2. How Bitcoin Blockchain works

Contents

1. **트랜잭션의 생성과 전파**
2. **노드 간 데이터 검증:**
   - 트랜잭션 검증
   - 블록 검증
3. **합의 알고리즘:**
   - PoW(작업 증명)와 PoS(지분 증명)의 비교
   - 분산 네트워크에서 신뢰 구축
4. **블록 생성 과정:**
   - 트랜잭션이 블록으로 묶이는 과정
   - 체인에 연결되는 원리
5. **블록체인의 장단점:**
   - 투명성과 보안성
   - 확장성과 속도 제한 문제

1. 트랜잭션의 생성과 전파

트랜잭션이란 ?

- 블록체인에서 **A가 B에게 비트코인을 보내는 요청**
- 트랜잭션은 **블록체인의 기본 단위**로, 모든 거래는 트랜잭션 형태로 기록됨

1. 트랜잭션의 생성과 전파

트랜잭션 데이터의 주요 요소 :

- **송신자(A) :** A의 공개키(Public Key)
- **수신자(B) :** B의 비트코인 주소(공개 키의 해시)
- **보낼 금액 :** ex. 1BTC
- **서명(Signature) :** 송신자가 트랜잭션의 무결성과 정당성을 증명
- **트랜잭션 ID :** 트랜잭션 데이터의 고유한 해시값

트랜잭션의 형태 (JSON 예시

```
{
  "sender": "A's Public Key",
  "receiver": "B's Address",
  "amount": 1,
  "signature": "d1a2d3e4...",
  "timestamp": "2024-12-06T12:00:00Z"
}
```

https://github.com/habit80/myBlockPanda

1. 트랜잭션의 생성과 전파

트랜잭션 서명의 생성 과정

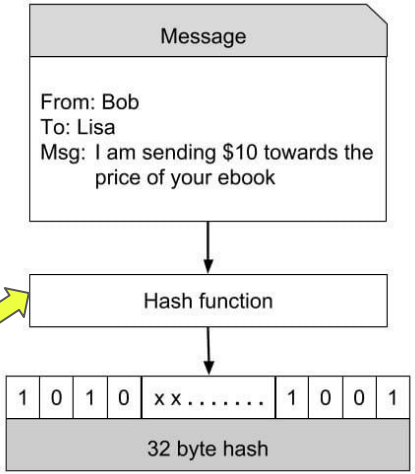
- **서명(Signature) 생성의 목적**
  - 송신자(A)가 해당 트랜잭션을 요청했음을 증명
  - 데이터 위변조 방지

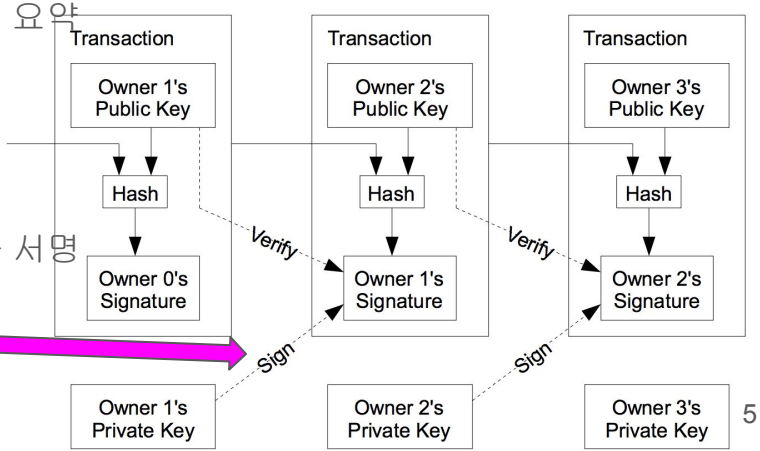
- **서명 생성 과정**
  - **트랜잭션 해시 생성**
    - 트랜잭션 데이터를 해시 **함수(SHA-256)**를 사용해 요약
    - 생성된 해시는 트랜잭션의 고유한 지문 역할

```
Hash = SHA256(Transaction Data)
```

  - **개인키로 서명**
    - 송신자(A)는 자신의 개인키(Private Key)로 해시를 서명

```
Signature = Sign(Hash, Private Key)
```





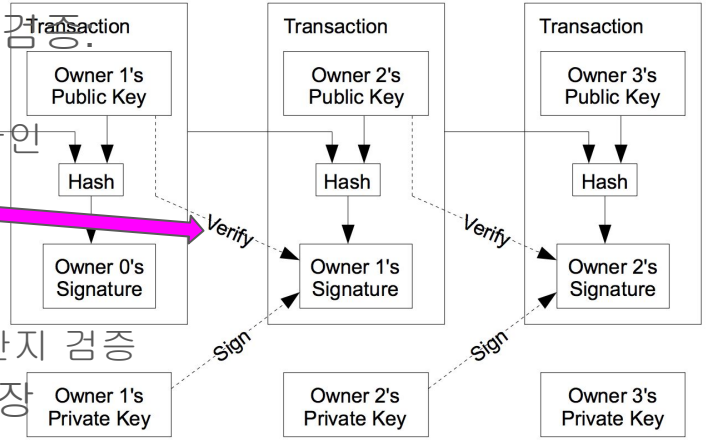
1. 트랜잭션의 생성과 전파

트랜잭션 검증 과정

- 네트워크에 브로드캐스트된 트랜잭션은 각 노드에서 검증:
  - **서명 검증 :**
    A의 공개 키와 서명을 사용해 트랜잭션이 정당한지 확인

```
Verify(Signature, Public Key) == Hash
```

  - **잔액 검증 :**
    송신자의 UTXO(미사용 잔액)를 확인해 잔액이 충분한지 검증
- 검증된 트랜잭션은 **Mempool**(트랜잭션 대기열)에 저장



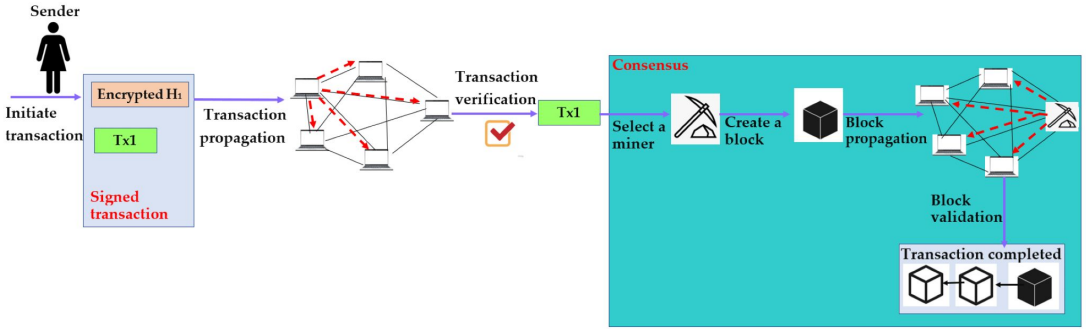
트랜잭션 검증 실패 시

- 서명 불일치 또는 잔액 부족 시, 해당 트랜잭션은 무효 처리

1. 트랜잭션의 생성과 전파

트랜잭션의 전파

- **P2P 네트워크에서의 전파 과정:**
  - 송신자가 트랜잭션을 생성해 자신의 노드에 전송
  - 노드는 해당 트랜잭션을 검증한 후, 주변 노드로 전파
  - 트랜잭션은 네트워크 전체에 Gossip Protocol 방식으로 퍼짐

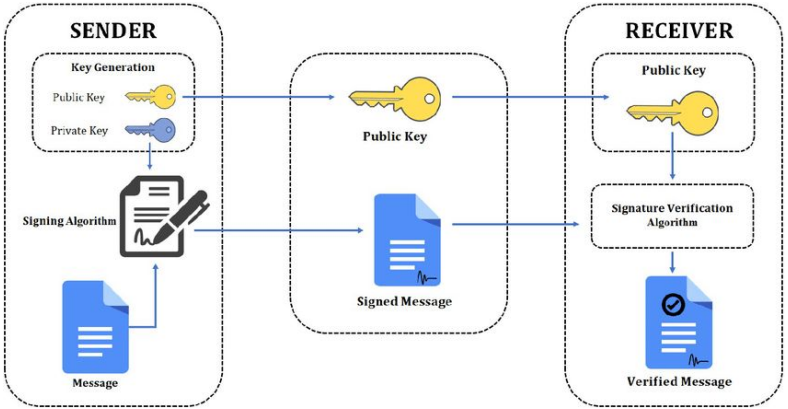


- **결과**
  - 트랜잭션이 네트워크의 모든 노드에 저장됨
  - 이후 채굴 노드가 트랜잭션을 선택해 블록에 포함

2. 노드 간 데이터 검증

트랜잭션 검증 과정

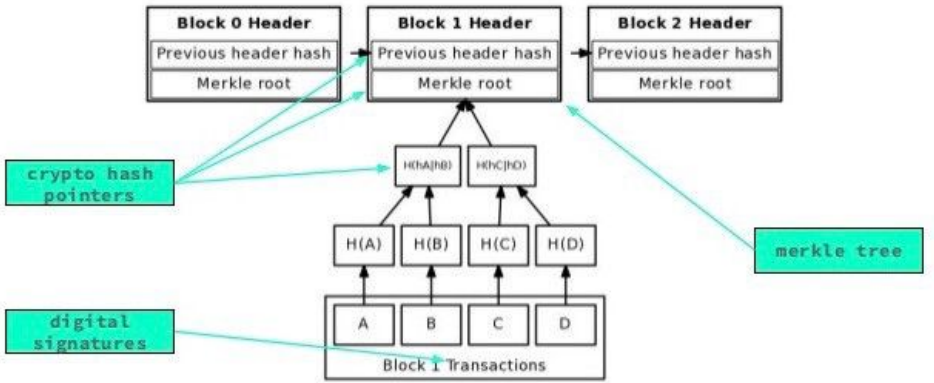
- **트랜잭션 서명 확인**
  - A의 개인 키(Private Key)로 서명한 데이터를 A의 공개 키(Public Key)로 복호화
  - 서명이 유효하면 신뢰
- **송신자의 잔액 확인**
  - UTXO(미사용 트랜잭션 출력)를 조회하여 송신자가 충분한 잔액을 보유하고 있는지 확인



2. 노드 간 데이터 검증

블록 검증

- **검증 요소**
  - 블록 헤더의 Hash가 난이도 목표(Target Difficulty) 를 만족하는지 확인
  - 블록 내 모든 트랜잭션이 유효한지 확인
- **Merkle Root로 트랜잭션 무결성 검증**
  - 블록 내 트랜잭션 Hash 들을 병합해 최상위 Hash (Merkle Root) 생성



Merkle tree connecting block transactions to block header merkle root

3. 합의 알고리즘

합의 알고리즘의 목적

- **블록 생성의 공정성 보장**
  - 중앙 기관 없이 분산된 노드가 동일한 조건에서 경쟁하여 블록 생성
  - 특정 노드의 독점을 방지하고 탈중앙화 유지

- **데이터 무결성과 신뢰성 보장**
  - PoW는 블록을 조작하거나 위변조하는 것을 어렵게 만들어 네트워크의 안정성과 신뢰성 유지

- **누구의 블록이 체인에 추가될지를 결정**
  - PoW(ex.비트코인)와 PoS(ex.이더리움)는 블록 생성자를 선택하는 대표적인 두 가지 방식

3. 합의 알고리즘 (PoW)

PoW의 기본 개념

- 블록 생성자는 **SHA-256 Hash 계산 문제**를 해결해야 함
- 계산된 Hash 값이 **난이도 목표(Target Hash)** 를 만족해야 블록을 생성할 수 있음
  Hash(Block Header + **Nonce**) < Target Hash
  ➢ **작업증명의 핵심은 Nonce 값을 구하는 것**
- 이 과정을 먼저 해결한 노드가 블록을 생성하고 보상을 받음

블록 헤더 구조 (PoW 관련 주요 필드)

필드	설명	크기
Previous Block Hash	이전 블록의 해시값 (체인 연결 보장)	32 Byte
Merkle Root	현재 블록에 포함된 모든 트랜잭션의 해시 요약값	32 Byte
Timestamp	블록 생성 시점	4 Byte
Nonce	PoW 문제를 해결하기 위해 반복적으로 변경되는 값	4 Byte
Difficulty Target	난이도 목표(Target Hash) 를 정의하는 압축된 형태	4 Byte

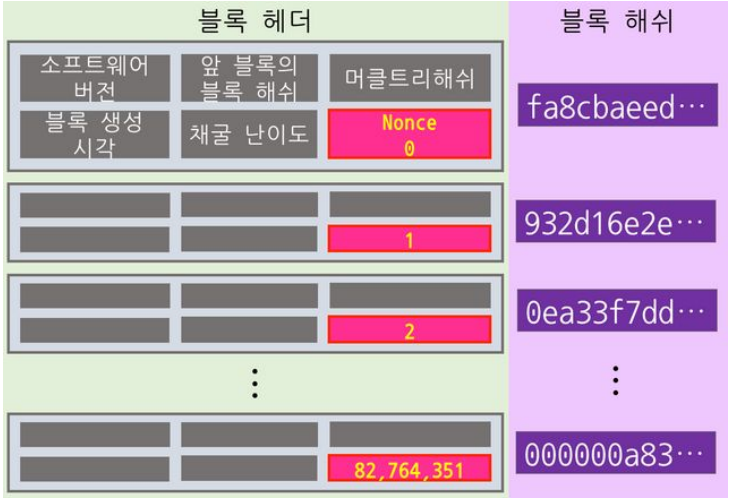
3. 합의 알고리즘 (PoW)

PoW의 실행 과정

- **블록 헤더 데이터 준비**
  - Previous Block Hash, Merkle Root, Timestamp, 초기 Nonce 값 설정

- **Nonce 조정 및 Hash 계산**
  - Nonce 값을 0부터 증가시키며 블록 헤더의 SHA-256 Hash를 계산
  - 계산된 Hash 값이 Target Hash 조건을 만족하면 성공
    Hash(Block Header + **Nonce**) < Target Hash

- **성공 시 블록 생성**
  - 찾는 Nonce 값과 함께 새로운 블록을 네트워크에 브로드캐스트



3. 합의 알고리즘 (PoW)

PoW 값이 저장되는 위치

- **Nonce**
  - PoW 문제를 해결한 결과로 찾은 값이며, 블록 헤더의 Nonce 필드에 저장
- **Hash 결과**
  - 블록 헤더 전체를 Hashing한 결과 값은 블록의 고유식별자(Block Hash)로 저장
- **체인 연결 역할**
  - 각 블록의 Previous Block Hash 필드에 이전 블록의 해시가 저장되어 체인을 연결

https://github.com/habit80/myBlockPanda

3. 합의 알고리즘 (PoW)

Python 코드 예제

```python
import hashlib

def mine_block(previous_hash, merkle_root, timestamp, difficulty):
    nonce = 0
    target = "0" * difficulty  # 난이도에 따른 목표 해시 (앞자리 0의 개수)
    while True:
        block_header = f"{previous_hash}{merkle_root}{timestamp}{nonce}".encode()
        block_hash = hashlib.sha256(block_header).hexdigest()
        if block_hash[:difficulty] == target:
            return {
                "block_hash": block_hash,
                "nonce": nonce,
                "header": {
                    "previous_hash": previous_hash,
                    "merkle_root": merkle_root,
                    "timestamp": timestamp,
                    "nonce": nonce,
                    "difficulty": difficulty
                }
            }
        nonce += 1
```

3. 합의 알고리즘 (PoS)

PoS의 기본 개념

- PoS는 블록 생성자를 보유한 코인 비율에 따라 확률적으로 선택
- PoW처럼 에너지를 소비하지 않으며, 스테이킹 된 코인이 많을수록 블록 생성자로 선택될 가능성이 높음

PoS 실행 과정

1. 네트워크 참여자는 자신이 보유한 코인을 스테이킹(Staking)하여 블록 생성자로 참여
2. 네트워크가 확률적으로 블록 생성자를 선택
3. 선택된 노드는 블록을 생성하고 트랜잭션 수수료를 보상으로 받음

3. 합의 알고리즘 (PoW vs PoS)

항목	Proof of Work (PoW)	Proof of Stake (PoS)
에너지 소비	매우 높음	낮음
보안	강력 (작업 증명 기반)	지분 기반 보안(지분 집중 가능성)
블록 생성 방식	SHA-256 해시 계산	지분량 기반 확률적 선택
블록체인 적용 사례	비트코인, 라이트코인 등	이더리움 2.0, 카르다노 등

4. 블록 생성 과정

트랜잭션이 블록으로 묶이는 전체 과정

1. 트랜잭션 수집

1.1. **Mempool** : 모든 노드는 Mempool(트랜잭션 대기열)에 전송된 트랜잭션을 저장
1.2. **유효성 검증** : 노드가 트랜잭션의 서명, 잔액 등을 확인하여 유효한 트랜잭션만 선택

2. 블록 생성 준비

2.1. 채굴자는 **Mempool에서 유효한 트랜잭션**을 선택하여 **블록에 포함**
* 트랜잭션의 수수료가 높은 순서로 우선 선택
2.2. 블록의 크기제한(비트코인 기준 1MB)에 맞춰 적합한 트랜잭션을 추가

3. Merkle Root 생성

3.1. 선택된 트랜잭션을 Hash 값으로 변환하여 **Merkle Tree** 를 생성
* 트랜잭션 Hash → 부모노드 Hash → 최종 **Merkle Root** 생성
3.2. Merle Root 는 블록 헤더에 저장

4. 블록 완성 및 체인 연결

4.1. PoW 문제를 해결한 후, 새로운 블록을 생성하고 네트워크에 브로드캐스트
4.2. 다른 노드가 블록을 검증한 뒤 체인에 추가
* **Previous Hash**가 일치해야 체인에 연결 가능

5. 블록체인의 장단점

장점		단점	
투명성	모든 노드가 동일한 데이터를 공유 거래 내역이 누구에게나 공개	확장성 문제	처리 속도 제한(TPS) (비트코인은 초당 약 7 트랜잭션 처리, 블록 크기와 속도 제한 문제)
보안성	데이터 암호화 및 해시를 통해 무결성 보장 네트워크 합의로 위변조 방지	에너지 소비	PoW 기반의 높은 전력 소모
탈중앙화	단일 실패 지점 없음	네트워크 지연	트랜잭션이 블록에 포함되기까지 시간이 걸림

https://github.com/habit80/myBlockPanda

End