차세대 스마트 컨트랙트와 탈중앙화된 어플리케이션 플랫폼

(A Next-Generation Smart Contract and Decentralized Application Platform)

사토시 나카모토가 2008[1a][1b]–2009년[1c][1d] 개발한 비트코인은 종종 화폐와 통화분야에서 매우 근본적인 혁신으로 묘사되어 왔는데, 이것은 비트코인이 어떤 담보나 내재적인 가치를[2] 가지지 않으며 중앙화된 발행기관이나 통제기관도 없는 디지털 자산의 첫번째 사례였기 때문이다. 하지만 비트코인 실험의 더욱 중요한 측면은 비트코인을 떠받치고 있는 분산합의수단으로서의 블록체인 기술이며, 이에 대한 관심이 급격하게 늘어나고 있다.

블록체인 기술을 이용한 대안적 어플리케이션들에는 다음과 같은 것들이 자주 거론되고 있다. 사용자 정의 화폐와 금융상품을 블록체인 위에 표현하는 컬러드 코인("colored coins"),[3] 물리적 대상의 소유권을 표현하는 스마트 자산("smart property"),[4] 도메인 이름과 같은 비동질적 자산을 기록하는 네임코인("Namecoin"),[5] 임의적인 계약규칙을 구현한 코드에 의해 다지털 자산을 관리하는 좀 더 복잡한 형태의 스마트 컨트랙트 ("smart contracts"),[6] 더 나아가 블록체인을 기반으로 한 탈중앙화된 자율 조직("decentralized autonomous organizations" , DAOs)[7] 등이다.

이더리움이 제공하려는 것은 완벽한 튜링완전(turing-complete) 프로그래밍 언어가 심어진 블록체인이다. 이 프로그래밍 언어는, 코딩된 규칙에 따라 '어떤 상태'를 다르게 변환시키는 기능(arbitrary state transition functions)이 포함된 "계약(contracts)"을 유저들이 작성할 수 있게 함으로써 앞서 설명한 시스템들을 구현 가능하게 할 뿐만 아니라 우리가 아직 상상하지 못한 다른 많은 어플리케이션도 매우 쉽게 만들 수 있도록 도와줄 것이다.

목차

비트코인과 기존 개념들에 대한 소개(Introduction to Bitcoin and Existing Concepts)

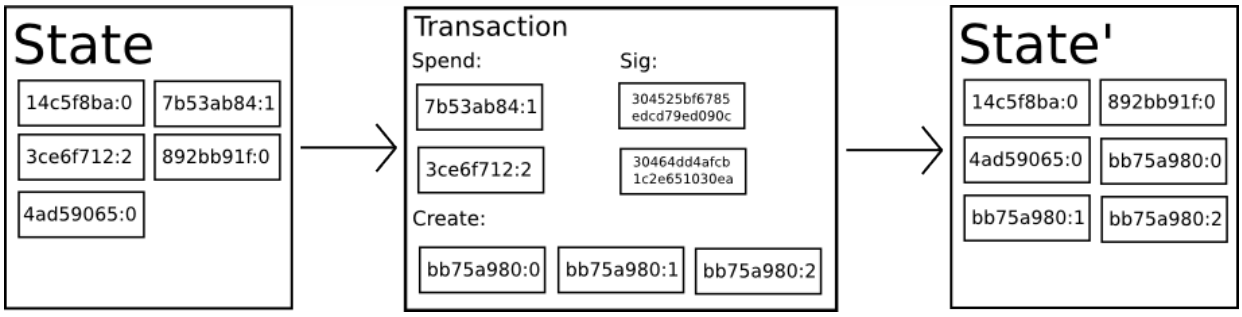
역사(History)

분산화된 디지털 통화의 개념은, 재산등록 같은 대안 어플리케이션과 마찬가지로 지난 수십 년간 우리 주변에 있었다. 1980~90년대 익명 e-cash 프로토콜은 주로 ‘Chaumian blinding’으로 알려진 ‘로우레벨 암호 알고리즘(cryptographic primitive)’에 기반하였고 개인정보를 강력하게 보호하는 화폐를 제공하였으나 중앙집권적인 중개인에 의존했기 때문에 별다른 주목을 받지 못했다. 1998년 'Wei Dai'의 b-money는 분산 합의와 계산 퍼즐을 풀게 하는 방식을 통해서 화폐를 발행하게 하는 아이디어를 최초로 제안하였지만 분산 합의를 실제로 어떻게 구현할지에 대한 자세한 방법은 제시하지 못했다. 2005년에 'Hall Finney'는 "재사용 가능한 작업증명(reusable proofs of work)" 개념을 소개하였다. 이 시스템은 b-money의 아이디어에 Adam Back의 ‘계산 난이도 해시캐시 퍼즐(computationally difficult Hashcash puzzles)’을 조합한 것이었다. 그러나 외부의 신뢰를 필요로 하는 컴퓨팅(trusted computing)을 그 기반에 둠으로써, 이상을 구현하는데에는 또 다시 실패했다. 2009년 사토시 나카모토에 의해 처음 실제적으로 구현된 탈중앙화된 화폐는 공개키 암호방식을 통한 소유권 관리를 위해 사용되던 기존의 알고리즘을 ‘작업 증명(proof of work)’이라고 알려진 합의 알고리즘과 결합함으로써 가능하게 되었다.

작업증명의 기반이 되는 작동방식은 매우 혁신적인 것이었는데, 이것은 두가지 문제를 동시에 해결하기 때문이다. 첫째, 이것은 간단하면서도 상당히 효과적인 합의 알고리즘을 제공해주었다. 즉, 네트워크 상에 있는 모든 노드들이 비트코인의 장부상태(state of the Bitcoin ledger)에 일어난 표준 업데이트의 집합(a set of canonical updates)에 공동으로 동의할 수 있도록 해주었다는 것이다. 둘째, 누구나 합의 프로세스에 참여할 수 있도록 허용해줌으로써 합의결정권에 대한 정치적 문제를 해결할 수 있을 뿐만 아니라 동시에 시빌공격(sybil attacks)도 방어해줄 수 있는 메커니즘을 제공했다. 이것은 합의 프로세스에 대한 참여의 조건으로 ‘특정한 리스트에 등록된 주체이어야만 한다’라는 어떤 형식적 장벽대신에, 경제적 장벽 - 각 노드의 결정권의 크기를 그 노드의 계산능력에 직접적으로 비례시키는 방식으로 대체하는 것이었다.

이후로, 지분증명(proof of stake)이라는 새로운 방식의 합의 알고리즘이 등장했는데, 이는 각 노드가 가진 계산능력이 아니라 화폐의 보유량에 따라 각 노드의 결정권 정도를 계산해야 한다는 것이다. 이 두 방식의 상대적인 장점들에 대한 논의는 이 백서에서는 다루지 않겠지만, 두 방법 모두 암호화화폐의 기반으로서 사용될 수 있다는 점은 지적해두고자 한다.

상태변환시스템으로서의 비트코인(Bitcoin As A State Transition System)



기술적인 관점에서 보았을 때, 비트코인과 같은 암호화 화폐의 장부는 하나의 상태변환시스템(state transition system)으로 생각해볼 수 있다. 이 시스템은, 현재 모든 비트코인의 소유권 현황으로 이루어진 하나의 “상태(state)” 와 이 현재 상태와 트랜잭션을 받아서 그 결과로써 새로운 상태를 출력해주는 “상태변환함수(state transition function)”로 구성되어 있다. 표준 은행 시스템에 비유하자면 상태는 모든 계좌잔고표(balance sheet)이고 트랜잭션은 A에서 B로 X를송금하라는요청이며, 상태변환함수에의해A의계좌에서는X가 감소하고 B의 계좌에서는 X가증가한다. 만약처음에A의계좌에있는금액이X 이하인 경우에는 상태변환함수가 에러를 리턴한다. 이러한 상태변환를 비트코인 장부에서는 다음과 같이 정의할 수 있다.

```
APPLY(S,TX) -> S' or ERROR
```

은행 시스템 예시에서는 다음과 같다.

```
APPLY({ Alice: $50, Bob: $50 },"send $20 from Alice to Bob") = { Alice:
$30, Bob: $70 }
```

```
APPLY({ Alice: $50, Bob: $50 },"send $70 from Alice to Bob") = ERROR
```

비트코인에서 "상태(state)"는 생성되었지만 아직 사용되지 않은 모든 코인들의 집합(기술적표현으로는 '소비되지 않은 트랜잭션 출력', UTXO(Unspent Transaction Outputs))이다. 각 UTXO들에는 각자의 코인금액이 표시되어 있고 이 UTXO의 소유자(20byte의 주소로 정의되는 암호화된 공개키(public key)[1])정보가 들어 있다. 트랜잭션은 하나 이상의 입력(inputs) 및 출력을 포함한다. 각 입력에는 보내는 쪽 지갑주소에서 선택된 기존 UTXO에 대한 참조정보와, 해당지갑주소에 대응되는 개인키(private key)가 생성한 암호화된 서명을 담고 있다. 그리고 각 출력들은 상태에 추가될 새로운 UTXO정보를 가지고 있다.

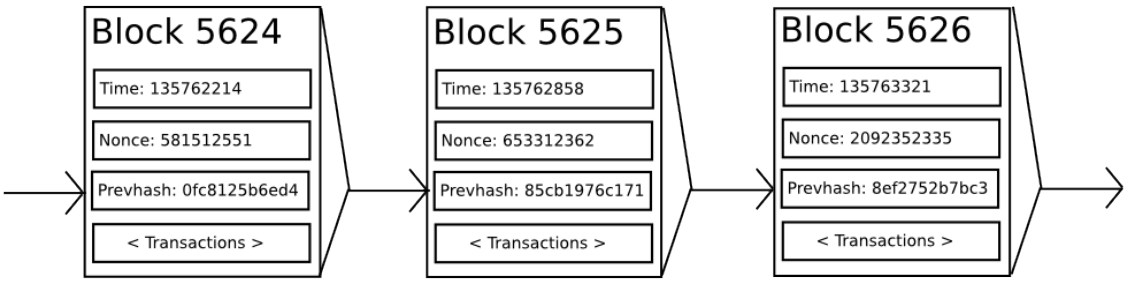
상태변환함수 `APPLY(S,TX) -> S'` 는 다음과 같이 정의할 수 있다.

1. TX의 각 입력에 대해 :
    - 만약 참조된 UTXO가 `S`에 없다면, 에러를 리턴.
    - 만약 서명이 UTXO의 소유자와 매치되지 않으면, 에러를 리턴.
2. 만약 입력에 사용된 UTXO들 금액의 합이 출력 UTXO들 금액의 합보다 작으면, 에러를 리턴.
3. 입력에 사용된 UTXO가 삭제되고 출력 UTXO가 추가된 `S`를 리턴.

여기서 1번의 첫번째 과정은 존재하지 않는 코인이 트랜잭션에 사용되는 것을 막기 위한 것이고 1번의 두번째 과정은 다른 사람의 코인이 트랜잭션에 사용되는 것을 막기 위한 것이다. 위 절차를 실제 비트코인 지불과정에 적용하면 다음과 같다. Alice가 Bob에게 11.7 BTC를 보내고 싶다고 가정하자. 먼저 Alice 지갑주소로부터 표시된 금액의 합이 적어도 11.7 BTC 이상인 UTXO의 집합을 찾는다. 실제 대부분의 경우에는 11.7 BTC를 정확히 바로 선택할 수 없다. Alice의 지갑주소에서 각각 6, 4, 2 BTC 가 표시된 3개의 UTXO를 참조할 수 있다고 하자. 이 3개의 UTXO가 트랜잭션의 input이 되고 2개의 output이 생성된다. Output 중 하나는 11.7 BTC가 표시된

새로운 UTXO이며 소유자는 Bob의 지갑주소가 된다. 그리고 다른 하나는 12(6+4+2) - 11.7 = 0.3 BTC의 "잔돈(change)"이 표시된 새로운 UTXO이며 소유자는 Alice 자신의 지갑주소가 된다.

채굴



만일 우리가 위에서 기술한 내용을 신뢰를 기반으로 하는 중앙집권화된 서비스 방식으로 구현하자면 매우 간단한 일이 될텐데, 왜냐하면 중앙 서버 하드드라이브에 상태변화의 과정을 저장만 하면 되기 때문이다. 그러나 비트코인에서는, 탈중앙화된 통화시스템을 구축하고자 하는 것이며, 이를 위해서는 모든 사람이 수긍할 수 있는 트랜잭션 순서 합의 시스템을 상태변화시스템과 결합해야만 한다. 비트코인의 분산 합의 과정은 네트워크에 "블록(blocks)"이라 불리는 트랜잭션 패키지를 계속적으로 생성하고자 시도하는 노드들을 필요로 한다. 이 네트워크는 약 10분마다 하나의 블록을 생성하도록 계획되어 있고 각 블록은 타임스탬프, 논스(nonce), 이전 블록에 대한 참조(이전 블록의 해시), 그리고 이전 블록 이후에 발생한 모든 트랜잭션의 목록을 포함한다. 이 과정을 통해서 지속적으로 성장하는 블록체인이 생성되게 되는데, 비트코인 장부의 최신상태(state)를 나타내기 위해 지속적인 업데이트가 이루어진다.

이 체계에서 하나의 블록이 유효한지 아닌지를 확인하기 위한 알고리즘은 다음과 같다.

1. 이 블록에 의해 참조되는 이전 블록이 존재하는지, 유효한지 확인한다.
2. 타임스탬프 값이 이전 블록의 타임스탬프 값보다 크면서 2시간 이내인지 확인한다.
3. 작업증명(proof of work)이 유효한지 확인한다.
4. `S[0]`를 이전 블록의 마지막 상태(state)가 되도록 설정한다.
5. `TX`를 `n`개의 트랜잭션을 가지는, 블록의 트랜잭션 목록으로 가정한다. 폐구간 `0...n-1`의 모든 i에 대해, `S[i+1] = APPLY(S[i], TX[i])`집합 중 어느 하나라도 에러를 리턴하면 거짓(false)을 리턴하며 종료한다.
6. 참(true)을 리턴하고, `S[n]`를 이 블록의 마지막 상태로 등록한다.

기본적으로 블록의 각 트랜잭션은 유효한 상태변환을 일으켜야 한다. 여기서 상태가 블록 내에 어떠한 방법으로로 기록되지 않았다는 점에 주목해보자. 상태는 유효성을 검증하는 노드가 매번 계산해서 기억해야 할 완전히 추상적 것(abstraction)인데, 이것은 원시상태(genesis state)부터 해당 블록까지의 모든 트랜잭션을 순차적으로 적용함으로써 계산될 수 있다. 채굴자가 블록에 포함시키는 트랜잭션의 순서에 주목해보자. 만약 어떤 블록에 A와 B라는 두 트랜잭션이 있고 B가 A의 출력 UTXO를 소비한다고 하자. 이때 A가 B이전의 트랜잭션인 경우 그 블록은 유효하지만, 그렇지 않을 경우 유효하지 않다.

블록 유효성 검증 알고리즘에서 특징적인 부분은 "작업증명(proof of work)"의 조건 즉, 256 비트 숫자로 표현되는 각 블록의 이중-SHA256 해시값이 동적으로 조정되는 목표값(이더리움 영문 백서를 작성하는 시점에서 대략 2^{187})보다 반드시 작아야 된다는 조건이다. 작업증명의 목적은 블록 생성을 계산적으로 어렵게 만들어서 sybil 공격자들이 마음대로 전체 블록체인을 조작하는 것을 방지하는 것이다. SHA256은 전혀 예측불가능한 유사난수 함수(pseudorandom function)로 설계되었기 때문에 유효 블록을 생성하기 위한 유일한 방법은 블록헤더의 논스(nonce) 값을 계속해서 증가시키면서, 생성되는 새로운 해시값이 위의 조건을 만족하는지 확인하는 과정을

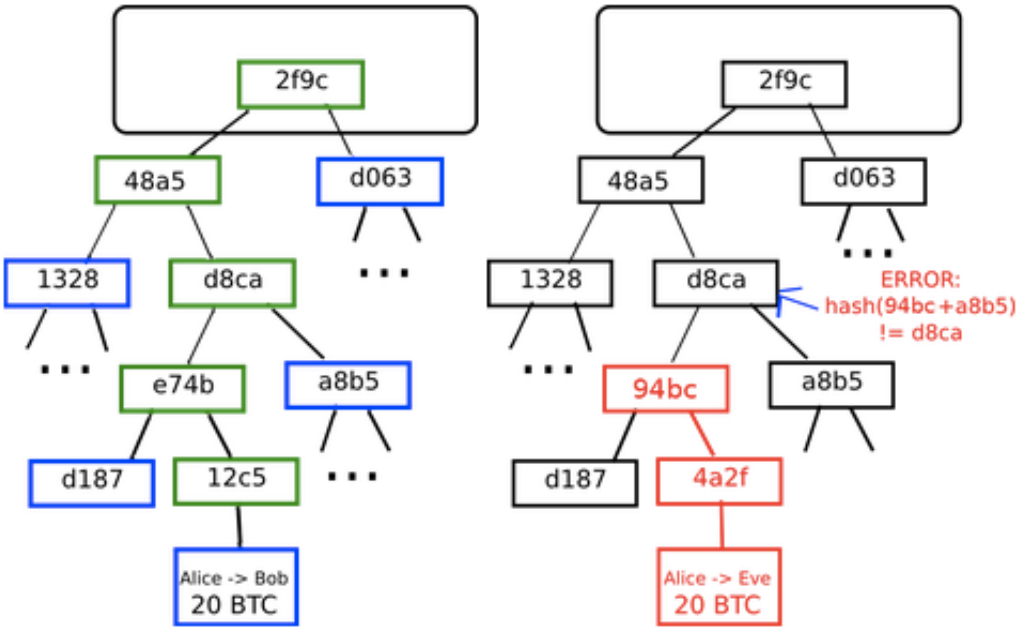
반복하는 것 뿐이다. 현재 목표값인 2^{187}하에서 하나의 유효블록을 발견하기 위해서 평균적으로 2^{64}번의 시도를 해야만 한다. 일반적으로 이 목표값은 매 2016개의 블록마다 네트워크에 의해 재조정되어서 네트워크의 현재 노드들이 평균적으로 10분마다 새로운 블록을 생성할 수 있도록 한다. 이러한 연산작업에 대한 보상으로 현 시점의 각 블록의 채굴자들은 25 BTC를 획득할 자격을 가진다. 그리고 출력금액보다 입력금액이 큰 트랜잭션이 있다면 그 차액을 "트랜잭션 수수료(transaction fee)"로 얻는다. 이것이 BTC가 발행되는 유일한 방법이며, 원시상태(genesis state)에는 아무런 코인이 포함되지 않았다.

채굴 목적을 더 잘 이해하기 위해서, 악의적인 공격자가 있을 때 어떤 일이 발생하는지 알아보자. 비트코인의 뼈대를 이루는 암호기법은 안전한 것으로 알려져 있다. 그러므로 공격자는 비트코인 시스템에서 암호기법에 의해 직접 보호되지 않는 부분인 '트랜잭션 순서'를 공격 목표로 잡을 것이다. 공격자의 전략은 매우 단순하다.

1. 어떤 상품(가급적이면 바로 전달되는 디지털 상품)을 구매하기 위해 판매자에게 100 BTC를 지불한다.
2. 상품이 전송되기를 기다린다.
3. 판매자에게 지불한 것과 같은 100 BTC를 공격자 자신에게 보내는 트랜잭션을 생성한다.(이중지불 시도)
4. 비트코인 네트워크가, 공격자 자신에게 보내는 트랜잭션이 판매자에게 지불하는 트랜잭션보다 먼저 수행된 것으로 인식하도록 한다.

1번 과정이 발생하고 몇 분 후에 몇몇 채굴자가 그 트랜잭션을 블록에 포함할 것이다. 이 블록 번호를 270000이라 하자. 대략 1시간 후에는 이 블록 다음의 체인에 5개의 블록들이 추가될 것이다. 이 5개의 블록들은 위 1번 트랜잭션을 간접적으로 가리킴으로써 "컨펌(confirming)"한다. 이 시점에서 판매자는 지불이 완료된 것으로 판단하고 상품을 전송할 것이다. 디지털 상품으로 가정했으므로 전송은 바로 끝난다. 이제 공격자는 판매자에게 보낸 것과 동일한 100 BTC를 공격자 자신에게 보내는 다른 트랜잭션을 생성한다. 만약 공격자가 그냥 단순하게 트랜잭션을 시도한다면, 채굴자들이 `APPLY(S,TX)`를 실행하고 이 `TX`는 상태에 더 이상 존재하지 않는 UTXO를 소비하려 한다는 것을 알아차리므로 이 트랜잭션은 진행되지 않는다. 그러므로 대신에, 같은 부모 블록 269999을 가리키지만 판매자에게 보낸 것을 대체하는 새로운 트랜잭션이 포함된 다른 버전의 블록 270000을 채굴함으로서 블록체인 "분기점(fork)"을 생성한다. 이 블록 정보는 원래 것과 다르므로 작업증명(proof of work)이 다시 수행되어야 한다. 그리고 공격자의 새버전 블록 270000은 기존 270000과 다른 해시를 가지므로 원래 블록 270001부터 270005는 공격자의 블록을 가리키지 않는다. 그러므로 원래 체인과 공격자의 새로운 체인은 완전히 분리된다. 이러한 분기점에서 비트코인 네트워크의 규칙은 가장 긴 블록체인을 참으로 인식하는 것이다. 공격자가 자신의 체인에서 혼자 작업을 하는 동안 정당한 채굴자들은 원래의 270005체인에서 작업할 것이기 때문에 공격자 자신의 체인을 가장 길게 만들기 위해서는 네트워크의 다른 노드들의 계산능력 조합보다 더 큰 계산능력을 가져야 한다.(이를 51% attack이라 한다.)

머클트리



왼쪽: Merkle tree(머클트리)의 몇몇 노드만 보아도 곁가지(branch)의 유효성을 입증하기에 충분하다.

오른쪽: Merkle tree의 어떤 부분을 바꾸려는 시도는 결국 상위 해시값 어딘가에 불일치를 만든다.

비트코인의 중요한 확장 기능은 블록이 여러 계층 구조(multi-layer data structure)에 저장된다는 것이다. 어떤 블록의 "해시(hash)"란 사실 블록헤더의 해시만을 의미한다. 이 블록헤더에는 타임스탬프, 논스(nonce), 이전 블록 해시, 그리고 블록에 포함된 모든 트랜잭션 정보에 의해 생성되는 머클트리(Merkle tree)의 루트 해시가 들어있는 200 바이트 정도의 데이터이다. 머클트리(Merkle tree)는 이진트리(binary tree)의 일종으로서 트리의 최하위에 위치하고 기저 데이터가 들어있는 수 많은 잎노드, 자기 자신 바로 하위에 있는 두 자식 노드의 해시로 구성된 중간 노드, 자기 자신 바로 하위에 있는 두 자식 중간노드의 해시로 구성된 트리의 "최상위(top)"에 있는 하나의 루트 노드의 집합이다. 머클트리(Merkle tree)의 목적은 어떤 블록의 데이터가 분리돼서 전달될 수 있도록 하는 것이다. 만약에 비트코인의 어떤 노드가 한 소스로부터 블록헤더만을 다운로드 받고, 이 블록헤더와 관계된 트랜잭션 정보는 다른 소스로부터 다운받아도 이 데이터들이 여전히 정확하다는 것이 보장된다. 이것이 가능한 이유는 Merkle tree에서 하위 노드들의 해시값이 상위 노드에 영향을 주기 때문에 어떤 악의적인 유저가 머클트리 최하위에 있는 트랜잭션 정보를 가짜로 바꿔치기 하면 상위 부모들의 해시값들이 변해서 결국 트리의 루트값이 바뀌므로, 결과적으로 이 블록의 해시가 달라지기 때문이다. 이렇게 되면 이 블록은 완전히 다른 블록으로 인식되게 되며, 이것은 유효하지 않은 작업증명을 가지고 있게 될 것이 분명하다.

머클트리 프로토콜은 비트코인 네트워크를 장기간 지속가능하게 만드는 기초가 된다. 비트코인 네트워크에서 각 블록의 모든 정보를 저장하고 처리하는 "완전노드(full node)"는 2014년 4월 기준으로 거의 15 GB의 디스크 공간을 필요로 하며 매달 1 GB 넘게 증가하고 있다. 현재 데스크탑 컴퓨터 정도에서는 수용할 수 있지만 스마트폰에서는 불가능하다. 그리고 나중에는 소수의 사업체들이나 풀 노드를 유지할 수 있을 것이다. 반면 "단순화된 지불검증(simplified payment verification, SPV)"으로 알려진 프로토콜은 "가벼운 노드(light node)"라고 불리는 또 다른 형태의 노드를 가능하게 해준다. 가벼운 노드는 블록헤더를 다운로드하고 그 블록헤더에서 작업증명을 검증한다. 그리고 관련 트랜잭션들에 대한 "곁가지들(branches)"만을 다운로드 한다. 이렇게 전체 블록체인의 매우 작은 비율만을 다운로드 함에도 불구하고 강한 안전성을 보장하면서도, 임의의 트랜잭션의 상태 및 잔고 상태를 알아낼 수 있게 한다.

블록체인 기술을 이용한 다른 응용 사례 (Alternative Blockchain Applications)

블록체인의 근본 아이디어를 확장해 다른 개념으로 응용하려는 아이디어 역시 오랜 역사를 가지고 있다. 2005년 'Nick Szabo'는 "소유주 권한을 통한 재산권 보장"이라는 글을 발표했다. 그는 정주(homesteading), 불법점유, 지공주의(Georgism) 등의 개념을 포함한 정교한 틀을 설계해 누가 어떤 땅을 가지고 있느냐라는 등기 문제를 블록체인 기반 시스템으로 처리할 수 있음을 보였다. 그는 이것이 "데이터베이스 복제 기술의 새로운 발전"덕분에 가능해졌다고 말했다. 하지만, 불행히도 그 당시에는 쓸만한 효과적인 파일 복제 시스템이 없었기때문에 Nick Szabo의 프로토콜은 실현되지 못했다. 하지만 2009년 이후 비트코인 분권 합의 시스템이 발전하면서 수 많은 대안 응용 사례가 빠르게 부각되기 시작했다.

- **네임코인** - 2010년에 만들어진 네임코인은 '탈중앙화된 명칭 등록 데이터베이스'라고 부르는 것이 가장 좋을 것이다. 토르, 비트코인, 비트메시지와 같은 탈중앙화된 자율조직 프로토콜을 이용할 때, 사용자는 타인과 서로 교류하기 위해 각자의 계정을 구분해내야 한다. 하지만 현존하는 가능한 구별 방법은 1LW79wp5ZBqaHW1jL5TCiBCrhQYtHagUWy와 같은 식의 의사난수 해쉬를 이용하는 방식이었다. 이상적으로는, 사용자가 "george"같은 일상적인 이름을 계정 이름으로 갖는 것이 좋겠지만 문제는 어떤 사용자가 "george" 라는 이름을 계정으로 만들 수 있다면, 다른 누구도 똑같이 "george"라는 계정을 등록해 흉내낼 수 있다는 점이다. 유일한 해답은 선출원주의로, 먼저 등록한 사람이 성공하고 두 번째 등록한 사람은 실패하도록 하는 것이다. 이는 이미 비트코인 합의 규약에 완벽히 적용된 문제이기도 하다. 네임코인은 이런 아이디어를 응용한 가장 오래되고 가장 성공적인 명칭 등록 시스템이다
- **컬러드 코인** - 컬러드 코인의 목적은 누구나 비트코인 블록체인 위에서 자신만의 고유한 디지털 화폐를 발행할 수 있는 프로토콜 역할을 하는 것이다. 또는 (그 디지털 화폐의 발행량이 한 단위 밖에 없는 단순한 경우로 환원할 수 있는) 자기 자신만의 디지털 토큰을 발행하는 프로토콜 역할을 하는 것이다. 컬러드 코인 프로토콜에서, 사용자는 특정 비트코인 UTXO에 공개적으로 색깔을 부여함으로써 새 화폐를 "발행"할 수 있다. 다른 UTXO의 색깔은 이미 소비된(혼합 색깔 입력의 경우에는 몇몇 특별한 규칙이 적용된다) 것으로 간주하는 거래의 입력과 같은 색깔이 되도록 재귀적으로 정의한다. 이 프로토콜은 블록체인을 처음부터 끝까지 역추적해 그들이받은 UTXO의 색깔을 정함으로써, 사용자가 특정 색깔을 가진 UTXO만 지갑에 간직하고 그 코인을 보통 비트코인처럼 여기저기 보낼 수 있게 한다.
- **메타코인** - 메타코인이 품고 있는 아이디어는, 비트코인 거래를 메타코인 거래 저장에 이용하되, 상태 이동 함수 APPLY' 를 다르게 가짐으로써, 비트코인 시스템 위에서 운영되는 프로토콜을 갖는 것이다. 메타코인 프로토콜만으로는 비트코인 블록체인 속에 무효 메타코인 거래가 나타나는 현상을 예방 수 없기 때문에, 규칙이 하나 더해진다. 즉 만약 APPLY'(S, TX)가 에러를 리턴하면, 프로토콜은 APPLY'(S,TX)=S로 정해진다. 비트코인 스스로는 내부 실행이 불가능한, 잠재적으로 더 발전된 성질을 가진 무작위 암호화폐 프로토콜을 만드는 쉬운 메커니즘이라고 할 수 있다. 반면 이 프로토콜의 개발비용은 적은데 왜냐하면 채굴과 네트워킹의 복잡성 문제가 이미 비트코인 프로토콜에 의해 처리되고 있기 때문이다.

일반적으로 합의 프로토콜을 건설하는 데 두 가지 접근방법이 있다. 하나는 독립적인 네트워크를 세우는 것이고 다른 하나는 비트코인 시스템과 연동되는 프로토콜을 세우는 것이다. 전자의 접근 방법은 네임코인 같은 응용 사례에서는 상당히 성공적이었지만, 실제 실행하는 데 어려움이 있다; 각 개별 실행주체가 모든 필요한 상태변환과 네트워킹 코드를 건설하고 점검해야 할 뿐만 아니라 독립적인 블록체인을 구동시켜야 한다. 나아가, 분권 합의 기술에 관한 어플리케이션의 집합이 멱함수분포를 따를 것으로 예상된다. 즉, 대다수 어플리케이션은 자기 자신의 블록체인을 보장하기에는 너무 작을 것이다. 그리고 또 거대한 클래스의 분권화된 어플리케이션, 즉 서로 교류를 하기 위한 분권화된 자율 기구(DAO)가 생겨날 것이라고 예상한다.

후자의 접근 방법, 즉, 비트코인에 기반한 접근 방법은 비트코인의 단순 지불 검증(SPV)특징을 물려받지 못한다는 단점이 있다. 단순지불검증은 비트코인에서는 작동한다. 왜냐하면 비트코인은 블록체인 깊이(depth)를 검증 대리 수단으로 이용할 수 있기 때문이다. 한 거래의 근원을 찾아 충분히 뒤로 돌아가보면, 그 상태의 정합성을 증명하는 부분이 있었다고 말해도 무방하다. 반면, 블록체인에 기반한 메타-프로토콜은 무효거래가 블록체인에 포함되지 않도록 막을 방법이 자기 자신의 프로토콜 자체에는 없다. 그렇기 때문에 완전히 안전보장이 된 단순지불검증 메타-프로토콜이라면, 어떤 거래가 유효한지 아닌지를 결정하기위해, 항상 비트코인 블록체인의 원점까지 돌아가 훑어보는 작업이 필요하다. 현재까지 비트코인에 기반한 메타-프로토콜의 모든 "간단한"(light) 클라이언트 구현은

자료를 제공하는 믿을 만한 서버에 의지하고 있는 형편이다. 우리가 암호화폐를 만든 가장 중요한 목적이 제3의 신용기구의 필요성을 없애는 것이었다는 걸 특히 되새겨본다면, 이것은 아주 분명하게도, 차선의 결과가 될 뿐이다.

스크립팅

별도의 확장없이도 비트코인 프로토콜은 낮은 수준의 "스마트 계약"의 개념을 가능하게 할 수 있다. 비트코인의 UTXO는 공개키만으로 획득할 수 있을 뿐만 아니라, 단순 스택-기반 프로그래밍 언어로 표현되는 더 복잡한 스크립트로도 획득할 수 있다. 이런 경우에, UTXO를 지출하는 거래는 그 스크립트를 만족하는 데이터를 제공해야만 한다. 사실, 기초적인 공개키 소유권 메커니즘도 스크립트를 통해 실행된다: 그 스크립트는 타원곡선서명을 '입력'으로 받아 그 거래와 UTXO를 가진 주소에 대해 검증을 하고 만약 검증이 성공하면 1을, 실패하면 0을 '출력'한다. 여러 다른 다양한 사용 사례에 대해 좀 더 복잡한 여러 스크립트들이 있을 수 있다.

예를 들어, 주어진 세 개의 개인 키 가운데 두 개로부터 서명을 받아야만 승인이 되도록 스크립트를 짤 수 있다. 이런 스크립트는 회사 계정, 보안 저축 계정, 상업 공탁 상황 등에 유용하게 쓰일 수 있다. 스크립트는 또한 어떤 계산 문제의 답에 대한 포상금을 지불하는데도 쓰일 수 있다. "만약 당신이 이 액면가의 도기코인 거래를 나에게 보냈다는 SPV 증명을 제공한다면, 이 비트코인 UTXO는 당신 것이다"라는 식으로 말하는 스크립트를 짤 수도 있다. 즉 근본적으로 탈중앙화된 상호-암호화폐 교환을 가능하게 한다.

하지만 비트코인에 구현된 스크립트 언어는 몇가지 중요한 한계가 있다.

- **튜링불완전성**: 비트코인 스크립트 언어로 할 수 있는 작업이 많긴 하지만, 모든 경우의 프로그래밍을 다 지원하지는 않는다. 특히 while이나 for와 같은 순환(loop) 명령 카테고리가 빠져 있다. 순환 명령어를 없앤 이유는 거래 증명을 할 때 무한 순환에 빠지는 것을 막기 위해서였다. 이론적으로는 튜링불완전성은 스크립트 프로그래머가 극복할 수 있는 장애물이기는 하다. 왜냐하면 어떤 순환 명령이든 단순히 하위 코드를 여러 차례 if 구문과 함께 반복함으로써 구현이 가능하기 때문이다. 하지만 이것은 아주 공간 비효율적인 프로그램이 된다. 예를 들어 대안 타원곡선서명 알고리즘을 실행하려면 코드 안에 있는 곱셈을 모두 개별적으로 256번 반복하는 것이 필요하다.
- **가치무지하다**: UTXO 스크립트만으로는 인출 액수를 세밀하게 통제할 방법이 없다. 예를 들어 신탁 계약의 강력한 실용 사례라 할 수 있는 헷지 계약을 살펴보자. A와 B가 1000어치의BTC를공동계좌에입금했다고하자. 시간이지나면비트코인의가격이오를수가있다. 두사람은30일후자동으로A가 1000어치 BTC를 받고 B는 공동계좌의 나머지 잔액을 받는 그런 계약을 맺고 싶다. 하지만 이 계약은 1BTC가 미국 달러로 얼마인지 정해줄 제3자를 필요로 한다. 만약 이런 계약이 실현가능하다면 지금 현존하는 완전 중앙집권적인 금융 시스템 아래에서도 고도로 발전된 계약 형태라고 볼 수 있을 것이다. 하지만 UTXO는 인출액 전부가 송금되거나 말거나 밖에 선택할 수가 없다. 즉 세부 작은 단위로 나눠질 가능성을 포함할 수 없는 것이다. 위에 예를 든 계약 거래를 실행할 유일한 방법은 변하는 UTXO의 액면가 단위를 아주 다양하게 양산하고(예를 들어 1부터 30까지의 모든 자연수 k에 대해 2의 k승의 1 UTXO를 만듦) A가 B에게 이중에서 필요한 금액에 맞는 것을 선택해서 보내게 하는 방식과 같이 매우 비효율적인 편법을 사용하는 길 뿐이다.
- **다양한 상태를 표현할 수 없다(Lack of State)**: UTXO가 표현할 수 있는 상태는 사용되었거나 안 되거나 둘 뿐이다. 그렇기 때문에 이 두가지 상태 이외에 다른 어떤 내부적 상태를 가지는 다중 단계 계약이나 스크립트를 만들 수가 없다. 이 점이 분산 환전 거래나 이중 암호 실행 프로토콜(계산 보상금을 보장하기 위해 필요하다)과 같은 다중 조건 계약을 어렵게 한다. 즉 UTXO은 단순하고 1회적인 계약에만 이용될 수 있을 뿐, 분산조직과 같은 더 복잡한 "상태적(stateful)" 계약에는 이용될 수 없고 메타프로토콜을 적용하기 어렵게 만든다.
- **블록체인을 해독할 방법이 없다(Blockchain-blindness)**: UTXO는 논스(Nonce), 타임스탬프,이전 블록해시같은 블록체인 자료를 해독하지 못한다. 이 단점으로 인해 스크랩트 언어 속에 잠재적으로 가치있을 무작위성이 빠지게 된다. 그래서 도박이나 여러 다른 분야의 어플리케이션을 만드는 데 한계를 보인다.

정리하자면, 발전된 어플리케이션을 만드는 데 3가지 접근법이 있다. 첫번째는 독립적인 블록체인을 만드는 것이고 두번째는 비트코인에 이미 내재된 스크립트를 이용하는 것이며, 세번째는 비트코인 상에서 작동되는 메타-규약을 건설하는 것이다. 독립적인 블록체인을 쓰면 무한히 자유로운 프로그램을 짤 수 있지만 개발 기간, 초기 셋업 작업, 보안 등의 비용을 치뤄야 한다. 비트코인에 내재된 스크립트를 이용하면 실행이 간단하고 표준화된다는 장점이 있지만, 이용범위가 제한적이다. 메타규약을 쓰는 것은 간단하긴 하지만, 확장성의 결함을 감수해야 한다.
이더리움을 통해 우리는 개발하기도 쉽고 더 강력한 라이트 클라이언트 기능을 가지는 동시에 경제적인 개발 환경과 블록체인 보안을 공유하는 어플리케이션을 만들 수 있는, 대안 프레임워크(alternative framework)를 건설하려고 한다.

이더리움

이더리움의 목적은 분산 어플리케이션 제작을 위한 대체 프로토콜을 만드는 것이다. 대규모 분산 어플리케이션에 유용할 것이라 생각되는 다른 종류의 제작기법을 제공하며, 빠른 개발 시간, 작고 드물게 사용되는 어플리케이션을 위한 보안, 다른 어플리케이션과의 효율적인 상호작용이 중요한 상황에 특히 주안점을 두고 있다. 이더리움은 튜링 완전 언어를 내장하고 있는 블록체인이라는 필수적이고 근본적인 기반을 제공함으로써 이 목적을 이루고자 한다. 누구든지 이 언어를 사용해 스마트 컨트랙트, 분산 어플리케이션을 작성하고 소유권에 대한 임의의 규칙, 트랜잭션 형식(transaction format), 상태변환 함수(state transition function) 등을 생성 할 수 있다. 네임코인의 기본적인 형태는 두 줄 정도의 코드로 작성할 수 있고, 통화나 평판 시스템 관련 프로토콜은 스무 줄 내외의 코드로 만들 수 있다. 어떤 값을 저장하고, 특정한 조건들을 만족했을 때만 그 값을 얻을 수 있게 하는 일종의 암호 상자인 스마트 컨트랙트 또한 이 플랫폼 위에 만들 수 있다. 이것은 비트코인의 스크립팅(scripting)이 제공하는 것보다 훨씬 강력한 기능들이 제공되기 때문에 가능한 것으로, 튜링-완전(Turing-completeness), 가치 인지능력(value-awareness), 블록체인 인지능력(blockchain-awareness), 상태(state)개념 등이 포함된다.

이더리움 어카운트

이더리움에서, 상태(state)는 어카운트(account)라고 하는 오브젝트(object)들로 구성되어 있다. 각각의 어카운트는 20바이트의 주소와 어카운트 간 값과 정보를 직접적으로 전달해 주는 상태변환(state transition)을 가지고 있다. 이더리움 어카운트는 다음 네 개의 필드를 가지고 있다.

- **논스(nonce)**: 각 트랜잭션이 오직 한번만 처리되게 하는 일종의 카운터
- 어카운트의 현재 **이더(ether) 잔고**
- 어카운트의 **계약 코드** (존재한다면)
- 어카운트의 **저장 공간** (초기설정(default) 상에서는 비어있음)

이더는 이더리움의 기본 내부 암호-연료(crypto-fuel) 이고, 트랜잭션 수수료를 지불하는데 사용된다. 보통 두가지 종류의 어카운트가 존재하는데, 프라이빗 키에 의해 통제되는 외부 소유 어카운트(Externally Owned Accounts)와 컨트랙트 코드에 의해 통제되는 컨트랙트 어카운트(Contract Accounts)가 있다. 외부 소유 어카운트는 아무런 코드도 가지고 있지 않으며, 이 어카운트에서 메시지를 보내기 위해서는 새로운 트랜잭션을 하나 만들고, 서명(signing)을 해야 한다. 컨트랙트 어카운트는 메시지를 받을 때마다, 자신의 코드를 활성화시키고, 이에 따라 메시지를 읽거나 내부 저장공간에 기록하고, 다른 메시지들을 보내거나, 컨트랙트들을 차례로 생성하게 된다. 이더리움에서 컨트랙트는, 수행되거나 컴파일 되어져야 할 어떤 것이라기 보다는, 이더리움의 실행 환경안에 살아있는 일종의 자율 에이전트(autonomous agents)로서, 메시지나 트랜잭션이 도착하면 항상 특정한 코드를 실행하고, 자신의 이더 잔고와, 영속적인 변수들을 추적하기 위해 자신의 키/값 저장소를 직접적으로 통제하는 역할을 한다.

메시지와 트랜잭션

이더리움에서 사용되는 트랜잭션(transaction)이란 용어는 외부 소유 어카운트가 보낼 메시지를 가지고 있는 서명된 데이터 패키지를 말한다. 이 트랜잭션은 다음을 포함하고 있다.

- 메시지 수신처
- 발신처를 확인할 수 있는 서명
- 발신처가 수신처로 보내는 이더의 양
- 선택적(optional) 데이터 필드
- STARTGAS 값, 트랜잭션 실행이 수행되도록 허용된 최대 계산 단계수
- GASPRICE 값, 매 계산단계마다 발신처가 지불하는 수수료

처음 세 항목은 암호 화폐에서는 거의 표준처럼 사용되는 값이다. 데이터 필드는 초기값으로 설정된 기능(function)은 가지고 있지 않지만, 버추얼 머신(virtual machine)은 컨트랙트가 이 데이터에 접근할 때 사용할 수행코드(opcode)를 가지고 있다. 예를 들어, 블록체인 위에 도메인 등록 서비스로 기능하고 있는 컨트랙트가 있을 경우, 이 컨트랙트로 보내지는 데이터는 두개의 필드를 가지고 있는 것으로 해석할 수 있다. 첫번째 필드는 등록하고자 하는 도메인이고, 두번째 필드는 IP 주소이다. 컨트랙트는 메시지 데이터로부터 이 값들을 읽어서 저장소 내 적당한 위치에 저장한다.

STARTGAS 와 GASPRICE 필드는 이더리움의 앤티-서비스거부(anti-DoS) 모델에 있어서 매우 중요한 역할을 한다. 코드내의 우연적이거나 악의적인 무한루프, 또는 계산 낭비를 방지하기 위해 각각의 트랜잭션은 사용할 수 있는 코드 실행의 계산 단계 수를 제한하도록 설정되어야 한다. 계산의 기본 단위는 gas이고 보통, 계산 단계는 1 gas의 비용이 소요되나, 어떤 연산은 더 비싼 계산 비용을 치루거나, 상태의 일부분으로 저장되어야 하는 데이터의 양이 많을 경우 더 많은 수의 gas 비용이 필요하게 된다. 또한 트랜잭션 데이터에 있는 모든 바이트는 바이트당 5 gas 의 수수료가 든다. 이러한 수수료 시스템의 의도는 어떤 공격자가 계산, 밴드위스, 저장소 등을 포함해 그들이 소비하는 모든 리소스에 비례하여 강제로 수수료를 지불하게 하는데 있다. 따라서, 이런 리소스중 어떤 것이라도 상당량을 소비하는 네트웍과 연관된 트랜잭션은 대략 증가분에 비례한 gas 수수료를 가지고 있어야 한다.

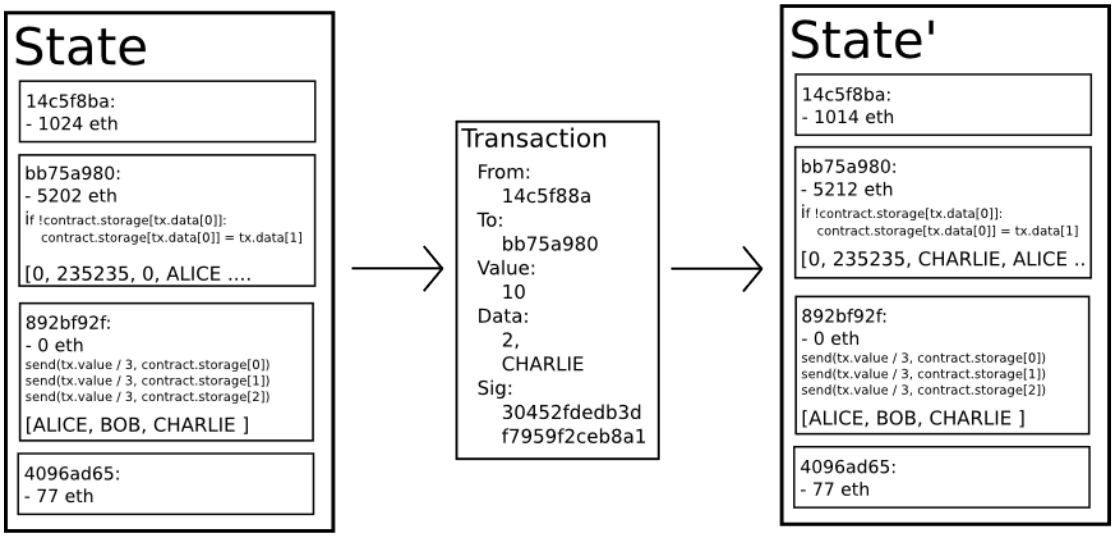
메시지(Messages)

컨트랙트는 다른 컨트랙트에게 "메시지"를 전달할 수 있다. 메시지는 따로 저장될 필요가 없는 이더리움의 실행 환경에서만 존재하는 가상의 오브젝트이다. 메시지는 다음의 것을 포함하고 있다.

- (암묵적으로) 메시지 발신처
- 메시지 수신처
- 메시지와 함께 전달되는 이더
- 선택적 데이터 필드
- STARTGAS 값

본질적으로, 메시지는 외부 실행자가 아닌 컨트랙트에 의해 생성된다는 것을 제외하면 트랜잭션과 유사하다. 현재 코드 수행을 하고 있는 컨트랙트가 메시지를 생성하고 실행하라는 CALL opcode를 만나게 되면 메시지를 생성한다. 트랜잭션과 마찬가지로, 메시지는 해당 코드를 실행하는 수신자 어카운트에 도달하게 된다. 따라서, 컨트랙트는 외부 실행자가 하는 것과 정확히 같은 방식으로 다른 컨트랙트와 관계를 맺을 수 있다. 트랜잭션이나 컨트랙트에 의해 할당된 gas 허용치는 그 트랜잭션과 모든 하위 실행에 의해 소모된 총 gas 에 적용된다. 예를 들어, 외부 실행자 A가 B에게 1000 gas와 함께 트랜잭션을 보내고, B는 600 gas를 소모한 뒤 C에게 메시지를 보내고, C의 내부 실행에 300 gas를 소모한 후 반환하면, B는 gas가 모두 소모되기 전에 100 gas를 더 사용할 수 있다.

이더리움 상태 변환 함수(Ethereum State Transition Function)



이더리움 상태 전이 함수 APPLY(S, TX) -> S' 는 다음처럼 정의될 수 있다. 트랜잭션이 형식에 제대로 맞는지(즉, 올바른 갯수의 값을 가지고 있는지) 체크하고, 서명이 유효한지, 논스가 발신처 어카운트의 논스와 일치하는지를 체크한다. 그렇지 않다면 오류를 반환한다. STARTGAS * GASPRICE 로 트랜잭션 수수료를 계산하고, 서명으로부터 발신처 주소를 결정한다. 발신처 어카운트 잔고에서 이 수수료를 빼고 발신자 논스를 증가시킨다. 발신처 잔고가 충분하지 않으면 오류를 반환한다. GAS = STARTGAS 로 초기화 한후, 트랜잭션에서 사용된 바이트에 대한 값을 지불하기 위해 바이트당 gas의 특정양을 차감한다. 발신처 어카운트에서 수신처 어카운트로 트랜잭션 값을 보낸다. 수신처 어카운트가 존재하지 않으면 새로 생성한다. 수신처 어카운트가 컨트랙트이면, 컨트랙트의 코드를 끝까지, 또는 gas가 모두 소모될 때 까지 수행한다. 발신처가 충분한 '돈'을 가지고 있지 못해서 값 전송이 실패하거나, 코드 수행시 gas가 부족하면, 모든 상태 변경를 원상태로 돌려놓는다. 단, 수수료 지불은 제외되고, 이 수수료는 채굴자 어카운트에 더해지게 된다. 그 외에는, 모든 남아있는 모든 gas에 대한 수수료를 발신처에게 돌려주고, 소모된 gas에 지불된 수수료를 채굴자에게 보낸다.

예를 들어, 다음과 같은 컨트랙트 코드를 가정해 보자.

```
if !self.storage[calldataload(0)]:
    self.storage[calldataload(0)] = calldataload(32)
```

실제로 컨트랙트 코드는 로우-레벨 EVM 코드로 작성되나, 이 예제는 이해하기 쉽게 하기 위해, 이더리움 하이-레벨 언어중 하나인 Serpent 로 작성하였다. 이 코드는 EVM 코드로 컴파일 될 수 있다. 컨트랙트의 스토리지는 비어있다고 가정하고, 트랜잭션이 10 ether, 2000 gas, 0.001ether gasprice, 64 바이트의 데이터(0-31 바이트까지는 숫자 2를 나타내고, 32-63 바이트는 CHARLIE 라는 문자열)를 보낸다고 가정하자. 이 경우 상태 변환 함수의 프로세스는 다음과 같다.

1. 트랜잭션이 유효하고 형식에 제대로 맞는지 확인한다.
2. 트랜잭션 발송처가 최소 2000 * 0.001 = 2 ether를 가지고 있는지 확인하고, 그럴 경우, 발송처의 어카운트에서 2 ether를 뺀다.
3. gas = 2000으로 초기화 한 후, 트랜잭션은 170바이트 길이를 가지고, 바이트당 수수료는 5라고 가정하면, 850을 빼야 하고 결국 1150 gas가 남게된다.
4. 발송처 어카운트에서 추가 10 ether를 빼고 이것을 컨트랙트 어카운트에 더한다.
5. 코드를 실행시킨다. 이 경우는 간단한데, 컨트랙트의 index 2에 해당하는 스토리지가 사용되었는지 확인하고 (이 경우, 사용되지 않았다.) index 2에 해당하는 스토리지 값을 CHARLIE 로 설정한다. 이 작업에 187 gas 가 소비됐다고 가정하면, 남아있는 gas 의 양은 1150 - 187 = 963 이 된다.

6. 963*0.001 = 0.963 ether를 송신처의 어카운트로 되돌려주고, 결과 상태를 반환한다.

트랜잭션의 수신처에 컨트랙트가 없으면, 총 트랜잭션 수수료는 제공된 GASPRICE와 트랜잭션의 바이트 수를 곱한 값과 같아지고, 트랜잭션과 함께 보내진 데이터는 관련이 없어지게 된다.

메시지는 트랜잭션과 마찬가지 방식으로, 상태를 원래 상태로 되돌린다는 것에 주목하자. 메시지 실행시 gas가 부족하게 되면, 그 메시지 실행과 그 실행에 의해 촉발된 다른 모든 실행들은 원래대로 되돌려지게 되지만, 그 부모 실행은 되돌려질 필요가 없다. 이것은 컨트랙트가 다른 컨트랙트를 호출하는 것은 안전하다는 것을 의미한다. A가 G gas를 가지고 B를 호출하면, A의 실행은 최대 G gas만을 잃는다는 것을 보장받게 된다. 컨트랙트를 생성하는 CREATE라는 opcode를 보면, 실행 방식은 대체로 CALL과 유사하나, 실행 결과는 새로 생성된 컨트랙트의 코드를 결정한다는 차이가 있다.

코드 실행(Code Execution)

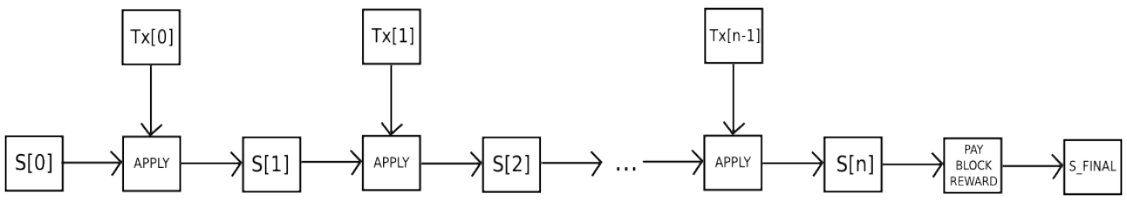
이더리움 컨트랙트를 구성하는 코드는 "이더리움 버추얼 머신 코드" 또는 "EVM 코드"로 불리는 로우-레벨, 스택 기반의 바이트코드 언어로 작성된다. 이 코드는 연속된 바이트로 구성되어 있고, 각각의 바이트는 연산(operation)을 나타낸다. 보통, 코드 실행은 0부터 시작하는 현재 프로그램 카운터를 하나씩 증가시키면서 반복적으로 연산을 수행하도록 구성된 무한 루프이고, 코드의 마지막에 도달하거나 오류, STOP, RETURN 명령을 만나면 실행을 멈추게 된다. 연산을 수행하기 위해서는 데이터를 저장하는 세가지 타입의 공간에 접근할 수 있어야 한다.

- **스택**: last-in-first-out 컨테이너로 여기에 값들을 밀어 넣거나(push) 하거나 뺄(pop) 수 있다.
- **메모리**: 무한대로 확장 가능한 바이트 배열
- 컨트랙트의 영속적인(long-term) **저장소(storage)**: 키/값 저장소. 계산이 끝나면 리셋되는 스택이나 메모리와는 달리 저장소는 영속적으로 유지된다.

코드는 또한 블록 헤더 데이터 뿐만 아니라 특정 값이나, 발송자 및 수신되는 메시지의 데이터에 접근할 수 있고, 결과값으로 데이터의 바이트 배열을 반환할 수도 있다.

EVM 코드의 공식 실행 모델은 놀랍도록 단순하다. 이더리움 버추얼 머신이 실행되는 동안, 모든 계산 상태는 (block_state, transaction, message, code, memory, stack, pc, gas) 튜플(tuple)로 정의될 수 있고, block_state는 모든 어카운트를 포함하는 전역상태(global state)로서 잔고와 저장소(storage)를 포함한다. 반복되는 매 코드 실행 순간의 시작시, code의 pc(프로그램 카운터)번째 바이트의 현재 명령이 실행되고, (pc 가 코드의 길이보다 크면(pc >= len(code)) pc 는 0), 각각의 명령은 튜플을 어떻게 변화시킬지 대한 그 자신의 정의를 알고 있다. 예를 들어, ADD는 스택에서 두개의 아이템을 꺼내(pop), 그 합을 구한 후 다시 스택에 넣고(push) gas를 1만큼 감소시키고, pc는 1 증가시킨다. SSTORE 는 스택에서 두개의 아이템을 꺼내 이 아이템의 첫번째 값이 가리키는 컨트랙트 저장소 인덱스에 두번째 아이템을 넣는다. 이더리움 버추얼 머신 환경을 JIT 컴파일을 통해 최적화 하는 많은 방법이 있지만, 기본적인 이더리움은 수백줄의 코드로 구현될 수 있다.

블록체인과 채굴(Blockchain and Mining)



이더리움 블록체인은 여러면에서 비트코인 블록체인과 유사하나, 어느정도 차이점들이 있다. 이더리움과 비트코인에서의 각 블록체인 구조에 대한 주요 차이점으로는 비트코인과는 달리 이더리움 블록은 트랜잭션 리스트와 가장 최근의 상태(state) 복사본을 가지고 있다는 것이다. 그것 외에도, 두개의 다른 값 - 블록 넘버와 difficulty - 이 또한 블록내에 저장된다. 기본적인 이더리움 블록 검증 알고리즘은 다음과 같다.

1. 참조하고 있는 이전 블록이 존재하는지 그리고, 유효한지 확인한다.
2. 현재 블록의 타임스탬프가 참조하고 있는 이전 블록의 그것보다 크면서, 동시에 현 시점을 기준으로 15분 후보다 작은 값인지 확인한다.
3. 블록 넘버, difficulty, 트랜잭션 루트, 삼촌 루트, gas 리미트등(기타 다양한 이더리움 로우 레벨 개념)이 유효한지 확인한다.
4. 블록에 포함된 작업 증명이 유효한지 확인한다.
5. S[0] 이 이전 블록의 마지막 상태(state)라고 가정 하자.
6. TX를 현재 블록의 n개의 트랜잭션 리스트라고 하자. 0 부터 n-1에 대해, S[i+1] = APPLY(S[i], TX[i]) 로 설정하자. 어플리케이션이 오류를 반환하거나, 이 시점까지 블록에서 소모된 총 gas가 GASLIMIT를 초과하면 오류를 반환한다.
7. 채굴자에게 지불된 보상 블록을 S[n] 덧붙인 후 이것을 S_FINAL 이라 하자.
8. 상태 S_FINAL의 머클 트리 루트가 블록 헤더가 가지고 있는 최종 상태 루트와 같은지를 검증한다. 이 값이 같으면 그 블록은 유효한 블록이며, 다르면 유효하지 않은 것으로 판단한다.

이러한 접근은 언뜻, 모든 상태를 각 블록에 저장할 필요성 때문에 매우 비효율적인 것처럼 보이지만, 실제로는 효율성의 측면에서는 비트코인과 비교할만 하다. 그 이유로는 상태가 트리 구조로 저장되고, 모든 블록 후에 단지 트리의 작은 부분만이 변경되기 때문이다. 보통, 인접한 두 개의 블록간에는 트리의 대부분의 내용이 같고, 따라서 한번 데이터가 저장되면 포인터(서브트리의 해쉬)를 사용하여 참조될 수 있다. 패트리시아 트리(Patricia tree)로 알려진 이러한 종류의 특별한 트리는 머클 트리 개념을 수정하여 노드를 단지 수정할 뿐만 아니라, 효율적으로 삽입되거나 삭제하여 이러한 작업을 수행할 수 있도록 해준다. 또한, 모든 상태 정보가 마지막 블록에 포함되어 있기 때문에, 전체 블록체인 히스토리를 모두 저장할 필요가 없어지게 된다. 이 방법을 비트코인에 적용한다면 5~20배의 저장 공간 절약의 효과가 생길 것이다.

물리적인 하드웨어 관점에서 볼 때, 컨트랙트 코드는 “어디에서" 실행되는가 하는 의문이 쉽게 들 수 있다. 간단한 해답은 다음과 같다. 컨트랙트 코드를 실행하는 프로세스는 상태 전환 함수 정의의 한 부분이고, 이것은 블록 검증 알고리즘의 부분이다. 따라서, 트랜잭션이 블록 B에 포함되면 그 트랜잭션에 의해 발생할 코드의 실행은 현재 또는 향후에 블록 B 를 다운로드 하고 검증하는 모든 노드들에 의해 실행될 것이다.

어플리케이션(Applications)

기본적으로, 이더리움을 이용하여 총 세 가지 카테고리의 어플리케이션을 제작할 수 있다. 첫번째 카테고리는 돈과 직접적으로 연관된 컨트랙트를 계약참여자로 하여금 보다 강력하게 설정-관리하게끔 하는 금융 어플리케이션이다. 이의 예는 하위화폐(=유로/달러 등의 상위화폐와 환율이 연동된 화폐를 지칭), 파생상품, 헷지컨트랙트, 예금용 전자지갑, 유언장, 그리고 최종적으로는 전면적인 고용계약 수준의 것들까지 포함한다. 두번째 카테고리는 준(準) 금융 어플리케이션이다. 금전이 관여되어 있지만, 상당부분 비(非)화폐적인 면이 존재하는 계약을 위한 어플리케이션이 이에 해당된다. 이의 좋은 예로는 어려운 연산 문제를 푸는 자에게 자동적으로 포상금이 지급되는 계약이다. 마지막으로, 온라인 투표와 분권형(分權形) 거버넌스(Governance)와 같이 금융과 관련성이 아예 없는 어플리케이션이 있다.

토큰 시스템(Token Systems)

블록체인토큰시스템(On-blockchain token system)은 미화/금 등과 연동된 하위화폐, 주식과 “스마트자산* (Smart Property: 비트코인의 블록체인 상에서 소유권이 컨트롤/관리되는 자산),” "위조불가능한(secure unforgeable)" 쿠폰, 그리고 통상적인 가치와 연결되어 있지 않은 기타 토큰시스템 (예, 인센티브 부여를 위한 포인트제도) 등에 이르기까지 다양한 형태의 거래시스템을 네트워크 상에서 구현하게끔 해주는 어플리케이션들을 갖고 있다. 이더리움에서 토큰시스템은 놀랍도록 쉽게 구현할 수 있다. 토큰시스템을 이해하는 데에 핵심은 아래와 같다.

- 모든 화폐 혹은 토큰시스템은 근본은 결국 한 가지 오퍼레이션만을 수행하는 데이터베이스이다.
- A라는 주체로부터 X 단위의 화폐/토큰을 차감하고, 차감한 X 단위의 화폐/토큰을 B에게 지급한다. 단, 거래 전, A는 최소 X단위를 보유하고 있었음
- A가 이 거래를 승인함

이더리움에서 유저는 바로 위의 로직을 컨트랙트에 반영 시키기만 하면 된다. Serpent 에서 토큰시스템을 실행하는 기본적은 코드는 아래와 같다:

```
def send(to, value):
    if self.storage[msg.sender] >= value:
        self.storage[msg.sender] = self.storage[msg.sender] - value
        self.storage[to] = self.storage[to] + value
```

이는 기본적으로 본 백서에서 설명한 “은행시스템”의 "상태변환함수(state transition function)"를 아무런 가공없이 그대로적용시킨 것이다. 통화의 단위를 정의하고 배급하기 위한 최초 작업을 위해서, 또는 더 나아가 여타 컨트랙트들이 계좌의 잔금에 대한 정보요청을 처리하기 위한, 몇 줄의 코드가 추가적으로 더 쓰여져야 할 수도 있다. 하지만, 그 정도가 토큰시스템을 만드는 데 필요한 전부이다. 이론적으로, 이더리움에 기반한 하위화폐 체계로서의 토큰시스템은 비트코인에 기반한 메타화폐 (=비트코인 블록체인 연동된 화폐)가 갖고 있지 않는 중요한 특성을 지니고 있을 수 있다: 거래비용을 거래 시 사용한 화폐로 직접 지불할 수 있다는 점이 그것이다. 다음과 같은 과정을 통하여 이 특성은 발현될 수 있다: 컨트랙트을 집행하기 위해서는 발송인에게 지불해야 하는 비용 만큼의 이더 잔고를 유지해야 한다. 그리고 컨트랙트 집행 시 수수료로 받는 내부화폐(하위화폐)를 (상시 돌아가고 있는 내부화폐-이더 거래소에서) 즉각 환전하여 이더 잔고로 충전할 수 있다. 유저들은 그렇게 이더로 그들의 계좌들을 “활성화”시켜야 하지만 각 컨트랙트를 통해 얻어지는 만큼의 금액을 이더로 매번 환전해 주기에, 한 번 충전된 이더는 재사용이 가능하다고 볼 수 있다.

파생상품과 가치안정통화

파생상품은 “스마트 컨트랙트”의 가장 일반적인 어플리케이션이며, 코드로 실행할 수 있는 가장 간단한 형태의 어플리케이션 중 하나다. 금융 컨트랙트를 실행하는 데 가장 주된 어려움은 대부분의 경우 계약에서 규정하는 자산에 대한 시세를 외부에서 참조해야 한다는 것이다. 예를 들어, 금융컨트랙트에 매우 필요한 것은 이더(또는 기타 가상화폐)-USD 변동성에 대해 헷지(hedge)하는 어플리케이션인데, 이 헷지컨트랙트를 실행하기 위해서는 ETH/USD의 환율을 제공할 수 있는 컨트랙트가 필요하다. 환율을 알기 위한 가장 쉬운 방법은 주식시장의 NASDAQ과 같은 특정한 제 3자가 실시간으로 제공하는 “데이터피드” 컨트랙트를 통해서이고, 관여주체는 필요할 때 마다 환율을 업데이트할 수 있어야 하며, 여타 컨트랙트들과 환율에 대한 메시지를 주고받을 수 있는 인터페이스를 제공할 수 있어야 한다.

상기 핵심 요건들을 가정하고, 위 언급한 헷지컨트랙트는 다음과 같은 구조를 띌 것이다:

1. A가 1000 이더를 입금할 때까지 기다린다
2. B가 1000 이더를 입금할 때까지 기다린다
3. 입금된 이더의 달러가치를 기록하며 (환율은 Data feed 컨트랙트로 쿼리를 보냄으로써 계산한다), 이를 $X라 한다

4. 30일 이후, 당시의 환율을 적용한 금액을 계산하여 A에게는 $X를 송금하고 당시 총금액에 나머지를 B에게 송금하도록 A 또는 B가 컨트랙트를 다시 활성화 시킬 수 있게끔 한다.

위와 같은 컨트랙트는 가상통화를 이용한 상거래의 향후 발전 가능성을 제시한다. 가상화폐 상거래 활성화의 장애물 중 하나는 가상화폐의 높은 변동성이다; 다수의 유저들과 상인들은 가상화폐 혹은 블록체인자산이 제공하는 보안성과 편의성에 대한 니즈가 있지만, 단 하루만의 그들의 자산가치가 23% 하락할지도 모른다는 리스크는 피하고 싶어한다. 이 문제에 대한 지금까지의 가장 보편적인 솔루션은 자산 발행자가 자산에 대한 보증을 서는 것이었다: 이는 곧, 발행자가 하위화폐를 만들어서 그를 통해서 통화량을 조절할 수 있는 권한을 갖고, 누군가가 일정 단위의 하위화폐를 지불하였을 때 그에 상응하는 특정한 베이스 자산 (예, USD, 금)으로 교환해주는 방식을 뜻한다. 이 방식을 본 사례에 적용한다면, 가상화폐 발행자는 가상화폐를 지불하는 자에게 그에 상응하는 베이스자산을 제공할 것이라고 공개적인 약속을 하는 것이다. 이 메커니즘은 비(非)가상화폐 혹은 비(非)디지털자산을 블록체인 자산화(化)자산화 시키는 결과를 낳는다—물론 가상화폐 발행자를 신뢰할 수 있다면 말이다.

다만, 현실적으로는 자산 발행인을 언제나 신뢰를 할 수 없으며, 몇몇 사례를 보면, 우리의 금융인프라는 자산보증 서비스가 존재하기에는 너무 취약하거나, 때로는 적대적이기도 하다. 파생상품은 이에 대한 대안을 제공해준다. 여기서 자산을 보증하기 위한 펀드를 제공하는 역활을 하나의 자산 발행자가 하는 것이 아니라 암호화 담보자산 (cryptographic reference asset, 예: 이더)의 가격이 올라갈 것이라는 데에 베팅을 하는 투자자들 (speculators)의 탈중앙화된 시장이 그 역할을 담당하게 된다. 파생상품을 통한 보증 또한 완전하게 탈중앙화된 방법론은 아니라는 것을 주의하기 바란다. 비록 자산 발행자을 통한 방법 보다 진입장벽(영업허가증 등이 필요 없음)이 없고 사기/조작 가능성이 줄어들기는 하지만, 신뢰성있는 제 3기관이 USD/ETH 시세 또는 환율을 제공해야 하기 때문이다.

신원조회 / 평판 시스템(Identity and Reputation Systems)

최초의 알트코인(비트코인 이후에 생겨난 가상화폐)인 네임코인은 비트코인과 유사한 블록체인을 이용하여 사용자가 공공DB에 다른 데이터와 함께 본인의 이름을 등록하는 명의등록 시스템을 만들어냈다. 이의 주된 사용례는 “bitcoin.org”와 (Namecoin의 경우에는 “bitcoin.bit”) 도메인명을 매핑하는 DNS 시스템이다. 다른 사용례에는 이메일 인증, 그리고 보다 진일보된 평판 시스템 등이 있다. 이더리움에서 네임코인과 같은 명의등록 시스템의 기본적인 컨트랙트는 아래와 같은 형태를 띈다.

```
def register(name, value):
    if !self.storage[name]:
        self.storage[name] = value
```

이 컨트랙트는 매우 단순하게도, 이더리움 네트워크 안에서 저장되어 있는, 추가할 수는 있지만 수정하거나 지울 수 없는 데이터 베이스일 뿐이다. 누구든지 소량의 이더를 이용하여 본인의 명의를 등록할 수 있으며, 한 번 등록하면 영구적으로 보존된다. 보다 정교한 명의등록 컨트랙트는 다른 컨트랙트가 보내는 쿼리에 반응할 수 있는 함수조건이 걸려 있을 것으며, 명의 소유자 (곧, 최초 등록자)가 데이터를 변경하거나 명의소유권을 이전할 수 있는 메커니즘이 장착되어 있을 것이다. 혹자는 평판이나 인터넷신용도 기능등을 그 위에 추가할 수도 있다.

분산형 파일 저장소(Decentralized File Storage)

지난 몇년 동안, 드롭박스와 같은 웹 상에 파일을 저장시켜주는 인기있는 스타트업이 다수 생겨났다 (월 정액에 유저들이 하드드라이브를 백업 시켜 놓고 백업파일에 액세스 할 수 있는 비즈니스 모델임). 그러나, 현시점에서 파일저장 시장은 종종 상대적으로 비효율적일 때가 많다. 현재 존재하는 솔루션들의 월정액 가격을 보면 (특히 무료 할당량도 기업 할인도 없는 20-200기가바이트 수준의 기업이 지불하는 월정액), 한달만 써도 전체 하드드라이브의 비용보다 더 비쌀 정도이다. 이더리움의 컨트랙트는 분산형 파일 저장소 생태계의 발전을 가능케한다. 이 생태계에서 유저 개개인은 본인의 하드드라이브를 대여해주는 대가로 소액의 돈을 받을 수 있으며 남는 하드디스크

공간은 파일저장의 비용을 더욱 낮추는 결과를 낳을 것이다. 분산형 파일 저장소의 핵심 기반은, 소위 “분산형 드롭박스 컨트랙트”가 될 것이다. 이 컨트랙트는 다음과 같이 작동한다: 1) 유저가 업로드하려는 데이터를 블록으로 잘라내고, 2) 프라이버시를 위해 해당 데이터를 암호화 시킨 후, 3) 그 데이터로 머클트리를 만든다. 위 데이터에 대한 컨트랙트는 아래와 같은 룰에 의해서 유지된다.

- N개의 블록 마다 무작위 방식으로 (컨트랙트 코드로 접근가능한 전 블록의 해쉬에 기반한 무작위 방식) 머클트리의 인덱스를 뽑는다.
- 유저가 올린 파일에 해당하는 트리의 특정 인덱스에 대하여, 해당 데이터를 저장해주겠다는 첫 주체에게 (간소화된 지불증명이자 소유권증명의 의미를 띄는) X 이더를 지불한다.

파일을 올린 유저가 다시 자신의 파일을 다운로드 하고 싶을 때에는, 소액결제 채널 프로토콜(예, 32킬로바이트에 1 szabo를 지불한다)을 사용해서 파일을 복원할 수 있다; 수수료 측면에서 가장 효율적인 접근방법은 파일을 업로드한 유저가 저장이 끝나는 마지막까지 파일에 대한 트랜스액션을 공표하지 않고, 매 32 킬로바이트 마다 동일한 Nonce를 갖고 있는 보다 수익성이 있는 트랜스액션으로 바꿔주는 방법이 있다.

비록 이 방식은 파일을 업로드한 유저가 다수의 랜덤한 노드들이 나의 파일을 계속 저장하고 있을 것이라고 믿어야 한다는 것을 전제하는 것 처럼 보이지만, 실제로는 유저는 업로드한 파일을 수많은 암호화된 조각으로 잘라내서 여러 노드들과 공유하고 또 컨트랙트를 통해서 외부 노드들이 내가 올린 파일을 저장하고 있다는 것을 모니터링함으로써 내가 올린 파일에 대한 분실 혹은 제 3자에의한 도용이라는 리스크를 거의 0에 가깝게 줄일 수 있다는 것이 분산형 드롭박스 컨트랙트 중요한 특징이다. 컨트랙트가 계속 돈을 지불하고 있다는 것은 곧 네트워크 상에서 누군가는 파일을 저장하고 있다는 것을 증명한다.

탈중앙화된 자율조직(Decentralized Autonomous Organizations)

“탈중앙화된 자율조직”의 기본적인 개념은특정한 집합의 구성원 또는 주주들을 갖고 있는 가상 독립체(virtual entity)가 필요한 수만큼의 구성원의 동의하에(예, 67% 다수) 조직자금운용 권한 및 코드 변경 권한을 갖는다는 것이다. 구성원들은 그 조직이 어떻게 운영자금을 배분할지를 공동으로 결정할 것이다. DAO의 자금을 배분하는 방식은 포상, 급여 형식부터 보다 색다른 내부화폐로 보상하는 형식까지 다양하다. 이것은 본질적으로 통상적인 기업이나 비영리재단에서 사용하는 법적인 장치들을 그대로 따르는 것이지만, 그 집행의 강제(enforcement)를 위해 암호화 블록체인 기술을 사용한다는 점이 차별점이다. 지금까지의 DAO에 대한 논의는 주로 "자본주의적(capitalist)" 모델인 "탈중앙화된 자율기업(decentralized autonomous corporation, DAC)"에 관한 것이었는데, 이 DAC는 배당을 받는 주주들과 매매가능한 지분을 가지고 있다. 이것에 대한 대안적인 형태로 "탈중앙화된 자율 커뮤니티(decentralized autonomous community)" 같은 개념도 생각해 볼 수 있는데, 이 안에서 구성원들은 의사결정에 있어서 모두 동일한 지분을 갖고있으며, 기존 구성원의 67%의 표결을 통한 동의가 있을 때 구성원을 충원하거나 탈퇴시킬 수 있을 것이다.그렇다면, 한사람이 오직 하나의 멤버십만을 가져야한다는 요건이 그 그룹에 의해 공동으로 시행될 필요가 있을 것이다. DAO 코딩에 관한 일반적인 개요는 다음과 같다. 가장 간단한 디자인은 단순하게도 구성원 2/3가 동의/거부하였을 시 저절로 코드가 변경되는 컨셉이다. 비록 이론적으로 한 번 세팅된 코드는 바뀔수 없어도, 별도의 코드들을 각각 다른 컨트랙트들로 분리시켜서, 이것을 변경가능한 저장공간에 각각 넣어둔 다음, 이 코드들을 불러낼 수 있는 주소들을 제공함으로써 우리는 실제적으로 코드가 변경된 것과 같은 효과를 만들 수 있다. 아주 간단한 DAO 컨트랙에는 3가지 종류의 트랙잭션들이 있을 수 있는데, 그 구분은 그 트랜잭션이 제공하는 데이터의 종류에 따른다:

- [0,i,K,V]는 저장공간 인덱스 k 에 있는 주소를 v 값으로 바꾸라는 인덱스 i 를 가진 제안을 등록
- [0,i]는 제안 i 에 찬성하는 투표를 등록
- [2,i] 는 충분한 투표가 이루어 졌을 때 제안 i 를 완결

컨트랙트는 상기 항목들 각각에 대한 조건절을 갖고 있을 것이다. 컨트랙은 모든 오픈스토리지에 일어난 변화들과 누가그 변화들에 대해 투표했는가하는 리스트를 보관유지하게 될 것이다. 컨트랙은 또한 전체 구성원 리스트도 보관한다. 어떤 스토리지 변경이던지 구성원의 2/3 의 투표를 받으면, 마지막으로 확정시키는 트랜잭션이 그 변경을 집행할 수 있게 된다. 이것보다 좀 더 발전된 형태는 내장 투표 기능을 이용해서 트랜잭션을 송신하거나, 구성원을 충원/탈퇴시키거나, 위임 민주주의 (Liquid Democracy 또는 Delegative Democracy)의 투표위임등의 기능도 추가할 수 있을 것이다. 이런 투표위임을 통해 누구에게나 자신을 위해 투표할 수 있도록 위임할 수 있고, 또 이 권한은 다른 사람에게 다시 전가가 될 수도 있다. A가 B에게 위임하고, B는 C에게 위임하면, C가 A의 투표를 결정한다. 이러한 설계를 통해서, DAO는 탈중앙화된 커뮤니티로 유기적으로 성장할 수 있으며, 더 나아가 누가 구성원인지 아닌지를 판단하는 기능을 전문가들에게 위임 할 수도 있도록 해줄 것이다. (물론 "현행시스템"과 달리, 각 커뮤니티 구성원들의 의견이 바뀜에 따라, 그러한 전문가들은 있을 수도/없을 수도 있게 된다.)

이것과 비교되는 다른 모델은 탈중앙화된 기업이라고 할 수 있는데, 여기에서 각 어카운트는 0 또는 그 이상의 지분을 가질 수 있고, 어떤 결정을 내리기 위해서는 지분의 2/3 가 필요하다. 그것의 가장 단순화된 핵심 골격은 자산 관리 기능, 지분을 매매할 수 있는 오퍼를 낼 수 있는 능력, 그리고 다른 오퍼들을 수락할 수 있는(아마도 컨트랙트내에 있는 주문매칭 메커니즘을 통해) 능력들을 포함하게 될 것이다. "이사회" 개념을 일반화하는 유동식 민주주의(Liquid Democracy) 스타일의 위임제도 또한 있게 될 것이다.

추가적인 어플리케이션들(Further Applications)

1. **예금용 "전자지갑"**. 펀드를 안전하게 보관하고 싶은 A가 펀드를 잃어버리거나 누군가에게 그녀의 Private key를 해킹당할 것을 걱정한다고 가정해 보자. 그녀는 이더를 B라는 은행과의 컨트랙트에 다음과 같은 방식으로 집어 넣을 것이다.

- A만이 하루에 그녀가 소유하는 펀드의 최대 1%를 출금할 수 있다.
- B또한 하루에 A가 소유하는 펀드의 최대 1%를 출금할 수 있지만, A는 그녀의 Private key를 통해 트랜스액션을 발송함으로써 B의 출금 권한을 없애버릴 수 있다.
- A와 B는 함께 어떤 금액도 출금할 수 있다.

일반적으로 하루의 1%라는 상한선은 A에게 충분하며, 그 이상의 금액을 출금하고 싶을 시 A는 B에게 상한 조정 허가를 요청할 수 있다. Alice의 Private key가 해킹 당하였을 경우, B에게 펀드를 새로운 컨트랙트로 이체 시키라고 요청할수 있다. A가 본인의 Private key를 분실하는 경우, B는 오랜 시간에 걸쳐서라도 펀드의 금액을 출금할 수 있다. B가 악당인 경우에는 A는 B의 출금 권한을 정지시킬 수 있다.

2. **작물보험**. 시세가 아닌 날씨 데이터피드를 이용해서 파생상품을 손쉽게 만들 수 있다. 아이오와주에 있는 농부가 강수량 데이터와 역비례하게 지불금이 산출되는 파생상품을 산다면, 가뭄이 있을 시 농부는 자동적으로 보상을 받을 수 있을 것이다. 이러한 어플리케이션은 자연재해 일반에 대한 보험상품으로 확대될 수 있을 것이다.
3. **탈중앙화된 데이터피드**. "쉘링코인 (SchellingCoin)"이라는 프로토콜을 사용하여, 변량 (Difference)을 다루는 금융계약(예, 로또)을 탈중앙화된 방식으로 운용할 수 있다. 쉘링코인은 다음과 같이 작동한다: 하나의 주어진 기준치(예, ETH/USD 시세)에 대해 N명의 참여자가 각각 자신들이 맞다고 생각하는 값을 시스템에 제공한다. 이러한 값들은 그 값의 크기에 따라 순위가 매겨지며, 이 때 25번째 퍼센타일과 75번째 퍼센타일 사이에 있는 값을 제시한 사람은 토큰 1개를 보상으로 받게 된다. 이렇게 되면 보상을 받기 위해서 모든 참가자들은 다른 사람들이 제시했을 똑같은 값을 제시하고자 할 것이고, 많은 다수의 참여자가 현실적으로 동의할 수 있는 유일한 값은 결국 명백한 기본값인 참값(truth)이 될 것이다. 이것은 ETH/USD 가격, 베를린의 온도, 심지어는 어려운 연산문제의 결과값까지도 포함하는, 어떤 수의 값들도 이론적으로 제공해줄 있는 탈중앙화된 프로토콜을 만들 수 있게 해준다.
4. **스마트 멀티시그 공탁 계좌**. 비트코인은 멀티시그 트랜잭션을 만들 수 있게 해주는데, 이는, 가령, 5개의 키 중 3개를 갖고 서명해야 출금을 허용하는 방식의 트랜잭션을 지칭한다. 이더리움은 보다 높은 세밀도를 제공한다. 예를 들어, 5개 중 4개가 있으면 기금은 전체를 사용할 수 있고, 5개 중 3개가 있으면 하루에 기금의

10%을 사용할 수 있고, 2개가 있으면 하루에 0.5%를 사용할 수 있다. 추가적으로, 이더리움의 멀티시그는 동시에 집행해야 할 필요가 없다. 즉, 두 주체는 다른 시기에 블록체인에 본인의 전자 서명을 등록할 수 있고 최종의 전자서명이 이루어졌을 시 자동적으로 트랜잭션이 네트워크로 보내진다.

5. **클라우드 컴퓨팅**. EVM 기술을 사용하여 입증 가능한 컴퓨팅 환경을 만들 수 있다. 입증 가능한 컴퓨팅 환경의 예시는 다음과 같다: 한 유저가 다른 유저에게 연산을 수행하게 한 후 랜덤한 시점에 연산을 수행한 주체에게 미리 설정된 연산 체크포인트가 올바른지에 대한 증명을 요구할 수 있다. 이 기술은 누구든지 자신의 데스크탑, 노트북, 또는 전문화된 서버를 갖고 참여할 수 있는 클라우드컴퓨팅 시장을 창조하게 될 것이며, 함께 연산의 정확성을 무작위 추출검사를 함으로써 시스템의 신뢰성이 더욱 강화될 것이다 (즉, 노드들이 이익을 내기 위해서는 유저들을 속일 수 없게 된다). 물론 그러한 시스템은 모든 작업들을 수행하는 데에 적합한 것은 아니다; 예를 들어, 높은 수준의 프로세스간 커뮤니케이션이 필요한 작업 같은 경우 여러 개의 클라우드 노드들로 수행하기에는 적합하지 않다. 하지만 그 외 작업들은 보다 수월하게 병렬진행이 가능하다; SETI@home, folding@home, 유전자 알고리즘 같은 경우는 분산화된 클라우드 컴퓨팅 플랫폼에서 쉽게 작업할 수 있는 프로젝트들이다.
6. **P2P 도박**. Frank Stajano나 Richard Clayton의 Cyberdice 같은 P2P 도박 프로토콜들은 모두 이더리움의 블록체인 위에 구현될 수 있다. 가장 단순한 도박 프로토콜은 다음 블록의 해시값의 차이에 대한 컨트랙트이며, 0에 가까운 수수료와 그 누구도 사기를 칠 수 없는 보다 발전된 프로토콜이 그 위에 얹혀질 수 있다.
7. **예측 시장**. 오라클(Oracle) 혹은 쉘링코인(Schelling Coin) 등을 갖고, Robin Hanson이 주창한 것과 같은 예측 시장도 쉽게 구현할 수 있다. 쉘링코인과 함께 예측시장은 분권화된 조직들에 대한 거버넌스 프로토콜로서 "퓨타키(Futarchy의 첫 번째 주류 어플리케이션이 될 수 있다.
8. **블록체인상의 탈중앙화된 장터**. 신원조회 / 평판 시스템을 기반으로 원활하게 돌아가는 P2P 장터를 구축할 수 있다.

그 밖의 이슈들

수정된 GHOST 도입(Modified GHOST Implementation)

GHOST(Greedy Heaviest Observed Subtree)프로토콜은 Yonatan Sompolinsky and Aviv Zohar 에 의해 2013년 12월에 처음 소개된 혁신이다. GHOST의 문제의식은, 현재 빠른 확인시간(confirmation times)을 가지고 있는 블록체인들이 높은 스테일(stale) 비율로 인해 보안성 저하라는 문제를 겪고 있다는 것인데, 이는 블록들이 네트워크를 통해 전파되는데 일정한 시간이 걸리기 때문이라는 것이다. 만일 채굴자 A가 하나의 블록을 채굴했는데, 이 블록이 채굴자 B에게 전파되기전에 채굴자 B가 다른 또 하나의 블록을 채굴했다고 하면, 채굴자 B의 블록은 결국 낭비될 것이고, 네트워크 보안에 기여하지 못하게 될 것이다.

게다가 중앙집중화(centralization) 이슈도 있다; 만일 채굴자 A가 30%의 해시파워를, 그리고 B가 10%의 해시파워를 가지고 있다면, A가 스테일 블록을 생산할 위험성은 매번 70%가 될 것이고(왜냐하면 다른 30%의 경우에는 A가 마지막 블록을 만들게 되었고, 따라서 즉각적으로 채굴데이터를 가지게 되기 때문이다), 반면 B는 매번 90%의 경우에 스테일 블록을 생산하게 될 위험성을 가지고 있다. 따라서 만일 블록 주기가 스테일 비율이 높은 것에 필요한 만큼 충분히 짧다면, A는 단순히 크기가 크다라는 사실 자체만으로 훨씬 더 높은 효율성을 가지게 된다. 이러한 두가지 효과가 결합되어서, 블록주기가 짧은 블록체인에서는, 높은 해시파워 점유율을 가진 단일한 풀이 채굴과정에 대한 사실상의 통제권을 가지게 될 가능성이 매우 높아진다.

Sompolinsky와 Zohar가 설명했듯이, GHOST는 어느 체인이 "가장 긴(longest)"것인지 계산할 때 스테일 블록도 포함으로써 위에서 제기한 첫번째 이슈, 즉 네트워크 보안 손실이라는 문제를 해결한다. 다시 말해서 어느 블록이 가장 큰 전체 작업증명을 가지고 있는지 계산함에 있어서, 그 블록의 모블록(parent)과 그 조상(ancestors) 뿐만 아니라, 그 블록의 스테일 자손(stale descendants, 이더리움의 용어로는 "삼촌")까지도 더한다는 것이다. 중앙화라는 두번째 문제를 해결하기 위해서 우리는 Sompolinsky와 Zohar가 설명한 프로토콜을 넘어서서, 스테일 블록들에 대해서도 블록보상을 제공한다. 스테일 블록도 기본 보상의 87.5%를 받게 되며, 그 스테일 블록을

포함하고 있는 사촌이 나머지 12.5%를 받는다. 하지만 수수료는 삼촌들에게는 주어지지 않는다.

이더리움은 7단계 레벨만 포함하는 단순화된 GHOST 버전을 구현한다. 그것은 다음과 같이 구체적으로 정의된다;

- 하나의 블록은 반드시 하나의 모블록을 지정해야 하며, 0 또는 그 이상의 삼촌을 지정해야 한다.
- 블록 B에 포함된 삼촌은 다음과 같은 속성들을 가지고 있어야 한다.
  - B의 k번째 조상의 직접적인 자손이어야 한다. 여기서 2 <= k <= 7.
  - B의 조상이어서는 안된다.
  - 유효한 블록 헤더여야 하지만, 이전에 확인되었을 필요도, 또는 심지어 유효한 블록일 필요도 없다.
  - 이전 블록들에 포함된 모든 삼촌들, 그리고 같은 블록에 포함된 모든 다른 삼촌들과는 달라야 한다 (중복포함방지)
- 블록 B에 있는 각 삼촌 U에 대해, B의 채굴자는 코인베이스 보상에 더해 추가로 3.125%를 더 받고, U의 채굴자는 기본 코인베이스 보상의 93.75%를 받는다.

단지 최대 7세대만 삼촌을 포함할 수 있는 제한된 GHOST 버전을 사용하는 이유는 두가지이다. 첫째, 무제한 GHOST는 하나의 블록에 대해 어떤 삼촌이 유효한지에 대한 계산을 매우 복잡하게 만든다.. 둘째, 만일 이더리움과 같은 방식의 보상을 하면서도 무제한 GHOST를 적용하게 되면 채굴자들이 공격자의 체인이 아니라 주체인(mainchain)에서 채굴를 할 동기를 잃게 될 것이다.

수수료

블록체인에 올려지는 각 트랜잭션은 그것을 다운로드하고 검증하기 위한 비용을 네트워크에 부과하기 때문에, 남용을 방지하는 어떠한 규제 메커니즘, 일반적으로는 트랜잭션 수수료가 필요하게 된다. 비트코인에서 사용되는 기본적인 접근방법은 순수하게 자발적인 수수료를 징수하면서, 채굴자들이 게이트키퍼(gatekeeper)로서의 역할을 하고 유동적으로 최저액을 설정하도록 하는 것이다. 이런 접근방법은 비트코인 커뮤니티에서 매우 환영 받아왔는데, 그것이 "시장-기반"이기 때문에, 채굴자와 트랜잭션 송신자들간의 수요와 공급이 그 가격을 결정한다는 이유에서였다.

하지만 이런식의 사고방식에는 문제가 있는데, 트랜잭션 처리는 시장에서 일어나는 것이 아니라는 점이다. 트랜잭션 처리를 채굴자가 송신자에 제공하는 하나의 서비스로 해석하는 것이 직관적으로 솔깃해 보이기는 하지만, 실제적으로는 채굴자가 포함하는 모든 트랜잭션들은 네트워크의 모든 노드들에 의해 처리되어야 하고, 따라서 트랜재션처리에 필요한 대부분의 비용은 제3자가 부담하는 것이지, 그 트랙재션을 포함할지 말지를 결정하는 채굴자들이 아니라는 것이다. 그러므로 공유지의 비극(tragedy-of-the-commons) 문제들이 매우 일어나기 쉽다는 것이다.

하지만, 이러한 시장기반 메커니즘의 결함은 어떤 부정확한 단순화 전제들이 주워졌을 때, 마술처럼 그 결함자체를 상쇄하게 된다. 그 주장은 다음과 같다. 다음을 전제해 보자:

1. 하나의 트랜잭션이 `k`개의 작업들(operations)을 초래하는데, 이 트랙잭션을 포함하는 채굴자에게 `kR`만큼의 보상을 제공하게 된다. 여기서 `R`은 송신자에 의해서 설정되고, `k` 와 `R`은 (대략적으로) 채굴자에게 사전에 노출된다.
2. 하나의 작업은 어떤 노드에 대해서든 C 만큼의 처리비용을 가진다(즉, 모든 노드들은 똑같은 효율성을 가지고 있다).
3. `N`개의 채굴노드들이 있고, 각각은 정확히 똑같은 처리파워(즉, 전체의 `1/N`)를 가지고 있다.
4. 채굴을 하지 않는 완전노드(full nodes)는 없다.

채굴자는 어떠한 트랜잭션이 그 비용보다 기대보상이 클 경우 처리하려고 할 것이다. 따라서, 기대 보상은 kR/N인데, 왜냐하면 채굴자는 다음번 블록을 처리할 1/N 확률을 가지고 있으며, 이 채굴자에게 처리비용은 단순히 kC이다. 그러므로 채굴자들은 kR/N > kC 이거나, R > NC일때 트랜잭션들을 포함하려 할 것이다. 여기서 R은 송신자에 의해 제공된 단위작업(pre-operation)당 수수료이고, 따라서 이것은 송신자가 그 트랜잭션에서 보게 될 혜택에 대한 하한값이 되고, NC는 하나의 작업을 처리하기 위해 전체 네트워크에 부과된 비용임을 주목하자. 따라서 채굴자들은 비용보다 전체 공리적인 혜택이 큰 트랜잭션들만 포함하려 하는 인센티브를 갖게 된다.

하지만 현실에서는 이러한 가정들이 맞지 않는 몇가지 중요한 차이들이 있다.

1. 채굴자는 다른 검증 노드들보다 트랜잭션을 처리하는데 더 많은 비용을 지불하게 되는데, 왜냐하면, 추가적인 검증시간은 블록전파를 지연시키고, 따라서 블록이 스테일되는 확률을 증가시키기 때문이다.
2. 비채굴 완전노드(full note)들이 존재한다.
3. 채굴 파워의 분포는 실제로 심각하게 불평등하게 될 수 있다.
4. 네트워크에 피해를 주는 이해관계를 가진 투기자들, 정치적 적, 그리고 일탈자들이 존재하고, 그들은 다른 검증노드가 지불하는 비용보다 훨씬 적은 비용이 들게 될 그런 컨트랙트들을 교묘하게 만들 수 있다.

(1)은 채굴자가 더 적은 수의 트랜잭션들을 포함하게 되는 경향을 제공하게 되고, (2)는 NC를 증가시키게 되며, 따라서 이 두가지의 효과들은 부분적으로는 서로를 상쇄한다. (3)과 (4)가 주요한 문제인데, 이것들을 해결하기 위해, 플로팅 상한값(floating cap)을 도입한다. 어떤 블록이던지 BLK_LIMIT_FACTOR 곱하기 장기 지수 이동평균(the long-term exponential moving average)보다 더 많은 오퍼레이션들을 가질 수 없다는 것이다. 정확히는:

```
blk.oplimit = floor((blk.parent.oplimit * (EMAFACTOR - 1) +
floor(parent.opcount * BLK_LIMIT_FACTOR)) / EMA_FACTOR)
```

BLK_LIMIT_FACTOR 와 EMA_FACTOR은 상수이며 각각 잠정적으로 65536와 1.5로 정해질 것이지만, 추후 분석 후에 바뀔 가능성이 많다.

비트코인에 있어서 큰 블록크기를 막는 또 다른 요인도 있다. 큰 블록이 전파되는데에 더 오래 걸리기 때문에, 스테일 될 가능성이 높다는 점이다. 이더리움에서도 높은 가스(GAS)사용 블록은 전파되는데 더 오래걸리는데, 그것은 크기가 물리적으로 크다는 점과, 트랜잭션 상태변환들(state transitions)을 검증 처리하는데 더 오래 걸린다는 점 때문에 그러하다. 이러한 지연 불이익(delay disincentive)은 비트코인의 경우에는 중요한 고려사항이지만, 이더리움의 경우에는 GHOST프로토콜 덕분에 중요도가 낮아진다. 따라서 조정된 블록리미트(block limit)로 인해, 보다 안정적인 기본기준(baseline)을 얻을 수 있게 된다.

연산과 튜링완전성(Computation And Turing-Completeness)

중요한 점은 이더리움 가상 머신(EVM)이 튜링-완전하다는 것이다. 즉 EVM은 무한 순환을 포함한 상상가능한 모든 계산 수행을 코딩할 수 있다. EVM코드는 순환 계산을 다음 두 가지 방법으로 수행한다. 첫번째는 JUMP 명령어로 코드의 이전 장소로 되돌아가고, JUMPI 명령어로 while x < 27: x = x * 2 같은 문장처럼 조건에 따라 건너뛰게 하는 것이다. 두번째는 한 계약이 재귀 반복을 통해 순환을 일으킬 가능성이 있는 다른 계약을 호출하는 것이다. 이것은 자연스럽게 어떤 문제를 야기한다: 악의적인 사용자가 계산을 무한 순환에 빠뜨리는 방법으로 채굴자와 풀 노드를 마비시켜버릴 수 있을까? 컴퓨터 학계에서 정지문제(halting problem)라고 알려진 유명한 문제를 통해 이런 이슈를 피할 수 없음을 알 수 있다. 일반적으로 어떤 주어진 문제가 궁극적으로 멈추는지 아닌지를 미리 판별할 방법은 없다. 상태변환 과정에서 설명했듯이, 한 거래에 최대로 계산할 수 있는 단계 수를 설정함으로써 우리는 해답을 얻을 수 있다. 만약 계산 단계가 그 최대수보다 더 많으면 계산은 원점으로 돌아가지만 수수료는 그대로 지불된다 메시지들도 같은 방법으로 작동한다. 우리가 제시한 해답의 의미를 더 잘 이해하기 위해, 아래와 같은 몇가지 보기를 생각해보자.

- 한 악의적 공격자가 무한 순환을 실행하는 계약을 만들어 채굴자로 하여금 무한 순환을 실행하도록 거래를 보냈다고 하자. 채굴자는 거래를 진행하고 무한 순환을 실행해 가스를 다 소모해서 실행 도중에 멈춘다고 하더라도, 거래는 여전히 유효하고 채굴자는 여전히 공격자에게 이미 실행된 각 계산 단계마다의 수수료를 요구할 수 있다.
- 한 악의적 공격자가 채굴자에게 계산을 오랫동안 계속하게 할 목적으로 아주 긴 무한 순환 프로그램을 짰다고 하자. 계산이 끝났을 때 몇몇 블럭이 추가로 생성되어 채굴자가 수수료를 요구하기 위해 그 거래를 포함하는게 불가능하게 만드는 게 악의적 공격자의 목적이다. 하지만, 그 공격자는 실제 실행되는 계산 단계의 상한선을 규정하는 STARTGAS 명령어에 대한 값을 제출해야만 하고, 따라서 채굴자는 해당 계산이 과도하게 많은 단계의 수를 필요로 한다는 것을 계산 전에 미리 알게 된다.
- 예를 들어send(A,contract.storage[A]); contract.storage[A] = 0, 같은 명령이 들어간 계약이 있다고 하자. 한 악의적 공격자가 이 계약을 본 후 첫번째 계산 단계만 실행시키고 두번째 단계는 실행할 수 없을 만큼의(예를 들어 예금 인출만 한 다음 장부에 기록되는 스텝은 실행되지 않게) 가스만 넣고 거래를 진행시켰다고 하자. 계약 작성자는 이런 공격에 대해 방어를 걱정할 필요가 없다. 왜냐하면 계산 실행이 도중에 멈추면, 해당 변화도 원상복구되기 때문이다.
- 어떤 금융 계약이 9개의 금융상품 자료값의 평균을 취해 위험을 최소화하도록 작동하고 있다고 하자. 그 중 DAOs 섹션에서 설명된 것 같은 가변주소요청 메커니즘을 통해 변경가능하도록 디자인 된 하나의 자료값을 악의적 공격자가 취한다고 하자. 그렇게 함으로써 이 금융 계약으로부터 펀드를 찾으려는 모든 시도에 대해 가스가 다 소모되도록 시도하게 된다. 하지만 금융 계약은 이 문제를 막기 위해 메시지 위에 가스 한도를 설정해 두는 것으로 공격을 방어할 수 있다..

튜링-완전에 대한 대칭적인 개념은 튜링-비완전이다. 즉 JUMP 명령어나 JUMPI 명령어가 존재하지 않으며, 그 어떤 주어진 시간에도 오직 각각의 계약의 복사본 하나만이 허용된다. 이런 시스템 아래에서는 위에 서술된 수수료 시스템이라든지 우리가 제시한 해답의 효율성을 둘러싼 불확실성에 관한 논쟁은 불필요할 것이다. 한 계약을 실행하는데 드는 비용은 프로그램의 크기에 따라 상한선이 정해질 것이기 때문이다. 나아가, 튜링-비완전성은 그리 큰 제한도 아니다. 우리가 현재까지 상상했던 계약 가운데, 순환 명령을 필요로 했던 것은 단 하나 뿐이었다. 그리고 그 순환 명령조차도 프로그램 코딩에서 한 문장을 26번 반복함으로써 없앨수 있었다. 튜링-완전이 함의하고 있는 심각성과 그 제한적인 이점을 생각해볼 때, 왜 튜링-불완전 언어를 쓰면 안되는 걸까? 하지만 현실적으로, 튜링-불완전성은 순환 문제와 악성 공격에 대한 깔끔한 해답이 아니다. 왜 그런지를 알기 위해 아래와 같은 예제 계약을 보자.

```
C0: call(C1); call(C1);
C1: call(C2); call(C2);
C2: call(C3); call(C3);
...
C49: call(C50); call(C50);
C50: (프로그램의 한 단계를 실행한 후 그 변화를 저장소에 기록한다.)
```

이제 A 에게 거래를 보내자. 51번의 거래에서 우리는 2 의 50승의 계산 단계를 계속하는 계약을 보낸다..
채굴자들은 각 계약에 따른 계산 단계의 최대 수와 다른 계약을 재귀적으로 호출하는 계약에 대한 계산 단계 수를 모두 확보함으로써, 이런 논리 폭탄을 사전에 감지하려고 시도할 수 있을지도 모른다. 하지만 이런 시도는 채굴자들이 다른 계약을 호출하는 계약은 다루지 못하게 만든다. (왜냐하면 위의 모든 26개 계약의 작성과 실행은 한 줄의 계약으로 쉽게 합쳐질 수 있기 때문이다.) 다른 문제적 지점은 메시지의 주소 필드는 변수라는 점이다. 그래서 일반적으로, 주어진 계약이 사전에 미리 호출하는 다른 계약이 뭔지를 판별하는 것조차 불가능할지도 모른다. 그래서,결국 우리는 놀라운 결론에 도달한다. 튜링-완전은 놀랍도록 다루기 쉬우며, 만약 튜링완전성이 없으면 정확히 같은 계약으로 대체할 수 없는 한, 마찬가지로 다루기가 놀랍도록 어렵다는 점이다. 그렇다면, 그냥 그 프로토콜을 튜링-완전하게 놔두는게 좋을 것이다.

통화 그리고 발행(Currency and Issuance)

이더리움(Ethereum) 네트워크는 그 안에서 자체적으로 통용되는, '이더(Ether)'라는 화폐를 가지고 있다. 이더는 여러가지 가상자산들간의 효율적인 교환을 가능케하는 매개물의 역할을 하며, 또한 트랜잭션 수수료(transaction fee)를 지불하기 위한 방법을 제공한다. 사용자의 편의와 향후 있을 지 모르는 논쟁을 예방하는 차원에서, 이더(Ether)의 각 단위에 대한 명칭은 다음과 같이 미리 정해졌다. (비트코인 명칭과 관련하여 벌어지는 논쟁 참조)

- 1: wei
- 10^{12}: szabo
- 10^{15}: finney
- 10^{18}: ether

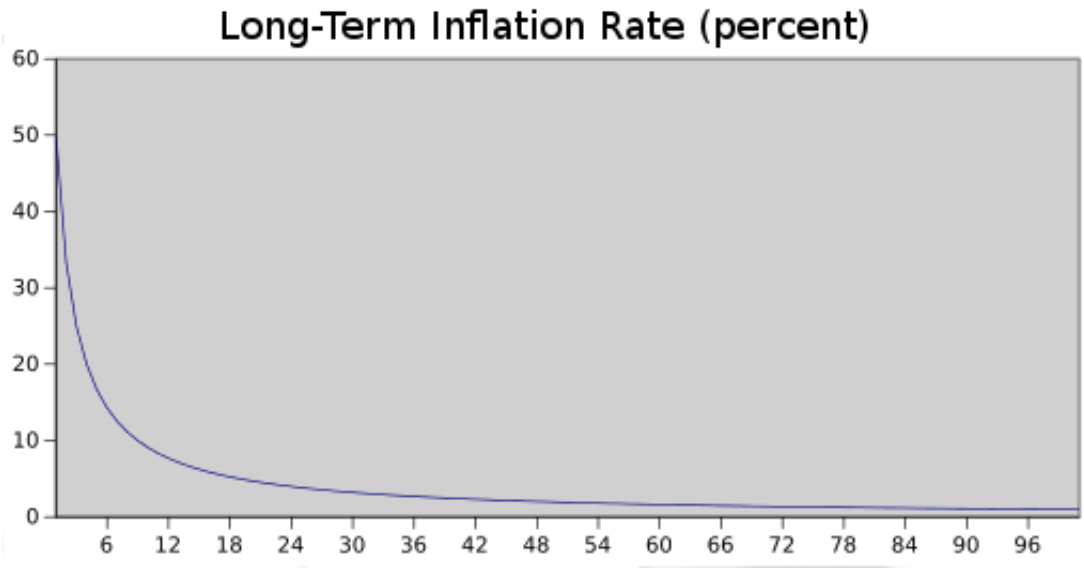
위 명칭들은, 미화 명칭인 "달러"와 "센트" 또는 비트코인의 "BTC"와 "사토시" 등의 확장개념으로 생각하면 이해하는데 도움이 될 것이다. 가까운 미래에, "이더(ether)"는 일반 거래(transaction)를 위해, "피니(finney)"는 소액결제를 위해, 그리고 "싸보(szabo)"와 "웨이(wei)"가 수수료나 프로토콜 도입 등과 관련된 기술적논의를 위해 사용될 것으로 기대된다. 나머지 명칭들은, 지금 당장은 클라이언트에 포함시키지 않는다.

화폐발행 모델:

- BTC 당 1000-2000개의 가격으로 이더를 판매한다. Matercoin이나 NXT와 같은 다른 암호화화폐 플랫폼에서 성공적으로 사용했던 방법으로, 이더리움 조직을 금전적으로 지원하고 개발에 필요한 비용을 댄다. 이 시기에 이더를 구매하는 구매자들은 큰 폭의 할인을 통해 저렴하게 이더를 얻게 된다. 이렇게 모인 자금은 전액, 개발자를 위한 월급과 보상, 그리고 여러가지의 이더리움 관련 영리와 비영리프로젝트를 위한 투자금으로써 사용된다.
- 판매 된 총 이더(60,102,216 ETH)의 0.099배 만큼(5,950,110)의 이더가 신규발행 되어, 이더리움 런칭 전의 초기기여자들과 '이더로 이미 발생 된 비용에 대한 미지급금'을 처리하기 위해 이더리움조직(Ethereum organization)에게 분배된다.
- 또 다른 0.099배 만큼의 이더는 장기보유금으로 신규 발행하여 적립해둔다.
- 채굴시점 이후부터 영구히 매년, 총 판매수량(60,102,216 ETH)의 0.26배 만큼(15,626,576)씩을 채굴자에게 신규 발행해준다.

분류	런칭시	1년후	5년후
초기판매 이더총량에 대한 배수	1.198X	1.458X	2.498X
구매자	83.5%	68.6%	40.0%
런칭전 미지급 적립금	8.26%	6.79%	3.96%
런칭후 적립급	8.26%	6.79%	3.96%
채굴자 채굴량	0%	17.8%	52.0%

** 이더 장기 공급 성장률(%)**



매년 신규발행량이 일정함에도 불구하고, 비트코인이 그러한 것처럼, 발행된 총 이더에 대한 신규 이더의 발행률은 그 비중이 0을 향하여 계속 줄어들게 된다.

위 모델에서 결정되어야할 두 가지 선택이 있다. (1) 하나는, '재단보유금(endowment pool)'의 존재유무와 그 규모이며, (2) 둘째는 총 발행코인량이 정해져 있는 비트코인과는 달리, 신규코인을 끊임없이 발행해야 하는지의 여부이다.

'재단보유금(endowment pool)'의 정당성에 대해서는 다음과 같이 설명할 수 있다. 만일 이러한 보유금이 없는 상황이라면, 같은 인플레이션율을 유지하기 위해서는 연간 발행량이 26%가 아닌, 21.7%로 줄어들어야 한다. 그렇게 되면 이더의 총량은 16.5% 줄어들게 되며, 각 이더의 가치는 19.8%증가하게 된다. 이 경우 균형을 위해서는 19.8%의 이더가 더 프리세일에서 판매되어야 한다. 그렇게 되면 각 경우의 이더 가치는 서로 정확히 동일해진다. 그렇게 되면 이더리움 재단이 1.198배의 BTC를 가지게 되는데, 이를 처음 BTC 액수(1배수)와 추가된 0.198배수의 BTC로 나누어보면 결국 상황이 동일해진다는 점을 알 수 있다. 그러나 한 가지 차이점은 이 경우 조직이 가진것은 이더가 아닌 BTC이므로, 이더의 가치를 높이기 위한 인센티브를 얻지 못한다는 점이다.

정해진 양의 이더를 영구적으로 신규발행하는 모델(permanent linear supply growth model)은 비트코인이 겪고있는 '부의 집중현상'을 완화시킬 수 있다. 또한 현재 또는 미래의 참여자들이 계속해서 이더를 시장이 아닌 채굴을 통해 얻을 수 있는 기회를 제공한다. 동시에, "공급성장률(Supply Growth Rate)"은 계속해서 0을 향해 줄어들게 된다. 우리측의 이론으로는 다음과 같은 현상을 예상해 볼 수 있다. 시간이 흐름에 따라 사용자들의 부주의, 죽음 등으로 인해 현실적으로 일부의 이더들이 계속해서 시장에서 사라지게 된다. 이렇게 사라지는 이더로 인해 점점 줄어드는 '시장유통가능 이더총량(the total currency supply in circulation)'은 매년 신규발행되는 이더에 의해 균형을 이루게 된다. (ex. 만일 총 이더량이 26배수(1,562,657,616 ETH)에 달했고, 매년 이 중 1%(0.26 배수)에 해당하는 이더가 소실된다면, 이는 매년 새로이 발행되는 0.26배수의 이더와 균형을 이루게 된다)

장래, 공급성장률을 약 '0에서 0.05배수 이내'가 되도록 수정을 하면서, POS로 채굴모델을 변경할 계획을 가지고 있다. 만일, '이더리움 재단(Ethereum organization)'이 보유금을 모두 잃거나, 또는 여타의 이유로 사라지게 되면, "사회적계약(social contract)"을 열어둘 것이다. 이를 통해, 이더 발행량을 최대 '60102216 * (1.198 + 0.26 * n)'를 넘지 않도록만(n은 첫 블록 생성 이후의 총 년수) 지킨다면, 누구든지 이더리움의 '후속버전(a future candidate version:RC버전)'을 만들 수 있을 것이다. 이 후속버전의 창시자는 개발/관리에 필요한 비용을 충당하기 위해서, 공개판매(crowd-sell)를 하거나, '총 가능 이더발행량'과 'POS를 통한 공급량' 간의 차액 중 일부나 전부를 이용할 수 있을 것이다. 만일 어떠한 창시자가 이러한 "사회적계약(social contract)"에 반하는 내용을 업데이트하게 된다면, 결국 대의에 의해 합당한(compliant) 버전에서 별개로 포크되어(forked)나와 탈락하게 될 것이다.

채굴 중앙집중화(Mining Centralization)

비트코인 채굴 방식은, 목표 값(현재 기준 약 2^{192})보다 낮은 값이 나올 때까지, 블록헤더에 대한 sha256 해싱 작업을 무한정 반복하는 것이다. 하지만 해당 방식에는 두 가지 약점이 존재한다.

첫번째는 현재 채굴참여에 대한 장벽이 매우 높아졌다는 것이다. 현재 채굴생태계는 ASIC(특수목적을 위해 전용으로 설계된 반도체로, 범용반도체에 비해 성능이 뛰어남)에 의해 완전히 잠식되었다. 이러한 ASIC채굴기는 일반 GPU채굴기 등에 비해 수 천배 이상의 효율을 가지는데, 따라서 ASIC이 아닌 일반컴퓨터를 통한 일반사용자들의 채굴행위는 경쟁력에서 밀려 효용을 잃게 되었다. 과거의 채굴행위가 분권화되고 이타적인 참여자 중심의 '생태계'였다면, 현재는 수십억원의 투자가 되어야만 참여가 가능한 재력가들의 '사업'으로 변질되고 말았다.

두번째는 채굴방식이다. 이전처럼 여러 지역에서 여러 참여자가 블록생성에 참여하는 것이 아니라, 중앙집중화된 채굴풀(Mining pool)이 제공하는 블록헤더(block header)에 의존하여 채굴에 참여한다는 점이다. 이로 인한 부작용이 상당한데, 현재 기준으로는, 3개 채굴풀들이 개인들의 컴퓨팅파워를 인계 받아서 무려 50%에 육박하는 해시를 간접적으로 통제하고 있다. 물론 해당 풀의 점유율이 50%를 넘어가기 전에 개인들이 다른 소규모 풀들로 이동을 할 수 있기 때문에, 풀들이 마음대로 자원을 남용할 수는 없겠지만, 이는 여전히 큰 문제이다.

이더리움의 채굴 방식은 조금 다르다. 각 채굴자가 상태정보(the state)에서 무작위의 정보를 가져와서, 무작위로 선택 된 최근 몇개의 블록내역을 해싱 작업하고 결과값을 내놓는 것이다. 이렇게 하게 되면 두가지 이점이 있다.

첫번째는 이더리움 계약이 모든 종류의 컴퓨터 계산방식을 포괄할 수 있다는 점이다. 따라서 자연히 ASIC도 모든 계산방식에 적합하게 설계되어야 하는데, 이렇게 되면 결국 ASIC이라기 보다는 일종의 고성능 CPU가 되는 셈이다. 즉 현실적으로 ASIC(주문형 전용반도체) 자체가 무용지물이 된다.

두번째로, 채굴자들은 작업 시 전체 블록체인을 다운 받아 모든 이체내역을 검증해야 한다는 점이다. 이렇게 되면 중앙집중화 된 대형 풀이 필요없게 된다. 물론 대형풀 자체는 신규블록생성 보상을 균일하게 참여자들에게 배분해 주는 효과가 있긴 하지만, 그러한 효과는 P2P형식의 풀(pool)을 통해서도 충분히 구현이 가능하다. 굳이 중앙집중형 풀(centralized pool) 방식을 사용할 필요가 없다.

물론 위의 채굴 모델이 아직 검증된 것은 아니다. 또한 ASIC장비에 대한 저항성을 높이는 작업도, 이론처럼 현실에서 적용이 될 수 있을지에 대하여는 의문의 여지가 있다. 하지만 한 가지 확실한 것은, 여러종류의 수많은 계약이 적용이 되면, 이를 모두 포괄하는 ASIC을 예전처럼 만들어 내기는 어렵다는 점이다. 또한 어떠한 종류의 작업에 특화 된 ASIC이 존재한다면, 이에 반하는 작업을 요하는 계약이 생성되는 것을 원치 않을 것이다. 그러면 해당 ASIC채굴자의 경쟁자는 그에 적대적인, 즉 비효율적인 작업을 요하는 계약들을 생성해 냄으로써 공격을 가할 것이다. 즉, 각 부분에 특화된 ASIC을 소유한 채굴자들은 서로에게 불리한 작업을 하게하는 계약들을 만들어 냄으로써 서로를 공격할 것이다. 물론 이러한 방법은 '기술적'인 접근이라기보다는 '경제학적 인간행동론'에 근거한 접근에 가깝다.

확장성(Scalability)

이더리움에 대한 한 가지 공통된 의문점은 확장성 부분이다. 비트코인과 마찬가지로 이더리움도 모든 이체작업이 네크워크 상의 전체 노드에 의해서 일일이 검증 및 작업이 되어야 한다는 약점이 있다. 비트코인의 경우, 현재 전체 블록체인의 크기가 약 15GB에 이르며, 그 크기는 매 시간 1MB씩 꾸준히 늘어나고 있다. VISA의 경우 초당 2,000여 건의 이체작업을 처리하는데, 이는 매 3초당 1MB씩의 확장(시간 당 1GB, 매 년 8TB)을 의미한다. 이더리움도 비슷한 문제를 겪을 것이고, 단순히 화폐로서의 역할 만하는 비트코인에 비한다면, 온갖 종류의 탈중앙화된 어플리케이션들(Dapps: Decentralized applications)을 포괄하는 이더리움은 이 부분에서 훨씬 더 많은 문제를 겪을 수도 있을 것이다. 하지만 한 가지 다른 점은, 이더리움은 '전체 블록체인 히스토리'가 아닌, 단지 '상태 정보(the state)'만 가지고 있으면 된다는 점이다.

만일 개개의 모든 노드가 전체 블록체인을 보관해야 한다면, 아래와 같은 문제가 생길 수 있다. 블록체인의 크기가 점점 커져 100TB에 육박하게 되었다고 생각해보자. 이 정도 수준으로 보관해야하는 블록체인의 크기가 커지면, 오직 소수의 사업가나 기업 형태의 참여자만이 이를 감당할 수 있게 된다. 다수의 일반 사용자들은 '라이트 SPV(Simple Payment Verification)' 노드만들 사용하게 될 것이다. 이렇게 되면, 전체 블록체인의 내역을 가진 소수의 참여자들이 결탁하여, 장부내역을 수정하거나 블록보상량을 바꿔치기 하는 등의 조작행위가 일어날 수 있을 것이다. 단순한 '라이트 노드(light node)'로서는 이러한 조작을 감지할 방법이 없다. 물론 '전체 블록체인를 소유한 노드(full node)' 중에서도 선의의 참가자가 있을지 모른다. 그러나 다수의 '완전노드(full node)'가 작심하여 블록체인 조작을 시도한다면, 이를 발견하는 시점에서는 이미 늦었다고 봐야 할 것이다. 실제로 비트코인이 현재 이와 비슷한 문제에 처할 위험이 있다고 경고받고 있으며, 해당 문제를 완화시키는 방법에 대하여는 Peter Todd에 의해 논의된 바 있다.

위의 문제를 해결키 위해, 가까운 시일 안에 두 가지의 전략을 추가로 도입할 예정이다. 첫번째로 이더리움도 기본적으로 블록체인 기술을 바탕으로 한 채굴 알고리즘을 사용하고 있기 때문에, 모든 채굴자들은 '완전노드(full node)'가 되도록 의무화 될 것이며, 이는 필요한 최소한의 완전노드 숫자를 확보할 수 있도록 해줄 것이다 . 두번째로, 이체내역 검증 작업 이후 블록체인에 '중간상태 트리루트(an intermediate state tree root)'를 도입하는 것이다. 이렇게 되면, 아무리 블록생성 작업이 소수의 노드에 집중되더라도, 단 하나의 선의의 노드(honest node)만 존재한다면 검증 프로토콜(verification protocol)을 통해 이 문제를 해결할 수 있다.

만일 어떠한 채굴노드가 전파한 블록이 검증오류(invalid)처리가 되었다면, 해당 블록의 '구성(format)'이 맞지 않거나 '상태내역 S [n]'이 틀린 경우일 것이다. 'S [0]' 상태가 옳은 것으로 간주되기 때문에, 'S[i-1]'이 맞다면, 'S[i]'에 오류가 있는 것이다. 검증작업에 참여하는 노드는, 'APPLY(S[i-1],TX[i]) -> S[i]' 작업(processing)을 하는 '페트리샤 트리 노드의 부분집합(the subset of Patricia tree)'을 통해 '검증오류증명(proof of invalidity)'과 '인덱스 i'를 제공한다. 노드들은, 위의 노드들을 이용해 해당 작업을 수행하며, 생성한 'S[i]'가 제공받은 'S[i]'와 일치하지 않음을 발견하게 된다.

또한 '불완전한 블록(incomplete block)'을 전파하려는 악의의 채굴노드들과 관련된 더욱 정교한 공격이 이루어질 수 있다. 블록을 검증하는데에 필요한 정보가 온전히 존재하지 않을 수도 있다. 이 경우, '질의-응답프로토콜(challenge-response protocol)' 기법이 사용될 수 있다. 검증노드가 '목표 블록의 인덱스 형태(target transaction indices)'로 '질문(challenge)'을 생성하고, 노드를 수신하는 라이트노드(light node)는 해당 블록(challenge)을 일단 검증오류블록으로 취급한다. 이후, 다른 노드(채굴노드이든 검증노드이든)가 '페트리샤 트리 노드의 부분집합(the subset of Patricia tree)'을 검증증명(proof of validity)으로써 제공한다면, 그때서 위의 블록은 검증된(유효한) 것으로 취급된다.

결론

이더리움 프로토콜은 본래 매우 범용적인 프로그래밍 언어를 통해 '블록체인상 에스크로나 인출한도설정, 금전계약, 도박 시장 등의 고급 기능'을 제공하는, 가상화폐의 업그레이드 버전으로 구상되었다. 이더리움 프로토콜은 이러한 어플리케이션들을 직접적으로 제공하는 것이 아니라, 튜링완전언어(Turing-complete programming language)를 통해 이론적으로 거의 모든 형태의 이체방식이나 어플리케이션을 만들어낼 수 있도록 지원한다. 더욱 흥미로운 점은, 이더리움은 단순한 '화폐'의 차원을 훨씬 뛰어넘는다는 점이다. 분산저장공간(DFS:decentralized file storage)이나, 분산컴퓨팅, 분산예측시장(decentralized prediction market) 프로토콜 등은 사실 수많은 응용개념들 중 일부에 불과하다. 이러한 새로운 개념들은 컴퓨팅 산업의 효율성을 폭발적으로 높일 수 있는 잠재력이 있으며, P2P프로토콜에 처음으로 '경제적인 차원(economic layer)'을 입힘으로써 엄청난 혁신을 가져올 수 있을 것이다. 마지막으로, 컴퓨팅이나 금융과 관련이 없는 분야들에서도 다양한 어플리케이션들이 나올 것이다.

이더리움 프로토콜이 제공하는 '임의상태변환(arbitrary state transition function)'이라는 개념은 고유의 잠재력을 지닌 플랫폼을 탄생시킨다. 기존의 자료저장공간이나 도박, 금융 등의 하나의 목적에 특화된 폐쇠형 구조(close-ended)와는 달리, 이더리움은 자유롭게 조정이 가능한 구조(open-ended)이다. 우리는 이것이 몇 년 이내에, 금융부문이든 비금융부문이든 엄청나게 많은 종류의 서비스를 설계할 수 있도록 돕는 것에 특화된 기반이

될 것이라고 믿는다.

주석, 참고문헌 및 추가자료

주석

1. 관찰력이 좋은 독자라면, 비트코인 주소는 '공개키(public key)'가 아니라, '타원곡선공개키의 해시(the hash of the elliptic curve public key)'로 이루어져 있다는 것은 눈치챘을 것이다. 물론, 암호학적 관점에서 보자면 '공개키 해시(public key hash)'로 부르든 단순히 '공개키(public key)'로 부르든 차이는 없다. 왜냐하면, '비트코인 암호기법' 자체가 '일종의 맞춤형 전자서명알고리즘'이고, 이 알고리즘에서는 공개키가 '타원곡선공개키의 해시(the hash of the Elliptic Curve public key)'를 포함하고 있고, 여기서의 '서명(signature)'은 '타원곡선서명(ECC signature)'과 연결된 '타원곡선공개키(ECC public key)'로 구성되어 있기 때문이다. 또한 '검증알고리즘'은 서명(signature) 안의 '타원곡선공개키(ECC public key)'를, 공개키로써 제공 된 '타원곡선공개키해시(the hash of the elliptic curve public key)'와 대조확인하고, 또한 '서명'을 '타원곡선공개키(ECC public key)'와 대조하여 검증하는 것이기 때문이다.
2. 기술적으로는, 이전 11개 블록의 중간값(median)이다.
3. 내부적으로는 2와 "CHARLIE" 모두 숫자이다. 다만 "CHARLIE"는 '빅 엔디언(big-endian)' 기반의 256 비트로 표시한 것이다. 숫자는 0부터 2^{256}-1까지 사용한다.

참고문헌

1a. Nakamoto, S. 31 October 2008. "Bitcoin: A Peer-to-Peer Electronic Cash System". Also known as the Bitcoin whitepaper. http://nakamotoinstitute.org/bitcoin/. http://bitcoin.org/bitcoin.pdf. https://github.com/saivann/bitcoinwhitepaper. Accessed 7 July 2017.
1b. Davis, J. 10 October 2011. "The Crypto-Currency: Bitcoin and its mysterious inventor". The New Yorker. http://www.newyorker.com/magazine/2011/10/10/the-crypto-currency. Retrieved 31 October 2014. Accessed 7 July 2017.

2. Unknown author. Unknown date. "Intrinsic value". http://bitcoinmagazine.com/8640/an-exploration-of-intrinsic-value-what-it-is-why-bitcoin-doesnt-have-it-and-why-bitcoin-does-have-it/. Unable to access 7 July 2017.
3. Assia, Y.; Vitalik, B.; Haki, M.; Meni R.; and Rotem, L. U.d. "Colored coins whitepaper". https://docs.google.com/a/buterin.com/document/d/1AnkP_cVZTCMLIzw4DvsW6M8Q2JC0lIzrTLuoWu2z1BE/edit. Accessed 7 July 2017.
4. Last edited on 10 May 2016. "Smart property". Bitcoin Wiki. https://en.bitcoin.it/wiki/Smart_Property. Accessed 7 July 2017.
5. Last modified 6 July 2017. "Namecoin". https://namecoin.org/. Accessed 7 July 2017.
6. Last edited on 24 June 2017. "Smart contracts". Bitcoin Wiki. https://en.bitcoin.it/wiki/Contracts. Accessed 7 July 2017.
7. Buterin, V. Sep 19, 2013. "Bootstrapping A Decentralized Autonomous Corporation: Part I". Bitcoin Magazine. http://bitcoinmagazine.com/7050/bootstrapping-a-decentralized-autonomous-corporation-part-i. Accessed 7 July 2017.
8. "Blind signature". Last modified 29 March 2017. Wikipedia. https://en.wikipedia.org/wiki/Blind_signature. Accessed 7 July 2017.
9. Dai, W. U.d. "B-money". http://www.weidai.com/bmoney.txt. Accessed 7 July 2017.
10. Hal, F. Reusable proofs of work: http://www.finney.org/~hal/rpow/. Unable to access 7 July 2017.
11. Back, A. U.d. Hashcash. http://www.hashcash.org/. Accessed 7 July 2017.

12. Last edited on 15 June 2017. "Sybil attack". Wikipedia. https://en.wikipedia.org/wiki/Sybil_attack. Accessed 7 July 2017.
13. Last edited on 30 June 2017. "SHA-2". Wikipedia.https://en.wikipedia.org/wiki/SHA-2. Accessed 7 July 2017.
14. Szabo, N. 1998. "Secure property titles with owner authority". http://szabo.best.vwh.net/securetitle.html. Unable to access 7 July 2017. Alternative link here: http://nakamotoinstitute.org/secure-property-titles/. Accessed 7 July 2017.
15. Last edited on 27 June 2017. "Elliptic Curve Digital Signature Algorithm". Wikipedia. https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm. Accessed 7 July 2017.
16. Dogecoin. http://dogecoin.com/. Accessed 7 July 2017.
17. Last edited on 29 June 2017. "Denial-of-service attack". Wikipedia. https://en.wikipedia.org/wiki/Denial-of-service_attack. Accessed 7 July 2017.

추가자료

1. Intrinsic value: http://bitcoinmagazine.com/8640/an-exploration-of-intrinsic-value-what-it-is-why-bitcoin-doesnt-have-it-and-why-bitcoin-does-have-it/
2. Smart property: https://en.bitcoin.it/wiki/Smart_Property
3. Smart contracts: https://en.bitcoin.it/wiki/Contracts
4. B-money: http://www.weidai.com/bmoney.txt
5. Reusable proofs of work: http://www.finney.org/~hal/rpow/
6. Secure property titles with owner authority: http://szabo.best.vwh.net/securetitle.html
7. Bitcoin whitepaper: http://bitcoin.org/bitcoin.pdf
8. Namecoin: https://namecoin.org/
9. Zooko's triangle: http://en.wikipedia.org/wiki/Zooko's_triangle
10. Colored coins whitepaper: https://docs.google.com/a/buterin.com/document/d/1AnkP_cVZTCMLIzw4DvsW6M8Q2JC0lIzrTLuoWu2z1BE/edit
11. Mastercoin whitepaper: https://github.com/mastercoin-MSC/spec
12. Decentralized autonomous corporations, Bitcoin Magazine: http://bitcoinmagazine.com/7050/bootstrapping-a-decentralized-autonomous-corporation-part-i/
13. Simplified payment verification: https://en.bitcoin.it/wiki/Scalability#Simplifiedpaymentverification
14. Merkle trees: http://en.wikipedia.org/wiki/Merkle_tree
15. Patricia trees: http://en.wikipedia.org/wiki/Patricia_tree
16. GHOST: http://www.cs.huji.ac.il/~yoni_sompo/pubs/15/btc_scalability_full.pdf
17. StorJ and Autonomous Agents, Jeff Garzik: http://garzikrants.blogspot.ca/2013/01/storj-and-bitcoin-autonomous-agents.html
18. Mike Hearn on Smart Property at Turing Festival: http://www.youtube.com/watch?v=Pu4PAMFPo5Y
19. Ethereum RLP: https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-RLP
20. Ethereum Merkle Patricia trees: https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-Patricia-Tree
21. Peter Todd on Merkle sum trees: http://sourceforge.net/p/bitcoin/mailman/message/31709140/