# Blockchain Merkle Trees

Difficulty Level : Medium • Last Updated : 19 Sep, 2022

Read Discuss

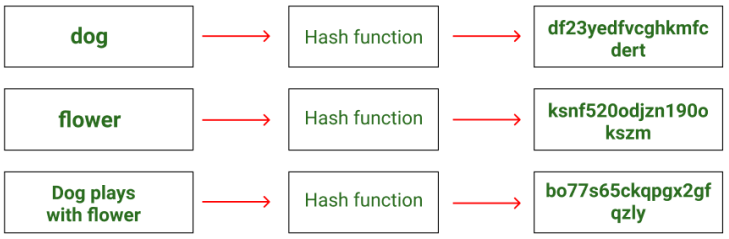
A hash tree is also known as **Merkle Tree.** It is a tree in which each leaf node is labeled with the hash value of a data block and each non-leaf node is labeled with the hash value of its child nodes labels. This article focuses on discussing the following topics in detail:

1. **What is a Cryptographic Hash?**
2. **What is Hash Pointer?**
3. **Blockchain Structure**
4. **Block Structure**
5. **Merkle Tree Structure**
6. **How Do Merkle Trees Work?**
7. **Why Merkle Trees are Important For Blockchain?**
8. **Proof of Membership**
9. **Merkle Proofs**
10. **Simple Payment Verification(SPV)**
11. **Advantages of Merkle Tree**

Let's discuss each of these topics in detail.

## What is a Cryptographic Hash?

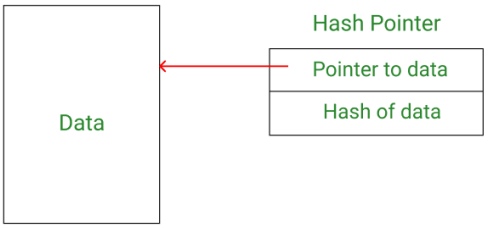
A cryptographic hash is a function that outputs a **fixed-size digest** for a variable-length input. A hash function is an important cryptographic primitive and extensively used in blockchain. For example, SHA-256 is a hash function in which for any variable-bit length input, the output is always going to be a 256-bit hash.




- From the above picture, it is clear that even the slightest change in an alphabet in the input sentence can drastically change the hash obtained. Therefore hashes can be used to **verify integrity**.
- Consider there is a text file with important data. Pass the contents of the text file into a hash function and then store the hash in the phone. A hacker manages to open the text file and changes the data.
- Now when you open the file again, you can compute the hash again and compare this hash with the one stored previously on the phone.
- It will be clearly evident that the two hashes do not match and hence the file has been tampered with.

### What is Hash Pointer?

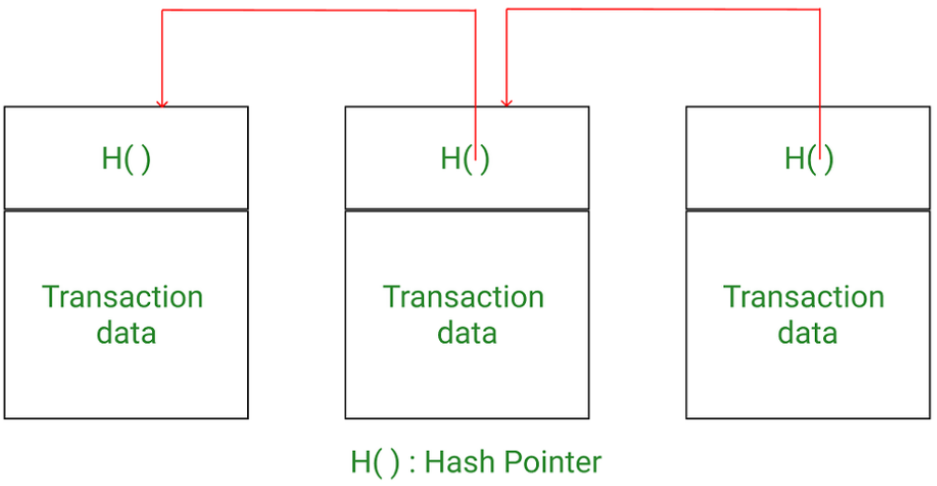
A regular pointer stores the memory address of data. With this pointer, the data can be accessed easily. On the other hand, a hash pointer is a pointer to where data is stored and with the pointer, the cryptographic hash of the data is also stored. So a hash pointer points to the data and also allows us to verify the data. A hash pointer can be used to build all kinds of data structures such as **blockchain and Merkle tree.**




## Blockchain Structure

The blockchain is a proficient combination of two hash-based data structures-

1. **Linked list:** This is the structure of the blockchain itself, which is a linked list of hash pointers. A regular linked list consists of nodes. Each node has 2 parts- data and pointer. The pointer points to the next node. In the blockchain, simply replace the regular pointer with a hash pointer.
2. **Merkle tree:** A Merkle tree is a binary tree formed by hash pointers, and named after its creator, Ralph Merkle.




*Blockchain as linked list with hash pointers*

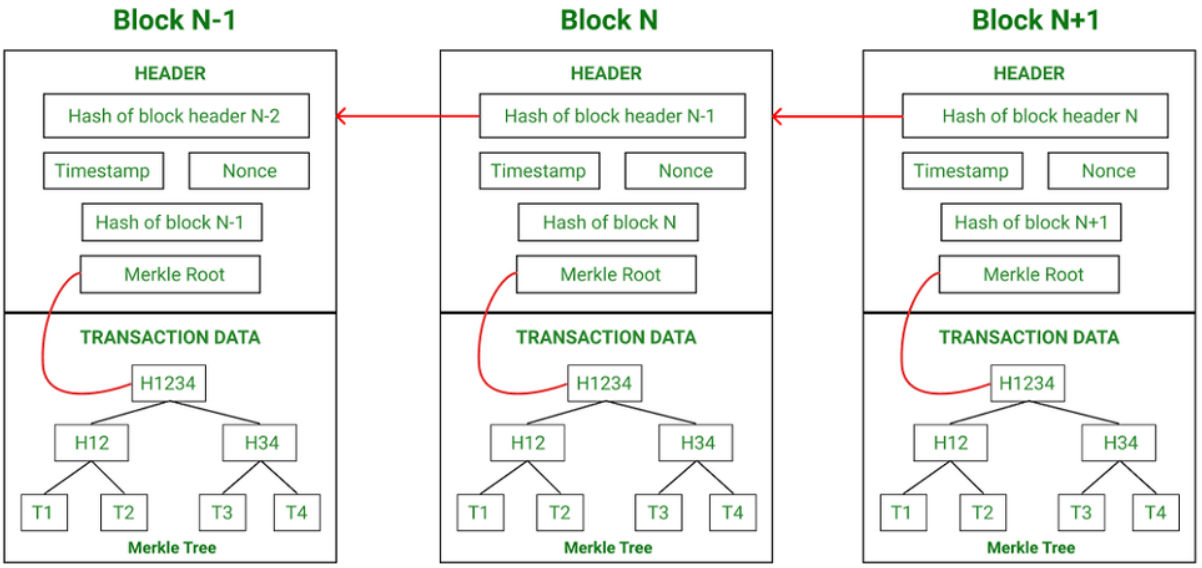
## Block Structure

**1. Block header:** The header data contains metadata of the block, i.e information about the block itself. The contents of the block header include-

- Hash of the previous block header.
- Hash of the current block.
- Timestamp.
- Cryptographic nonce.
- Merkle root.

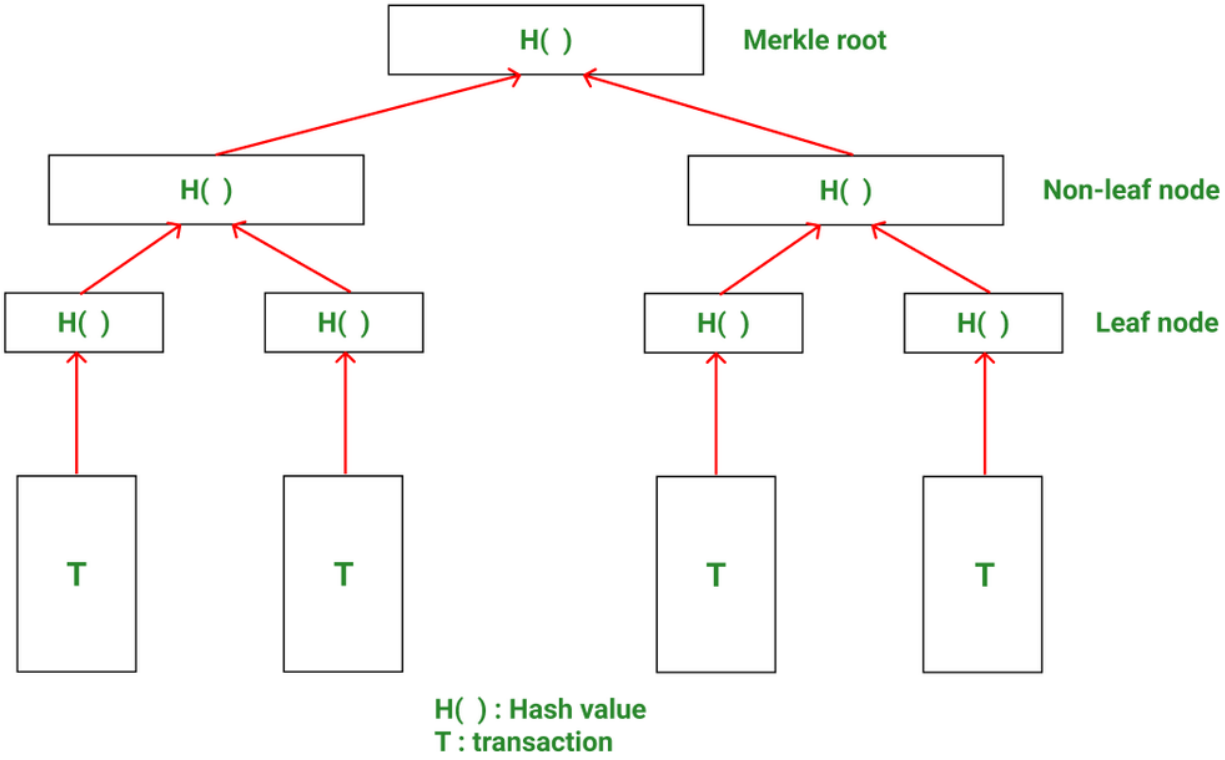
**2. Merkle tree:** A Merkle tree is a binary tree formed by hash pointers, and named after its creator, Ralph Merkle.

- As mentioned earlier, each block is supposed to hold a certain number of transactions. Now the question arises, how to store these transactions within a block? One approach can be to form a hash pointer-based linked list of transactions and store this complete linked list in a block. However, when we put this approach into perspective, it does not seem practical to store a huge list of hundreds of transactions. What if there is a need to find whether a particular transaction belongs to a block? Then we will have to traverse the blocks one by one and within each block traverse the linked list of transactions.
- This is a huge overhead and can reduce the efficiency of the blockchain. Now, this is where the Merkle tree comes into the picture. Merkle tree is a per-block tree of all the transactions that are included in the block. It allows us to have a hash/digest of all transactions and provides proof of membership in a time-efficient manner.
- So to recap, the blockchain is a hash-based linked list of blocks, where each block consists of a header and transactions. The transactions are arranged in a tree-like fashion, known as the Merkle tree.



*Each block comprises of block header + Merkle tree*

## Merkle Tree Structure




*Structure of Merkle tree*

**1.** A blockchain can potentially have thousands of blocks with thousands of transactions in each block. Therefore, memory space and computing power are two main challenges.

**2.** It would be optimal to use as little data as possible for verifying transactions, which can reduce CPU processing and provide better security, and this is exactly what Merkle trees offer.

**3.** In a Merkle tree, transactions are grouped into pairs. The hash is computed for each pair and this is stored in the parent node. Now the parent nodes are grouped into pairs and their hash is stored one level up in the tree. This continues till the root of the tree. The different types of nodes in a Merkle tree are:

- **Root node:** The root of the Merkle tree is known as the Merkle root and this Merkle root is stored in the header of the block.
- **Leaf node:** The leaf nodes contain the hash values of transaction data. Each transaction in the block has its data hashed and then this hash value (also known as transaction ID) is stored in leaf nodes.
- **Non-leaf node:** The non-leaf nodes contain the hash value of their respective children. These are also called intermediate nodes because they contain the intermediate hash

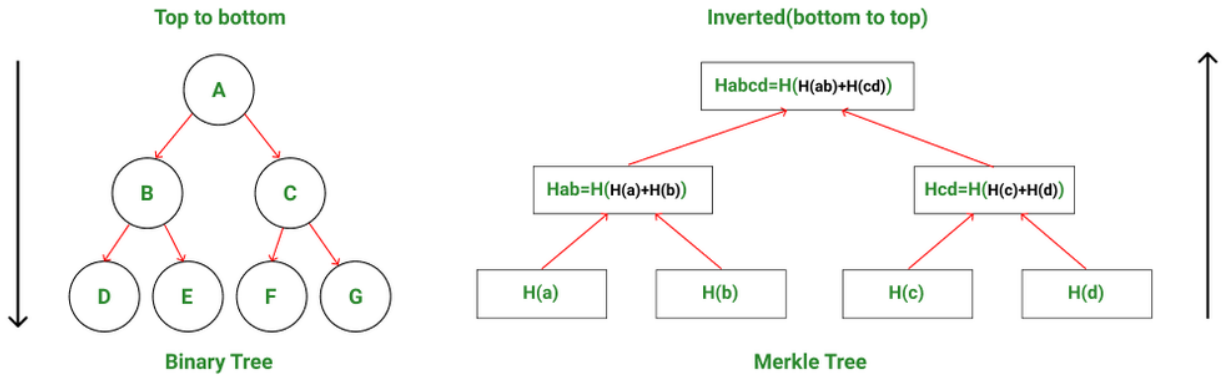
values and the hash process continues till the root of the tree.

**4.** Bitcoin uses the SHA-256 hash function to hash transaction data continuously till the Merkle root is obtained.

**5.** Further, a Merkle tree is **binary in nature**. This means that the **number of leaf nodes needs to be even** for the Merkle tree to be constructed properly. In case there is an odd number of leaf nodes, the tree duplicates the last hash and makes the number of leaf nodes even.

### How Do Merkle Trees Work?

- A Merkle tree is constructed from the leaf nodes level all the way up to the Merkle root level by grouping nodes in pairs and calculating the hash of each pair of nodes in that particular level. This hash value is propagated to the next level. This is a **bottom-to-up** type of construction where the hash values are flowing from down to up direction.
- Hence, by comparing the Merkle tree structure to a regular binary tree data structure, one can observe that Merkle trees are actually **inverted down**.




*Binary tree direction vs Merkle tree direction*

**Example:** Consider a block having 4 transactions- T1, T2, T3, T4. These four transactions have to be stored in the Merkle tree and this is done by the following steps-

**Step 1:** The hash of each transaction is computed.

> ***H1 = Hash(T1).***

**Step 2:** The hashes computed are stored in leaf nodes of the Merkle tree.

**Step 3:** Now non-leaf nodes will be formed. In order to form these nodes, leaf nodes will be paired together from left to right, and the hash of these pairs will be calculated. Firstly hash of

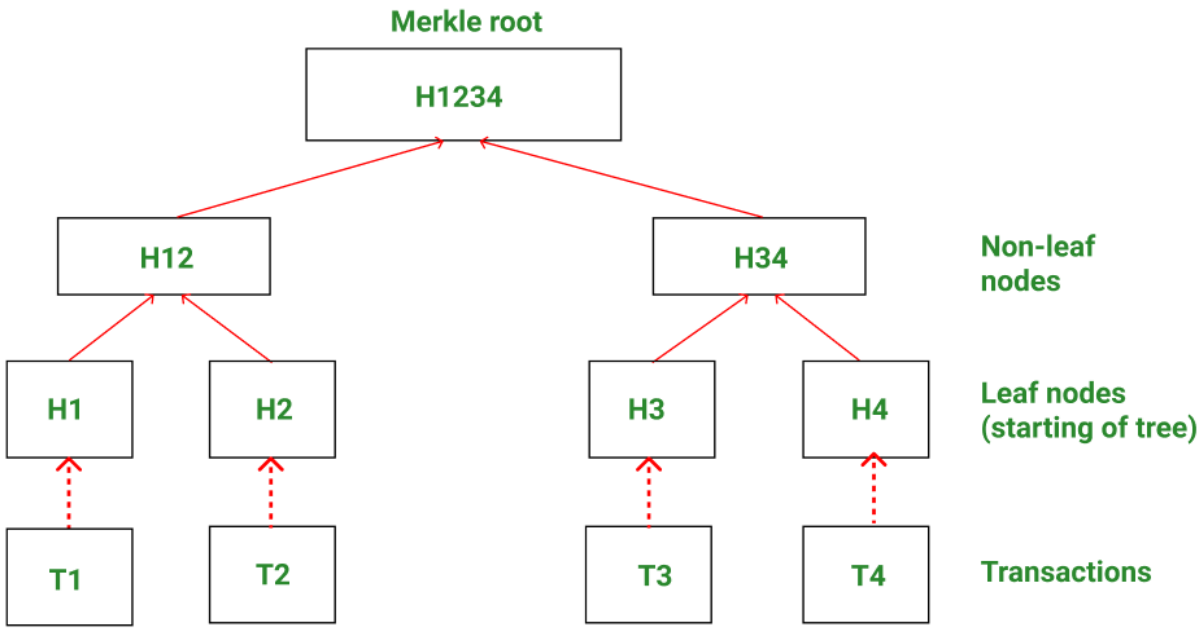
H1 and H2 will be computed to form H12. Similarly, H34 is computed. Values H12 and H34 are parent nodes of H1, H2, and H3, H4 respectively. These are non-leaf nodes.

***H12 = Hash(H1 + H2)***

***H34 = Hash(H3 + H4)***

**Step 4:** Finally H1234 is computed by pairing H12 and H34. H1234 is the only hash remaining. This means we have reached the root node and therefore H1234 is the Merkle root.

***H1234 = Hash(H12 + H34)***




*Merkle tree works by hashing child nodes again and again till only one hash remains.*

**Key Points:**

- In order to check whether the transaction has tampered with the tree, there is only a need to remember the root of the tree.
- One can access the transactions by traversing through the hash pointers and if any content has been changed in the transaction, this will reflect on the hash stored in the parent node, which in turn would affect the hash in the upper-level node and so on until the root is reached.
- Hence the root of the Merkle tree has also changed. So Merkle root which is stored in the block header makes transactions tamper-proof and validates the integrity of data.

- With the help of the Merkle root, the Merkle tree helps in eliminating duplicate or false transactions in a block.
- It generates a digital fingerprint of all transactions in a block and the Merkle root in the header is further protected by the hash of the block header stored in the next block.

### Why Merkle Trees are Important For Blockchain?

- In a centralized network, data can be accessed from one single copy. This means that nodes do not have to take the responsibility of storing their own copies of data and data can be retrieved quickly.
- However, the situation is not so simple in a distributed system.
- Let us consider a scenario where blockchain does not have Merkle trees. In this case, every node in the network will have to keep a record of every single transaction that has occurred because there is no central copy of the information.
- This means that a huge amount of information will have to be stored on every node and every node will have its own copy of the ledger. If a node wants to validate a past transaction, requests will have to be sent to all nodes, requesting their copy of the ledger. Then the user will have to compare its own copy with the copies obtained from several nodes.

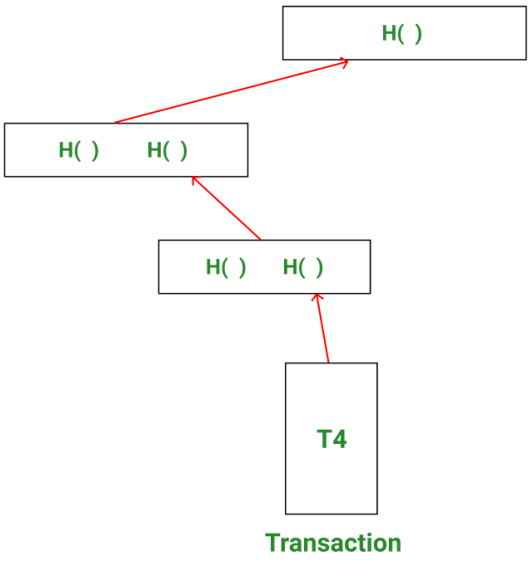
- Any mismatch could compromise the security of the blockchain. Further on, such verification requests will require huge amounts of data to be sent over the network, and the computer performing this verification will need a lot of processing power for comparing different versions of ledgers.
- Without the Merkle tree, the **data itself has to be transferred all over the network** for verification.
- Merkle trees allow comparison and verification of transactions with **viable computational power and bandwidth**. Only a small amount of information needs to be sent, hence compensating for the huge volumes of ledger data that had to be exchanged previously.

> *Merkle trees use a one-way hash function extensively and this hashing separates the proof of data from data itself*

### Proof of Membership

A very interesting feature of the Merkle tree is that it provides **proof of membership**.

**Example:** A miner wants to prove that a particular transaction belongs to a Merkle tree Now the miner needs to present this transaction and all the nodes which lie on the path between the transaction and the root. The rest of the tree can be ignored because the hashes stored in the intermediate nodes are enough to verify the hashes all the way up to the root.




*Proof of membership: verifying the presence of transactions in blocks using the Merkle tree.*

If there are n nodes in the tree then only log(n) nodes need to be examined. Hence even if there are a large number of nodes in the Merkle tree, proof of membership can be computed in a relatively short time.

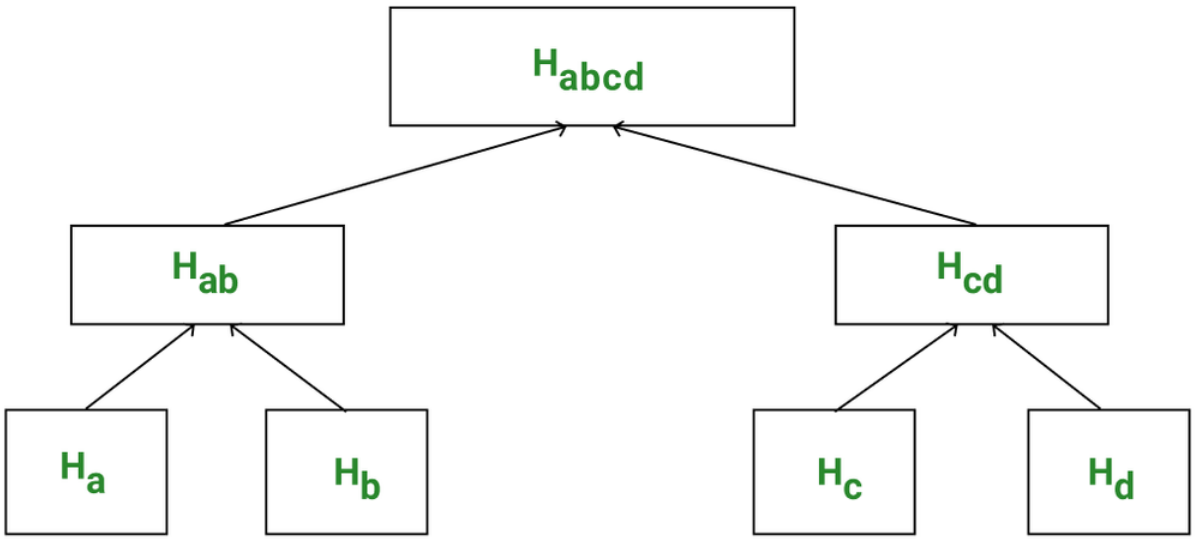
## Merkle Proofs

A Merkle proof is used to decide:

1. If data belongs to a particular Merkle tree.
2. To prove data belongs to a set without the need to store the whole set.
3. To prove a certain data is included in a larger data set without revealing the larger data set or its subsets.

> *Merkle proofs are established by hashing a hash's corresponding hash together and climbing up the tree until you obtain the root hash which is or can be publicly known.*

Consider the Merkle tree given below:




Let us say we need to prove that transaction 'a' is part of this Merkle tree. Everyone in the network will be aware of the hash function used by all Merkle trees.

1. H(a) = Ha as per the diagram.
2. The hash of Ha and Hb will be Hab, which will be stored in an upper-level node.
3. Finally hash of Hab and Hcd will give Habcd. This is the Merkle root obtained by us.
4. By comparing the obtained Merkle root and the Merkle root already available within the block header, we can verify the presence of transaction 'a' in this block.

From the above example, it is clear that in order to verify the presence of 'a', 'a' does not have to be revealed nor do 'b', 'c', 'd' have to be revealed, only their hashes are sufficient. Therefore Merkle proof provides an efficient and simple method of verifying inclusivity, and is synonymous with "proof of inclusion".
A **sorted Merkle tree** is a tree where all the data blocks are ordered using an ordering function. This ordering can be alphabetical, lexicographical, numerical, etc.

**Proof of Non-Membership:**

- It is also possible to test non-membership in logarithmic time and space using a sorted Merkle tree. That is, it is possible to show that a given transaction does not belong in the Merkle tree.
- This can be done by displaying a path to the transaction that is immediately before the transaction in question, as well as a path to the item that is immediately following it.
- If these two elements in the tree are sequential, this proves that the item in issue is not included or else it would have to go between the two things shown if it was included, but there is no room between them because they are sequential.

**Coinbase Transaction:**

A coinbase transaction is a unique Bitcoin transaction that is included in the Merkle tree of every block in the blockchain. It is responsible for creating new coins and also consists of a coinbase parameter that can be used by miners to insert arbitrary data into the blockchain.

### Simple Payment Verification(SPV)

- SPV makes it extremely easy for a client to verify whether a particular transaction exists in a block and is valid **without having to download the entire blockchain.** The users will only require a copy of the block headers of the longest chain.
- This copy of headers is stored in the SPV wallet and this wallet uses the SPV client to link a transaction to a Merkle branch in a block. SPV client requests **proof of inclusion(Merkle proof),** in the form of a Merkle branch. The fact that the transaction can be linked to a Merkle branch is proof that the transaction exists.
- Now by assessing the blocks which are being mined on top of the transaction's block, the client can also conclude that majority of the nodes have built more blocks on top of this chain by using consensus mechanisms like Proof of Work, and hence this is the longest, valid blockchain.

### Advantages of Merkle Tree

1. **Efficient verification:** Merkle trees offer efficient verification of integrity and validity of data and significantly reduce the amount of memory required for verification. The proof of verification does not require a huge amount of data to be transmitted across the blockchain network. Enable trustless transfer of cryptocurrency in the peer-to-peer, distributed system by the quick verification of transactions.
2. **No delay:** There is no delay in the transfer of data across the network. Merkle trees are extensively used in computations that maintain the functioning of cryptocurrencies.
3. **Less disk space:** Merkle trees occupy less disk space when compared to other data structures.
4. **Unaltered transfer of data:** Merkle root helps in making sure that the blocks sent across the network are whole and unaltered.
5. **Tampering Detection:** Merkle tree gives an amazing advantage to miners to check whether any transactions have been tampered with.
    - Since the transactions are stored in a Merkle tree which stores the hash of each node in the upper parent node, any changes in the details of the transaction such as the amount to be debited or the address to whom the payment must be made, then the change will propagate to the hashes in upper levels and finally to the Merkle root.
    - The miner can compare the Merkle root in the header with the Merkle root stored in the data part of a block and can easily detect this tampering.

6. **Time Complexity:** Merkle tree is the best solution if a comparison is done between the time complexity of searching a transaction in a block as a Merkle tree and another block that has transactions arranged in a linked list, then-
   - **Merkle Tree search:** O(logn), where n is the number of transactions in a block.
   - **Linked List search:** O(n), where n is the number of transactions in a block.



## Related Articles

1. Implementation of Blockchain in Java
2. Flutter and Blockchain - Hello World Dapp
3. Introduction to Blockchain technology | Set 2
4. What is Blockchain Wallet?
5. Smart Contracts in Blockchain
6. Blockchain Forks
7. Important Blockchain terminologies
8. Benefits of Blockchain Technology
9. Types of Blockchain and Chain Terminology
10. Blockchain and Data Privacy

Next ›

Important Blockchain terminologies

## Article Contributed By :

yashi4001
@yashi4001

## Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

**Improved By :** rashi_garg

**Article Tags :** Picked, Blockchain

Improve Article | Report Issue