

테조스(Tezos)

해당 번역본은 이용자가 디지털자산 관련 정보를 편리하게 확인할 수 있도록
단순 참고용으로 번역된 것이며, 투자 권유를 목적으로 하지 않습니다.
그 내용이 정확하지 않을 수 있으므로 정확한 정보 습득을 위해서는
원문을 참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다.
두나무는 해당 번역본 내용의 진실성 및 정확성을 보장하지 않고,
제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며,
투자 손실은 투자자에게 귀속됩니다.

테조스(Tezos) – 자기 수정 암호 원장

백서

L.M Goodman

2014년 9월 2일

원 논문과 현재 구현 간의 차이점은 빨간색으로 표시됨.

“우리의 논증은 완전히 순환적인 것은 아니지만, 그와 비슷하다.” — 윌러드 반 오만 콰인(Willard van Orman Quine)

초록(Abstract)

우리는 테조스(Tezos)라는 범용적이고 자기 수정(self-amending) 가능한 암호 원장을 제시한다. 테조스는 어떤 블록체인 기반 원장도 구현할 수 있다. 일반적인 블록체인의 연산들은 네트워크 작업을 담당하는 셸(shell)로 추상화된 순수 함수형 모듈로 구현된다. 비트코인(Bitcoin), 이더리움(Ethereum), 크립토노트(Cryptonote) 등은 모두 네트워크 계층에 대한 적절한 인터페이스를 구현함으로써 테조스 내에서 표현될 수 있다.

가장 중요한 점은, 테조스는 메타 업그레이드(meta upgrades)를 지원한다는 것이다. 즉, 프로토콜이 자체 코드를 수정함으로써 진화할 수 있다. 이를 달성하기 위해, 테조스는 이해관계자들이 프로토콜 수정안을 승인하는 절차를 정의하는 시드 프로토콜(seed protocol)로 시작한다. 이 수정 절차에는 투표 절차 자체에 대한 수정도 포함된다. 이는 철학자 피터 서먼스(Peter Suber)가 루이스 캐럴(Lewis Carroll)의 유명한 "거북이가 아킬레우스를 설득하다"(What the Tortoise Said to Achilles)를 해석하면서 지적인 메타 규칙(meta-rules) 개념과 유사하다.

목차(Contents)

- 1 서론(Introduction)
- 2 자기 수정 암호 원장(Self-amending cryptolledger)
 - 2.1 수학적 표현(Mathematical representation)
 - 2.2 네트워크 셸(The network shell)
 - 2.2.1 시계(Clock)
 - 2.2.2 체인 선택 알고리즘(Chain selection algorithm)
 - 2.2.3 네트워크 수준 방어(Network level defense)
 - 2.3 함수적 표현(Functional representation)
 - 2.3.1 체인 검증(Validating the chain)
 - 2.3.2 프로토콜 개정(Amending the protocol)
 - 2.3.3 RPC
- 3 시드 프로토콜(Seed protocol)
 - 3.1 경제(Economy)
 - 3.1.1 코인(Coins)
 - 3.1.2 채굴 및 서명 보상(Mining and signing rewards)
 - 3.1.3 분실된 코인(Lost coins)
 - 3.1.4 개정 규칙(Amendment rules)
 - 3.2 지분증명 메커니즘(Proof-of-stake mechanism)
 - 3.2.1 개요(Overview)
 - 3.2.2 시계(Clock)
 - 3.2.3 무작위 시드 생성(Generating the random seed)
 - 3.2.4 Follow-the-coin 절차(Follow-the-coin procedure)
 - 3.2.5 블록 채굴(Mining blocks)
 - 3.2.6 블록 서명(Signing blocks)
 - 3.2.7 체인의 가중치(Weight of the chain)
 - 3.2.8 고발(Denunciations)
 - 3.3 스마트 컨트랙트(Smart contracts)
 - 3.3.1 컨트랙트 타입(Contract type)
 - 3.3.2 오리지네이션(Origination)
 - 3.3.3 트랜잭션(Transactions)
 - 3.3.4 저장소 수수료(Storage fees)
 - 3.3.5 코드(Code)
 - 3.3.6 수수료(Fees)
- 4 결론(Conclusion)

1 서론(Introduction)

이 논문의 첫 번째 부분에서는 추상적 블록체인의 개념과 자기 수정(self-amending) 암호 원장의 구현에 대해 논의한다. 두 번째 부분에서는 우리가 제안하는 시드 프로토콜(seed protocol)을 설명한다.

2.1 수학적 표현(Mathematical representation)

블록체인 프로토콜은 근본적으로 전역 상태(global state)의 동시적 변형(concurrent mutations)을 모나딕(monadic) 방식으로 구현한 것이다. 이는 “블록(blocks)”을 전역 상태에 작용하는 연산자로 정의함으로써 달성된다. 생성(genesis) 상태에 작용하는 블록의 자유 모노이드(free monoid)는 트리 구조를 형성한다. 특정 순서에 따라 최소 말단 노드(leaf)를 전역 정준(canonical) 상태로 정의한다.

이는 다음과 같은 추상적 표현을 시사한다:

- $(S, \leq)$: 가능한 상태들의 전순서(total order)를 갖는 가산 집합(countable set).
- $\emptyset \notin S$: 특수한 무효 상태(invalid state).
- $B \subset S^{S \cup \{\emptyset\}}$: 블록들의 집합. 유효한 블록 집합은 $B \cap S^S$.

S 위의 전순서(total order)는 $\forall s \in S, \emptyset < s$ 가 되도록 확장된다. 이 순서는 블록 트리 내에서 어떤 리프가 정준(canonical) 것으로 간주되는지를 결정한다. B 내의 블록들은 상태에 작용하는 연산자로 본다.

결국, 어떤 블록체인 프로토콜(비트코인(Bitcoin), 라이트코인(Litecoin), 피어코인(Peercoin), 이더리움(Ethereum), 크립토노트(Cryptonote) 등)이든 다음의 튜플(tuple)에 의해 완전히 규정될 수 있다:

$$(S, \leq, \emptyset, B \subset S^{S \cup \{\emptyset\}})$$

[^1]

네트워크 프로토콜은 이들 블록체인 간에 본질적으로 동일하다. “채굴(mining)” 알고리즘은 블록 생성에 대한 인센티브가 주어졌을 때 네트워크의 창발적 속성(emergent property)에 불과하다.

테조스(Tezos)에서는 블록이 프로토콜 자체에 작용하도록 하여 블록체인 프로토콜을 자기 성찰적(introspective)으로 만든다. 그런 다음 우리는 프로토콜 집합을 다음과 같이 재귀적으로 표현할 수 있다:

$$P = \{(S, \leq, \emptyset, B \subset S^{(S \times P) \cup \{\emptyset\}})\}$$

2.2 네트워크 셸(The network shell)

이 형식적인 수학적 설명은 블록 트리를 어떻게 구축할지에 대해서는 알려주지 않는다. 이는 가십 네트워크(gossip network)와 프로토콜 사이의 인터페이스 역할을 하는 네트워크 셸(network shell)의 몫이다.

네트워크 셸은 클라이언트가 알고 있는 최적의 체인을 유지하는 방식으로 작동한다. 셸은 세 가지 유형의 객체를 인식한다. 첫 번째와 두 번째는 트랜잭션과 블록이며, 이는 유효하다고 판단될 경우에만 네트워크를 통해 전파된다. 세 번째는 프로토콜로, 기존 프로토콜을 수정(amend)하기 위해 사용되는 OCaml 모듈이다. 이에 대해서는 뒤에서 더 자세히 설명한다. 현재로서는 트랜잭션과 블록에 집중한다.

네트워크 셀에서 가장 힘든 부분은 노드를 서비스 거부 공격(denial-of-service attack)으로부터 보호하는 것이다.

2.2.1 시계(Clock)

모든 블록은 네트워크 셀이 볼 수 있는 타임스탬프(timestamp)를 포함한다. 시스템 시간으로부터 몇 분 이내라면 미래에서 온 것처럼 보이는 블록도 버퍼링되지만, 그렇지 않으면 거부된다. 프로토콜 설계는 클라이언트의 합리적인 시계 드리프트(clock drift)를 허용해야 하며, 타임스탬프가 조작될 수 있다는 점을 가정해야 한다.

2.2.2 체인 선택 알고리즘(Chain selection algorithm)

셀은 전체 블록 트리가 아니라 단일 체인만 유지한다. 이 체인은 클라이언트가 더 우수한 체인을 알게 되었을 때만 덮어쓰인다.

트리를 유지하는 것은 네트워크 통신 측면에서는 더 절약적일 수 있지만, 공격자가 낮은 점수(low-scoring)이지만 유효한 포크를 대량으로 생성하는 서비스 거부 공격에 취약할 수 있다.

그러나 노드가 특정 체인의 점수에 대해 거짓말을 하는 것은 여전히 가능하며, 클라이언트는 잠재적으로 많은 블록을 처리한 후에야 그 거짓을 밝혀낼 수 있다. 하지만 그러한 노드는 이후 무시될 수 있다.

다행히도, 프로토콜은 점수가 낮은 체인에서는 블록 생성률이 낮게 나타나는 속성을 가질 수 있다. 따라서 클라이언트는 “약한(weak)” 포크의 소수 블록만 고려한 후, 공표된 점수가 거짓임을 결론지을 수 있다.

2.2.3 네트워크 수준 방어(Network level defense)

게다가 셀은 “방어적(defensive)”이다. 이는 다양한 IP 범위에 걸쳐 다수의 피어에 연결을 시도한다. 또한 연결이 끊어진 피어를 감지하고, 악의적인 노드를 차단한다.

특정한 서비스 거부 공격으로부터 보호하기 위해, 프로토콜은 블록과 트랜잭션 크기에 대해 맥락 의존적인(context dependent) 제한을 셀에 제공한다.

2.3 함수적 표현(Functional representation)

2.3.1 체인 검증(Validating the chain)

우리는 다음의 OCaml 타입들을 통해 추상적 블록체인 구조의 거의 모든 범용성을 효율적으로 포착할 수 있다. 먼저, 블록 헤더는 다음과 같이 정의된다:

```
type raw_block_header = {  
  pred: Block_hash.t;  
  header: Bytes.t;  
  operations: Operation_hash.t list;  
  timestamp: float;  
}
```

우리는 헤더 필드를 의도적으로 강한 타입으로 정의하지 않았는데, 이는 임의의 내용을 표현할 수 있도록 하기 위함이다. 그러나 셀의 작동에 필요한 필드들은 타입을 지정한다. 여기에는 이전 블록의 해시, 연산 해시들의 목록, 그리고 타임스탬프가 포함된다. 실제로, 블록에 포함된 연산들은 네트워크 수준에서 블록과 함께 전송된다. 연산은 임의의 블롭(blob)으로 표현된다.

```
type raw_operation = Bytes.t
```

상태(state)는 Context 모듈의 도움으로 표현되며, 이는 디스크 기반 불변 키-값 저장소(immutable key-value store)를 캡슐화한다. 키-값 저장소의 구조는 다재다능하여 다양한 상태를 효율적으로 표현할 수 있다.

```
module Context = sig
  type t
  type key = string list
  val get: t -> key -> Bytes.t option Lwt.t
  val set: t -> key -> Bytes.t -> t Lwt.t
  val del: t -> key -> t Lwt.t
  (*...*)
end
```

디스크 연산에서 블로킹을 피하기 위해 함수들은 비동기 모나드 Lwt[4]를 사용한다. Context에 대한 연산은 순수 함수형이다: get은 예외를 던지지 않고 옵션 모나드를 사용하며, set과 del은 모두 새로운 Context를 반환한다. Context 모듈은 메모리 캐싱과 디스크 저장소를 조합하여 불변 저장소의 모습을 효율적으로 제공한다.

이제 임의의 블록체인 프로토콜의 모듈 타입을 정의할 수 있다:

```
type score = Bytes.t list
module type PROTOCOL = sig
  type operation
  val parse_block_header : raw_block_header -> block_header option
  val parse_operation : Bytes.t -> operation option
  val apply :
    Context.t ->
    block_header option ->
    (Operation_hash.t * operation) list ->
    Context.t option Lwt.t
  val score : Context.t -> score Lwt.t
  (*...*)
end
```

이제 상태를 직접 비교하지 않고, `score` 함수를 사용해 `Context`를 바이트 리스트로 사영(projection)한다. 바이트 리스트는 길이에 따라 먼저 정렬되고, 그다음 사전식(lexicographic) 순서에 따라 정렬된다. 이는 소프트웨어 버전 관리에서 사용되는 구조와 유사한 일반적인 구조로, 다양한 순서를 표현하는 데 상당히 유용하다.

왜 프로토콜 모듈 내에 비교 함수를 정의하지 않을까? 첫째, 그러한 함수가 전순서(total order)를 나타내야 한다는 요구사항을 강제하기 어렵다. 반면 점수 사영(score projection)은 항상 이를 검증한다(동물은 마지막 블록의 해시를 기반으로 해결할 수 있다). 둘째, 원칙적으로는 서로 다른 프로토콜 간 상태를 비교할 수 있어야 한다. 특정 프로토콜 개정 규칙에 의해 실제로는 그런 일이 일어나지 않을 가능성이 크지만, 네트워크 셀은 이를 알 수 없다.

`parse_block_header`와 `parse_operation` 연산은 셀에 노출되어 있으며, 셀이 완전한 타입의 연산과 블록을 프로토콜에 전달하고, 이들이 올바른 형태인지 확인한 후 연산을 릴레이하거나 블록을 로컬 블록 트리 데이터베이스에 추가할 수 있도록 한다.

`apply` 함수는 프로토콜의 핵심이다:

- 블록 헤더와 연관된 연산 리스트가 전달되면, `Context`에 가해진 변경 사항을 계산하고 수정된 사본을 반환한다. 내부적으로는 블록의 해시를 버전 핸들로 사용해 버전 관리 시스템처럼 차이점만 저장한다.
- 연산 리스트만 전달될 경우, 가능한 한 많은 연산을 탐욕적으로 적용한다. 이 함수는 프로토콜 자체에는 필수적이지 않지만, 유효한 블록을 만들려는 채굴자들에게는 큰 도움이 된다.

2.3.2 프로토콜 개정(Amending the protocol)

테조스(Tezos)의 가장 강력한 기능은 자기 수정(self-amendment) 가능한 프로토콜을 구현할 수 있다는 점이다. 이는 프로토콜에 두 가지 절차적 함수(procedural functions)를 노출함으로써 달성된다:

- `set_test_protocol`: 테스트넷에서 사용되는 프로토콜을 새로운 프로토콜로 교체한다(보통 이 해관계자의 투표로 채택된 것).
- `promote_test_protocol`: 현재 테스트 중인 프로토콜을 현재 프로토콜로 교체한다.

이 함수들은 `Context`를 변환하여 연관된 프로토콜을 변경한다. 새로운 프로토콜은 다음 블록이 체인에 적용될 때 효력을 발휘한다.

```
module Context = sig
  type t
  (*...*)
  val set_test_protocol: t -> Protocol_hash.t Lwt.t
  val promote_test_protocol: t -> Protocol_hash.t -> t Lwt.t
end
```

`protocol_hash`는 `.ml`과 `.mli` 파일을 tarball로 묶어 sha256 해시를 계산한 값이다. 이 파일들은 즉시 컴파일되며, 작은 표준 라이브러리에 접근할 수 있지만 샌드박스 내에서 실행되며 시스템 호출은 불가능하다.

이 함수들은 프로토콜의 `apply` 함수를 통해 호출되며, 새로운 `Context`를 반환한다.

많은 조건이 프로토콜 변경을 유발할 수 있다. 가장 단순한 버전에서는 이해관계자의 투표가 프로토콜 변경을 유발한다. 더 복잡한 규칙들은 점진적으로 투표로 도입될 수 있다. 예를 들어, 이해관계자들이 원한다면 새로운 개정안이 특정 속성을 준수함을 컴퓨터가 검증 가능한 증명으로 제공하도록 요구하는 개정안을 통과시킬 수 있다. 이는 사실상 “합헌성(constitutionality)”에 대한 알고리즘적 검증이다.

2.3.3 RPC

GUI 구축을 쉽게 하려면, 프로토콜이 JSON-RPC API를 제공한다. API 자체는 다양한 절차의 타입을 나타내는 JSON 스키마에 의해 기술된다. 보통 `get_balance`와 같은 함수들이 RPC로 구현될 수 있다.

```
type service = {  
  name : string list ;  
  input : json_schema option ;  
  output : json_schema option ;  
  implementation : Context.t -> json -> json option Lwt.t  
}
```

``name'`은 네임스페이스 구성을 위해 문자열 리스트로 정의한다. 입력과 출력은 선택적으로 JSON 스키마로 설명된다.

호출은 주어진 `Context`에서 수행되며, 보통 최고 점수 리프(highest scoring leaf)의 최근 조상에 해당한다. 예를 들어, 최고 점수 리프로부터 6 블록 위의 `Context`를 질의하면 6회의 확정(confirmations)이 있는 원장의 상태를 보여준다.

UI 자체는 특정 버전의 프로토콜에 맞춰 조정될 수도 있고, JSON 사양으로부터 일반적으로 유도될 수도 있다.

3 시드 프로토콜(Seed protocol)

블록체인이 제네시스 해시(genesis hash)에서 시작하는 것처럼, 테조스(Tezos)는 시드 프로토콜(seed protocol)에서 시작한다. 이 프로토콜은 사실상 어떤 블록체인 기반 알고리즘도 반영하도록 수정(amend)될 수 있다.

3.1 경제(Economy)

3.1.1 코인(Coins)

초기에는 100억(10,000,000,000) 개의 코인이 존재한다(토큰 공급의 초기 규모는 실제로는 크라우드세일(crowdsale)에서 발행된 토큰 수가 되며, “100억”은 단순히 임시 표기였을 뿐이다. 이러한 크기 변경은 논의 중인 원칙에는 영향을 미치지 않는다). 코인은 소수점 둘째 자리까지 나눌 수 있다(실제로는 정밀성을 위해 소수점 이하 8자리까지 사용할 수 있다).

우리는 하나의 코인을 “tez”라고 부를 것을 제안하며, 최소 단위는 단순히 센트(cent)라고 부른다. 또한, ₮ (\ua729, “Latin small letter tz”) 기호를 테즈(tez)를 나타내는 데 사용할 것을 제안한다. 따라서 $1 \text{ cent} = ₮0.01 = 1/100 \text{ 테즈}$ 가 된다.

3.1.2 채굴 및 서명 보상(Mining and signing rewards)

원칙(Principle) 우리는 모든 분산형 통화의 보안은 참여자들에게 금전적 보상을 제공해야만 유지될 수 있다고 추정한다(현재 보상 일정은 최종 확정 중이다). 입장문(position paper)에서 설명했듯이, 거래 수수료에만 의존하는 것은 공유지의 비극(tragedy of the commons)을 야기한다. 테조스에서는 채권(bond)과 보상의 조합을 사용한다.

채권은 채굴자(miner)가 구매하는 1년(현재는 기회비용이 높고 보안에 기여하는 이익이 적으므로 채권 기간은 한 사이클로 제한된다)의 보증금이며, 서명자(endorser) 또한 채권을 구매해야 한다. 이중 서명(double signing)이 발생할 경우 채권은 몰수된다.

1년(사이클)이 지난 후 채굴자(및 서명자)는 기회비용을 보상받기 위해 채권과 함께 보상을 받는다. 보안은 주로 채권의 가치에 의해 제공되며, 보상은 그 가치의 일부 비율에 불과하다.

채권의 목적은 필요한 보상의 규모를 줄이고, 손실회피 효과(loss aversion effect)를 네트워크의 이익으로 활용하는 것이다.

세부 사항(Specifics) 시드 프로토콜에서 블록을 채굴하면 ₮512의 보상이 주어지며, ₮1536의 채권이 필요하다. 블록 서명은 ₮32 ΔT^{-1} 의 보상을 제공하는데, 여기서 ΔT 는 해당 블록과 직전 블록 사이의 시간 간격(분 단위)이다. 블록당 최대 16개의 서명이 가능하며, 서명에는 채권이 필요 없다. 이 수치는 토큰 공급량 100억 개를 기준으로 설정되었으며, 상황에 따라 조정될 수 있다. 시뮬레이션 결과에 따르면 블록당 서명 수를 늘리면 포크(fork)의 난이도를 크게 높일 수 있으므로, 이 수는 향후 증가할 수 있다.

따라서 블록 생성 속도가 분당 1개라고 가정하면, 첫 해가 끝난 후 전체 화폐량의 약 8%가 안전 채권 형태로 보유될 것으로 예상된다(위의 매개변수 조정에 따라 변경될 수 있음).

보상 일정은 최대 5.4%의 명목 인플레이션률을 의미한다(총 블록 보상은 연간 약 5% 수준에서 시작되지만, 토큰 총량에 점근적 상한을 추가할 수 있다. 우리는 거버넌스 모델이 토큰 보유자의 이익과 일치할 경우 이는 무관하다고 생각하지만, 일부 사람들에게는 중요한 사안이므로 마지못해 고려 중이다). 명목 인플레이션은 중립적이며, 누구를 부유하게 하거나 빈곤하게 하지 않는다¹.

또한, 1년의 기간은 블록 개수가 아니라 블록의 타임스탬프에 의해 결정된다. 이는 채굴자들이 수행한 약정의 길이에 대한 불확실성을 제거하기 위함이다.

전망(Looking forward) 제안된 보상은 채굴자들에게 채권 대비 33%의 수익을 제공한다(현재 이 매개변수를 수정 중이며 곧 모든 이해관계자에게 타당한 방식으로 확정할 예정이다). 채굴자와 서명자들이 잠재적으로 변동성이 큰 자산을 한 사이클 동안 보유해야 하므로, 초기에는 높은 수익률이 필요하다.

그러나 테조스가 성숙해짐에 따라, 이 수익률은 점차 시장 금리 수준으로 낮아질 수 있다. 명목 인플레이션율을 1% 이하로 유지하는 것도 가능하겠지만, 그렇게 하는 데 특별한 의미가 있을지는 불분명하다.

3.1.3 분실된 코인(Lost coins)

화폐량에 대한 불확실성을 줄이기 위해, 1년 이상 활동이 없는 주소(타임스탬프로 판단)는 해당 주소가 보유한 코인과 함께 폐기된다(백서 최초 제안과 달리, 현재는 1년 이상 비활성인 주소가 자금을 잃는 일은 없다. 단지 다시 활성화될 때까지 스테이킹 권리만 잃게 된다. 즉, 비활성 주소는 블록 생성에 선택되지 않으며(이는 합의 알고리즘을 느리게 할 수 있으므로), 재활성화될 때까지 투표할 수도 없다(참여율에 대한 불확실성을 피하기 위해서다)).

3.1.4 개정 규칙(Amendment rules)

개정안은 $N = 2^{17} = 131\,072$ 블록마다 선거 주기(election cycle)를 거쳐 채택된다. 블록 간격이 1 분이라면 이는 약 3개월에 해당한다. 선거 주기는 다시 $2^{15} = 32\,768$ 블록 단위의 4분기로 나뉜다. 이 주기는 초기 개선을 장려하기 위해 비교적 짧게 설정되었으며, 이후 개정안을 통해 점차 길어질 것으로 예상된다(프로토콜 업그레이드 투표는 첫해에 훨씬 더 빈번히 이루어져 빠른 반복(iteration)을 가능하게 한다).

보안 조치로, 테조스 재단은 투표 절차의 문제점을 해결할 때까지 12개월간 거부권을 가진다. 개정안 채택에는 일정 수준의 정족수(quorum)가 요구된다. 정족수는 $Q = 80\%$ 에서 시작해 평균 참여율을 반영하도록 동적으로 조정된다. 이는 분실된 코인을 처리하는 데에도 필요하다.

1분기 프로토콜 개정안은 새로운 프로토콜을 나타내는 .ml과 .mli 파일을 tarball로 묶어 해시를 제출함으로써 제안된다. 이해관계자는 이러한 프로토콜 중 원하는 수만큼 승인할 수 있다. 이는 “승인 투표(approval voting)”로 알려져 있으며, 특히 견고한 투표 절차이다.

2분기 1분기에서 가장 많은 승인을 받은 개정안이 이제 투표에 부쳐진다. 이해관계자는 찬성, 반대 또는 명시적 기권(explicit abstention)을 선택할 수 있다. 기권 또한 정족수에 포함된다.

3분기 정족수가 충족되고(명시적 기권 포함), 개정안이 80%의 찬성을 받으면 개정안은 승인되어 테스트 프로토콜을 대체한다. 그렇지 않으면 기각된다. 정족수가 q 라고 가정하면, 최소 정족수 Q 는 다음과 같이 갱신된다:

$$Q \leftarrow 0.8Q + 0.2q.$$

이 갱신의 목적은 분실된 코인으로 인해 투표 절차가 시간이 지나며 정체되는 것을 방지하는 것이다. 최소 정족수는 각 선거에서 달성된 정족수의 지수 이동 평균(exponential moving average)이다.

4분기 개정안이 승인된 경우, 3분기 시작부터 테스트넷에서 실행된다. 이해관계자는 두 번째 투표를 통해 테스트 프로토콜을 메인 프로토콜로 승격할지 여부를 확인한다. 이 역시 정족수 충족과 80%의 초다수결(supermajority)을 요구한다.

우리는 개정 절차에 대해 보수적인 접근을 의도적으로 선택했다. 그러나 이해관계자들은 필요하다면 이 정책을 완화하거나 강화하는 개정안을 자유롭게 채택할 수 있다.

3.2 지분증명 메커니즘(Proof-of-stake mechanism)

3.2.1 개요(Overview)

우리의 지분증명 메커니즘은 Slasher[1], chain-of-activity[2], proof-of-burn 등을 포함한 여러 아이디어의 혼합이다. 아래는 알고리즘에 대한 간단한 개요이며, 구성 요소들은 이어지는 절에서 더 자세히 설명한다.

각 블록은 무작위 이해관계자(채굴자)가 채굴하며, 이전 블록에 대해 무작위 이해관계자(서명자)들이 제공한 다중 서명을 포함한다. 채굴과 서명은 모두 소규모 보상을 제공하지만, 동시에 이중 채굴(double mining) 또는 이중 서명(double signing)이 발생할 경우 몰수되는 1년짜리(최초 제안과 달리 언본딩은 1년이 아니라 한 사이클 후에 이뤄진다. 사이클을 넘는 장기 보류는 보안 개선 효과가 미미한 반면 많은 자본을 고정시키므로 연장하지 않는다) 안전 예치금(safety deposit)을 요구한다.

프로토콜은 2048 블록의 사이클로 전개된다. 각 사이클의 시작 시점에, 무작위 시드는 끝에서 두 번째 사이클에서 채굴자들이 선택하고 커밋한 숫자들과 마지막 사이클에서 공개한 숫자들로부터 도출된다. 이 무작위 시드를 사용하여, 다음 사이클에 대해 특정 주소에 채굴 권리와 서명 권리를 배정하기 위해 follow the coin 전략을 사용한다. 그림 1을 보라.

3.2.2 시계(Clock)

프로토콜은 블록 간 최소 지연 시간을 강제한다. 원칙적으로 각 블록은 어떤 이해관계자라도 채굴할 수 있다. 그러나 주어진 블록에 대해 각 이해관계자는 무작위 최소 지연 시간에 종속된다. 가장 높은 우선순위를 받은 이해관계자는 이전 블록 이후 1분이 지나면 블록을 채굴할 수 있다. 두 번째 우선순위는 2분 후, 세 번째는 3분 후, ... 와 같이 진행된다.

이는 소수의 이해관계자만 기여하는 포크에서 블록 생성률이 낮게 나타나도록 보장한다. 만약 그렇지 않다면, 매우 높은 점수를 주장하는 아주 긴 체인을 노드가 검증하도록 속여 CPU 기반 서비스 거부 공격이 가능해진다.

3.2.3 무작위 시드 생성(Generating the random seed)

채굴된 모든 블록은 채굴자가 선택한 무작위 수에 대한 해시 커밋먼트를 포함한다. 이 숫자들은 다음 사이클에서 공개되어야 하며, 공개하지 않으면 안전 예치금이 몰수된다. 이 강력한 페널티는 시드 엔트로피를 약화시키는 선택적 미공개(selective withholding)를 억제하기 위한 강한 억지 장치다.

다음 사이클의 악의적 채굴자들이 이러한 공개를 검열하려 시도할 수 있지만, 하나의 블록에서 여러 숫자를 공개할 수 있으므로 성공 가능성은 매우 낮다.

사이클 내에서 공개된 모든 숫자는 해시 리스트로 결합되며, scrypt 키 파생 함수로 루트에서 시드를 도출한다. 키 파생은 일반 데스크톱 PC에서 블록 하나를 검증하는 평균 시간의 일부 퍼센트에 해당하는 비용이 들도록 조정해야 한다.

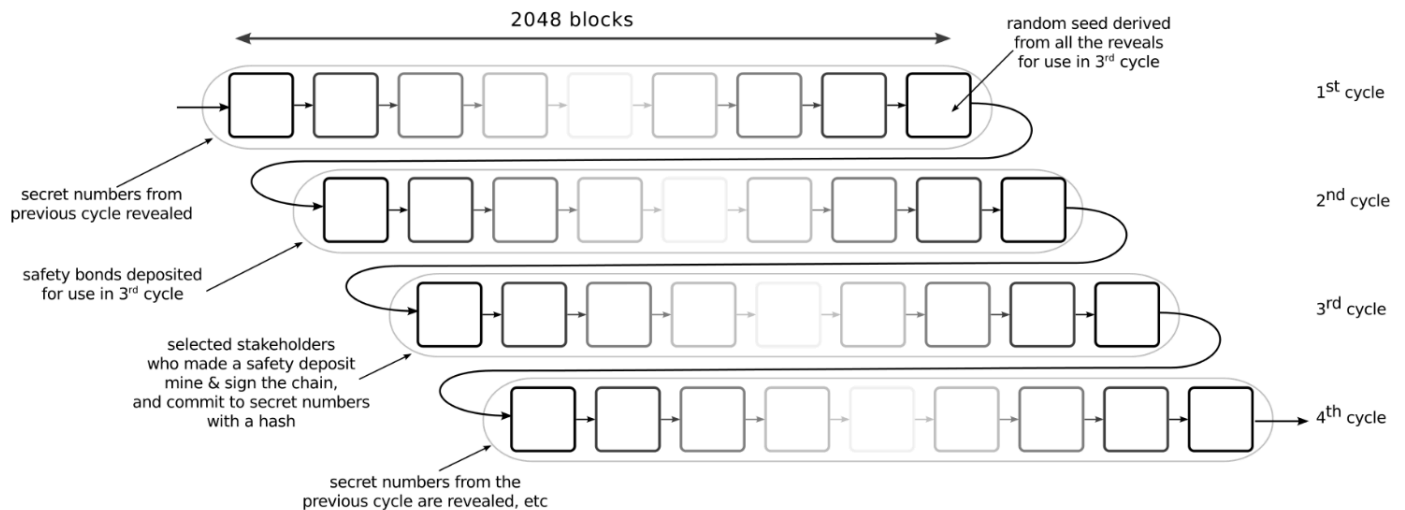


그림 1: 지분증명 메커니즘의 네 개 사이클

3.2.4 Follow-the-coin 절차(Follow-the-coin procedure)

무작위로 이해관계자를 선택하기 위해 follow the coin 절차를 사용한다.

원리(Principle) 이 아이디어는 비트코인에서 follow-the-satoshi로 알려져 있다. 이 절차는 마치 발행된 모든 사토시가 고유 일련번호를 가진 것처럼 “작동한다(as-if)”. 사토시는 생성 시간에 따라 암묵적으로 정렬되며, 무작위 사토시 하나가 추출되어 블록체인을 통해 추적된다. 물론 개별 센트 단위가 직접 추적되는 것은 아니다. 대신, 여러 출력에 걸쳐 입력이 결합되고 사용될 때 무엇이 일어나는지를 서술하는 규칙이 적용된다.

결국 알고리즘은 각 키에 연관된 구간(interval) 집합을 추적한다. 각 구간은 사토시의 “범위”를 나타낸다. 불행히도 시간이 지남에 따라 데이터베이스 단편화가 심해져 클라이언트 측 부담이 증가한다.

코인 롤(Coin Rolls) 우리는 10,000 테즈(tez)로 구성된 큰 “코인 롤”을 구성하여 이전 알고리즘을 최적화한다. 따라서 약 100만 개의 롤이 존재한다. 데이터베이스는 각 롤을 현재 소유자에게 매핑한다.

각 주소는 특정 롤들의 집합과 잔돈(loose change)을 보유한다. 한 롤의 일부만 지출하려면 롤을 분해하고, 그 일련번호를 롤의 LIFO 큐(일종의 “림보(limbo)”)에 보낸다. 모든 트랜잭션은 깨진(분해된) 롤의 수를 최소화하는 방식으로 처리된다. 주소가 롤을 재구성하기에 충분한 코인을 보유하게 되면, 큐에서 일련번호를 꺼내 다시 롤을 형성한다.

LIFO 우선순위는 공격자가 비밀 포크에서 계정 간 잔돈을 교환(shuffling)하여 자신이 보유한 코인을 바꾸는 것을 방지한다.

이 접근의 작은 단점은 지분이 가장 가까운 정수 개수의 롤로 내림(round down) 처리된다는 점이다. 그러나 follow-the-satoshi 접근에 비해 효율성이 크게 향상된다.

롤에 번호가 매겨져 있지만, 이 방법이 Zerocash와 같은 대체가능성(fungibility) 보존 프로토콜의 사용을 배제하지는 않는다. 이러한 프로토콜도 동일한 “림보” 큐 기법을 사용할 수 있다.

동기(Motivation) 이 절차는 단순히 잔액에 비례하여 무작위 주소를 추출하는 것과 기능적으로 다르다.

실제로 비밀 포크에서 채굴자는 무작위 시드 생성을 통제하고, 미리 적절한 주소들을 생성해 자신에게 서명 및 발행 권한을 배정하려 시도할 수 있다. 룰이 무작위로 선택되는 경우, 비밀 포크는 특정 룰의 소유권을 위조할 수 없으므로, 자신에게 서명 및 발행 권한을 배정하려면 시드에 적용된 해시 함수의 프리이미지(preimage)를 구하려 시도해야 한다.

길이 $N = 2048$ 인 사이클에서, 룰의 일부 f 를 보유한 사람은 평균적으로 fN 개의 채굴 권한을 받으며, 실제 받은 비율 f' 의 표준편차는

$$\sqrt{\frac{1}{N} \frac{1-f}{f}}.$$

공격자가 서로 다른 W 개의 시드를 무차별 대입(brute-force)할 수 있다면, 기대되는 이점은 최대 ²

$$\left(\sqrt{\frac{2 \log(W)}{N} \frac{1-f}{f}}\right) fN \text{ 블록.}$$

예를 들어, $f = 10\%$ 의 룰을 통제하는 공격자는 사이클당 약 205개 블록을 채굴할 것으로 예상된다. 비밀 포크에서 시드를 통제하려 시도하고, 1조 번 이상의 해시를 계산했다고 가정하면, 약 302개(즉 전체의 약 14.7%) 블록을 자신에게 배정할 수 있다.

주의할 점:

- 시드를 도출하는 해시는 비용이 큰 키 파생 함수이므로 무차별 대입이 비현실적이다.
- 채굴 블록 수에서 선형적인 이득을 얻으려면, 공격자는 제곱 지수적인(quadratically exponential) 노력이 필요하다.

3.2.5 블록 채굴(Mining blocks)

무작위 시드는 룰을 반복적으로 선택하는 데 사용된다. 첫 번째로 선택된 룰의 이해관계자는 이전 블록 이후 1분이 지나면 블록을 채굴할 수 있고, 두 번째는 2분 후, ... 와 같이 진행된다.

이해관계자가 시드를 확인하고 다음 사이클에서 높은 우선순위 블록을 채굴할 수 있음을 알게 되면, 안전 예치금을 예치할 수 있다.

특정 블록을 채굴하기 위해 어떤 이해관계자도 안전 예치금을 예치하지 않는 잠재적 문제 상황을 피하기 위해, 16분의 지연 후에는 예치금 없이 블록을 채굴할 수 있다.

채권(bond)은 해당 체인에서 블록을 채굴하지 않은 경우 즉시 묵시적으로 구매자에게 반환된다.

3.2.6 블록 서명(Signing blocks)

현 상태로도 거의 동작하는 지분증명 시스템에 가깝다. 체인의 가중치를 블록 수로 정의할 수도 있다. 그러나 이는 일종의 이기적 채굴(selfish mining)을 초래할 수 있다.

따라서 서명 체계를 도입한다. 블록이 생성되는 동안 무작위 시드를 사용해 16개의 서명 권한을 16개의 룰에 무작위로 할당한다.

서명 권한을 받은 이해관계자들은 생성 중인 블록을 관찰한 뒤 해당 블록의 서명을 제출한다. 이러한 서명은 채굴자들이 자신의 부모 블록의 포함을 보장하기 위해 다음 블록에 포함시킨다.

서명자가 받는 서명 보상은 해당 블록과 그 직전 블록 사이의 시간 간격에 반비례한다.

따라서 서명자들은 특정 시점에 생성된 가장 우수한 블록이라고 진정으로 믿는 블록에 서명할 강한 유인을 가진다. 또한 서명 보상은 블록이 실제로 블록체인에 포함되어야만 지급되므로, 어떤 블록에 서명할지 합의할 강한 유인도 있다.

최고 우선순위 블록이 채굴되지 않은 경우(예: 채굴자가 오프라인) 서명자들에게는 채굴자가 늦을 경우를 대비해 잠시 대기하려는 유인이 있을 수 있다. 그러나 다른 서명자들이 최고 우선순위 블록에 서명하기로 결정할 수 있고, 새로운 블록이 그 서명들을 포함하여 바로 생성될 수 있으므로, 지연 전략은 배제될 가능성이 크다.

반대로, 서명자들이 다른 서명자들이 먼저 서명할 것을 두려워해, 보자마자 첫 블록에 서명하기 시작하는 균형(equilibrium)을 상상해볼 수도 있다. 그러나 이는 누구에게도 이익이 되지 않는 매우 부자연스러운 상황이다. 서명자들이 이러한 균형이 일어날 것이라 생각할 인센티브는 없고, 그런 방식으로 동작하도록 프로그램 코드를 고칠 인센티브는 더더욱 없다. 운영을 교란하려는 악의적 이해관계자가 이 전략을 따르더라도, 다른 이들이 따를 가능성이 낮으므로 결국 자신만 손해를 보게 된다.

3.2.7 체인의 가중치(Weight of the chain)

가중치는 서명 수이다.

3.2.8 고발(Denunciations)

블록의 이중 발행(double minting) 또는 이중 서명을 방지하기 위해, 채굴자는 자신의 블록에 고발(denunciation)을 포함할 수 있다.

이 고발은 두 개의 서명 형태를 취한다. 각 발행 서명 또는 블록 서명은 블록의 높이(height)를 서명하므로, 위반에 대한 증거는 매우 간결하다.

누구에게나 위반을 고발할 수 있도록 허용할 수도 있지만, 채굴자 이외의 누구에게 허용해도 실질적인 이점은 없다. 실제로 채굴자는 어떤 위반 증거든 단순히 복사해 자신의 발견인 것처럼 제출할 수 있기 때문이다.³

어떤 당사자가 이중 발행이나 이중 서명에 대해 유죄 판결을 받으면, 안전 예치금은 몰수된다.

3.3 스마트 컨트랙트(Smart contracts)

3.3.1 컨트랙트 타입(Contract type)

UTXO(미사용 트랜잭션 출력) 대신, 테조스는 상태를 가지는 계정(stateful accounts)을 사용한다. 이러한 계정이 실행 가능한 코드를 포함하면 일반적으로 컨트랙트라고 불린다. 계정도 컨트랙트의 일종(실행 코드가 없는 컨트랙트)이므로, 우리는 일반적으로 둘 다 “컨트랙트”라고 부른다.

각 컨트랙트에는 “관리자(manager)”가 있으며, 계정의 경우 단순히 소유자가 관리자이다. 컨트랙트가 지출 가능(spendable)으로 표시된 경우, 관리자는 컨트랙트와 연관된 자금을 지출할 수 있다. 또한 각 컨트랙트는 지분증명 프로토콜에서 블록을 서명하거나 채굴하는 데 사용되는 공개키 해시를 지정할 수 있다. 개인키는 관리자가 제어할 수도 있고 그렇지 않을 수도 있다.

형식적으로, 컨트랙트는 다음과 같이 표현된다:

```
type contract = {  
  counter: int; (* 반복 공격 방지를 위한 카운터 *)  
  manager: id; (* 컨트랙트 관리자 공개키의 해시 *)  
  balance: Int64.t; (* 보유 잔액 *)  
  signer: id option; (* 서명자의 id *)  
  code: opcode list; (* 명령어(opcode) 리스트로 구성된 컨트랙트 코드 *)  
  storage: data list; (* 컨트랙트의 저장소 *)  
  spendable: bool; (* 관리자가 자금을 지출할 수 있는가? *)  
  delegatable: bool; (* 관리자가 서명 키를 변경할 수 있는가? *)  
}
```

컨트랙트의 핸들은 초기 콘텐츠의 해시이다. 이미 존재하는 컨트랙트와 해시가 충돌하는 컨트랙트를 생성하려는 시도는 유효하지 않은 연산이며, 유효한 블록에 포함될 수 없다.

데이터는 합집합 타입(union type)으로 표현된다:

```
type data =  
| STRING of string  
| INT of int
```

여기서 INT는 부호 있는 64비트 정수이며, 문자열은 최대 1024바이트의 배열이다. 저장 용량은 16,384바이트로 제한되며, 정수는 8바이트, 문자열은 해당 길이만큼 계산한다.

3.3.2 오리지네이션(Origination)

오리지네이션(Origination) 연산은 새로운 컨트랙트를 생성하는 데 사용된다. 이 연산은 컨트랙트의 코드와 초기 저장소 내용을 지정한다. 핸들이 이미 기존 컨트랙트의 핸들과 일치한다면, 오리지ンは 거부된다(실수 또는 악의적 행위가 아니라면 일어나지 않을 상황이다).

컨트랙트는 활성 유지를 위해 최소 1 XTZ의 잔액이 필요하다. 잔액이 이 값 이하로 떨어지면 컨트랙트는 파기된다.

3.3.3 트랜잭션(Transactions)

트랜잭션은 한 컨트랙트에서 다른 컨트랙트로 보내는 메시지이며, 다음과 같이 표현된다:

```

type transaction = {
  amount: amount; (* 송금되는 금액 *)
  parameters: data list; (* 스크립트에 전달되는 매개변수 *)
  (* 반복 공격 방지를 위한 카운터 (송장 id) *)
  counter: int;
  destination: contract hash;
}

```

이러한 트랜잭션은 관리자의 키로 서명하면 컨트랙트에서 보낼 수 있고, 컨트랙트 내에서 실행되는 코드에 의해 프로그래밍 방식으로 보낼 수도 있다. 트랜잭션이 수신되면, 금액은 대상 컨트랙트의 잔액에 추가되고, 대상 컨트랙트의 코드가 실행된다. 이 코드는 전달된 매개변수를 활용할 수 있으며, 저장소를 읽고 쓰고, 서명 키를 변경하며, 다른 컨트랙트에 트랜잭션을 발송할 수 있다.

카운터의 역할은 재사용 공격(replay attack)을 방지하는 것이다. 트랜잭션은 컨트랙트의 카운터 값과 트랜잭션의 카운터 값이 같을 때만 유효하다. 트랜잭션이 적용되면 카운터는 1 증가하여, 동일한 트랜잭션이 재사용되지 못하도록 한다.

트랜잭션은 또한 클라이언트가 유효하다고 간주하는 최근 블록의 해시를 포함한다. 공격자가 포크를 이용해 긴 재구성을 강제하더라도, 이러한 트랜잭션을 포함할 수 없으므로 포크가 명백히 가짜임이 드러난다. 이는 최후의 방어선으로, TAPOS는 긴 재구성을 방지하는 데에는 훌륭하지만 단기적인 이중 지불(double spending)을 막는 데에는 좋지 않다.

(account_handle, counter) 쌍은 비트코인의 미사용 출력과 대략적으로 동일하다.

3.3.4 저장소 수수료(Storage fees)

저장소는 네트워크에 비용을 발생시키므로, 저장소가 1바이트 증가할 때마다 최소 $\mathfrak{t}1$ 의 수수료가 부과된다. 예를 들어, 트랜잭션 실행 후 저장소에 정수가 하나 추가되고 기존 문자열에 10자가 덧붙여졌다면, 컨트랙트 잔액에서 $\mathfrak{t}18$ 이 차감되어 소각된다.

3.3.5 코드(Code)

언어는 스택 기반이며, 고수준 데이터 타입과 원시 연산(primitives), 엄격한 정적 타입 검사를 갖춘다. 이 설계는 Forth, Scheme, ML, Cat에서 영감을 받았다. 명령어 집합의 전체 사양은 [5]에 기술되어 있다. 이 사양은 언어의 명령어 집합, 타입 시스템, 의미론을 완전히 제시하며, 쉽게 배우기 위한 것이 아니라 정밀한 참고 매뉴얼을 제공하기 위한 것이다.

3.3.6 수수료(Fees)

지금까지 설명한 시스템은 이더리움(Ethereum)이 트랜잭션을 처리하는 방식과 유사하다. 그러나 우리는 수수료 처리 방식에서 차이가 있다. 이더리움은 프로그램 실행 시간이 선형적으로 증가할수록 수수료가 증가하도록 하여 임의의 길이의 프로그램 실행을 허용한다. 이는 한 채굴자가 트랜잭션을 검증할 유인은 제공하지만, 동일한 트랜잭션을 검증해야 하는 다른 채굴자들에게는 동일한 유인이 제공되지 않는다.

실제로 스마트 컨트랙트에 사용될 흥미로운 프로그램들은 대부분 매우 짧다. 따라서 우리는 프로그램이 실행될 수 있는 단계 수에 대해 하드 캡(hard cap)을 부여하는 방식으로 구조를 단순화한다.

하드 캡이 일부 프로그램에 너무 엄격하다면, 프로그램은 실행을 여러 단계로 나누고 여러 트랜잭션을 사용해 완전하게 실행할 수 있다. 테조스는 개정(amendment) 가능하므로, 이 하드 캡은 미래에 변경될 수 있고, 새로운 원시 연산이 새로운 명령어(opcode)로 추가될 수도 있다.

계정이 허용한다면, 서명 키는 변경 요청 서명 메시지를 발행하여 바꿀 수 있다.

4 결론(Conclusion)

우리는 매력적인 시드 프로토콜을 구축했다고 생각한다. 그러나 테조스(Tezos)의 진정한 잠재력은 이해관계자들이 자신들에게 가장 적합하다고 생각하는 프로토콜을 결정할 수 있도록 권한을 부여하는 데 있다.

참고문헌(References)

- [1] Vitalik Buterin. *Slasher: A punitive proof-of-stake algorithm*. <https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/>, 2014.
- [2] Ariel Gabizon, Iddo Bentov and Alex Mizrahi. *Cryptocurrencies without proof of work*. <http://www.cs.technion.ac.il/~iddo/CoA.pdf>, 2014.
- [3] Peter Suber. *Nomic: A game of self-amendment*. <http://legacy.earlham.edu/~peters/writing/nomic.htm>, 1982.
- [4] Jérôme Vouillon. *Lwt: a cooperative thread library*. 2008.
- [5] Tezos project. *Formal specification of the tezos smart contract language*. <https://tezos.com/pages/tech.html>, 2014.

각주(Footnotes)

-
- 1. 대조적으로, 비트코인의 채굴 인플레이션은 비트코인 보유자 전체를 빈곤하게 만들고, 중앙은행제도는 저축자의 희생을 대가로 금융 부문을 부유하게 만든다. ↩
 - 2. 이는 W 개의 정규 분포 변수를 대상으로 한 최대값의 기댓값에 대한 표준적 경계값이다. ↩
 - 3. 영지식 증명(zk-proof)은 누구나 위법 행위를 고발하여 이익을 얻을 수 있도록 하지만, 이것이 실제로 큰 이익을 가져다주는지는 분명하지 않다. ↩