Article blockchain

Merkle Tree and Ethereum Objects - Ethereum Yellow Paper Walkthrough (2/7)

Lucas Saldanha
11 Dec 2018 • 11 min read

Hi everyone! This is another post in our series exploring the Ethereum Yellow Paper. In this post, we will learn more about the main objects in Ethereum and their role. We'll also briefly discuss how Merkle trees are used in Ethereum.

I hope that after reading this post, you'll know what Merkle trees are and the role that they play in Ethereum, what the world state and account state are, and what a transaction and a

block look like.

If you missed the first post in the series, talking about Ethereum and the blockchain paradigm, check it out here!

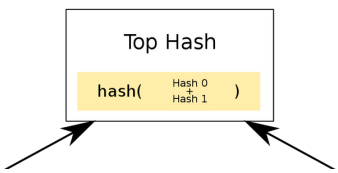
(DISCLAIMER: this post is based on the Byzantium version of the Yellow Paper, version e94ebda from 5th June 2018)

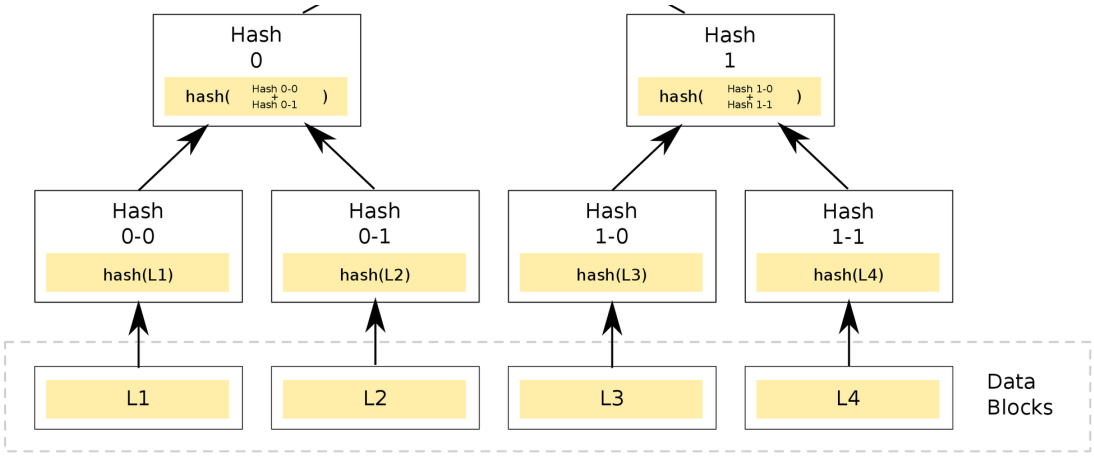
Merkle Trees

Before discussing the main data objects in Ethereum, I'd like to present a summary of what Merkle trees are and the properties that make them useful.

The Yellow Paper states that it assumes implementations will maintain the world state and transactions in a modified Merkle Patricia tree (trie). This data structure is described in Appendix D.
The Merkle Patricia tree (trie) has some really interesting properties and if you want to learn more about the Ethereum implementation, I'd recommend reading this article.

In a Merkle tree, **the leaf nodes contains the hash of a block of data and the non-leaf nodes contains the hash of its children nodes**.





A diagram representing a Merkle tree (with nodes and their relationship)

(Image from: https://en.wikipedia.org/wiki/Merkle_tree)

In a Merkle tree any change to the underlying data causes the hash of the node referring to the data to change. Since each parent node hash depends on the data of its children, any change to the data of a child node causes the parent hash to change. This happens to each parent node up to the root node. Therefore, **any change to the data at the leaf nodes causes the root node hash to change**. From this, we can derive two important properties:

1. **We don't need to compare all the data across the leaf nodes to know if they have the same data. We can just compare the root node hash.**
2. If we want to prove that specific data is part of the tree, we can use a technique called Merkle proofs. We won't dive into details here but it is an easy and effective way to prove that a piece of data is in the Merkle tree.

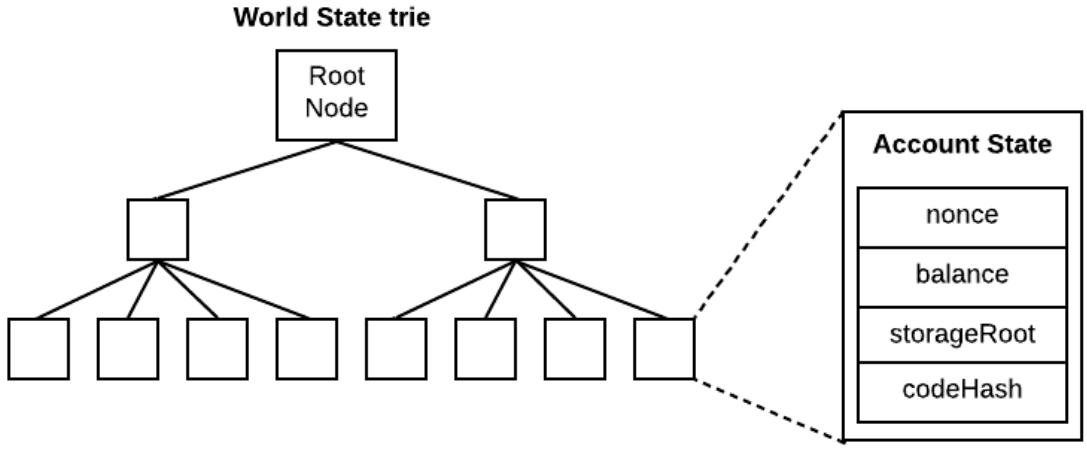
The first property is important because it makes it possible to store only a hash of the root node to represent the data at that point in time. **This means we only need to store the root hash of the tree representing the block on the blockchain** (as opposed to storing all the data in the blockchain) and still keep the data immutable.

Now we have an idea of what a Merkle tree root hash represents, let's talk about Ethereum main objects.

World State

The **world state is a mapping between addresses (accounts) and account states**. The world state is not stored on the blockchain but the Yellow Paper states it is expected implementations store this data in a trie (also referred as the state database or state trie).The world state can be seen as the global state that is constantly updated by transaction executions. If you remember the discussion in the first post of the series about the Ethereum network being like a

decentralized computer, the world state is considered this computer's hard drive.
All the information about Ethereum accounts live in the world state and is stored in the world state trie. If you want to know the balance of an account, or the current state of a smart contract, you query the world state trie to retrieve the account state of that account. We'll describe how this data is stored shortly.



World state trie and Account storage

Account State

In Ethereum, there are two types of accounts: External Owned Accounts (EOA) and Contract Accounts. An EOA account is the account that you and I would have, that we can use to send Ether to one another and deploy smart contracts. A contract account is the account that is created when a smart contract is deployed. Every smart contract has its own Ethereum account.

The account state contains information about an Ethereum account. For example, it stores how much Ether an account has and the number of transactions sent by the account. Each account has an account state.

Let's take a look into each one of the fields in the account state:

- **nonce**
  - Number of transactions sent from this address (if this is

an External Owned Account - EOA) or the number of contract-creations made by this account (don't worry about what *contract-creations* means for now).

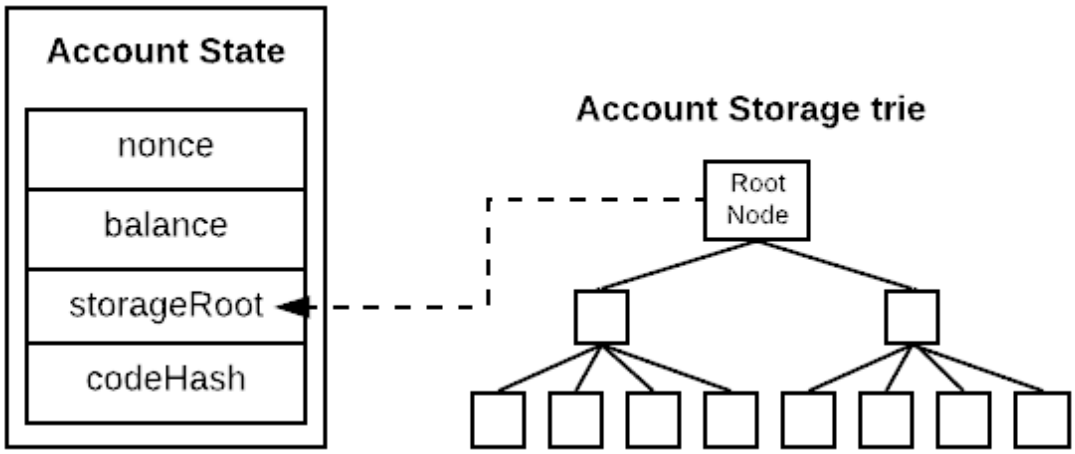
- **balance**
  - Total Ether (in Wei) owned by this account.
- **storageRoot**
  - Hash of the root node of the account storage trie (we'll see what the account storage is in a moment).
- **codeHash**
  - For contract accounts, hash of the EVM code of this account. For EOAs, this will be empty.

One important details about the account state is that **all fields (except the codeHash) are mutable**. For example, when one account sends some Ether to another, the nonce will be incremented and the balance will be updated to reflect the new balance.

One of the consequences of the codeHash being immutable is that if you deploy a contract with a bug, you can't update the same contract. You need to deploy a new contract (the buggy version will be available forever). This is why it is important to use Truffle to develop and test your smart contracts and follow the best practices when working with Solidity.

The **Account Storage trie is where the data associated with an account is stored**. This is only relevant for Contract Accounts,

as for EOAs the storageRoot is empty and the codeHash is the hash of an empty string. All smart contract data is persisted in the account storage trie as a mapping between 32-bytes integers. We won't discuss in details how the contract data is persisted in the account state trie. If you really want to learn about the internals, I suggest reading this post. The hash of an account storage root node is persisted in the storageRoot field in the account state of the respective account.



Account state and Account Storage trie

Transaction

If you read my last post about the Ethereum model, you remember that **transactions are what makes the state change from the current state to the next state**. In Ethereum, we have three types of transactions:

1. Transactions that **transfer value** between two EOAs (e.g, change the sender and receiver account balances)
2. Transactions that **send a message call** to a contract (e.g, set

a value in the smart contract by sending a message call that executes a setter method)

3. Transactions that **deploy a contract** (therefore, create an account, the contract account)

(technically, types 1 and 2 are the same... transactions that send message calls that affect an account state, either EOA or contract accounts. But is it easier to think about them as three different types)

These are the fields of a transaction:

- **nonce**
  - Number of transactions sent by the account that created the transaction.
- **gasPrice**
  - Value (in Wei) that will be paid per unit of gas for the computation costs of executing this transaction.
- **gasLimit**
  - Maximum amount of gas to be used while executing this transaction.
- **to**
  - If this transaction is transfering Ether, address of the EOA account that will receive a value transfer.
  - If this transaction is sending a message to a contract (e.g, calling a method in the smart contract), this is address of the contract

address of the contract.

- If this transactions is creating a contract, this value is always empty.

- **value**
  - If this transaction is transfering Ether, amount in Wei that will be transferred to the recipient account.
  - If this transaction is sending a message to a contract, amount of Wei payable by the smart contract receiving the message.
  - If this transaction is creating a contract, this is the amount of Wei that will be added to the balance of the created contract.

- **v, r, s**
  - Values used in the cryptographic signature of the transaction used to determine the sender of the transaction.

- **data** (only for value transfer and sending a message call to a smart contract)

  - Input data of the message call (e.g, imagine you are trying to execute a setter method in your smart contract, the data field would contain the identifier of the setter method and the value that should be passed as parameter).

- **init** (only for contract creation)

  - The EVM-code utilized for initialization of the contract.

Don't try to grasp all of this at once... Some fields like the *data*

contract.

field or the *init* field require you to have a deeper understanding of the internals of Ethereum to really understand what they mean and how to use them. This is not the time to deeply understand any of these fields.

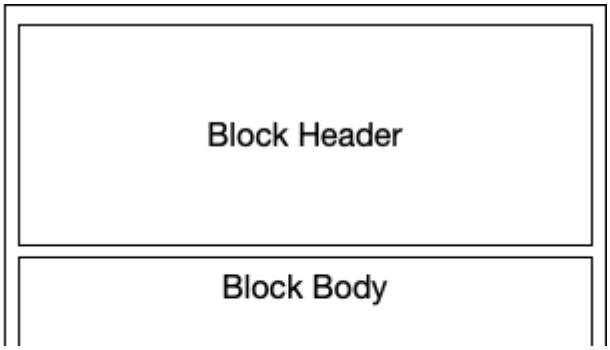
Not surprisingly, all transactions in a block are stored in a trie. And the root hash of this trie is stored in the... block header! Let's take a look into the anatomy of an Ethereum block.

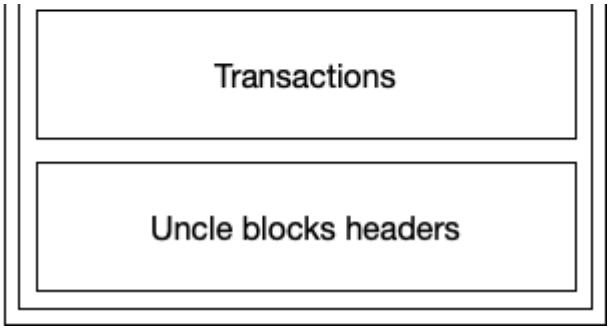
Block

The block header is divided in two parts, the **block header** and the **block body**.

The block header is the blockchain part of Ethereum. This is the structure that contains the hash of its predecessor block (also known as parent block), building a cryptographically guaranteed chain.

The block body contains a **list of transactions** that have been included in this block and a **list of uncle (ommer) blocks headers** (if you want to know more about uncle blocks I recommend this post).





A high level diagram of the Ethereum block

Let's take a look at all fields in the block header:

- **parentHash**
  - Hash of the block header from the previous block. Each block contains a hash of the previous block, all the way to the first block in the chain. This is how all the data is protected against modifications (any modification in a previous block would change the hash of all blocks after the modified block).
- **ommersHash**
  - Hash of the uncle blocks headers part of the block body.
- **beneficiary**
  - Ethereum account that will get fees for mining this block.
- **stateRoot**
  - Hash of the root node of the world state trie (after all transactions are executed).
- **transactionsRoot**

- Hash of the root node of the transactions trie. This trie contains all transactions in the block body.

- **receiptsRoot**
  - Every time a transactions is executed, Ethereum generates a transaction receipt that contains information about the transaction execution. This field is the hash of the root node of the transactions receipt trie.

- **logsBloom**
  - Bloom filter that can be used to find out if logs were generated on transactions in this block (if you want more details check this Stack Overflow answer). This avoids storing of logs in the block (saving a lot of space).

- **difficulty**
  - Difficulty level of this block. This is a measure of how hard it was to mine this block (I'm not diving into the details of how this is calculated in this post).

- **number**
  - Number of ancestor blocks. This represents the height of the chain (how many blocks are in the chain). The genesis block has number zero.

- **gasLimit**
  - Each transaction consumes gas. The gas limit specifies the maximum gas that can be used by the transactions

included in the block. It is a way to limit the number of transactions in a block.

- **gasUsed**
  - Sum of the gas cost of each transaction in the block.
- **timestamp**
  - Unix timestamp when the block was created. Note that due to the decentralized nature of Ethereum, we can't trust in this value (specially when implementing smart contracts that have time related business logic).
- **extraData**
  - Arbitrary byte array that can contain anything. When a miner is creating the block, it can choose to add anything in this field.
- **mixHash**
  - Hash used to verify that a block has been mined properly (if you want to really understand this, read about the Ethash proof-of-work function).
- **nonce**
  - Same as the mixHash, this value is used to verify that a block has been mined properly.

Phew... This is a long list... Take your time to read it! Again, the point here is not to memorize each one of the fields and what they represent (you can always Google it later). What I

intended was to describe each field in a simple way (at least simpler than the Yellow Paper) to help you to understand what they mean. Think of it as an *Ethereum objects for dummies*! :)

Conclusion

Let's do a quick recap about what we just saw! Basically, Ethereum has 4 types of tries:

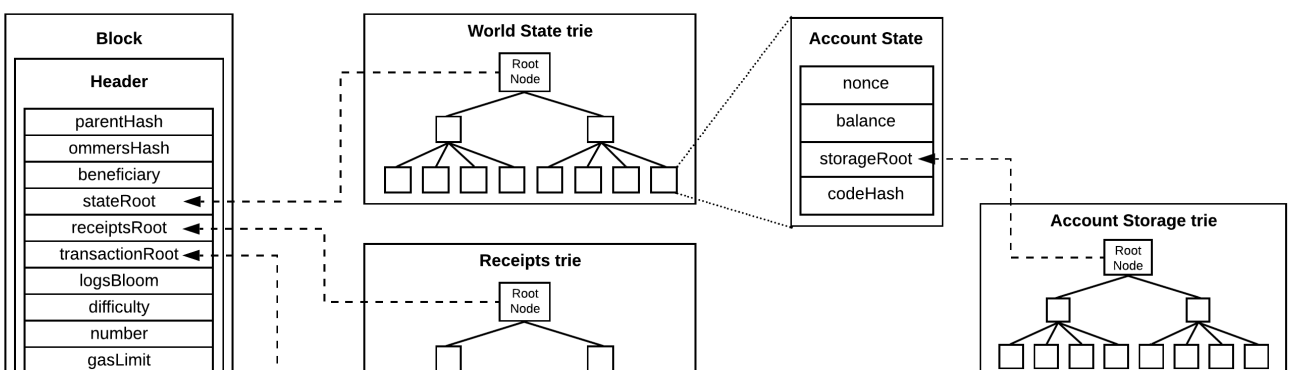
1. **The world state trie contains the mapping between addresses and account states**. The hash of the root node of the world state trie is included in a block (in the **stateRoot** field) to represent the current state when that block was created. We only have one world state trie.
2. **The account storage trie contains the data associated to a smart contract**. The hash of the root node of the Account storage trie is included in the sccount state (in the **storageRoot** field). We have one Account storage trie for each account.
3. **The transaction trie contains all the transactions included in a block**. The hash of the root node of the Transaction trie is included in the block header (in the **transactionsRoot** field). We have one transaction trie per block.
4. **The transaction receipt trie contains all the transaction receipts for the transactions included in a block**. The hash of the root node of the transaction receipts trie is included in also included in the block header (in the **receiptsRoot** field); We have one transaction receipts trie

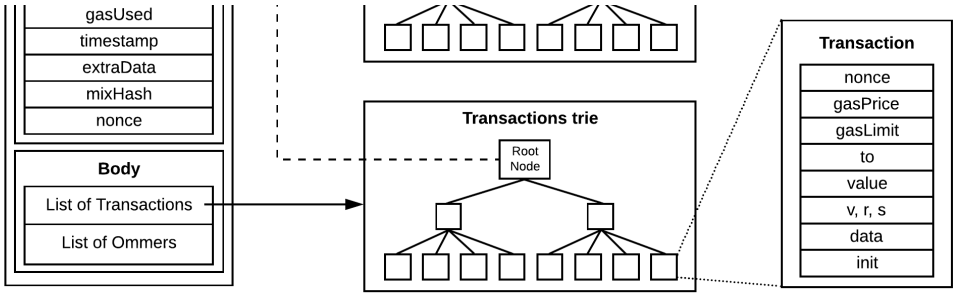
per block.

And the objects that we discussed today are:

1. **World state**: the hard drive of the distributed computer that is Ethereum. It is a mapping between addresses and account states.
2. **Account state**: stores the state of each one of Ethereum's accounts. It also contains the storageRoot of the account state trie, that contains the storage data for the account.
3. **Transaction**: represents a state transition in the system. It can be a funds transfer, a message call or a contract deployment.
4. **Block**: contains the link to the previous block (parentHash) and contains a group of transactions that, when executed, will yield the new state of the system. It also contains the **stateRoot**, the **transactionRoot** and the **receiptsRoot**, the hash of the root nodes of the world state trie, the transaction trie and the transaction receipts trie, respectively.

I tried to capture all the information contained in this post in this diagram:





Block, transaction, account state objects and Ethereum tries

I hope that you have found this post useful. As I said before, the main goal was to describe each one of the main Ethereum objects in words that someone relatively new to the blockchain and Ethereum ecosystem could understand. In my experience, learning from the Yellow Paper is not easy and requires a lot of patience.

Working with Ethereum protocol is interesting. Every day, I have the opportunity to learn something new (and also relearn things that I thought that I already knew). If you spot anything in this post that isn't quite clear (or even something that is clearly wrong), please comment and I'll try to address it as soon as possible. After all, we don't want to propagate misconceptions about the protocol.

In my next post of the series, we'll talk about why Ethereum has Gas and about the Transaction Execution Model. Hold on as we dive deeper into the Yellow Paper!

See you in the next one. Cheers!

References

References

- Merkle Trees
- Merkle Proofs
- How data is stored in Ethereum?
- Diving into Ethereum's world state
- How does Ethereum work anyway?
- A (Practical) Walkthrough of Smart Contract Storage
- Inside an Ethereum transaction
- Life Cycle of an Ethereum Transaction
- Ethereum Design Rationale

Topic blockchain ethereum Share

Show Comments

Gas and Payment -...

Another day, another Ethereum Yellow Paper blog post! In this...

04 Mar 2019

The Blockchain Paradigm ...

Hi folks! It's been a long time without posting anything here....

14 Feb 2018