guardtime

MAY 2020

Blockchain Receipts using History Trees

by

Risto Alas, Hema Krishnamurthy

Abstract

A blockchain receipt (also known as a “light client proof”) provides proof that some data existed at a specific time, and that the data was approved by the validators of the blockchain. The receipt contains all the information needed to prove an individual input was indeed part of an approved block. This paper proposes a patented method to optimize the way old receipts are verified against new blocks, by the use of history trees.

To provide additional evidence to the users that the older parts of the history tree were not altered after the fact, we propose timestamping the older parts of the tree with KSI, which periodically publishes the root hash of its calendar (history) database in physical printed newspapers and thus provides an additional security anchor.

Guardtime’s KSI® blockchain is a data-centric security technology relying on hash functions and binary hash trees only. Whilst KSI is based on a calendar history blockchain, signing and verification times are extremely fast relative to other blockchains. Signatures can be obtained in around a second, and verification responses can be obtained in milliseconds.

Keywords: blockchain, Merkle trees, receipts, history trees, KSI®

Introduction

A blockchain receipt (also known as a “light client proof”) provides proof that some data existed at a specific time, and that the data was approved by the validators of the blockchain. The receipt contains all the information needed to prove an individual input was indeed part of an approved block. Receipts are useful if one does not wish to fully process and validate the entire blockchain, but rather wants to rely on the consensus of validators to make claims about the blockchain state (for example, the validator consensus can claim that a balance of a particular account was a specific value). Receipts can also be used to simply prove that a transaction was published in the blockchain. In general, the meaning of a receipt is defined by the operating rules of the blockchain that issued it.

The need for short proofs may come from performance requirements — for example, a smartphone may not have the computation power to process the blockchain in full. Privacy considerations can also play a role, in which case all the data needed for full re-validation may not be available. Additionally, blockchains typically do not wish to fully re-validate each other’s transactions when they are interacting.

Blockchain receipts can be generated by anybody who has (enough) access to the corresponding information on the blockchain. All the information for generating receipts is present on the blockchain itself; indeed, a receipt is essentially just a block from the blockchain, with some irrelevant parts torn off. The receipts can also provide a mechanism for accountability of the blockchain client — i.e., a full blockchain client can produce the electronic receipt to prove a transaction he/she was part of. The immutable proof can be submitted to system auditors for later audit.

The generalised idea of blockchain receipts has been around for a while. For example, Bitcoin has always had the idea of Simple Payment Verification. For Proof-of-Stake blockchains (permissioned blockchains generally fall into that category), the receipts generally contain the signatures of sufficient majority (e.g., more than 2/3) of the validator set on a particular block hash, and then a Merkle proof regarding specific data under the same block hash.

Blockchain receipts using history trees

Typically, each block in the blockchain contains a hash of the previous block — but here we propose keeping a whole or partial tree of previous blocks to increase efficiency of traversal. Efficient tree structures are used to store the history trees so the storage doesn't grow too much.

Merkle trees, named after Ralph Merkle who originally described them, work in the following way: inputs are arranged in leaves of a binary tree; the value of each parent node is computed as the hash value of the concatenation of the values of its child nodes; by tracing the path from the root to the leaf containing the target value, one can generate a hash chain that proves participation of the target value in the tree, without having to know the entire tree.

History trees are Merkle trees whose leaf nodes represent blocks in a blockchain, which in turn may contain transactions, various metadata, state information and/or any other information of the specific blockchain. Each node in the history tree is labeled with a cryptographic hash, which fixes the contents of the subtree rooted at that node.

Generally, with blockchains, it is not possible to authenticate old blocks on their own, for different reasons. For instance, for Proof-of-Stake blockchains that utilise "slashing"-type consensus algorithms (to punish dishonest validators), only the more recent blocks have "real money staked on them". If double-signing/forking of a more recent block is found, its validators will lose large amounts of money; but that does not hold for old blocks — their validators may no longer hold their deposits in the system, and therefore the malicious parties cannot be punished directly by the blockchain. Hence, one can directly authenticate only the most recent blocks, but not old blocks.

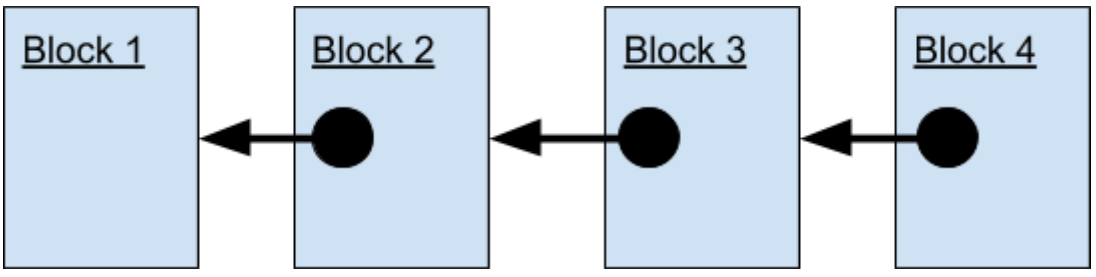

Figure 1. A typical blockchain data structure where every block contains a hash of its preceding block. The circles represent hash values; the arrows point at the objects being hashed.

To authenticate the data in the old blocks, one would normally have to traverse all the way back through the blockchain, block-by-block, and check that every block contains the hash of its preceding block (Figure 1). When one finally reaches the old block successfully, the old block is considered validated. The same goes for old receipts, which are essentially slices out of old blocks.

While the above verification process is secure, it is rather inefficient: a blockchain may generate millions of blocks per year, and thus a verification of an old block would take millions of hashing steps for every year of block time. Hence, we propose the use of history trees to shorten such proofs (Figure 2). Every block will not only contain a hash of its direct predecessor block, but rather an entire tree of hashes of all predecessor blocks. This will significantly shorten the proofs for authenticating older receipts/blocks.

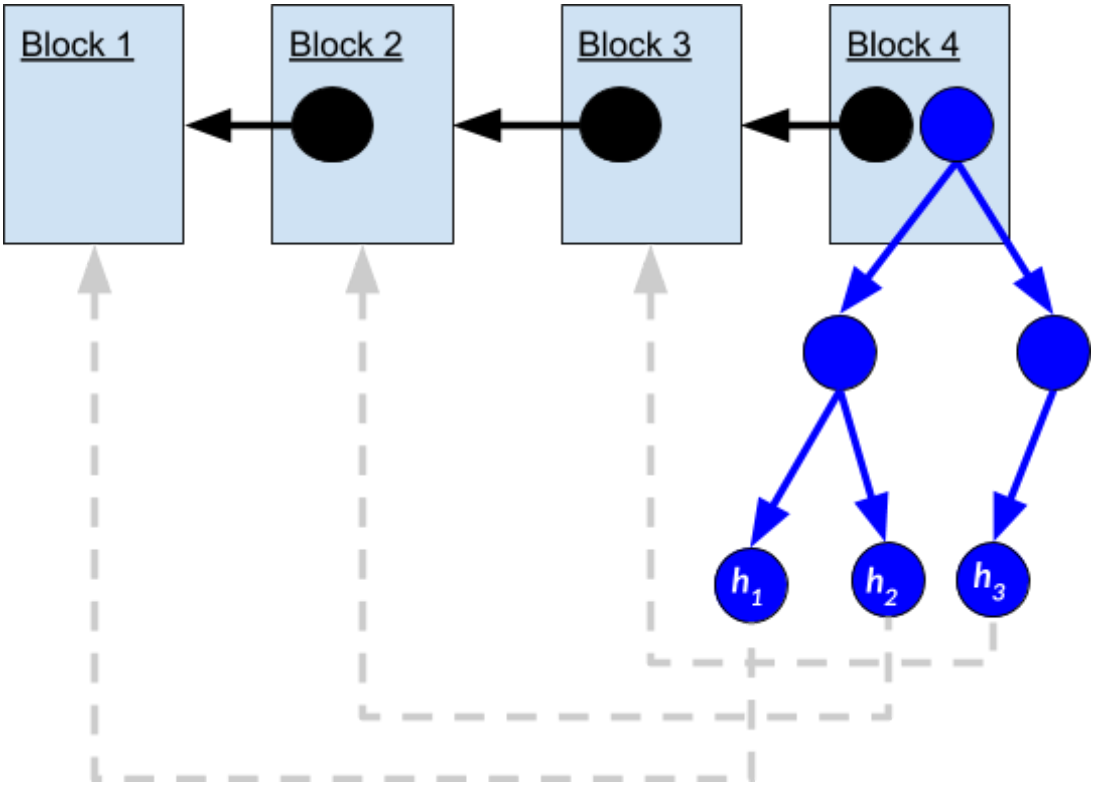

Figure 2. A blockchain with a history tree added, represented with dark blue color. For simplicity, we have shown the tree only in block number 4. In practice, the other blocks would contain similar trees that would point to their respective preceding blocks. The circles represent hash values; the arrows point at the objects being hashed. The values h_1, h_2 and h_3, represent hashes of blocks 1, 2, and 3 respectively.

Note that the history trees do not take much space — they take noticeably less memory than storing the blockchain itself: indeed, the history tree leaves will only store one hash per every block (plus additional intermediate hashes, which will

roughly only double the total amount of hashes). Additionally, as the old blocks never change, the history trees in successive blocks share large sub-trees; therefore, much of the history tree data can be shared among multiple successive trees. Such incremental storage of a history tree will take merely log(*T*) hash values in the worst case for a new block numbered *T*. The average case performance is even better: only 2 new hash values are required on average per each new block (the rest of the data will be shared with the tree from the previous block).

Nonetheless, to save space, a whole "new" history tree can be created periodically i.e. older parts of the history tree can be archived/pruned and the "frozen root of the old tree" used as the first leaf of the new tree. We also consider the option of including only important blocks in the history tree (e.g., only the blocks that contain high profile transactions would get aggregated into the history tree).

We also consider another solution to enable efficient verification of old blocks/receipts. The idea involves simply recording the old validator public keys, and then verifying old blocks/receipts the same way as new blocks would be verified: by verifying them against their validators' signatures. However, this solution would not be compatible with Proof-of-Stake incentives, as the old validators no longer have any stake in the system, and thus they would be free to sign any old invalid blocks after the fact, without fear of punishment. Note that trusted timestamping does not fix this issue, because technically, the malicious validators could have signed a parallel fork of a blockchain a long time ago and simply kept it a secret for a while, and then release the malicious fork only after their staked currency is no longer bonded (that is, it is no longer assigned to the same keys and cannot be slashed by the consensus algorithm).

Compared to this idea of validating against the old validator set, the history tree is stronger against blockchain forks (just like the idea of traversing millions of hashes, but more efficient) — i.e., it fixes a specific blockchain history in place, unlike just fixing a validator set which may have been compromised at some point in the past, thus forking the blockchain. And because the history tree is compatible with Proof-of-Stake implementations, it is also fully compatible with other developments like the Tendermint consensus protocol. Examples of publicly known big Proof-of-Stake implementations are Tendermint's Cosmos blockchain and Ethereum's Casper.

To ensure that malicious actors do not modify older parts of the history tree after the fact, we propose that they be timestamped with KSI, which periodically

publishes the root hash of its calendar (history) database in physical printed newspapers and thus provides an additional security anchor.

KSI operates in fixed-length rounds. In each round inputs are aggregated into a hash tree and the root hash of the tree is published in a distributed “history” database (also called a “calendar” database) that every customer has a copy of. For every asset included in the tree, a signature is returned to the customer that identifies the computation path, through the hash tree, from the asset's hash value, up to the root hash value of the current state of the history database. The signature also includes “sibling” values necessary to recreate the root hash. With access to the public history database, anyone, anywhere, can receive data and, without reliance on a central trust authority, verify the signature which includes proof of time and integrity and attribution of origin. Ideally, Guardtime’s KSI signatures themselves can be viewed as receipts.

Another option is for auditors to periodically anchor snapshots of the history tree into other blockchains. Tamper-evident hardware could also be used to securely store the history trees.

In blockchains where the validators are known entities, as is typically the case in permissioned blockchains, the receipts could also embed CA certificates for the validator keys, thus making the validators more transparent all the time. This might even allow the receipt verification code to verify the receipts using validation keys it does not yet know about, but that would require the users to believe that the given signing identities were held by honest participants at the time of signing (perhaps the signing organisations are generally trusted by the user), as well as trusting the certificate issuing CAs.

As a convenience, for some use-cases the receipts could even be human-readable, for example, PDF files signed by PKI signatures of the validators; the (separate) executable for verifying the blockchain might still contain the validator set history, so the user would not have to manually verify the signer identities.

One advantage of our approach is that the validator set can completely change over time, and as long as the companies of the last validators are trusted, the latest validator set can directly be authenticated. A downside is that 10 years later, one might not remember which companies were supposed to even host the blockchain back then; however, if and when this becomes a problem, the new validators could always keep a list of the old validators around ie as long as you trust the new validators, you can get the entire validator list from them.

Conclusion

Typically, each hash in the blockchain contains a hash of the previous block – but our approach of using history trees wherein we use a whole or modified partial tree of previous blocks to increase efficiency of traversal is novel. This makes the verification of old receipts/blocks much more efficient. We also propose using KSI to sign the history tree, which gives us the added advantage of getting a quicker timestamp (under 2 seconds) with long term verification capabilities since the calendar root hashes are published to a newspaper.

References

Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees https://eprint.iacr.org/2013/834

Event Verification Receipt System and Methods - http://www.freepatentsonline.com/y2020/0104294.html