Introduction

This report provides an overview of key aspects of blockchain technology. It explores the block structure, use of hashing algorithms, proof-of-work algorithm, mining, transaction execution, and communication between nodes. Understanding these concepts is crucial for comprehending the functioning and applications of blockchain technology.

Block structure

1. Block Hash: Each block has a unique hash that serves as its identifier. The hash is calculated by applying a hashing algorithm, such as SHA-256, to the block's data, including the previous block's hash, timestamp, nonce and merkle root. The block hash ensures the integrity of the block and its data.

2. Previous Block Hash: To establish a chronological order and maintain the chain-like structure of the blockchain, each block contains a reference to the hash of the previous block. This creates a link between blocks and forms a chain.

3. Timestamp: The timestamp indicates the time at which the block was mined.

4. Transactions: A block contains a collection of transactions. A transaction includes sender and recipient addresses, transaction amounts, additional data, the public key, the signature for verification purposes and the hash of the transaction.

5. Nonce: The nonce is a number used in the proof of work algorithm. Miners iterate through different nonce values to find a suitable hash that satisfies certain conditions, such as a specific number of leading zeros. The specific number of leading zeros is defined by the difficulty of the blockchain(e.g. a difficulty of 5 means each hash must start with 5 zeros). The inclusion of the nonce allows miners to demonstrate the computational effort they have expended to mine the block.

By defining the data model of the blocks in this way, we establish a structured format for storing and organizing information in the blockchain. Each block's hash and the reference to the previous block's hash create a chain of blocks, ensuring the immutability and integrity of the blockchain. The inclusion of other attributes, such as the timestamp, transactions, merkle root and nonce, enables various functionalities like transaction verification, consensus, and security through the proof of work mechanism.

Use of hashing algorithms

1. Block Hashing: When a new block is created, we calculate its hash by applying the SHA-256 hashing algorithm to the block's data. The block's data includes the previous block's hash, timestamp, merkle root, and nonce. By hashing the block's data, we obtain a unique hash that represents the block's identity. This hash ensures that any modification to the block's data will result in a completely different hash value.

2. Transaction Hashing: Within each block, individual transactions are also hashed using the same hashing algorithm. The transaction hash serves as a unique identifier for each transaction. It allows efficient indexing and retrieval of transactions from the blockchain when needed.

3. Merkle Tree: To optimize the verification of multiple transactions within a block, we constructed a Merkle tree. A Merkle tree is a binary tree structure in which the leaves represent the hashed transactions, and each internal node is the hash of its child nodes. By repeatedly hashing pairs of transactions until we reach the root of the tree, we obtain a single hash known as the Merkle root. The Merkle root summarizes the entire set of transactions within the block. Verifying the inclusion of a specific transaction within the block only requires a few hash computations and comparisons.

4. Validation and Consensus: Hashing algorithms are employed during the consensus process to validate the integrity of blocks. Miners and nodes in the network perform calculations using the hashing algorithm to find a suitable hash that meets certain criteria, such as a specific number of leading zeros. This process, known as proof of work, demonstrates that a certain amount of computational effort has been expended to mine the block. By relying on the cryptographic properties of the hashing algorithm, consensus is achieved, and the blockchain maintains its immutability and security.

Creating the Merkle Tree and the Merkle Root

The Merkle Root and the Merkle Tree are created by repeatedly iterating through the list of transaction hashes and hashing them together in pairs of 2, until there is only 1 hash left. The iterations create the Merkle Tree, while the hash that remains at the end is the Merkle Root.

Proof of Work algorithm

We implemented a proof-of-work (PoW) algorithm to secure the network, validate blocks, and control the rate at which new blocks are added to the blockchain. The PoW algorithm serves as the basis for mining and consensus in the network. Here's an explanation of how we implemented the PoW algorithm:

1. Target Difficulty: We defined a target difficulty level for mining a new block. This difficulty level determines the computational effort required to find a valid block hash. A higher difficulty means that more computational work is needed to find a valid hash, thus making the mining process take more time on average.

2. Block Header: Each block in the blockchain contains a header that includes various fields such as the previous block's hash, a timestamp, a nonce, a list of transactions, the name of the miner that mined the block and the merkle root. The nonce is a value that miners adjust to find a valid hash that meets the target difficulty.

3. Hashing Algorithm: We used the cryptographic hash function SHA-256, to compute the hash of the block header. Miners repeatedly modify the nonce value in the block header and hash it until a hash is found that meets the target difficulty.

4. Mining Process: Miners start by creating a candidate block, which includes the transactions to be included in the new block. They then increment the nonce value in the block header and compute the hash. If the resulting hash meets the target difficulty (has a sufficient number of leading zeros), the miner has successfully mined a block.

5. Validating the PoW: When a miner finds a block hash that meets the target difficulty, they broadcast the new block to the network. Other nodes in the network can easily verify the validity of the block by checking the proof of work. They can independently compute the hash of the block header and verify that it is a valid block.

The implementation of the PoW algorithm ensures that the process of mining new blocks is resource-intensive and time-consuming. Miners must expend computational power and electricity to search for a valid hash, which provides security to the network by making it difficult and economically costly to manipulate the blockchain. The PoW algorithm also ensures that the creation of new blocks is distributed among participants, preventing the situation where only one miner mines all of the blocks.

Mining a new block

1. Block Candidate: Miners start by selecting a block candidate . This block candidate includes a set of pending transactions chosen from the mempool that need to be added to the blockchain. It also includes a block header with relevant information such as the previous block's hash, a timestamp, and a nonce.

2. Nonce Iteration: Miners increment the nonce value in the block header and compute the hash of the block using the SHA-256 hashing algorithm. The nonce serves as a variable that miners can adjust to find a valid block hash.

3. Hash Evaluation: The computed hash is evaluated against a predefined target difficulty level. This difficulty level determines the number of leading zeros required in the hash. Miners aim to find a hash that meets the difficulty requirement.

4. Finding a Valid Hash: Miners continue incrementing the nonce and recalculating the hash until they find a hash that meets the target difficulty. This process involves a significant amount of computational work, and miners need to perform numerous iterations until a valid hash is found.

5. Block Validation: Once a miner finds a valid hash, they have successfully mined a new block. The miner broadcasts the mined block to the network. Each node validates the block by checking the validity of the blockchain if that node was added, before adding it to the blockchain.

Mining serves multiple purposes in the blockchain system. It secures the network by making it computationally expensive to alter past transactions, adds new blocks to the blockchain in a decentralized manner, and enables the consensus mechanism to achieve agreement on the state of the blockchain among participants.

By implementing the mining process, we ensure the integrity, security, and decentralization of the public blockchain system, allowing for the creation of new blocks and the inclusion of valid transactions in the blockchain.

Executing Transactions

1. Transaction Creation: Users initiate transactions by creating a transaction object that contains the necessary information, such as the sender's address, recipient's address, and the amount being transferred. After a transaction is initiated, it is signed by the wallet of the user using its private key and sent to a node in the network.

2. Transaction Verification: Before a transaction is sent to other nodes, executed or added to a block, it needs to be verified. This is done by confirming that the transaction signature is valid.

3. Transaction Propagation: Once a transaction is verified, it needs to be propagated to the network. The transaction is broadcasted to the connected nodes. This ensures that the transaction reaches a wide network of nodes for validation and inclusion in the blockchain.

4. Transaction Inclusion in a Block: Miners or validators in the network collect valid transactions and store them in the mempool. When a miner starts to mine a new block, it will pick all of the transactions from its mempool and include them in a new block they are mining.

5. Block Mining and Consensus: Miners or validators compete to solve a computational puzzle (proof-of-work). The miner who successfully mines the block broadcasts it to the network, including the executed transactions.

6. Block Validation: Other nodes in the network receive the newly mined block and validate its contents, including the executed transactions, before adding the block to their blockchain.

Communication Between Nodes

When a new node starts, because our blockchain was designed to run only locally, it will try to start on localhost, port 8000. If there already is a node running on this port, it will try to run on port 8001 and so on, until it successfully runs.

When a node is run successfully, the first thing it does is try to connect to the other nodes that are running inside the network. It does this inside the connectToNodes function, where it connects to all of the nodes that are currently running in the network. When the node connects to a node that is already running, it sends a NewNodeMessage and establishes a connection with that. After the connection is established, the socket its added to the nodes

list of nodes, so that it can broadcast relevant information to it in the future, while also calling the listenToNode function, so that it can also listen for Transactions, Blocks or Sync Requests from this node.

Each node implements a listen function. This function is used to listen for any new connections. Here is where both wallets and other nodes can connect to a node. If a wallet connects to the node it can send Transactions to it, while if a new node connects to an already running node, immediately after connecting it will send a NewNodeMessage. This way the running node knows that it needs to add the socket to its list of nodes, so that it can broadcast information to that node in the future. After the initial connection is done, nodes can send between each other transactions, blocks and even the full blockchain, in order to make sure that they are up to date with the network.
Synchronization between nodes

Each time a new node is run, it has to sync to the latest version of the blockchain in the network. This version is determined by the longest valid chain present in the network. In order to achieve this, the node will send a SyncRequest to each of the nodes that it has connections to. After sending a SyncRequest, the node will receive back an instance of the blockchain. For each blockchain received, it will check if that blockchain is valid and if it is longer than the current version of the blockchain stored in its database. The longest valid version of the blockchain, whether it is the one that was already stored in the database or one that was received, will be used further.

Conclusion

Overall, building this project has helped me learn about the fundamental components of a blockchain, the use of hashing algorithms for data integrity, the proof of work algorithm for mining and consensus, the process of executing transactions, and the communication and synchronization between nodes in a network.