ChessChain

Authors:
Levan Dalbashvili
Irakli Kereleishvili
Giorgi Kurtanidze

Chess-Specific Data Model

- **Move Representation**: MoveData class with fields: match_id, id, player, move, timestamp, signature
- **Match Results**: ChessTransaction class with match_id, winner, moves_hash, nonce, public_key, signature
- **Move Verification**: Cryptographically verifies each move's authenticity
- **Move Storage**: Uses composite keys ({match_id}_{move_id}) in the LMDB database for efficient retrieval
- **Fake Match Generation:** Includes functionality to generate and propagate test matches with 3 predefined moves

Proof-of-Stake Consensus Mechanism

- **Initial Stake Allocation**: Each new node receives exactly 120 tokens (INITIAL_STAKE = 120)
- **Minimum Participation Threshold**: Nodes need at least 10 tokens (MIN_STAKE = 10) to participate in consensus
- **Selection**: The lottery_selection function in utils.py uses a deterministic random selection based on SHA-256 hash of the round seed + participant ID
- **Consensus Threshold**: Requires 67% stake approval (QUORUM_RATIO = 0.67) for block confirmation
- **Block Proposer Rewards**: Proposers receive 2 tokens for each successfully confirmed block
- **Round Timing**: Consensus rounds occur every 20 seconds (POS_ROUND_INTERVAL = 20)

Cryptography Implementation

- **Ed25519 Curve**: Implements high-security elliptic curve signatures using Python's cryptography library
- **Multi-layer Signature Verification**:
  - Transaction signatures: {**match_id**}:{**winner**}:{**nonce**}:{**proposer_pubkey_hex**}
  - Block signatures: {**round_seed_hex**}:{**merkle_root**}:{**proposer_pubkey_hex**}:{**previous_block_hash**}:{**timestamp**}
  - Vote signatures: {**round_seed_hex**}:{**block_merkle_root**}:{**proposer_pubkey_hex**}:{**validator_pubkey_hex**}:{**vote**}
- **Per-node Keypair**: Each node generates a unique Ed25519 keypair at initialization

Efficient Merkle Tree Implementation

- **Binary Tree Structure**: Complete binary tree with parent-child relationships
- **SHA-256 Hashing**: Uses standard SHA-256 for all hash operations
- **String/Binary Compatibility**: Accepts both string and binary inputs, normalizing to bytes
- **Odd-Node Handling**: Duplicates the last node when constructing a level with an odd number of nodes
- **Hexadecimal Output**: Returns root hash as hex string via get_root() method
- **Empty Tree Handling**: Properly handles empty trees by returning None

Robust Blockchain Architecture

- **Genesis Block**: Initializes chain with special genesis block containing the timestamp 1714501200
- **Block Structure**: Includes round seed, transaction hashes, Merkle root, proposer information, timestamp, and previous block hash
- **Fork** : Uses the "longest chain wins" rule
- **Chain Traversal**: Can traverse the blockchain backward from any point using get_chain_from_hash()
- **Transaction State Management**: Properly tracks which transactions are included in confirmed blocks(Pending, Processed)

Gossip Protocol with Optimizations

- **Smart Peer Selection**: select_propagation_peers() selects a subset of 5 peers (configurable) for efficient propagation
- **Load Distribution**: Uses round number modulo peer count to cycle through different peers in each round
- **Mempool Management**: Tracks transactions in mempool and pending_transactions dictionaries
- **Duplicate Detection**: Prevents redundant message propagation through the sent set
- **Message Forwarding**: Validators forward votes and confirmations to enhance network connectivity

Fault Tolerance

- **Network Partition Recovery**: Can recover from network partitions through fork resolution
- **Transaction Reprocessing**: During chain reorganization, reprocess_transactions() properly moves transactions back to mempool
- **Block Request System**: BlockSyncRequest and BlockSyncResponse provide catch-up functionality
- **Chain Integrity**: Verifies block signatures and Merkle roots for every received block
- **Resynchronization**: resolve_fork_with_retry() attempts multiple strategies to resolve inconsistencies

Interactive Command Interface

- **Command Set**: Includes 8 commands (help, stake, show, send, clearmempool, showstakes, showmoves)
- **Stake Management**: Allows users to add stake with stake <amount>
- **Transaction Simulation**: Creates test chess matches and transactions with send
- **Mempool Inspection**: Views and manages pending transactions
- **Transaction Tracking**: Lists all stored transactions including their winners and match IDs
- **Move Inspection**: Retrieves and displays all moves for a particular match with showmoves <match_id>
- **Help System**: Provides detailed command documentation through the help command