Merkle Trees

Taylor Blair

March 21, 2024

Abstract

Merkle trees are secured data structures whose operations generate proofs that use $\mathcal{O}(\log(n))$ hashes. The runtime complexity combined with other properties related to trees makes it a useful component in several open-source projects.

Contents

1 Introduction

Merkle trees were invented by Ralph Merkle. Ralph Merkle initially patented Merkle trees for digital signatures. After the patent expired Merkle trees were implemented in a variety of open source projects.

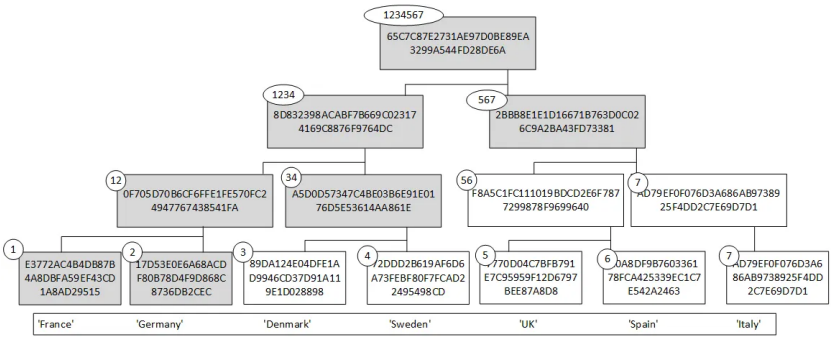


Figure 1: Basic Merkle Tree[6]

Merkle trees are a data structure where the node values represent a hash of the nodes below. They have several desirable properties related to the runtime of operations on the Merkle trees.

The security of Merkle trees is related to the ability to collide against the hash function. As such, a strong hash function is required for constructing the Merkle trees.

1.1 History

1987 The patent a 'Method of providing digital signatures' is filed by Ralph C. Merkle[5].

1999 The original patent expires.

2009 Bitcoin uses Merkle Trees for 'block header commitment.'[4]

2009 BitTorent uses Merkle Trees for data integrity[1].

As stated above, Merkle trees were created in 1987 by Ralph Merkle for digital signatures[5]. In the initial patent, the nodes were used for signing and verification:

> To sign a message M_i the signer selects a previously unused node (i.e., node i) from the infinite tree. The message signing key at this node is then used to sign this message. The sequence of nodes from the root of the tree (i.e., node 1) to node i is then used to verify that the message signature is correct and has not been tampered with.[5]

Once the Merkle tree patent expired Merkle trees were implemented in various open-source projects including Bitcoin and BitTorrent which is expanded upon in a section below.

2 Literature Review

Merkle Trees are a component of several projects, as such many papers provide incremental changes towards certain operations on Merkle trees. This paper references the original patent by Ralph Merkle [5] in addition to descriptions of Merkle tree operations given by Boneh and Shoup [2]. As secondary sources, the implementation of Merkle trees in Bitcoin [4] provides a real example of the impact of hash functions in addition to a whitepaper from the BitTorrent project[1].

3 Construction

Merkle trees are constructed from the bottom up by hashing data as the leaf nodes.

Algorithm 1 Merkle tree construction

for $i = 1, \ldots, n$ **do** ▷ Compute leaf node hashes

$y_i \leftarrow h(x_i)$

end for

for $j = 1, \ldots, n-1$ **do** ▷ Compute intermediate Nodes from $y_{n+1}, \ldots, y_{2n-1}$

$y_{i+n} \leftarrow h(y_{2i-1}, y_{2i})$ ▷ Hash leaf nodes below for new hash

end for

return Y ▷ Return tree where y_{2n-1} is the root

When referring to the parts of a Merkle tree the most common terminology is "root hash" which refers to the hashed value of the root of the tree and "leaf hash" which refers to the hash for a given data block.

4 Operations

There are several operations can for Merkle trees unrelated to tree manipulation that generate a proof resulting in a root hash. The root hash from the proof is then verified against a known hash to check for correctness. The most common operations are as follows.

4.1 Proving Equality of Merkle Trees

```
     ROOT1                ROOT2
    /     \              /     \
   L1       R1          L2       R2
  /  \     /  \        /  \     /  \
L1.1  ....  R1.2     L2.1 ... R2.2
```

Proving the equality of two Merkle trees is an $O(1)$ operation. The operation compares the root nodes of the two trees. If the root nodes match, then the trees are equal.

4.2 Proving Membership

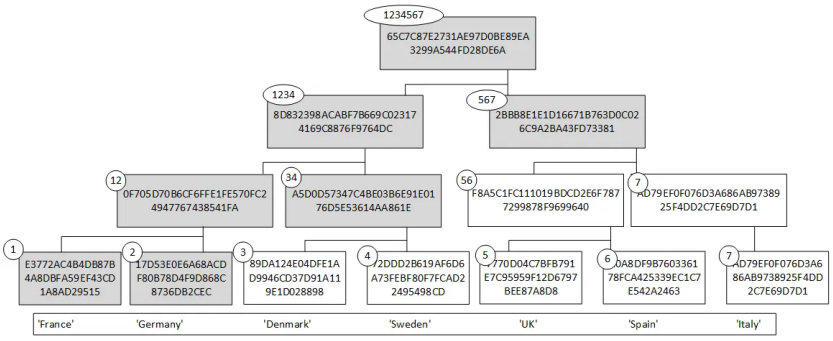


Figure 2: Show "*Germany*" exist in the tree[3]

Using the figure above as an example, to prove "*Germany*" is a member of the tree the verifier hashes "*Germany*" and hashes it with its sibling node to find the parent node. The parent node is then hashed with its sibling and the process repeats until

the root node. The generated root node is compared with the known root node, should the hashes match then the element is verified. Proving membership requires checking $\log(n)$ tree levels, thus the operation takes $O(\log(n))$ time.

Another advantage to checking membership using a Merkle tree is that checking the membership of multiple leaf nodes can be done in $O(\log(n))$. In the example above by proving "*Germany*" was a member of the tree, the verifier also verified that "*France*" was a member of the tree.

4.3 Proving Non-Membership

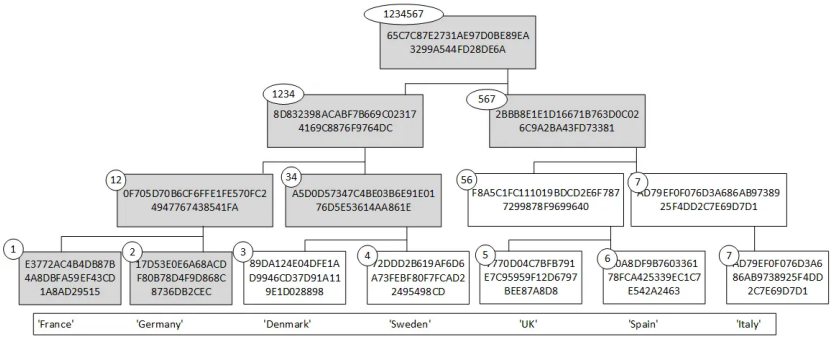


Figure 3: Show x is not in the tree[2]

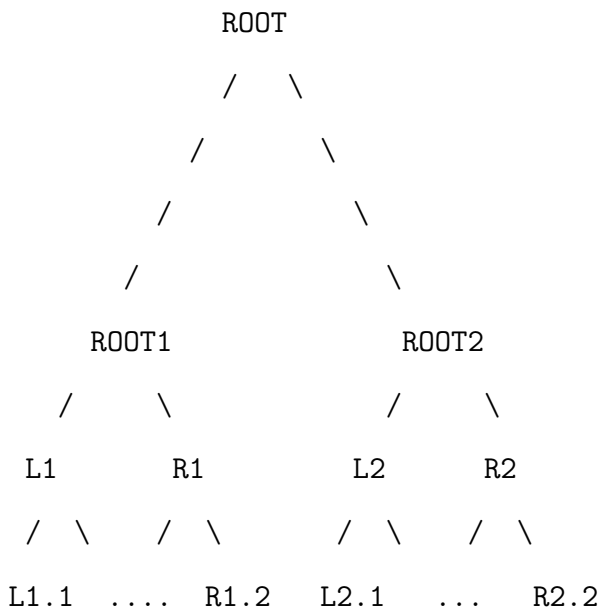
While hashing a value and checking if it appears in the leaf nodes might appear to Non membership cannot be verified using a standard Merkle tree. Non membership can only be proven with a sorted Merkle tree. Dan Boneh and Victor Shoup provide the following algorithm:

> To produce the proof, the prover first locates the two adjacent leaves x_i and x_{i+1}in T that bracket x, namely $x_i < x < x_{i+1}$. Next, the prover

provides a Merkle proof that x_i is in position i in T, and that x_{i+1} is in position $i+1$ in T. The verifier can check that these two leaves are adjacent, and that $x_i < x < x_{i+1}$, and this proves that x is not in T.

This operation takes $O(\log(n))$, but the additional requirement that the Merkle tree be sorted limits what operations can be used on the Merkle tree, notably merging.

4.4 Merging Merkle Trees



Merging Merkle trees is an $O(1)$ operation of the root of two other Merkle trees and creates a new root node. This operation can be used to add individual nodes to a Merkle tree, but it creates an unbalanced tree.

5 Security

5.1 Authenticated Data Structure

As defined by Boneh and Soup, Merkle Trees are secure data structures composed of three algorithms $D = (H, P, V)$[2]:

- H a secure hash function
- P a proof function that takes an element, location, and structure (i, x, T) and returns a proof of a given element
- V a verification algorithm $V(i, x, y, \pi)$ that "accepts" or "rejects" a proof.

5.2 Merkle Tree Attack Game

The attack game for Merkle trees is defined in Boneh and Shoup using the definition of an authenticated data structure. Given a Merkle tree $D = (H, P, V)$ defined over $(\mathscr{X}^n, \mathscr{Y})$, and a given adversary $\mathscr{A}$:

> The adversary A outputs a $y \in \mathscr{Y}$, a position $i \in \{1, \ldots, n\}$, and two pairs (x, π) and (x', π') where $x, x' \in \mathscr{X}$.

$\mathscr{A}$ wins the game if $x \neq x'$ and $V(i, x, y, \pi) = V(i, x', y, \pi') =$accept. Define $\mathscr{A}$'s advantage with respect to $\mathscr{D}$, denoted ADSadv$[\mathscr{A}, \mathscr{D}]$, as the probability that $\mathscr{A}$ wins the game[2].

5.3 Security Proof

Using the definition of the authenticated data structure combined with the attack game above, the security of the Merkle tree is dependent on the underlying hash function.

The goal of the attacker is to generate a proof of membership at a given location. This can be reduced to creating a hash collision where the root node will be the same.

In the case of a Merkle tree with only one leaf node, the attacker would need to generate a collision c such that if $H(x) = x'$, then $H(c) = x'$ but $x \neq c$.

In the case of a Merkle tree with two leaf nodes, the attacker can create a collision by colliding on the hash of the leaf node or by colliding against the root node. Let a, b be the data blocks and let $H(a) = a', H(b) = b'$ which gives a root node $H(a'b') = r$. The attacker can either find $H(c) = c'$ such that $c' = a'$, $c' = b'$ or $H(c'b') = r$, $H(a'c') = r$. All attacks depend on a flaw in the hash function.

More generally, $x \in T$ with hash $H(x) = h$ at position i. Let the attacker create a collision such that $H(y) = h$ but $x \neq y$. Thus the attacker's advantage against a Merkle tree is the same as the attacker's advantage against the hash function used by the Merkle tree.

6 Use Cases

Merkle trees are used as authenticated data structures with optimal complexity for proving membership and comparing against other structures.

6.1 Bitcoin

Bitcoin, and other cryptocurrencies, use Merkle trees for the commitment header. The *chain* in blockchain refers to the chain of receipts from processed transactions. It would be too slow to create a receipt for each transaction, so Bitcoin groups transactions into sections. The smaller group of transactions is a Merkle tree where the leaf nodes represent a single transaction. Thus the group of transactions results in a root hash which is used as the commitment header.

Merkle trees were incorrectly implemented in the Bitcoin protocol. This implementation resulted in DOS attacks due to over-hashing and duplicate nodes (CVE-2012-2459). This bug was patched in a proposal that replaced the Merkle tree implementation[4].

6.2 BitTorrent Data Integrity

BitTorrent uses Merkle trees for checking the integrity of torrented files. The torrented data is hashed into a Merkle Tree and the root hash is compared with a trusted peer to verify integrity[1]. If the root hashes do not match then the nodes below are compared with the trusted peer. This process repeats until a non-matching block is reached. The advantage of using a Merkle tree over another data structure is finding corrupted data blocks in $O(\log(n))$ time.

7 Closing

Merkle trees were not the first authenticated data structure, the Merkle Damgard construction provides several of the same operations [2], but the number of properties in Merkle trees makes them desirable across a multitude of applications. The core construct of Merkle Trees, the tree, enables $O(\log(n))$ time operations in addition to speed-ups for proofs of many members. Their usage in various applications, notably in most cryptocurrencies, serves as evidence that Merkle trees will continue to propagate.

References

Bakker, A. (2009). Bep 0030: Merkle hash torrent extension [[Online; accessed 4-May-2023]].

Boneh, D., & Shoup, V. (2020). A graduate course in applied cryptography. *Draft 0.5*.

Buchannen, B. (2022, January). Bloom filters, merkle trees and... accumulators. https://medium.com/asecuritysite-when-bob-met-alice/bloom-filters-merkle-trees-and-accumulators-27bc2f7baf5a

Friedenbach, M., & Alm, K. (2017, August). Fast merkle trees proposal. https://github.com/bitcoin/bips/blob/master/bip-0098.mediawiki

Merkle, R. C. (1979). Method of providing digital signatures. *Patent US4309569A*.

Wikipedia contributors. (2022). Merkle tree — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Merkle_tree&oldid=1123544588