---

title: Verkle trees description: A high level description of Verkle trees and how they will be used to upgrade Ethereum lang: en summaryPoints: - Discover what Verkle trees are - Read why Verkle Trees are a useful upgrade for Ethereum

---

Verkle trees {#verkle-trees}

Verkle trees (a portmanteau of "Vector commitment" and "Merkle Trees") are a data structure that can be used to upgrade Ethereum nodes so that they can stop storing large amounts of state data without losing the ability to validate blocks.

Statelessness {#statelessness}

Verkle trees are a critical step on the path to stateless Ethereum clients. Stateless clients are ones that do not have to store the entire state database in order to validate incoming blocks. Instead of using their own local copy of Ethereum's state to verify blocks, stateless clients use a "witness" to the state data that arrives with the block. A witness is a collection of individual pieces of the state data that are required to execute a particular set of transactions, and a cryptographic proof that the witness is really part of the full data. The witness is used *instead* of the state database. For this to work, the witnesses need to be very small, so that they can be safely broadcast across the network in time for validators to process them within a 12 second slot. The current state data structure is not suitable because witnesses are too large. Verkle trees solve this problem by enabling small witnesses, removing one of the main barriers to stateless clients.

Ethereum clients currently use a data structure known as a Patricia Merkle Trie to store its state data. Information about individual accounts are stored as leaves on the trie and pairs of leaves are hashed repeatedly until only a single hash remains. This final hash is known as the "root". To verify blocks, Ethereum clients execute all the transactions in a block and update their local state trie. The block is considered valid if the root of the local tree is identical to the one provided by the block proposer, because any differences in the computation done by the block proposer and the validating node would cause the root hash to be completely different. The problem with this is that verifying the blockchain requires each client to store the whole state trie for the head block and several historical blocks (the default in Geth is to keep state data for 128 blocks behind the head). This requires clients to have access to a large amount of disk space, which is a barrier to running full nodes on cheap, low power hardware. A solution to this is to update the state trie to a more efficient structure (Verkle tree) that can be summarized using a small "witness" to the data that can be shared instead of the full state data. Reformatting the state data into a Verkle tree is a stepping stone for moving to stateless clients.

What is a witness and why do we need them? {#what-is-a-witness}

Verifying a block means re-executing the transactions contained in the block, applying the changes to Ethereum's state trie, and calculating the new root hash. A verified block is one whose computed state root hash is the same as the one provided with the block (because this means the block proposer really did the computation they say they did). In today's Ethereum clients, updating the state requires access to the entire state trie, which is a large data structure that must be stored locally. A witness only contains the fragments of the state data that are required to execute the transactions in the block. A validator can then only use those fragments to verify that the block proposer has executed the block transactions and updated the state correctly. However, this means that the witness needs to be transferred between peers on the Ethereum network rapidly enough to be received and processed by each node safely within a 12 second slot. If the witness is too large, it might take some nodes too long to download it and keep up with the chain. This is a centralizing force because it means only nodes with fast internet connections can participate in validating blocks. With Verkle trees there is no need to have the state stored on your hard drive; *everything* you need to verify a block is contained within the block itself. Unfortunately, the witnesses that can be produced from Merkle tries are too large to support stateless clients.

Why do Verkle trees enable smaller witnesses? {#why-do-verkle-trees-enable-smaller-witnesses}

The structure of a Merkle Trie makes witness sizes very large - too large to safely broadcast between peers within a 12 second slot. This is because the witness is a path connecting the data, which is held in leaves, to the root hash. To verify the data it is necessary to have not only all the intermediate hashes that connect each leaf to the root, but also all the

"sibling" nodes. Each node in the proof has a sibling that it is hashed with to create the next hash up the trie. This is a lot of data. Verkle trees reduce the witness size by shortening the distance between the leaves of the tree and its root and also eliminating the need to provide sibling nodes for verifying the root hash. Even more space efficiency will be gained by using a powerful polynomial commitment scheme instead of the hash-style vector commitment. The polynomial commitment allows the witness to have a fixed size regardless of the number of leaves that it proves.

Under the polynomial commitment scheme, the witnesses have manageable sizes that can easily be transferred on the peer-to-peer network. This allows clients to verify state changes in each block with a minimal amount of data.

The witness size varies depending on the number of leaves it includes. Assuming the witness covers 1000 leaves, a witness for a Merkle trie would be about 3.5MB (assuming 7 levels to the trie). A witness for the same data in a Verkle tree (assuming 4 levels to the tree) would be about 150 kB - **about 23x smaller**. This reduction in witness size will allow stateless client witnesses to be acceptably small. Polynomial witnesses are 0.128 -1 kB depending on which specific polynomial commitment is used).

What is the structure of a Verkle tree? {#what-is-the-structure-of-a-verkle-tree}

Verkle trees are `(key,value)` pairs where the keys are 32-byte elements composed of a 31-byte*stem* and a single byte *suffix*. These keys are organized into*extension* nodes and *inner* nodes. Extension nodes represent a single stem for 256 children with different suffixes. Inner nodes also have 256 children, but they can be other extension nodes. The main difference between the Verkle tree and the Merkle tree structure is that the Verkle tree is much flatter, meaning there are fewer intermediate nodes linking a leaf to the root, and therefore less data required to generate a proof.

Read more about the structure of Verkle trees

Current progress {#current-progress}

Verkle tree testnets are already up and running, but there are still substantial outstanding updates to clients that are required to support Verkle trees. You can help accelerate progress by deploying contracts to the testnets or running testnet clients.

Explore the Beverly Hills Verkle testnet

Watch Guillaume Ballet explain the Condrieu Verkle testnet(note that the Condrieu testnet was proof-of-work and has now been superseded by the Kaustinen testnet).

Further reading {#further-reading}

- Dankrad Feist explain Verkle trees on PEEPanEIP
- Guillaume Ballet explain verkle trees at ETHGlobal
- "How Verkle trees make Ethereum lean and mean" by Guillaume Ballet at Devcon 6
- Piper Merriam on stateless clients from ETHDenver 2020
- Dankrad Fiest explan Verkle trees and statelessness on Zero Knowledge podcast
- Vitalik Buterin on Verkle trees
- Dankrad Feist on Verkle trees
- Verkle tree EIP documentation