varnothing \\ c(I, i) & RLP\ c(I, i) < 32인\ 경우 \\ KEC\ RLP(c(I, i))\ 그렇지\ 않은\ 경우 & \end{cases}$$

기수 트리와 유사한 방식으로 트리가 루트에서 리프로 이동할 때 단일 키-값 쌍을 만들 수 있습니다.
키는 순회를 통해 누적되며 각 분기 노드에서 단일 니블을 획득합니다(기수 트리와 마찬가지로).
기수 트리와 달리 동일한 접두사를 공유하는 여러 키의 경우 또는 고유한 접미사를 갖는 단일 키의 경우 두 개의 최적화 노드가 제공됩니다. 따라서 트래버스하는 동안 다른 두 노드 유형(확장 및 리프) 각각 에서 잠재적으로 여러 개의 니블을 얻을 수 있습니다 . 트리에는 세 가지 종류의 노드가 있습니다. 리프(Leaf): 첫 번째 항목이 루트에서 이동한 키 및 가지의 누적 으로 아직 설명되지 않은 키의 니블에 해당하는 2개 항목 구조입니다 . hex-prefix 인코딩 방법이 사용되며 함수에 대한

두 번째 매개 변수는 1이어야 합니다.

확장: 첫 번째 항목이 니블의 키와 루트에서 이동하는 분기 키의 누적을 지나 적어도 두 개의 개별 키가 공유하는 크기보다 큰 일련의 니블에 해당하는 두 항목 구조입니다 . hex-prefix 인코딩 방법이 사용되며 함수에 대한 두 번째 매개 변수는 0이어야 합니다.

분기: 처음 16개 항목이 탐색의 이 시점에서 키 에 대해 가능한 16개의 니블 값 각각에 해당하는 17개 항목 구조입니다 . 17번째 항목은 이것이 종결자 노드인 경우에 사용되며 따라서 탐색의 이 지점에서 키가 종료됩니다.

분기는 필요할 때만 사용됩니다. 0이 아닌 단일 항목만 포함하는 분기 노드는 존재할 수 없습니다. 우리
구조 구성 함수 c를 사용하여 이 구조를 공식적으로 정의할 수 있습니다.

(206) $$c(I,i) \equiv \begin{cases} HP(I0[i..(I0 - 1)], 1), I1 & \text{if } I = 1 \text{ 여기서 } \exists I : I \in I \\ HP(I0[i..(j - 1)], 0), n(I, j)(u(0), u(1).), ..., u(15), v) & \text{if } i = j \text{ where } j = \max\{x : \exists l : l = x \wedge \forall I \in I : I0[0..(x \quad 1)] = l\} \text{ 그렇지 않으면 } u(j) \equiv n(\{I : I \in I \wedge I0[i] = j\}, i + 1) \\ I1 & \text{만약 } \exists I : I \in I \wedge I0 = i \\ () \in B\ 그렇지\ 않으면 & \end{cases}$$

안으로 =

D.1. 트라이 데이터베이스. 따라서 어떤 데이터가 저장되고 어떤 데이터가 저장되지 않는지에 대한 명시적인 가정이 이루어지지 않습니다. 이는 구현 관련 고려 사항이기 때문입니다. 우리는 단순히 키-값 세트 I를 32바이트 해시에 매핑하는 항등 함수를 정의하고 모든 I에 대해 그러한 해시가 하나만 존재한다고 주장합니다 . 엄밀히 말하면 사실은 아니지만 Keccak 해시의 충돌 저항이 주어진 허용 가능한 정밀도 내에서 정확합니다. 실제로 합리적인 구현은 각 세트에 대한 트리 루트 해시를 완전히 다시 계산하지 않습니다.

합리적인 구현은 다양한 시도의 계산에서 결정된 노드의 데이터베이스를 유지하거나 더 공식적으로 함수 c를 기억합니다. 이 전략은 시도의 특성을 사용하여 내용을 쉽게 기억합니다.

이전의 키-값 세트를 사용하고 이러한 세트를 매우 효율적인 방식으로 여러 개 저장합니다. 종속 관계로 인해,
Merkle-proofs는 특정 리프가 존재해야 함을 입증할 수 있는 O(log N) 공간 요구 사항 으로 구성될 수 있습니다.
주어진 루트 해시의 트라이 내에서.

부록 E. 미리 컴파일된 계약

미리 컴파일된 각 계약에 대해 out-of-gas 검사를 구현하는 템플릿 기능인 ΞPRE를 사용합니다 .

$$\Xi_{PRE}(\sigma, g, A, I) \equiv \begin{cases} (\varnothing, 0, A, ()) & g < gr \text{ 인 경우} \\ (\sigma, g - gr, A, o) & \text{그렇지 않은 경우} \end{cases} \tag{207}$$

미리 컴파일된 계약은 각각 이러한 정의를 사용하고 o (출력 데이터) 및 gr 에 대한 사양을 제공합니다 .
가스 요구 사항.

우리는 ΞECREC을 타원 곡선 디지털 서명 알고리즘(ECDSA) 공개 키 복구를 위한 사전 컴파일된 계약으로 정의합니다.
기능(ecrecover). 함수 ECDSARECOVER 및 상수 secp256k1n의 정의에 대해서는 부록 F를 참조하십시오. 우리
또한 필요에 따라 0을 추가하여 무한 길이에 대해 잘 정의된 입력 데이터로 d를 정의합니다. 의 경우
유효하지 않은 서명은 출력을 반환하지 않습니다.

(208) $\Xi_{ECREC} \equiv \Xi_{PRE}$ 여기서:

(209) $gr = 3000$

(210) $$= \begin{cases} 0 & v \notin \{27, 28\} \vee r = 0 \vee r \geq secp256k1n \vee s = 0 \vee s \geq secp256k1n \\ ECDSARECOVER(h, v - 27, r, s) = \varnothing \text{인 경우 } 0 \\ \text{그렇지 않으면 } 32 \end{cases}$$

(211) $o = 32$인 경우:

(212) $o[0..11] = 0$

(213) $o[12..31] = KEC\ ECDSARECOVER(h, v \quad 27, r, s) \quad [12..31]$ 여기서:

(214) $d[0..(Id - 1)] = Id$

(215) $d[ID..] = (0, 0, ...)$

(216) $시간 = d[0..31]$

(217) $v = d[32..63]$

(218) $r = d[64..95]$

(219) $s = d[96..127]$

우리는 ΞSHA256 및 ΞRIP160을 SHA2-256 및 RIPEMD-160 해시 함수를 구현하는 사전 컴파일된 계약으로 정의합니다.
각기. 가스 사용량은 가장 가까운 단어 수로 반올림된 요소인 입력 데이터 크기에 따라 달라집니다.

(220) $\Xi_{SHA256} \equiv \Xi_{PRE}$ 여기서:

(221) $$gr = 60 + 12 \quad \frac{ID}{32}$$

(222) $o[0..31] = SHA256(Id)$

(223) $\Xi_{RIP160} \equiv \Xi_{PRE}$ 여기서:

(224) $$gr = 600 + 120 \quad \frac{ID}{32}$$

(225) $o[0..11] = 0$

(226) $o[12..31] = RIPEMD160(Id)$

여기에서는 RIPEMD-160에 대해 잘 정의된 표준 암호화 기능이 있다고 가정합니다.
다음 형식의 SHA2-256:

(227) $$SHA256(i \in B) \equiv o \in B32$$

(228) $$RIPEMD160(i \in B) \equiv o \in B20$$

네 번째 계약인 항등 함수 ΞID는 단순히 출력을 입력으로 정의합니다.

(229) $\Xi_{ID} \equiv \Xi_{PRE}$ 여기서:

(230) $$gr = 15 + 3 \quad \frac{ID}{32}$$

(231) $오 = 아이디$

다섯 번째 계약은 모듈로에서 임의 정밀도 지수를 수행합니다. 여기서 00을 1로 하고 x mod 0
모든 x 에 대해 0입니다 . 입력의 첫 번째 단어는 첫 번째 음수가 아닌 정수 B 가 차지하는 바이트 수를 지정합니다 .
입력의 두 번째 단어는 두 번째 음수가 아닌 정수 E가 차지하는 바이트 수를 지정합니다. 세 번째

입력의 단어는 음이 아닌 세 번째 정수 M 이 차지하는 바이트 수를 지정합니다 . 이 세 단어 뒤에는 B, E 및 M이 옵니다. 나머지 입력은 버려집니다. 입력이 너무 짧을 때마다 누락된 바이트는 0으로 간주됩니다. 출력은 M과 동일한 형식으로 빅 엔디안으로 인코딩됩니다.

(232) ΞEXPMOD ≡ ΞPRE 제외:

(233) $$gr = \text{최대}\ 200, \quad \frac{f\ \text{최대}(M, B)\ \text{최대}(E, 1)}{Gquaddivisor}$$

(234) 쿼드디바이저 ≡ 3

(235) $$\text{에프}(\text{엑스}) \equiv \frac{\text{엑스}^{2}}{8}$$

(236) $$= \begin{cases} 0 & \text{한약재}\ E \leq 32 \wedge E = 0 \\ \log2(E) & \text{한약재}\ E \leq 32 \wedge E = 0 \\ 8(E - 32) + \log2(i[(96 + B)..(127 + B)]) & 32 < E \wedge i[(96 + B)..(127 + B)] = 0 \text{ 인 경우} \\ 8(E - 32) & \text{그렇지 않으면} \end{cases}$$

(237) $$o = B^{E \bmod M} \in N8M$$

(238) B ≡ i[0..31]

(239) E ≡ i[32..63]

(240) M ≡ i[64..95]

(241) B ≡ i[96..(95 + B)]

(242) E ≡ i[(96 + B)..(95 + B + E)]

(243) M ≡ i[(96 + B + E)..(95 + B + E + M)]

(244) $$i[\text{엑스}] \equiv \begin{cases} Id[x] & \text{if } x < Id\ 0 \\ & \text{그렇지 않으면} \end{cases}$$

E.1. zkSNARK 관련 미리 컴파일된 계약. 우리는 소수인 두 개의 숫자를 선택합니다.

(245) p ≡ 21888242871839275222246405745257275088696311157297823662689037894645226208583 q ≡

(246) 75088548364400416034343698204186575808495617

p 는 소수 이므로 {0, 1, . . . , p 1}은 덧셈 및 곱셈 모듈로 p를 사용하여 필드를 형성합니다 . 우리는 이 필드를 Fp라고 부릅니다.
다음과 같이 집합 C1을 정의합니다.

(247) $$C1 \equiv \{(X, Y) \in Fp \times Fp \mid \text{와이}^{2} = X^{\text{삼}} + 3\} \cup \{(0, 0)\}$$

다음과 같이 고유한 요소 (X1, Y1),(X2, Y2) 에 대해 C1 에서 이진 연산 +를 정의합니다.

(248) $$(X1, Y1) + (X2, Y2) \equiv \begin{cases} (X, Y)\ X1 = X2\ (0, 0)\text{이면 그} \\ \text{렇지 않으면} \end{cases}$$

$$\lambda \equiv \frac{Y2 - Y1}{X2 - X1}$$

$$X \equiv \lambda^{2} - X1 - X2$$

$$Y \equiv \lambda(X1 - X) - Y1$$

(X1, Y1) = (X2, Y2) 인 경우 , 다음과 같이 C1 에 +를 정의합니다.

(249) $$(X1, Y1) + (X2, Y2) \equiv \begin{cases} (X, Y)\ Y1 = 0\text{인 경우 (0,} \\ 0)\ \text{그렇지 않은 경우} \end{cases}$$

$$\lambda \equiv \frac{3X1^{2}}{2년1}$$

$$X \equiv \lambda^{2} \quad 2X1$$

$$Y \equiv \lambda(X1 - X) - Y1$$

(C1, +)는 그룹을 형성하는 것으로 알려져 있습니다. 우리는 스칼라 곱셈을 정의합니다.

(250) $$n \cdot P \equiv (0, 0) + P + \cdots + P \quad (N\text{번})$$

자연수 n과 C1의 점 P에 대해.
P1을 C1의 점(1, 2) 으로 정의합니다 . G1을 P1에 의해 생성된 (C1, +) 의 부분군이라고 합니다 . G1 은 차수가 q 인 순환 그룹 으로 알려져 있습니다 . G1 의 점 P 에 대해 logP1 (P)을 n · P1 = P를 만족하는 가장 작은 자연수 n 으로 정의합니다 . logP1 (P)은 최대 q 1입니다.

Fp2를 필드 Fp [i]/(i ^{2} + 1). 다음과 같이 세트 C2를 정의합니다.

(251) C2 ≡ {(X, Y) ∈ Fp2 × Fp2 | 와이 ^{2} = X ^{삼} + 3(i + 9) 1 } ∪ {(0, 0)}

동일한 방정식 (248), (249) 및 (250)을 사용하여 이진 연산 + 및 스칼라 곱셈 을 정의합니다 . (C2, +)도 그룹인 것으로 알려져 있습니다. 다음과 같이 C2 에서 P2를 정의합니다.

(252) P2 ≡ (11559732032986387107991004021392285783925812861821192530917403151452391805634 × i
+10857046999023057135944570762232829481370756359578518086990519993285655852781,
40823678758634336813322034031454355683168513275934012081057410762141200935 31 × 나는
+84956539231234314176049732474892724384181905872636001487702806493069581019 30)

우리는 G2를 P2에 의해 생성된 (C2, +) 의 하위 그룹으로 정의합니다 . G2 는 C2 에서 차수가 q 인 유일한 순환 그룹인 것으로 알려져 있습니다 .
G2 의 점 P 에 대해 logP2 (P)를 n · P2 = P를 만족하는 가장 작은 자연수 n 으로 정의합니다 . 이 정의에서 logP2 (P)는 최대 q 1입니다.

GT를 Fq 12 밑에 있는 곱셈 아벨 그룹이라고 합니다 . Non-degenerate bilinear map e : G1 × G2 → GT가 존재하는 것으로 알려져 있다. 이 쌍선형 맵은 유형 3 페어링입니다. 이러한 쌍선형 맵이 여러 개 있으며 e로 선택되는 것은 중요하지 않습니다. PT = e(P1, P2), a 는 G1 의 k 포인트 집합 , b 는 G2 의 k 포인트 집합 이라고 합니다 . 페어링의 정의에 따르면 다음은 동등합니다 .

(253) logP1 (a1) × logP2 (b1) + · · · + logP1 (ak) × logP2 (bk) ≡ 1 mod q

(254) $\prod_{i=0}^{n}$ e (ai, bi) = 태평양 표준시

따라서 페어링 동작은 검증 방법을 제공한다(253).

32바이트 숫자 x ∈ P256 은 Fp 의 요소를 나타낼 수도 있고 나타내지 않을 수도 있습니다 .

(255) δp(x) ≡ x인 경우 x < p
∅ 그렇지 않은 경우

64바이트 데이터 x ∈ B512 는 G1 의 요소를 나타낼 수도 있고 나타내지 않을 수도 있습니다 .

(256) δ1(x) ≡ g1 ∈ G1 이면 g1
∅ 그렇지 않으면

(257) g1_ _ (x, y) if x = ∅ ∧ y = ∅ ∅ 아니면

(258) x ≡ δp(x[0..31]) y ≡

(259) δp(x[32..63])

128바이트 데이터 x ∈ B1024 는 G2 의 요소를 나타낼 수도 있고 나타내지 않을 수도 있습니다 .

(260) δ2(x) ≡ g2 이면 g2 ∈ G2
∅ 그렇지 않으면

(261) g2 ≡ ((x0i + y0),(x1i + y1)) if x0 = ∅ ∧ y0 = ∅ ∧ x1 = ∅ ∧ y1 = ∅ 아니면
∅

(262) x0 ≡ δp(x[0..31]) y0 ≡

(263) δp(x[32..63]) x1 ≡

(264) δp(x[64..95]) y1 ≡

(265) δp(x[96..127])

우리는 ΞSNARKV를 zkSNARK 검증에서 의도된 사용을 위해 (253) 보유 여부를 확인하는 미리 컴파일된 계약으로 정의합니다 .

(266) ΞSNARKV ≡ ΞPRE 제외:

(267)ΞSNARKV(σ, g, A, I) = (∅, 0, A,()) if F

(268) F ≡ (Id mod 192 = 0 ∨ (∃j.aj = ∅ ∨ bj = ∅))

(269) 케이 = ID / 192

(270) gr = 34000k + 45000

(271) o[0..31] ≡ v ∧ ¬F인 경우 0x0000000000000000000000000000000000000000000000000000000000000001
0x0000000000000000000000000000000000000000000000000000000000000000 ¬v ∧ ¬F인 경우

(272) v ≡ (logP1 (a1) × logP2 (b1) + · · · + logP1 (ak) × logP2 (bk) ≡ 1 모드 q)

(273) a1 ≡ δ1(Id[0..63])

(274) b1 ≡ δ2(Id[64..191])

⋮

(275) ak ≡ δ1(Id[(Id - 192)..(Id - 129)])

(276) bk ≡ δ2(Id[(Id - 128)..(Id - 1)])

우리는 G1에 추가하기 위해 미리 컴파일된 계약을 정의합니다 .

(277) ΞBN_ADD ≡ ΞBN PRE 제외:

(278) ΞBN_ADD(σ, g, A, I) = (∅, 0, A,()) if x = ∅ ∨ y = ∅

(279) gr = 150

(280) o ≡ d $^{-1}_{1}$ (x + y) 여기서 +는 G1 의 그룹 연산입니다.

(281) x ≡ δ1 I¯d[0..63]

(282) y ≡ δ1 I¯d[64..127]

(283) I?d[x] ≡ ID[x] if x < ID
0 그렇지 않으면

우리는 G1에서 스칼라 곱셈을 위해 미리 컴파일된 계약을 정의합니다 . 여기서 Id는 (283)에서 정의됩니다.

(284) ΞBN_MUL ≡ ΞPRE 제외:

(285) ΞBN_MUL(σ, g, A, I) = (∅, 0, A,()) if x = ∅

(286) gr = 6000

(287) o ≡ d $^{-1}_{1}$ (n · x) 여기서 ·는 G1 의 스칼라 곱셈입니다.

(288) x ≡ δ1 I¯d[0..63]

(289) n ≡ I¯ d[64..95]

E.2. BLAKE2 사전 컴파일된 계약. Hess 등의 EIP-152. [2016] ΞBLAKE2 F를 사전 컴파일된 계약으로 정의 _ BLAKE2 암호화 해싱 알고리즘에 사용되는 압축 함수 F를 구현합니다. F 압축 _

기능은 Saarinen 및 Aumasson [2015]의 RFC 7693에 지정되어 있습니다.

(290) ΞBLAKE2 F ≡ ΞPRE 제외:

(291) ΞBLAKE2 F(σ, g, A, I) = (∅, 0, A,()) if Id = 213 ∨ f /∈ {0, 1}

(292) gr = r

(293) o ≡ LE8(시간 0) · ··· · LE8(시간7)

(294) (h 0, ..., h 7) r 라운드 및 w = 64인 ≡ F(h, m, tlow, thigh, f)

(295) BE4(r) ≡ ID[0..4]

(296) LE8(h0) ≡ ID[4..12]

(297) ...

(298) LE8(h7) ≡ Id[60..68]

(299) LE8(m0) ≡ Id[68..76]

(300) ...

(301) LE8(m15) ≡ ID[188..196]

(302) LE8(tlow) ≡ Id[196..204]

(303) LE8(허벅지) ≡ 아이디[204..212]

(304) f ≡ ID[212]

여기서 r ∈ B32, ∀i ∈ 0..7 : hi ∈ B64, ∀i ∈ 0..15 : mi ∈ B64, tlow ∈ B64, 허벅지 ∈ B64, f ∈ B8, BEk 는 k-byte big-endian 표현 - 비교(193):

$$\text{BEk}(x) \equiv (b0, b1, ..., bk\text{-}1) : x = \sum_{n=0}^{k-1} \text{십억} \cdot 256^{k-1-n} \tag{305}$$

LEk 는 k바이트 리틀 엔디안 표현입니다.

$$\text{LEk}(x) \equiv (b0, b1, ..., bk\text{-}1) : x = \sum_{n=0}^{k-1} \text{십억} \cdot 256^{n} \tag{306}$$

부록 F. 트랜잭션 서명

트랜잭션은 복구 가능한 ECDSA 서명을 사용하여 서명됩니다. 이 방법은 설명한 대로 SECP-256k1 곡선을 활용합니다.
Courtois 외. [2014], Gura et al. [2004] 페이지. 15 중 9, 단락. 삼.

보낸 사람이 임의로 선택한 양의 정수인 유효한 개인 키 pr을 가지고 있다고 가정합니다 (
길이가 32인 바이트 배열(빅 엔디안 형식) 범위 [1, secp256k1n 1].

ECDSAPUBKEY, ECDSASIGN 및 ECDSARECOVER 함수가 있다고 가정합니다 . 이들은 공식적으로
문헌, 예를 들어 Johnson et al. [2001]. (307) (308)

$$\text{ECDSAPUBKEY}(pr \in B32) \equiv pu \in B64 \tag{309}$$

$$\text{ECDSASIGN}(e \in B32, pr \in B32) \equiv (v \in B1, r \in B32, s \in B32)$$

$$\text{ECDSARECOVER}(e \in B32, v \in B1, r \in B32, s \in B32) \equiv pu \in B64$$

여기서 pu 는 크기가 64인 바이트 배열(각각 < 2 256 의 두 양의 정수를 연결하여 형성됨)인 공개 키이고, pr 은 크기가 32인 바이트 배열(또는 단일 양의 정수인 개인 키)입니다. 앞서 언급한 범위) e 는 트랜잭션의 해시 h(T)입니다. v 는 '복구 식별자'라고 가정합니다 . 복구 식별자는 r이 x값인 곡선점 좌표의 패리티 및 유한성을 지정하는 1바이트 값입니다 . 이 값은 [0, 3] 의 범위에 있지만 무한 값을 나타내는 위의 두 가지 가능성을 무효로 선언합니다. 값 0은 짝수 y 값을 나타내고 1은 홀수 y 값을 나타냅니다.

다음 조건이 모두 참이 아닌 한 ECDSA 서명이 유효하지 않음을 선언합니다.

$$0 < r < \text{secp256k1n} \tag{310}$$

$$0 < s < \text{secp256k1n} \div 2 + 1 \tag{311}$$

$$\text{in} \in \{0, 1\} \tag{312}$$

어디:

$$\text{secp256k1n} = 115792089237316195423570985008687907852837564279074904382605163141518161494337 \tag{313}$$

s 에 대한 이 제한은 ΞECREC 사전 컴파일 의 제한 210보다 더 엄격합니다 . 자세한 내용은 Buterin [2015]의 EIP-2를 참조하십시오 .

주어진 개인 키 pr 에 대해 해당하는 이더리움 주소 A(pr) (160비트 값)는 해당 ECDSA 공개 키의 Keccak-256 해시의 가장 오른쪽 160비트로 정의됩니다.

$$A(pr) = B96..255\ \text{KEC}\ \text{ECDSAPUBKEY}(pr) \tag{314}$$

서명 할 메시지 해시 h(T)는 트랜잭션의 Keccak-256 해시입니다. 서명의 세 가지 다른 맛
계획을 사용할 수 있습니다:

$$\text{LX}(T) \equiv \begin{cases} (Tn, Tp, Tg, Tt, Tv, p) & Tx = 0 \wedge Tw \in \{27, 28\} \text{ 인 경우} \\ (Tn, Tp, Tg, Tt, Tv, p, \beta, (), ()) & \text{if } Tx = 0 \wedge Tw \in \{2\beta + 35, 2\beta + 36\} \\ (Tc, Tn, Tp, Tg, Tt, Tv, p, TA) & Tx = 1 \text{ 인 경우} \end{cases} \tag{315}$$

어디

$$\text{피} \equiv \begin{cases} Ti & \text{if } Tt = \varnothing \\ Td & \text{그렇지 않으면} \end{cases}$$

$$h(T) \equiv \begin{cases} \text{KEC}(\text{RLP}(\text{LX}(T))) & Tx = 0 \text{ 인 경우} \\ \text{KEC}(Tx \cdot \text{RLP}(\text{LX}(T))) & \text{그렇지 않으면} \end{cases} \tag{316}$$

서명된 트랜잭션 G(T, pr) 은 다음과 같이 정의됩니다. (317) (318)

$$G(T, pr) \equiv T \text{ 제외:}$$

$$(Ty, Tr, Ts) = \text{ECDSASIGN}(h(T), pr)$$

이전에서 반복: (319) (320)

$$Tr = r$$

$$TS = 초$$

레거시 변환의 Tw 는 27 + Ty 또는 2β + 35 + Ty입니다.

그런 다음 트랜잭션의 발신자 기능 S를 다음과 같이 정의할 수 있습니다.

(321) $$S(T) \equiv B_{96..255}(\text{KEC}(\text{ECDSARECOVER}(h(T), v, T_r, T_s)))$$

(322) $$v \equiv \begin{cases} T_w - 27 & \text{if } T_x = 0 \wedge T_w \in \{27, 28\} \\ (T_w - 35) \bmod 2 & \text{if } T_x = 0 \wedge T_w \in \{2\beta + 35, 2\beta + 36\} \\ T_y & \text{if } T_x = 1 \end{cases}$$

서명된 트랜잭션의 발신자가 서명자의 주소와 같다는 주장은 자명해야 합니다.

(323) $$\forall T : \forall pr : S(G(T, pr)) \equiv A(pr)$$

부록 G. 수수료 일정

요금표 G는 상대적인 비용(가스)에 해당하는 스칼라 값의 튜플입니다.
트랜잭션이 영향을 줄 수 있는 작업.

이름	값 설명
G_{zero}	0 집합 W_{zero}의 연산에 대해 지불된 것이 없습니다.
$G_{jumpdest}$	1 JUMPDEST 작업 에 대해 지불할 가스의 양 .
G_{base}	2 설정된 W_{base}의 작동에 대해 지불할 가스의 양.
$G_{verylow}$	3 설정된 $W_{verylow}$의 작업에 대해 지불할 가스의 양.
G_{low}	5 설정된 W_{low}의 작업에 대해 지불할 가스의 양.
G_{mid}	8 설정된 W_{mid}의 작업에 대해 지불할 가스의 양.
G_{high}	10 설정된 W_{high}의 작업에 대해 지불할 가스의 양.
$G_{warmaccess}$	100 웜 계정 또는 스토리지 액세스 비용.
$G_{accesslistaddress}$	2400 액세스 목록이 있는 계정 워밍업 비용.
$G_{accessliststorage}$	1900 액세스 목록이 있는 스토리지 워밍업 비용.
$G_{coldaccountaccess}$	2600 콜드 계정 액세스 비용.
$G_{coldsload}$	2100 콜드 스토리지 액세스 비용.
G_{sset}	20000 저장 값이 0에서 0이 아닌 값으로 설정된 경우 SSTORE 작업에 대해 지불합니다.
G_{sreset}	2900 저장 값의 0이 변경되지 않은 경우 SSTORE 작업에 대해 지불하거나 0으로 설정됩니다.
R_{sclear}	보관 가치가 0으로 설정되면 15000 환불 제공(환불 카운터에 추가됨) 0이 아닙니다.
$R_{selfdestruct}$	계정 자체 파괴에 대해 24000 환불(환불 카운터에 추가).
$G_{selfdestruct}$	5000 SELFDESTRUCT 작업 에 대해 지불할 가스의 양입니다 .
G_{create}	32000 CREATE 작업에 대해 지불했습니다.
$G_{codedeposit}$	200 CREATE 작업이 코드를 상태에 배치하는 데 성공하기 위해 바이트당 지불합니다 .
$G_{callvalue}$	9000 CALL 작업 의 일부로 0이 아닌 값 전송에 대해 지불했습니다 .
$G_{callstipend}$	2300 0이 아닌 값 전송에 대해 Gcallvalue 에서 차감된 호출된 계약에 대한 급여입니다 .
$G_{newaccount}$	25000 계정을 만드는 CALL 또는 SELFDESTRUCT 작업에 대해 지불했습니다.
G_{exp}	10 EXP 작업에 대한 부분 지불.
$G_{expbyte}$	50 EXP 연산 에 대한 지수의 바이트 수를 곱할 때 부분 지불 .
G_{memory}	3 메모리 확장 시 모든 추가 단어에 대해 지불합니다.
$G_{txcreate}$	32000 홈스테드 전환 후 모든 계약 생성 트랜잭션에 의해 지불됩니다.
$G_{txdataeevery}$	4 트랜잭션에 대한 모든 0바이트의 데이터 또는 코드에 대해 지불합니다.
$G_{txdatanonzero}$	16 트랜잭션에 대한 데이터 또는 코드의 0이 아닌 모든 바이트에 대해 지불합니다.
$G_{transaction}$	모든 거래에 대해 21000 지불.
G_{log}	375 LOG 작업 에 대한 부분 지불입니다 .
$G_{logdata}$	8 LOG 작업 데이터의 각 바이트에 대해 지불됩니다.
$G_{logtopic}$	375 LOG 작업의 각 주제에 대해 지불했습니다.
$G_{keccak256}$	30 각 KECCAK256 작업에 대해 지불됩니다.
$G_{keccak256word}$	6 KECCAK256 연산 에 대한 입력 데이터의 각 단어(반올림)에 대해 지불됩니다 .
G_{copy}	3 복사된 단어를 곱하고 반올림한 *COPY 작업에 대한 부분 지불.
$G_{blockhash}$	20 각 BLOCKHASH 작업에 대한 지불.

부록 H. 가상 머신 사양

256비트 이진 값을 정수로 해석할 때 표현은 빅 엔디안입니다.
256비트 기계 데이텀이 160비트 주소 또는 해시로 변환되거나 그 반대로 변환되면 오른쪽(BE의 경우 하위)
20바이트가 사용되고 가장 왼쪽 12바이트는 버리거나 0으로 채워집니다.
빅 엔디안으로 해석됨)은 동일합니다.

H.1. 가스 비용. 일반 가스 비용 함수 C는 다음과 같이 정의됩니다.

(324)

$$C(\sigma, \mu, A, I) \equiv C_{mem}(\mu_i') - C_{mem}(\mu_i) + \begin{cases} C_{SSTORE}(s, \mu, A, I) & w = \text{S스토어 인 경우} \\ G_{exp} & w = \text{EXP} \wedge \mu_s[1] = 0 \text{ 인 경우} \\ G_{exp} + G_{expbyte} \times (1 + \log_{256}(\mu_s[1])) & w = \text{EXP} \wedge \mu_s[1] > 0 \text{ 인 경우} \\ G_{verylow} + G_{copy} \times \mu_s[2] \div 32 & \text{if } w \in W_{copy} \\ C_{aaccess}(\mu_s[0] \bmod 2^{160}, A) + G_{copy} \times \mu_s[3] \div 32 & w = \text{EXTCODECOPY 인 경우} \\ C_{aaccess}(\mu_s[0] \bmod 2^{160}, A) & \text{if } w \in W_{extaccount} \text{ if} \\ G_{log} + G_{logdata} \times \mu_s[1] & w = \text{LOG0} \\ G_{log} + G_{logdata} \times \mu_s[1] + G_{logtopic} & w = \text{LOG1 인 경우} \\ G_{log} + G_{logdata} \times \mu_s[1] + 2G_{logtopic} & w = \text{LOG2 인 경우} \\ G_{log} + G_{logdata} \times \mu_s[1] + 3G_{logtopic} & w = \text{LOG3 인 경우} \\ G_{log} + G_{logdata} \times \mu_s[1] + 4G_{logtopic} & w = \text{LOG4 인 경우} \\ C_{CALL}(s, \mu, A) & \text{if } w \in W_{call} \text{ if } w \\ C_{SELFDESTRUCT}(s, \mu) & = \text{SELFDESTRUCT} \\ G_{create} & w = \text{생성 인 경우} \\ G_{create} + G_{keccak256word} \times \mu_s[2] \div 32 & w = \text{CREATE2 인 경우} \\ G_{keccak256} + G_{keccak256워드} \times \mu_s[1] \div 32 & w = \text{KECCAK256 인 경우} \\ G_{점프스트} & w = \text{JUMPDEST 인 경우} \\ C_{SLOAD}(\mu, A, I) & w = \text{SLOAD 인 경우} \\ G_{지제로} & \text{if } w \in W_{zero} \text{ if } w \\ \text{그것을 얻으십시오} & \in W_{base} \text{ if } w \in \\ G_{그베리로우} & W_{verylow} \text{ if } w \in W_{low} \\ \text{붙타는 돗판 빛깔} & \\ G_{지미드} & w \in W_{mid} \text{ 인 경우} \\ G_{다리} & \text{if } w \in W_{high} \text{ if } w \\ G_{blockhash} & = \text{BLOCKHASH} \end{cases}$$

(325)

$$\equiv \text{에서} \quad \begin{cases} \mu_{pc} < I_b \text{ 인 경우 } I_b[\mu_{pc}] \\ \text{그렇지 않으면 중지} \end{cases}$$

어디:

(326)

$$C_{mem}(a) \equiv G_{메모리} \cdot a + \frac{시^2}{512}$$

(327)

$$C_{aaccess}(x, A) \equiv \begin{cases} x \in A_a \text{ 인 경우 } G_{warmaccess} \\ \text{그렇지 않으면 } G_{coldaccountaccess} \end{cases}$$

아래 해당 섹션에 지정된 대로 CCALL, CSELFDESTRUCT, CSLOAD 및 CSSTORE를 사용합니다 . 우리는 다음을 정의합니다 지침의 하위 집합:

W_{zero} = {중지, 복귀, 되돌리기}
W_{base} = {ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, GASPRICE, COINBASE, TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, CHAINID, RETURNDATASIZE, POP, PC, MSIZE, GAS}
$W_{verylow}$ = {ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, SHL, SHR, SAR, CALLDATALOAD, MLOAD, MSTORE, MSTORE8, 푸시*, DUP*, 스왑*}
W_{low} = {MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND, SELFBALANCE}
W_{mid} = {ADDMOD, MULMOD, 점프}
와이 = {JUMPI}
W_{copy} = {CALLDATACOPY, CODECOPY, RETURNDATACOPY}
W_{call} = {CALL, CALLCODE, DELEGATECALL, STATICCALL}
$W_{extaccount}$ = {잔액, EXTCODESIZE, EXTCODEHASH}

G_{memory} 의 곱으로 주어지는 메모리 비용 구성 요소 와 최대 0 및 메모리가 현재 단어 수 이상이어야 하는 단어 수의 상한선 μ_i' 에 유의하십시오. 읽거나 쓰십시오. 이러한 액세스는 0이 아닌 바이트 수에 대한 것이어야 합니다.

길이가 0인 범위를 참조하는 경우(예: CALL에 대한 입력 범위로 전달 시도) 메모리를 범위 시작 부분까지 확장할 필요가 없습니다 . μ_i' 는 활성 메모리의 새로운 최대 단어 수로 정의됩니다. 이 두 가지가 같지 않은 특별한 경우가 주어집니다.

또한 Cmem 은 메모리 비용 함수입니다(확장 함수는 전후 비용의 차이임). 이것은 고차 계수가 나누어지고 바닥이 있는 다항식이므로 최대 704B의 메모리가 사용되며 그 이후에는 훨씬 더 많은 비용이 듭니다.

명령어 세트를 정의하는 동안 범위 함수 M에 대한 메모리 확장을 정의했습니다.

$$M(s, f, l) \equiv \begin{cases} s & l = 0 \text{인 경우} \\ \max(s, (f + l) \div 32) & \text{그렇지 않으면} \end{cases} \tag{328}$$

또 다른 유용한 함수는 다음과 같이 정의되는 "64분의 1을 제외한 모든" 함수 L입니다.

$$L(n) \equiv n - n/64 \tag{329}$$

H.2. 명령어 세트. 섹션 9에서 이전에 명시된 바와 같이 이러한 정의는 최종 컨텍스트에서 발생합니다. 특히 우리는 O가 EVM 상태 진행 함수라고 가정하고 다음과 같이 다음 주기의 상태 (σ , μ)와 관련된 항을 정의합니다.

$$O(\sigma, \mu, A, I) \equiv (\sigma , \mu , \dagger , I) \text{ 명시된 바와 같이 예외가 있음} \tag{330}$$

여기에 J 및 C 의 추가 명령어별 정의와 함께 각 명령어에 대해 지정된 섹션 9에 주어진 상태 전이 규칙에 대한 다양한 예외가 제공됩니다. 각 명령어에 대해 또한 지정된 α, 스택에 배치된 추가 항목 및 δ, 섹션 9에 정의된 대로 스택에서 제거된 항목입니다.

0s: 중지 및 산술 연산 모든 산술 연산은 달리 명시되지 않는 한
모듈로 2256 입니다 . 영 00 의 영승은 1로 정의된다.

값 니모닉 d 설명

0x00 정지 0 0 실행을 중지합니다.

0x01 추가 2 1 더하기 연산. [0] ≡ μs [0] + μs [1]
중

0x02 MUL 2 1 곱셈 연산. [0] ≡ μs [0] × μs [1]
중

0x03 하위 2 1 빼기 연산. [0] ≡ μs [0] - μs [1]
중

0x04 DIV 2 1 정수 나누기 연산. μs [1] = 0 μs [0] ÷ μs [1]
중 [0] ≡ 0 인 경우 그렇지 않은 경
우

0x05 SDIV 2 1 부호 있는 정수 나누기 연산(잘림). μs [1] = 0 인 경우 μs [0] = 2 인 경우
0
중 [0] ≡ -2 255 255 ∧ μs [1] = -1
sgn(μs [0] ÷ μs [1])|μs [0] ÷ μs [1]| 그렇지 않으면 모든 값이
2의 보수 부호 있는 256비트 정수로 처리됩니다.
2일 때 오버플로 시맨틱에 유의하십시오. 255 부정된다.

0x06 MOD 2 1 모듈로 나머지 연산.
중 [0] ≡ 0 if μs [1] = 0 μs [0]
mod μs [1] 그렇지 않은 경우

0x07 SMOD 2 1 부호 있는 모듈로 나머지 연산.
중 [0] ≡ 0 if μs [1] = 0
sgn(μs [0])(|μs [0]| mod |μs [1]|) 그렇지 않은 경우
여기서 모든 값은 2의 보수 부호 있는 256비트 정수로 처리됩니다.

0x08 애드모드 3 1 모듈로 추가 연산.
중 [0] ≡ 0 μs [2] = 0 인 경
우 (μs [0] + μs [1]) mod μs [2] 그렇지 않은 경우
이 작업의 모든 중간 계산은 2256 의 적용을 받지 않습니다.
기준 치수.

0x09 물모드 3 1 모듈로 곱셈 연산.
중 [0] ≡ 0 μs [2] = 0 (μs
[0] × μs [1]) 인 경우 mod μs [2] 그렇지 않은 경우
이 작업의 모든 중간 계산은 2256 의 적용을 받지 않습니다.
기준 치수.

0x0a 경험치 2 1 지수 연산. [1] [0] ≡ μs [0] μs
중

0x0b SIGNEXTEND 2 1 2의 보수 부호 있는 정수의 길이를 확장합니다. μs [1]t i인 경우 μs [1]i 그렇지 않은 경우
t 여기서 t = 256 8(μs [0] + 1)
∀i ∈ [0..255] : μ [0]i ≡

μs [x]i는 μs [x] 의 i번째 비트(0부터 계산)를 제공합니다.

10초: 비교 및 비트 논리 연산

값	니모닉	d	설명
0x10	LT	2	1 보다 작음 비교. μs [0] < μs [1] 인 경우 1 중 [0] ≡ 그렇지 않으면 0
0x11	GT	2	1 보다 큼 비교. 중 [0] ≡ μs [0] > μs [1] 인 경우 1 그렇지 않은 경우 0
0x12	일반 대중교통	2	1 부호 있는 보다 작음 비교. μs [0] < μs [1] 인 경우 중 [0] ≡ 1 그렇지 않으면 0 여기서 모든 값은 2의 보수 부호 있는 256비트 정수로 처리됩니다.
0x13	SGT	2	1 부호가 보다 큼 비교. 중 [0] ≡ μs [0] > μs [1] 인 경우 1 그렇지 않은 경우 0 여기서 모든 값은 2의 보수 부호 있는 256비트 정수로 처리됩니다.
0x14	EQ	2	1 평등 비교. μs [0] = μs [1] 인 경우 1 그 중 [0] ≡ 렇지 않은 경우 0
0x15	ISZERO	1	1 단순한 연산자가 아닙니다. μs [0] 중 [0] ≡ = 0 인 경우 1 그렇지 않으면 0
0x16	및	2	1 비트 AND 연산. ∀i ∈ [0..255] : μ [0]i ≡ μs [0]i ∧ μs [1]i
0x17	또는	2	1 비트 OR 연산. ∀i ∈ [0..255] : μ [0]i ≡ μs [0]i ∨ μs [1]i
0x18	XOR	2	1 비트별 XOR 연산. ∀i ∈ [0..255] : μ [0]i ≡ μs [0]i ⊕ μs [1]i 1 비
0x19	아님	1	트 NOT 연산. μs [0]i = 0 인 경우 1 ∀i ∈ [0..255] : μ [0]i ≡ 그렇지 않으면 0
0x1a	바이트	2	1 워드에서 단일 바이트를 검색합니다. ∀i ∈ [0..255] : μ [0]i ≡ μs [1][i 248+8μs [0]) i ≥ 248 ∧ μs [0] < 32 인 경우 0 그렇지 않으면 N번째 바이트의 경우 왼쪽부터 계산합니다(예: N=0이 빅 엔디안에서 가장 중요함).
0x1b	SHL	2	1 좌측 시프트 연산. 중 [0] ≡ (μs [1] × 2μs [0]) mod 2256
0x1c	SHR	2	1 논리적 우측 시프트 연산. [0] ≡ μs [1] ÷ 2 μs [0] 중
0x1d	SAR	2	1 산술(부호 있는) 오른쪽 시프트 연산. [0] ≡ μs [1] ÷ 2 μs [0] μ 여기서 μ 둥 [0] 및 μs [1]은 2의 보수 부호 있는 256비트 정수로 취급됩니다. 안 μs [0]은 무부호로 처리됩니다.

20대: KECCAK256

값	니모닉	d	설명
0x20	KECCAK256	2	1 Keccak-256 해시를 계산합니다. [0] ≡ 중 KEC(μm[μs [0]... (μs [0] + μs [1] 1)]) ≡ M(μi , μs [0], 나는 μs [1])

30대: 환경 정보

값 니모닉	d 설명
0x30 주소	0 1 현재 실행 중인 계정의 주소를 얻습니다. [0] ≡ Ia 1 주어진 계정의 잔액을 중. 가져옵니
0x31 밸런스	1 다. σ[μs [0] mod 2160]b if σ[μs [0] mod 2160] = ∅ 중. [0] ≡ 0 그렇지 않으면 ᛏ ≡ Aa ∪ {μs [0] 모드 2160}
0x32 오리진	0 1 실행 시작 주소를 얻습니다. [0] ≡ 이오 μ . 이것은 원래 트랜잭션의 발신자입니다. 비어 있지 않은 연결된 코드가 있는 계정이 아닙니다.
0x33 발신자	0 1 발신자 주소를 얻습니다. [0] ≡ 이다 μ . 이것은 이 실행을 직접 담당하는 계정의 주소입니다.
0x34 콜밸류	0 1 이 실행을 담당하는 명령/트랜잭션에 의해 예치된 값을 가져옵니다. 중. [0] ≡ IV
0x35 콜데이터로드 1	1 현재 환경의 입력 데이터를 가져옵니다. [0] ≡ Id[μs [0] . . .(μs μ . [0] + 31)] Id[x] = 0인 경우 x Id 이것은 메시지 호출 명령 또는 트랜잭션과 함께 전달되는 입력 데이터와 관련됩니다.
0x36 CALLDATASIZE 0 1 현재 입력 데이터의 크기 가져오기 환경.	μ . [0] ≡ 아이디 이것은 메시지 호출 명령 또는 트랜잭션과 함께 전달되는 입력 데이터와 관련됩니다.
0x37 CALLDATACOPY 3 0 현재 환경의 입력 데이터를 메모리에 복사합니다.	∀i∈{0 . . . μs [2] 1}: μm [μs [0] + i] ≡ Id[μs [1] + i] μs [1] + i < Id 인 경우 0 그렇지 않으면 μs [1] + i 단위의 추가는 2256 모듈 로가 적용되지 않습니다 . 것 ≡ M(μi , μs [0], μs [2]) μ i 이 은 메시지 호출 명령 또는 트랜잭션과 함께 전달되는 입력 데이터와 관련됩니다.
0x38 코드 크기	0 1 현재 환경에서 실행 중인 코드의 크기를 가져옵니다. [0] ≡ Ib 중.
0x39 코드복사	3 0 현재 환경에서 실행 중인 코드를 메모리에 복사합니다. μs [1] + i < Ib ∀i∈{0 . . . μs [2] 1}: μm [μs [0] + i] ≡ Ib[μs [1] + i] 인 경우 그렇지 않은 경우 멈추다 μs ≡ M(μi , μs [0], μs [2]) μ i [1] + i 의 추가는 2256 모듈 로의 영향을 받지 않습니다 .
0x3a 가스프라이스	0 1 현재 환경에서 가스 가격을 가져옵니다. [0] ≡ IP 중. 원래 거래에서 지정한 가스 가격입니다.
0x3b 외부 코드 크기 1	1 계정 코드의 크기를 가져옵니다. 중. [0] ≡ b if σ[μs [0] 모드 2160] = ∅ 0 그렇지 않으면 여기서 KEC(b) ≡ σ[μs [0] mod 2160]c ᛏ ≡ Aa ∪ {μs [0] 모드 2160}
0x3c EXTCODECOPY 4 0 계정 코드를 메모리에 복사합니다.	∀i∈{0 . . . μs [3] 1}: μm [μs [1] + i] ≡ b[μs [2] + i] μs [2] + i < b 인 경우 그렇지 않은 경우 멈추다 여기서 KEC(b) ≡ σ[μs [0] mod 2160]c σ[μs [0] mod 2160] = ∅ 인 경우 b ≡ ()라고 가정합니다 . 나는 ≡ M(μi , μs [1], μs [3]) μs [2] + i 단위의 추가는 2256 모듈 로의 적용을 받지 않습니다 . ≡ Aa ∪ {μs [0] 모드 2160} ᛏ

0x3d RETURNDATASIZE 0 1 현재 환경에서 이전 호출의 출력 데이터 크기를 가져옵니다.

중ₒ [0] 오

0x3e RETURNDATACOPY 3 0 이전 호출의 출력 데이터를 메모리에 복사합니다. μo [μs [1] + i]

∀i∈{0 . . . μs [2] 1}: μm [μs [0] + i] ≡ 0 μs [1] + i < μo 인 경우 그렇지 않으면

μs [1] + i 단위의 추가는 2256 모듈로가 적용되지 않습니다 .

나는 _ ≡ M(μi , μs [0], μs [2])

0x3f 외부코드해시 1 1 계정 코드의 해시를 가져옵니다. 죽은 경우(σ, μs [0] mod 2160)

중ₒ [0] ≡ 0 σ[μs [0] mod 2160]c 그렇지 않은 경

ㅏ ᵢ 우 ≡ Aa ∪ {μs [0] mod 2160}

40대: 블록 정보

값 니모닉 d 설명

0x40 블록해시 1 1 가장 최근에 완료된 256개 블록 중 하나의 해시를 가져옵니다. [0] ≡ P(IHp , μs [0], 0) 여기

중ₒ 서 P는 최대 연령까지 특정 숫
자의 블록 해시입니다. 찾는 블록 번호가 현재 블록 번호보다 크거나 같거나 현재 블록 뒤에 256개 이상의 블록이
있으면 스택에 0이 남습니다.

P(h, n, a) ≡ 0 n > Hi ∨ a = 256 ∨ h = 0 인 경우
사만 n = 안녕 인 경우
그렇지 않으면 P(Hp, n, a + 1)

h가 0이 아닌 한 해시 h에서 헤더 H를 결정할 수 있다고 주장합니다.
(제네시스 블록의 상위 해시의 경우와 동일).

0x41 코인베이스 0 1 현재 블록의 수신자 주소를 얻습니다. [0] ≡ IHC

중ₒ

0x42 TIMESTAMP 0 1 현재 블록의 타임스탬프를 가져옵니다.

중ₒ [0] ≡ IH

0x43 숫자 0 1 현재 블록 번호를 가져옵니다.

중ₒ [0] ≡ 아이하이

0x44 DIFFICULTY 0 1 현재 블록의 난이도를 가져옵니다.

중ₒ [0] ≡ IHD

0x45 가스 제한 0 1 현재 블록의 가스 한도를 가져옵니다. [0] ≡ IH1

중ₒ

0x46 체인 ID 0 1 체인 ID를 가져옵니다. [0] ≡ β

중ₒ

0x47 SELFBALANCE 0 1 현재 실행 중인 계정의 잔액을 가져옵니다. [0] ≡ σ[Ia]b

중ₒ

	50대: 스택, 메모리, 스토리지 및 흐름 작업	
값 니모닉 d 설명		
0x50 팝	1 0 스택에서 항목을 제거합니다.	
0x51 로드	1	1 메모리에서 단어를 로드합니다. [0] ≡ 중 μm[μs [0] . . .(μs [0] + 31)] ≡ max(μi , (μs [0] + 32) ÷ 32)는 2256 모듈 로가 적용 μ i 의 계산에 추가 되지 않습니다 .
0x52 MSTORE 2 0 워드를 메모리에 저장합니다. μ m[μs [0] . . .(μs [0] +		31)] ≡ μs [1] ≡ max(μi , (μs [0] + 32) ÷ 다 . 32) μ i 는 2256 모듈 로가 적용되지 않습니 μ i 의 계산에 추가
0x53 MSTORE8 2 0 바이트를 메모리에 저장합니다. μ m[μs [0]] ≡ (μs		[1] mod 256) ≡ max(μi , (μs [0] + 1) 않 ÷ 32) μ i 는 2256 모듈러스 가 적용되지 μ 의 계산에 추가
0x54 슬로드	1	1 저장소에서 단어를 로드합니다. 중 [0] ≡ σ[Ia]s[μs [0]] K ≡ AK ∪ {(Ia, μs [0])} CSLOAD(μ, A, I) ≡ Gwarmaccess if (Ia, μs [0]) ∈ AK 지콜드로드 그렇지 않으면
0x55 S스토어	2 0 단어를 저장소에 저장합니다. σ	[Ia]s[μs [0]] ≡ μs [1] K ≡ AK ∪ {(Ia, μs [0])} CSSTORE(σ, μ) 및 A 우리는 다음과 같이 EIP-2200에서 지정합니다. 체크포인트("원래") 상태 σ0이 현재 트랜잭션을 되돌릴 경우의 상태임을 독자에게 상기시킵니다 . v0 = σ0[Ia]s[μs [0]]를 스토리지 슬롯의 원래 값으로 둡니다 . v = σ[Ia]s[μs [0]]를 현재 값으로 둡니다. v = μs [1]을 새 값으로 설정합니다. 그 다음에: CSSTORE(σ, μ, A, I) ≡ 0 if (Ia, μs [0]) ∈ AK 그렇지 않으면 Gcoldsload v = v ∨ v0 = v 인 경우 Gwarmaccess + G세트 v = v ∧ v0 = v ∧ v0 = 0 인 경우 GS 리셋 v = v ∧ v0 = v ∧ v0 = 0 인 경우 알스클리어 v = v ∧ v0 = v ∧ v = 0 인 경우 ≡ + 와 함께 v = v ∧ v0 = v 인 경우 rdirtyclear + rdirtyreset 그렇지 않은 경우 0 어디 Rsclear if v0 = 0 ∧ v = 0 더티클리어 ≡ 알스클리어 v0 = 0 ∧ v = 0 인 경우 0 그렇지 않으면 v0 = v ∧ v0 = 0 인 경우 Gsset - Gwarmaccess rdirtyreset ≡ Gsreset - v0 = v ∧ v0 = 0 인 경우 Gwarmaccess 그렇지 않은 경우 0
0x56 점프	1 0 프로그램 카운터를 변경합니다.	JJUMP(μ) ≡ μs [0] 이것은 μpc에 해당 값을 쓰는 효과가 있습니다 . 섹션 9를 참조하십시오.
0x57 점프	2 0 프로그램 카운터를 조건부로 변경합니다. μs [0] if μs [1] = 0 μpc	J점프(μ) ≡ + 1 그렇지 않으면 해당 값을 μpc 에 쓰는 효과가 있습니다 . 섹 션 9를 참조하십시오.
0x58 PC	0 1 이 명령어에 해당하는 증분 이전의 프로그램 카운터 값을 가져옵니다.	중 [0] ≡ μpc

0x59 도와주세요	0 1 활성 메모리의 크기를 바이트 단위로 가져옵니다. [0] ≡ 32μi 중			
0x5a 가스	0 1 해당 감소를 포함하여 사용 가능한 가스 양을 가져옵니다. 이 교육의 비용. 중 [0] ≡ μg			
0x5b JUMPDEST 0 0 유효한 점프 목적지를 표시합니다. 이 작업은 실행 중 컴퓨터 상태에 영향을 주지 않습니다.				

60년대 및 70년대: 푸시 작업

값	니모닉	d	설명
0x60	푸시1	0 1	스택에 1바이트 항목을 놓습니다. [0] ≡ c(μpc + 1) 중 여기서 c(x) ≡ 일[x] if x < 일 / 0 그렇지 않으면 바이트는 프로그램 코드의 바이트 배열에서 라인으로 읽혀집니다. 함수 c는 바이트가 한계를 넘어 확장되는 경우 기본 바이트가 0이 되도록 합니다. 바이트는 오른쪽으로 정렬됩니다(빅 엔디안에서 최하위 유효 위치를 차지함).
0x61	푸시2	0 1	2바이트 항목을 스택에 놓습니다. [0] ≡ c (μpc + 중 1). . .(μpc + 2) c(x) ≡ (c(x0), ..., c(xx 1)) 와 함께 c는 위에서 정의한 대로입니다. 바이트는 오른쪽으로 정렬됩니다(빅 엔디안에서 최하위 중요 위치를 차지함).
⋮	⋮	⋮ ⋮	⋮
0x7f	푸시32	0 1	32바이트(전체 단어) 항목을 스택에 놓습니다. [0] ≡ c (μpc + 1). . .(μpc μ + 32) 여기서 c는 위와 같이 정의됩니다. 바이트는 오른쪽으로 정렬됩니다(빅 엔디안에서 최하위 중요 위치를 차지함).

80년대: 복제 작업

값	니모닉	d	설명
0x80	DUP1	1 2	첫 번째 스택 항목을 복제합니다. [0] ≡ μs [0] 중
0x81	DUP2	2 3	두 번째 스택 항목을 복제합니다. 중 [0] ≡ μs [1]
⋮	⋮	⋮ ⋮	⋮
0x8f	DUP16	16 17	중복된 16번째 스택 항목입니다. [0] ≡ μs [15] 중

90년대: Exchange 운영

값	니모닉	d	설명
0x90	스왑1	2 2	1차와 2차 스택 아이템을 교환합니다. [0] ≡ μs [1] [1] ≡ μs 중 [0] 중
0x91	스왑2	3 3	1차와 3차 스택 아이템을 교환합니다. [0] ≡ μs [2] [2] ≡ μs [0] 중 중
⋮	⋮	⋮ ⋮	⋮
0x9f	스왑16	17 17	1차와 17차 스택 아이템을 교환합니다. [0] ≡ μs [16] [16] ≡ μs 중 [0] 중

a0s: 로깅 작업

모든 로깅 작업의 경우 상태 변경은 하위 상태의 로그 시리즈에 추가 로그 항목을 추가하는 것입니다.
그렉 ≡ Al · (la, t, µm[µs [0] . . (µs [0] + µs [1] 1)]) 및 메모
리 소비 카운터 업데이트: ≡ M(µi , µs [0], µs [1]) µ i 항목의 주제
다. 시리즈 t는 그에 따라 다릅니

값	니모닉	d	설명
0xa0	로그	0	2 0 주제가 없는 로그 레코드를 추가합니다. 티 ≡ ()
0xa1	로그1		3 0 하나의 주제로 로그 레코드를 추가합니다. t ≡ (µs [2])
⋮	⋮	⋮ ⋮	⋮
0x4	로그4		6 0 4개의 주제로 로그 레코드를 추가합니다. t ≡ (µs [2], µs [3], µs [4], µs [5])

f0s: 시스템 작업

값 니모닉 d 설명

0xf0 만들기 3 1 연결된 코드로 새 계정을 만듭니다. i ≡ μm[μs [1] . . .(μs [1] + μs [2] - 1)] ζ ≡ ∅

$$(s', g', A, z, o) \equiv \begin{cases} \Lambda(\sigma^*, A, I_a, I_o, L(\mu_g'), I_p, \mu_s[0], i, I_e + 1, \zeta, I_w) & \text{if } \mu_s[0] \sigma[I_a]_b \wedge I_e < 1024 \\ (\sigma, L(\mu_g'), A, 0, ()) & \text{그렇지 않으면} \end{cases}$$

제외 σ ≡ μg L(μg) [Ia]n = σ[Ia]n + 1 ≡ σ

지 _ + g [0] ≡ x

μ

여기서 z = 0인 경우 x = 0, 즉 계약 생성 프로세스가 실패하거나 Ie = 1024(최대 호출 깊이 제한에 도달함) 또는 μs [0] > σ[Ia]b (호출자의 잔액이 너무 낮음) 가치 이전을 이행하기 위해) 그렇지 않으면 x = ADDR(Ia, σ[Ia]n, ζ, i), 새로 생성된 계정의 주소(86). ≡ M(μi , μs [1], μs [2]) μ i () if z = 1 o 그렇지 않으면 따라서 피연산자 순서는 값, 입력 오프셋, 입력 크기입니다.

중 ≡

0xf1 호출 7 1 계정으로 메시지 호출. i ≡ μm[μs [3] . . .(μs [3] + μs [4] - 1)] Θ(σ, A , Ia, Io, t, t, CCALLGAS(σ, μ, A), Ip, μs [2], μs [2] , i, Ie + 1, Iw) (σ, 만약 μs [2] σ[Ia]b ∧ 즉 <1024

$$(s', g', \dagger, x, o) \equiv \begin{cases} \text{CCALLGAS}(\sigma, \mu, A), A, 0, ()) & n \equiv \min(\{\mu_s[6], o\}) \\ \mu\, m[\mu_s[5] \ldots (\mu_s[5] + n - 1)] = o[0 \ldots (n-1)] & \text{그렇지 않으면} \end{cases}$$

중 =

지 _ ≡ μg CCALLGAS(σ, μ, A) + g [0] ≡

중 x ≡ A 제

† 외 A † ≡ 아 ∪ {t}

t ≡ μs [1] 모드 2160

μ ≡ M(M(μi , μs [3], μs [4]), μs [5], μs [6])

i 여기서 이 작업에 대한 코드 실행이 실패하거나 μs [2] > σ[Ia]b (자금 부족) 또는 Ie = 1024(호출 깊이 제한 도달)인 경우 x = 0입니다. 그렇지 않으면 x = 1입니다.

따라서 피연산자 순서는 gas, to, value, in offset, in size, out offset, out size입니다.

$$\text{CCALL}(\sigma, \mu, A) \equiv \text{CGASCAP}(\sigma, \mu, A) + \text{CEXTRA}(\sigma, \mu, A)$$

$$\text{CCALLGAS}(\sigma, \mu, A) \equiv \begin{cases} \text{CGASCAP}(\sigma, \mu, A) + G_{callstipend} & \text{if } \mu_s[2] = 0 \\ \text{CGASCAP}(\sigma, \mu, A) & \text{그렇지 않으면} \end{cases}$$

$$\text{CGASCAP}(\sigma, \mu, A) \equiv \min\{L(\mu_g - \text{CEXTRA}(\sigma, \mu, A)), \mu_s[0]\} \text{ if } \mu_g \geq \text{CEXTRA}(\sigma, \mu, A)\ \mu_s[0] \text{ 그렇지 않은 경우}$$

$$\text{CEXTRA}(\sigma, \mu, A) \equiv C_{aaccess}(t, A) + \text{CXFER}(\mu) + \text{CNEW}(\sigma, \mu)$$

$$\text{CXFER}(\mu) \equiv \begin{cases} \mu_s[2] = 0 \text{ 이면 } G_{callvalue} & \text{그렇지 않으면} \\ 0 & \end{cases}$$

$$\text{CNEW}(\sigma, \mu) \equiv \begin{cases} G_{newaccount} & \text{if DEAD}(\sigma, t) \wedge \mu_s[2] = 0\ 0 \\ & \text{그렇지 않으면} \end{cases}$$

0xf2 CALLCODE 7 1 대체 계정 코드를 사용하여 이 계정으로 메시지 호출합니다.

다음을 제외하고 CALL 과 정확히 동일 : Θ(σ, A , Ia, Io, Ia, t, CCALLGAS(σ, μ, A), Ip, μs [2], μs [2], i, Ie + 1, Iw) (σ, CCALLGAS(σ, μ, A), A, 0,()) 만약 μs [2] σ[Ia]b ∧ 즉 <1024 그렇지 않으면

$$(s', g', \dagger, x, o) \equiv$$

두 번째 스택 값 μs [1](CALL에서와 같이) 에서 현재 주소 Ia로의 호출 Θ에 대한 네 번째 매개 변수의 변경 사항에 유의하십시오. 이는 수신자가 실제로 현재와 동일한 계정임을 의미하며 단순히 코드를 덮어쓴다는 것입니다.

0xf3 RETURN 2 0 출력 데이터를 반환하는 실행을 중지합니다.

HRETURN(μ) ≡ μm[μs [0] . . .(μs [0] + μs [1] - 1)]

이렇게 하면 출력이 정의된 이 시점에서 실행을 중지하는 효과가 있습니다.

섹션 9 참조. ≡

M(μi , μs [0], μs [1]) μ i

0xf4 DELEGATECALL 6 1	대체 계정의 코드를 사용하여 이 계정으로 메시지 호출하지만 송신자 및 값에 대한 현재 값을 유지합니다. CALL 과 비교하여 DELEGATECALL은 인수가 하나 더 적습니다. 생략된 인수는 μs [2]입니다. 결과적으로 CALL 의 정의에서 μs [3], μs [4], μs [5] 및 μs [6] 은 각각 μs [2], μs [3], μs [4] 및 μs [5]. 그렇지 않으면 다음 을 제외 하고 는 CALL 과 동일 합니다 . _ _ _ _ _ _ _ _ _ , CCALLGAS(σ, μ, A), A, 0,() (s , g , A, x, o) ≡ Ie < 1024 인 경우 / 그렇지 않으면 호출 Θ에 대한 두 번째 및 아홉 번째 매개변수의 변경 사항(네 번째 매개변수에 추가)에 유의하십시오. 이것은 받는 사람이 실제로 현재와 동일한 계정이라는 것을 의미합니다. 단순히 코드를 덮어쓰고 컨텍스트가 거의 완전히 동일하다는 것입니다.
0xf5 생성2	4 1 연결된 코드로 새 계정을 만듭니다. 소금 ζ ≡ μs [3]를 제외하면 CREATE 와 정확히 동일합니다 .
0xfa 정적 호출	6 1 계정으로 정적 메시지 호출. 다음을 제외하고는 CALL 과 완전히 동일합니다 . 인수 μs [2]가 0으로 대체됩니다. 더 깊은 인수 μs [3], μs [4], μs [5] 및 μs [6]은 각각 μs [2], μs [3], μs [4] 및 μs [5] 로 대체됩니다 . Θ의 마지막 인수는 ⊥입니다.
0xfd 되돌리기	2 0 상태 변경을 되돌리면서 실행을 중지하지만 데이터와 남은 가스를 반환합니다. HRETURN(μ) ≡ μm[μs [0] . . .(μs [0] + μs [1] - 1)] 이 작업의 효과는 (143)에 설명되어 있습니다. 가스 계산을 위해 메모리 확장 기능을 사용합니다. 나는 ≡ M(μi , μs [0], μs [1])
0xfe 유효하지 않음	∅ ∅ 무효 명령 지정.
0xff SELFDESTRUCT 1 0	실행을 중지하고 나중에 삭제할 수 있도록 계정을 등록합니다. ≡ As ∪ {Ia} ≡ Aa ∪ {r} σ [r] ≡ ∅ (σ[r]n, σ[r]b + σ[Ia]b, σ[r]s, σ[r]c) if r = Ia (σ[r]n, 0, σ[r] s, σ[r]c) 그렇지 않으면 — σ[r] = ∅ ∧ σ[Ia]b = 0 인 경우 여기서 r = μs [0] mod 2160 σ [Ia]b = 0 CSELFDESTRUCT(σ, μ) ≡ Gselfdestruct + 0 (만약 r ∈ Aa 그렇지 않으면 Gcoldaccountaccess) + Gnewaccount if DEAD(σ, r) ∧ σ[Ia]b = 0 그렇지 않으면 0

부록 I. 창세기 블록

제네시스 블록은 15개 항목이며 다음과 같이 지정됩니다.

(331) 0256, KEC RLP(), 0160, stateRoot, 0, 0, 02048, 2^{34}, 0, 0, 3141592, 시간, 0, 0256, KEC(42),(),()

여기서 0256은 모두 0인 256비트 해시인 상위 해시를 나타냅니다. 0160은 모두 0인 160비트 해시인 수혜자 주소를 나타냅니다. 02048은 모두 0인 2048비트인 로그 블룸을 나타냅니다. 234는 난이도를 나타냅니다. 트랜잭션 트리 루트, 영수증 트리 루트, 사용된 가스, 블록 번호 및 추가 데이터는 모두 0이며 빈 바이트 배열과 동일합니다. ommer와 transaction의 시퀀스 는 모두 비어 있고 ()로 표시됩니다. KEC (42) 는 길이가 1인 바이트 배열의 Keccak-256 해시를 참조하며 첫 번째이자 유일한 바이트의 값은 42이며 난스에 사용됩니다. KEC RLP () 값은 RLP에서 ommer 목록의 해시를 나타내며 둘 다 빈 목록입니다.

개념 증명 시리즈에는 개발 사전 채굴이 포함되어 상태 루트 해시를 일부 값 stateRoot로 만듭니다. 또한 시간은 제네시스 블록의 초기 타임스탬프로 설정됩니다. 해당 값에 대해서는 최신 설명서를 참조해야 합니다 .

부록 J. Ethash

J.1. 정의. 우리는 다음 정의를 사용합니다.

이름	값	설명
Jwordbytes	2^{4}	단어의 바이트.
Jdatasetinit	2^{30}	제네시스에서 데이터세트의 바이트입니다.
Jdataset성장	2^{23}	에포크당 데이터 세트 성장.
Jcacheinit	2^{24}	기원 시 캐시의 바이트.
Jcache성장	2^{17}	에포크당 캐시 성장.
네	에포크당 30000 블록.	
jmixbytes	128	바이트의 혼합 길이.
Jhashbytes	64	바이트 단위의 해시 길이.
제이패런트	256	각 데이터 세트 요소의 부모 수입니다.
Jcacherounds	3	캐시 생산의 라운드 수.
J액세스	64	하시모토 루프의 액세스 수입니다.

J.2. 데이터 세트 및 캐시의 크기. Ethash의 캐시 $c \in B$ 및 데이터 세트 $d \in B$ 의 크기는 epoch에 따라 다릅니다.
회전은 블록 번호에 따라 다릅니다.

(332) $$\text{에포크}(\text{하이}) = \frac{\text{안녕}}{\text{네}}$$

Jdatasetgrowth 바이트 에 의한 데이터 세트 크기 및 Jcachegrowth 바이트 에 의한 캐시 크기 , 모든 에포크. ~ 안에
주기적 동작으로 이어지는 규칙성을 피하기 위해 크기는 소수여야 합니다. 따라서 크기는 a만큼 줄어듭니다.
데이터 세트의 경우 여러 Jmixbytes , 캐시의 경우 Jhashbytes입니다 . dsize = d를 데이터세트의 크기로 둡니다 . 어느
를 사용하여 계산

(333) $$\text{dsize} = \text{Eprime}(\text{Jdatasetinit} + \text{Jdatasetgrowth} \cdot \text{Eepoch} - \text{Jmixbytes}, \text{Jmixbytes})$$

캐시 크기 csize는 다음을 사용하여 계산됩니다.

(334) $$\text{csize} = \text{Eprime}(\text{Jcacheinit} + \text{Jcachegrowth} \cdot \text{Epoch} \quad \text{Jhashbytes}, \text{Jhashbytes})$$

(335) $$\text{인쇄}(x, y) = \begin{cases} \text{엑스} & x/y \in \text{N인 경우} \\ \text{그렇지 않으면 Eprime}(x \quad 2 \cdot y, y) \end{cases}$$

J.3. 데이터 세트 생성. 데이터 세트를 생성하려면 바이트 배열인 캐시 c가 필요합니다. 때에 따라 다르지
캐시 크기 csize 및 시드 해시 $s \in B32$에서.

J.3.1. 시드 해시. 시드 해시는 시대마다 다릅니다. 첫 번째 에포크의 경우 32개 시리즈의 Keccak-256 해시입니다.
0의 바이트. 다른 모든 시대에는 항상 이전 시드 해시의 Keccak-256 해시입니다.

(336) $$s = \text{Cseedhash}(\text{하이})$$

(337) $$\text{Cseedhash}(\text{하이}) = \begin{cases} 032 & \text{에포크}(\text{Hi}) = 0 \text{ 인 경우} \\ \text{KEC}(\text{Cseedhash}(\text{Hi} - \text{Jepoch})) & \text{그렇지 않은 경우} \end{cases}$$

032 는 32바이트의 0입니다.

J.3.2. 은닉처. 캐시 생성 프로세스는 시드 해시를 사용하여 먼저 csize 바이트를 순차적으로 채우는 작업을 포함합니다.
그런 다음 Lerner[2014]가 만든 RandMemoHash 알고리즘의 Jcacherounds 패스를 수행합니다. 초기 캐시
씨 단일 바이트 배열의 배열인 은 다음과 같이 구성됩니다.
64개의 단일 바이트로 구성된 배열 ci를 초기 캐시의 i번째 요소로 정의합니다.

(338) $$\text{거기} = \begin{cases} \text{KEC512}(\text{들}) & \text{내가} = 0\text{인 경우} \\ \text{그렇지 않으면 KEC512}(\text{ci-1}) \end{cases}$$

따라서 c는 다음과 같이 정의할 수 있습니다.

(339) $$c[i] = ci \ \forall \ i < n$$

(340) $$\text{엔} = \frac{\text{크기}}{\text{Jhashbytes}}$$

캐시는 초기 캐시 c에 대해 RandMemoHash 알고리즘의 Jcacherounds 라운드를 수행하여 계산됩니다.

(341) $$c = \text{Ecacherounds}(c, \text{Jcacherounds})$$

(342) $$\text{Ecacherounds}(x, y) = \begin{cases} \text{엑스} & y = 0\text{인 경우} \\ \text{ERMH}(x) & y = 1\text{인 경우} \\ \text{그렇지 않으면 Ecacherounds}(\text{ERMH}(x), y \quad 1) \end{cases}$$

여기서 단일 라운드는 다음과 같이 캐시의 각 하위 집합을 수정합니다.

(343) ERMH(x) = Ermh(x, 0), Ermh(x, 1), ..., Ermh(x, n - 1)

(344) Ermh(x, i) = KEC512(x [(i 1 + n) mod n] ⊕ x [x [i][0] mod n])

x = x 제외 x [j] = Ermh(x, j) ∀ j < i

J.3.3. 전체 데이터 세트 계산. 본질적으로 우리는 Jparents 의사 무작위로 선택한 캐시 노드 의 데이터와 해시를 결합합니다.
데이터 세트를 계산하는 것입니다. 그런 다음 전체 데이터 세트는 각 Jhashbytes 바이트 크기의 여러 항목으로 생성됩니다 .

(345) $$d[i] = Edatasetitem(c, i) \forall i < \frac{\text{크기}}{Jhashbytes}$$

단일 항목을 계산하기 위해 경우에 따라 FNV 해시(Glenn Fowler [1991])에서 영감을 받은 알고리즘을 사용합니다.
XOR에 대한 비연관 대체물.

(346) EFNV(x, y) = (x · (0x01000193 ⊕ y)) mod 232

이제 데이터 세트의 단일 항목을 다음과 같이 계산할 수 있습니다.

(347) Edatasetitem(c, i) = Eparents(c, i, 1, ∅)

(348) 부모(c, i, p, m) = Eparents(c, i, p + 1, Emix(m, c, i, p + 1)) if p < Jparents 2
그렇지 않으면 Emix(m, c, i, p + 1)

(349) 에믹스(m, c, i, p) = KEC512(c[i mod csize] ⊕ i) if p = 0
EFNV m, c[EFNV(i ⊕ p, m[p mod Jhashbytes/Jwordbytes]) mod csize] 그렇지 않은 경우

J.4. 작업 증명 기능. 본질적으로 우리는 "mix" Jmixbytes 바이트 폭을 유지하고 반복적으로 순차적으로 가져옵니다.
전체 데이터 세트에서 Jmixbytes 바이트를 가져오고 EFNV 기능을 사용하여 믹스와 결합합니다. 순차 Jmixbytes 바이트
액세스는 알고리즘의 각 라운드가 항상 RAM에서 전체 페이지를 가져오도록 사용되어 변환 색인을 최소화합니다.
ASIC이 이론적으로 피할 수 있는 버퍼 미스.

이 알고리즘의 출력이 원하는 목표보다 낮으면 nonce가 유효합니다. 추가 신청 참고
마지막에 KEC는 최소한 작은
작업량이 완료되었습니다. 이 빠른 외부 PoW 검증은 안티 DDoS 목적으로 사용될 수 있습니다. 제공하는 역할도 합니다
결과가 편향되지 않은 256비트 숫자라는 통계적 보증.

PoW 기능은 압축된 믹스가 첫 번째 항목이고 Keccak-256 해시가 있는 배열을 반환합니다.
압축된 믹스와 시드 해시를 두 번째 항목으로 연결:

(350) 작업 증명(Hn, Hn, d) = {mc(KEC(RLP(LH(Hn))), Hn, d), KEC(sh(KEC(RLP(LH(Hn))), Hn) + mc(KEC(RLP(LH(Hn))), Hn, d))}

Hn은 nonce가 없는 헤더의 해시입니다. 압축 믹스 mc 는 다음과 같이 얻습니다.

(351) $$mc(h, n, d) = Ecompress(Eaccesses(d, \sum_{i=0}^{nmix} sh(h, n), sh(h, n), -1), -4)$$

시드 해시는 다음과 같습니다.

(352) sh(h, n) = KEC512(h + Erevert(n))

Erevert(n)은 nonce n의 되돌린 바이트 시퀀스를 반환합니다.

(353) Erevert(n)[i] = n[n i]

두 바이트 시퀀스 사이의 "+" 연산자는 두 시퀀스의 연결을 초래합니다.
데이터 세트 d는 섹션 J.3.3에 설명된 대로 얻습니다.
믹스에서 복제된 시퀀스의 수는 다음과 같습니다.

(354) $$nmix = \frac{jmixbytes}{Jhashbytes}$$

임의의 데이터 세트 노드를 혼합에 추가하기 위해 Eaccesses 기능이 사용됩니다.

(355) E접근(d, m, s, i) = Emixdataset(d, m, s, i) if i = Jaccesses 2
그렇지 않으면 Eaccesses(Emixdataset(d, m, s, i), s, i + 1)

(356) Emixdataset(d, m, s, i) = EFNV(m, Enewdata(d, m, s, i))

Enewdata는 nmix 요소 가 있는 배열을 반환합니다 .

(357) $$Enewdata(d, m, s, i)[j] = d[EFNV(i \oplus s[0], m[i \bmod \frac{jmixbytes}{Jwordbytes}]) \text{ 맞서 } \frac{dsize/Jhashbytes}{nmix} \cdot nmix + j] \forall j < nmix$$

믹스는 다음과 같이 압축됩니다.

(358) 압축(m, i) = 중 im - 8인 경우
Ecompress(EFNV(EFNV(EFNV(m[i + 4], m[i + 5]), m[i + 6]), m[i + 7]), i + 8) 그렇지 않으면

부록 K. 메인 네트워크의 이상 현상

K.1. 가스 부족에도 불구하고 계정 삭제. 블록 2675119에서 트랜잭션 0xcf416c536ec1a19ed1fb89e

4ec7ffb3cf73aa413b3aa9b77d60e4fd81a4296ba, 주소 0x03의 계정이 호출되어 가스 부족이 발생했습니다.
호출. 등식(207)에 대해 이것은 터치된 주소 집합에 0x03을 추가했으며 이 트랜잭션은 σ[0x03] 으로 바뀌었습니다.
∅로.

부록 L. 수학 기호 목록

기호	Latex 명령	설명
	\bigvee	이것은 작동하는 모든 요소의 최소 상한, 상한 또는 조인입니다. 따라서 그것은 이러한 요소 중 가장 큰 요소입니다(Davey and Priestley [2002]).