BLOCKCHAIN

EXPLAINER

Merkle Trees

#BLOCKCHAIN
#MERKLETREE
#HASHES

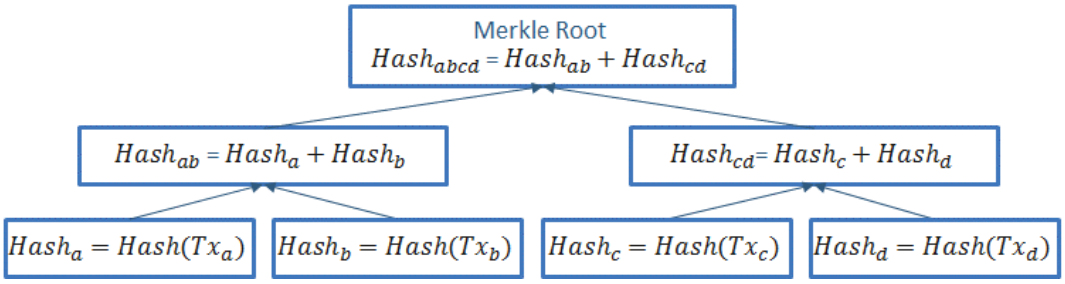
A Merkle Tree is a branching data structure that contains cryptographic hashes. It is constructed by hashing pairs of nodes until there is only one hash. This hash is called the Merkle root.

Features of Merkle Trees

Merkle Trees are efficient tools to determine if an element exists in a set without needing to store the set. In a blockchain, Merkle Trees are used to create a digital fingerprint that summarizes a set of transactions in a given block. It is important to note that Merkle Trees store hashes (or double hashes, in the case of Bitcoin) of a transaction's data rather than the transaction data itself.

Structure of the Tree:

The Merkle Tree is constructed recursively, in that the solution (the **Merkle Root**) relies on solutions to smaller instances of the same problem. The Merkle Root is solved by first hashing individual transactions; hashed transactions are called **leaves or leaf nodes** and are displayed at the bottom of the diagram below. Then, siblings or consecutive pairs of leaves are hashed to create **nodes or non-leaf nodes**. For example, to construct the parent node $Hash_{ab}$ the hashes of the children are concatenated to create a 64-byte string which is then hashed to create the parent node's hash. Finally, sibling nodes are hashed until there is a single node left which is called the **Merkle Root**. Although a Merkle Tree can be created from any number of transaction, the Merkle Root will always be 32-bytes (SHA256 outputs a fixed length 32-byte hash regardless of the input).



Balanced Trees:

Due to the fact that Merkle Trees are binary trees, they require an even set of leaves. In the event of an odd number of transactions, the last transaction's hash will essentially be duplicated to create an even number of leaf nodes. When this occurs, the tree may be referred to as a **balanced tree**.

Proving an element is included in the Tree:

Merkle Trees are efficient ways to prove that an element is included in the set without storing that set. One can check if an element (node or leaf) is included in the Merkle Tree by performing at most 2*log2(N) calculations. This set of calculations, which connects the element to the Merkle Root is called a **Merkle Path**. Since non-leaf nodes are hashes of the concatenation of its child nodes, in order to proof that a given element exists, one only needs the sibling nodes of all the elements in the Merkle Path. For example, to prove $Hash_{ab}$ exists in $Hash_{abcd}$, one only needs the sibling node: $Hash_{cd}$, to perform the calculation and so $Hash_c$ and $Hash_d$ become irrelevant. The efficiency of Merkle Tree becomes more apparent in larger Merkle Trees.

Double Hashing:

Some blockchains such as bitcoin prefer to double hash each element, bitcoin, such that $Hash_a = Hash(Hash(Tx_a))$

Sources: http://chimera.labs.oreilly.com/books/1234000001802/ch07.html/