Execution Layer

Blockchain @ uninsubria

Roberto Vicario

2024/2025

Contents

1 Execution Layer **2**

2 UTXO (Unspent Transaction Output) Model **2**

2.1 Merkle Tree 2

2.2 Lezione 4: Ethereum, Account Model e Smart Contracts 3

2.2.1 4.1 Account-based model 3

2.2.2 4.2 Transazioni e messaggi 3

2.2.3 4.3 Ethereum Virtual Machine (EVM) 3

2.2.4 4.4 Smart Contract ERC-20 4

1 Execution Layer

The *Execution Layer* is a critical component of blockchain systems, responsible for executing transactions and maintaining the state of the blockchain. It operates on top of the consensus layer, which ensures that all nodes agree on the order of transactions.

2 UTXO (Unspent Transaction Output) Model

The *UTXO* model is the key concept in Bitcoin and other cryptocurrencies. It represents the state of the blockchain as a set of unspent transaction outputs, which can be used as inputs for new transactions.

$\boxed{DEFINITION}$

Given a UTXO block U_i in the blockchain, and let $TX_i = \{tx_1, ..., tx_n\}$ be the set of transactions included in the block, organized in a Merkle tree with root hash h_{root}. The block U_i can be defined as:

$$U_i = (H_{i-1}, MT(h_{\text{root}})) \tag{1}$$

2.1 Merkle Tree

A *Merkle Tree* is a data structure that allows efficient verification that a transaction is included in a block, without needing all the data.

$\boxed{SEARCHING}$

1. **Initialization:** Client stores the root hash of the Merkle tree h_{root}.
2. **Forwarding:** Prover sends to the client the transaction hash TX_i and the Merkle proof:

   $$\Pi = \{h_1, h_2, ..., h_k\}$$

   Where each h_i is a hash at some level of the tree.
3. **Verification:** Starting from TX_i, the client combines it with each h_i using the hash function, following the tree path:

$$h' = H(H(H(TX_i \parallel h_1) \parallel h_2) \parallel ... \parallel h_k)$$

4. **Termination:** The client accepts if the final hash equals the stored Merkle root:

$$h' = h_{\text{root}}$$

2.2 Lezione 4: Ethereum, Account Model e Smart Contracts

2.2.1 4.1 Account-based model

- Stato: [S_t = { (addr, balance, nonce, code, storage) }] dove `addr` può essere EOA o contract.
- Due tipi:
  - **EOA**: firmato da chiave privata.
  - **Contract account**: contiene codice immutabile + storage mutabile.

2.2.2 4.2 Transazioni e messaggi

- Una tx include:
  - `sig`, `from`, `to`, `value`, `gasLimit`, `gasPrice`, `data`.
- I contract possono generare **messaggi** ad altri contract (non persistenti su blockchain).

2.2.3 4.3 Ethereum Virtual Machine (EVM)

- Stack machine, max depth 1024.
- Ogni istruzione ha un **costo in gas**:
  - E.g., `SSTORE`: 20k gas (scrittura), 15k refund (cancellazione).
- Se gas esaurito → tx rollback.
- La fee pagata è: [GasUsed ×GasPrice]

2.2.4 4.4 Smart Contract ERC-20

- Stato:
  - `balanceOf: mapping(address => uint256)`
  - `allowance: mapping(address => mapping(address => uint256))`
- Funzioni principali:
  - `transfer`, `approve`, `transferFrom`
- Eventi:
  - `Transfer`, `Approval`