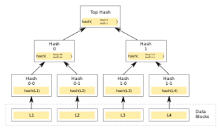
WIKIPEDIA

Merkle tree

In cryptography and computer science, a **hash tree** or **Merkle tree** is a tree in which every leaf node is labelled with the cryptographic hash of a data block, and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. Hash trees allow efficient and secure verification of the contents of large data structures. Hash trees are a generalization of hash lists and hash chains.



An example of a binary hash tree. Hashes 0-0 and 0-1 are the hash values of data blocks L1 and L2, respectively, and hash 0 is the hash of the concatenation of hashes 0-0 and 0-1.

Demonstrating that a leaf node is a part of a given binary hash tree requires computing a number of hashes proportional to the logarithm of the number of leaf nodes of the tree;[1] this contrasts with hash lists, where the number is proportional to the number of leaf nodes itself.

The concept of hash trees is named after Ralph Merkle who patented it in 1979.[2][3]

Contents

Uses

Hash trees can be used to verify any kind of data stored, handled and transferred in and between computers. They can help ensure that data blocks received from other peers in a peer-to-peer network are received undamaged and unaltered, and even to check that the other peers do not lie and send fake blocks.

Hash trees are used in hash-based cryptography. Hash trees are also used in the IPFS, Btrfs and ZFS file systems[4] (to counter data degradation[5]); Dat protocol; Apache Wave protocol;[6] Git and Mercurial distributed revision control systems; the Tahoe-LAFS backup system; Zeronet; the Bitcoin and Ethereum peer-to-peer networks;[7] the Certificate Transparency framework; and a number of NoSQL systems such as Apache Cassandra, Riak, and Dynamo.[8] Suggestions have been made to use hash trees in trusted computing systems.[9]

The initial Bitcoin implementation of Merkle trees by Satoshi Nakamoto applies the compression step of the hash function to an excessive degree, which is mitigated by using Fast Merkle Trees.[10]

Overview

A hash tree is a tree of hashes in which the leaves are hashes of data blocks in, for instance, a file or set of files. Nodes further up in the tree are the hashes of their respective children. For example, in the picture *hash 0* is the result of hashing the concatenation of *hash 0-0* and *hash 0-1*. That is, *hash 0 = hash(hash(0-0) + hash(0-1))* where + denotes concatenation.

Most hash tree implementations are binary (two child nodes under each node) but they can just as well use many more child nodes under each node.

Usually, a cryptographic hash function such as SHA-2 is used for the hashing. If the hash tree only needs to protect against unintentional damage, unsecured checksums such as CRCs can be used.

In the top of a hash tree there is a *top hash* (or *root hash* or *master hash*). Before downloading a file on a p2p network, in most cases the top hash is acquired from a trusted source, for instance a friend or a web site that is known to have good recommendations of files to download. When the top hash is available, the hash tree can be received from any non-trusted source, like any peer in the p2p network. Then, the received hash tree is checked against the trusted top hash, and if the hash tree is damaged or fake, another hash tree from another source will be tried until the program finds one that matches the top hash.[11]

The main difference from a hash list is that one branch of the hash tree can be downloaded at a time and the integrity of each branch can be checked immediately, even though the whole tree is not available yet. For example, in the picture, the integrity of *data block L2* can be verified immediately if the tree already contains *hash 0-0* and *hash 1* by hashing the data block and iteratively combining the result with *hash 0-0* and then *hash 1* and finally comparing the result with the *top hash*. Similarly, the integrity of *data block L3* can be verified if the tree already has *hash 1-1* and *hash 0*. This can be an advantage since it is efficient to split files up in very small data blocks so that only small blocks have to be re-downloaded if they get damaged. If the hashed file is very big, such a hash tree or hash list becomes fairly big. But if it is a tree, one small branch can be downloaded quickly, the integrity of the branch can be checked, and then the downloading of data blocks can start.

Second preimage attack

The Merkle hash root does not indicate the tree depth, enabling a second-preimage attack in which an attacker creates a document other than the original that has the same Merkle hash root. For the example above, an attacker can create a new document containing two data blocks, where the first is hash 0-0 + hash 0-1, and the second is hash 1-0 + hash 1-1.[12][13]

One simple fix is defined in Certificate Transparency: when computing leaf node hashes, a 0x00 byte is prepended to the hash data, while 0x01 is prepended when computing internal node hashes.[11] Limiting the hash tree size is a prerequisite of some formal security proofs, and helps in making some proofs tighter. Some implementations limit the tree depth using hash tree depth prefixes before hashes, so any extracted hash chain is defined to be valid only if the prefix decreases at each step and is still positive when the leaf is reached.

Tiger tree hash

The Tiger tree hash is a widely used form of hash tree. It uses a binary hash tree (two child nodes under each node), usually has a data block size of 1024 bytes and uses the Tiger hash.[14]

Tiger tree hashes are used in Gnutella,[15] Gnutella2, and Direct Connect P2P file sharing protocols[16] and in file sharing applications such as Phex,[17] BearShare, LimeWire, Shareaza, DC++[18] and Valknut.[19]

Example

Base32: `R5T6Y8UYRYO5SUXGER5NMUOEZ5O6E4BHPP2MRFQ`

URN: `urn:tree:tiger:R5T6Y8UYRYO5SUXGER5NMUOEZ5O6E4BHPP2MRFQ`

magnet: `magnet:?xt=urn:tree:tiger:R5T6Y8UYRYO5SUXGER5NMUOEZ5O6E4BHPP2MRFQ`

See also

- Binary tree
- Blockchain (database)
- Distributed hash table
- Hash table
- Hash trie
- Linked timestamping
- Radix tree

References

1. Becker, Georg (2008-07-18). "Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis" (http://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker_1.pdf) (PDF). Ruhr-Universität Bochum. p. 16. Retrieved 2013-11-20.
2. Merkle, R. C. (1988). "A Digital Signature Based on a Conventional Encryption Function". *Advances in Cryptology — CRYPTO '87*. Lecture Notes in Computer Science. **293**. pp. 369–378. doi:10.1007/3-540-48184-2_32 (https://doi.org/10.1007%2F3-540-48184-2_32). ISBN 978-3-540-18796-7.
3. US patent 4309569 (https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=US4309569), Ralph Merkle, "Method of providing digital signatures", published Jan 5, 1982, assigned to The Board Of Trustees Of The Leland Stanford Junior University
4. Bonwick, Jeff. "ZFS End-to-End Data Integrity" (https://blogs.oracle.com/bonwick/entry/zfs_end_to_end_data). *Jeff Bonwick's Blog*.
5. Likai Liu. "Bitrot Resistance on a Single Drive" (http://lifecs.likai.org/2014/06/bitrot-resistance-on-single-drive.html). *likai.org*.
6. "General Verifiable Federation" (https://web.archive.org/web/20180408150608/http://www.waveprotocol.org/whitepapers/wave-protocol-verification). *Google Wave Protocol*. Archived from the original (http://www.waveprotocol.org/whitepapers/wave-protocol-verification) on 2018-04-08. Retrieved 2017-03-09.
7. Koblitz, Neal; Menezes, Alfred J. (January 2016). "Cryptocash, cryptocurrencies, and cryptocontracts". *Designs, Codes and Cryptography*. **78** (1): 87–102. CiteSeerX 10.1.1.701.8721 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.701.8721). doi:10.1007/s10623-015-0148-5 (https://doi.org/10.1007%2Fs10623-015-0148-5). S2CID 16594958 (https://api.semanticscholar.org/CorpusID:16594958).
8. Adam Marcus. "The NoSQL Ecosystem" (http://www.aosabook.org/en/nosql.html). *aosabook.org*. "When a replica is down for an extended period of time, or the machine storing hinted handoffs for an unavailable replica goes down as well, replicas must synchronize from one another. In this case, Cassandra and Riak implement a Dynamo-inspired process called anti-entropy. In anti-entropy, replicas exchange Merkle trees to identify parts of their replicated key ranges which are out of sync. A Merkle tree is a hierarchical hash verification: if the hash over the entire keyspace is not the same between two replicas, they will exchange hashes of smaller and smaller portions of the replicated keyspace until the out-of-sync keys are identified. This approach reduces unnecessary data transfer between replicas which contain mostly similar data."
9. Kilian, J. (1995). "Improved efficient arguments (preliminary version)" (https://link.springer.com/content/pdf/10.1007/3-540-44750-4_25.pdf) (PDF). *CRYPTO*. doi:10.1007/3-540-44750-4_25 (https://doi.org/10.1007%2F3-540-44750-4_25).
10. Mark Friedenbach: Fast Merkle Trees (https://github.com/bitcoin/bips/blob/master/bip-0098.mediawiki)
11. Laurie, B.; Langley, A.; Kasper, E. (June 2013). "Certificate Transparency" (https://www.rfc-editor.org/info/rfc6962). *IETF*: RFC6962. doi:10.17487/rfc6962 (https://doi.org/10.17487%2Frfc6962).
12. Elena Andreeva; Charles Bouillaguet; Orr Dunkelman; John Kelsey (January 2009). "Herding, Second Preimage and Trojan Message Attacks beyond Merkle–Damgård". *Selected Areas in Cryptography*. Lecture Notes in Computer Science. **5867**. SAC. pp. 393–414. doi:10.1007/978-3-642-05445-7_25 (https://doi.org/10.1007%2F978-3-642-05445-7_25). ISBN 978-3-642-05443-3.
13. Elena Andreeva; Charles Bouillaguet; Pierre-Alain Fouque; Jonathan J. Hoch; John Kelsey; Adi Shamir; Sebastien Zimmer (2008). "Second Preimage Attacks on Dithered Hash Functions". In Smart, Nigel (ed.). *Advances in Cryptology – EUROCRYPT 2008*. Lecture Notes in Computer Science. **4965**. Istanbul, Turkey. pp. 270–288. doi:10.1007/978-3-540-78967-3_16 (https://doi.org/10.1007%2F978-3-540-78967-3_16). ISBN 978-3-540-78966-6.
14. Chapweske, J.; Mohr, G. (March 4, 2003). "Tree Hash EXchange format (THEX)" (https://web.archive.org/web/20090803220648/http://open-content.net/specs/draft-jchapweske-thex-02.html). Archived from the original (http://open-content.net/specs/draft-jchapweske-thex-02.html) on 2009-08-03.
15. "tigertree.c File Reference" (http://gtk-gnutella.sourceforge.net/doxygen/tigertree_8c.html). *Gtk-Gnutella*. Retrieved 23 September 2018.
16. "Audit: P2P DirectConnect Application" (https://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=21587). *Symantec*. Retrieved 23 September 2018.
17. Arne Babenhauserheide (7 Jan 2007). "Phex 3.0.0 released" (http://www.phex.org/mambo/content/view/80/2/). *Phex*. Retrieved 23 September 2018.
18. "DC++'s feature list" (http://dcplusplus.sourceforge.net/features.html). *dcplusplus.sourceforge.net*.
19. "Development" (http://gtk-gnutella.sourceforge.net/en/?page=devel). *GTK-Gnutella*. Retrieved 23 September 2018.

Further reading

- Merkle tree patent 4,309,569 (https://www.google.com/patents/US4309569) – explains both the hash tree structure and the use of it to handle many one-time signatures
- Tree Hash EXchange format (THEX) (https://web.archive.org/web/20080316033726/http://www.open-content.net/specs/draft-jchapweske-thex-02.html) – a detailed description of Tiger trees

External links

- A C implementation of a dynamically re-sizeable binary SHA-256 hash tree (Merkle tree) (https://github.com/IAIK/merkle-tree)
- Merkle tree implementation in Java (https://github.com/richpl/merkletree)
- Tiger Tree Hash (TTH) source code in C# (https://www.codeproject.com/articles/9336/thexcs-tth-tiger-tree-hash-maker-in-c), by Gil Schmidt
- Tiger Tree Hash (TTH) implementations in C and Java (http://sourceforge.net/projects/tigertree/)
- RHash (http://rhash.sourceforge.net/), an open source command-line tool, which can calculate TTH and magnet links with TTH
- A Golang implementation: Package merkleTree (https://godoc.org/github.com/keybase/go-merkle-tree) is a generic Merkle Tree implementation, for provably publishing much data under one succinct tree root
- Another Golang implementation: (https://godoc.org/github.com/ontio/ontology/merkle)
- Another Golang implementation: Package merkle is a fixed merkle tree implementation (https://godoc.org/github.com/xsleonard/go-merkle)
- Another Golang implementation: (https://godoc.org/github.com/33cn/chain33/common/merkle)
- Another Golang implementation: Package merkle computes a deterministic minimal height Merkle tree hash (https://godoc.org/github.com/tendermint/tendermint/crypto/merkle)
- Another Golang implementation: Package merkletree (https://godoc.org/github.com/NebulousLabs/merkletree) provides tools for calculating the Merkle root of a dataset, for creating a proof that a piece of data is in a Merkle tree of a given root, and for verifying proofs that a piece of data is in a Merkle tree of a given root.