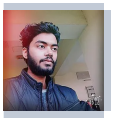
Getting Deep Into Ethereum: How Data Is Stored In Ethereum?

Originally published by Vaibhav Saini on July 29th 2018

49,560 reads

7

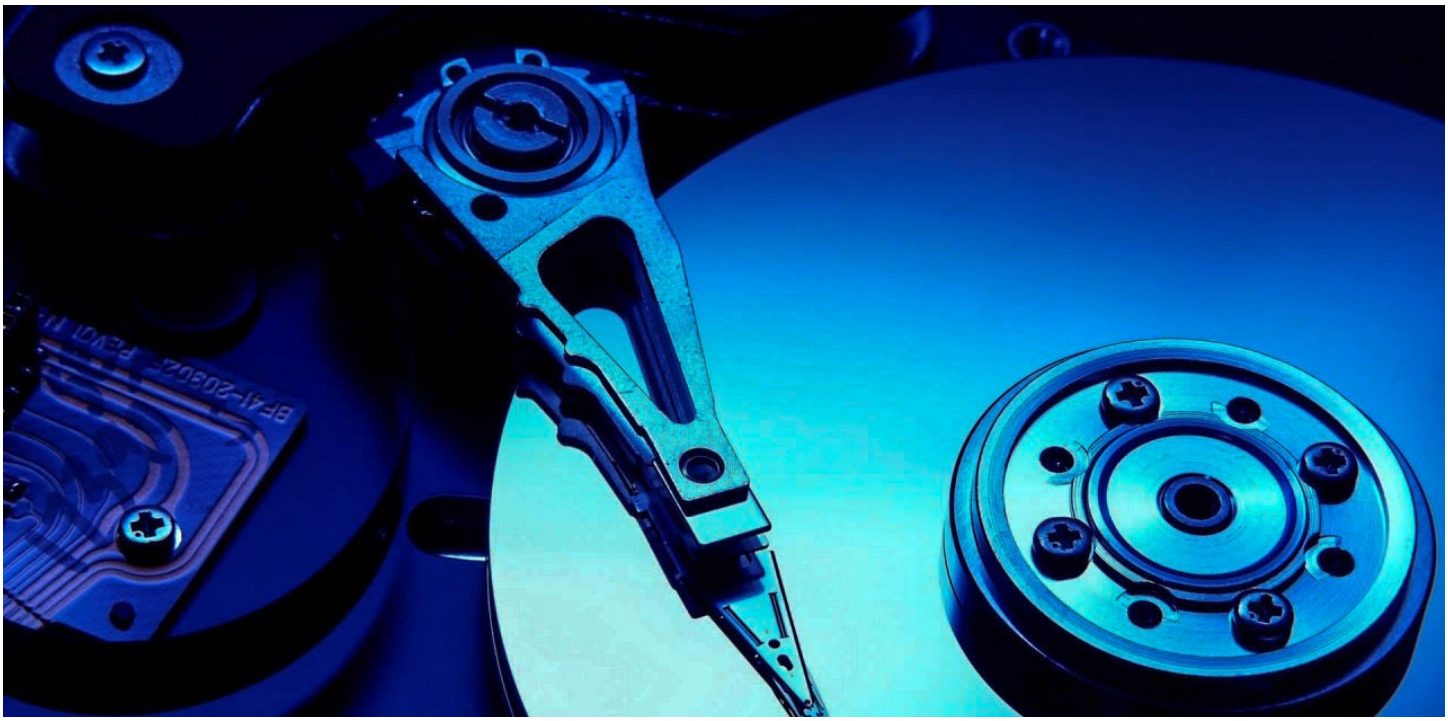


@vasa

Vaibhav Saini

Entrepreneur | Co-founder @tbc_inc, an MIT CIC incubated startup | Speaker |https://vaibhavsaini.com

In this post, we will see how states and transactions are stored in Ethereum and how it is different from Bitcoin.



This post is a continuation of my ***Getting Deep Into Series*** started in an effort to provide a deeper understanding of the internal workings and other cool stuff about Ethereum and blockchain in general which you will not find easily on the web. Here are the other parts of the Series:

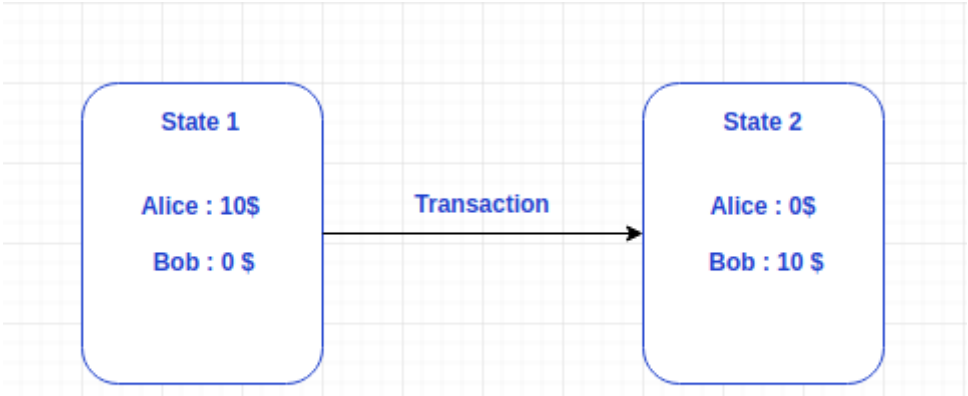
Getting Deep Into Geth: Why Syncing Ethereum Node Is Slow
*Downloading the blocks is just a small part. There is a lot of stuff going on... This post marks the first in a new…*hackernoon.com

Getting Deep Into EVM: How Ethereum Works Backstage
*An Ultimate, In-depth Explanation of What EVM is and How EVM Works.*hackernoon.com

In this post, we will dive into Ethereum's data storage layer. We will introduce the concept of blockchain "state". We will cover the theory behind the Patricia Trie data structure and demonstrate Ethereum's concrete implementation of tries using Google's leveldb database.

What do we store in the storage layer?

First of all, we have to see that what all things we need to store for making the blockchain system work. Let's take a simple example of Alice giving Bob 10$.



As we can see here that we can change the state by executing a transaction on it.

Here we have to keep track of the balances and other details of different people(**states**) and the details of what happens between them on blockchain(**transactions**). Different platforms handle this differently. Here we will see how Bitcoin and Ethereum handle this.

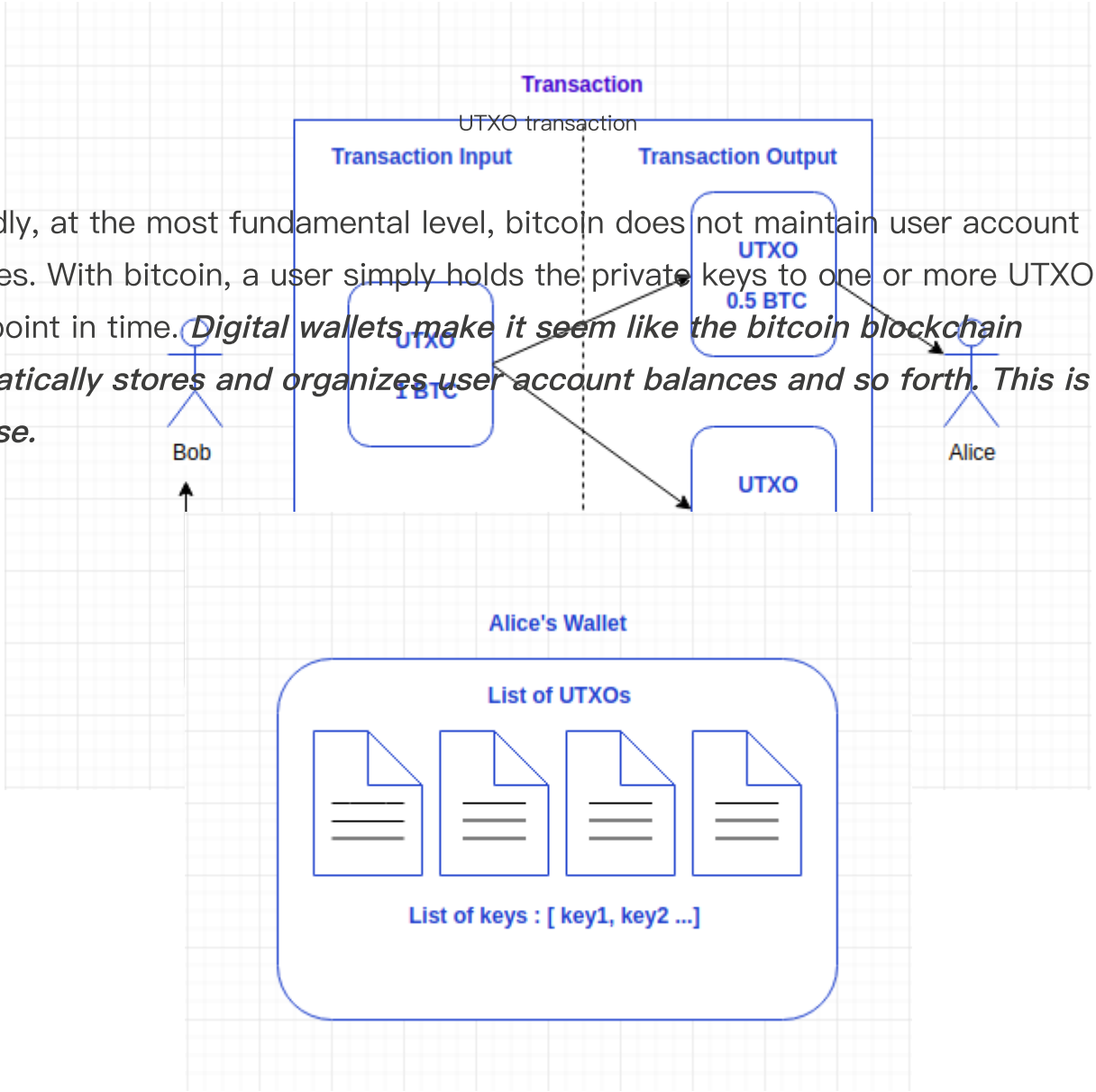
Bitcoin

Bitcoin's "state" is represented by its global collection of Unspent Transaction Outputs (UTXOs). The transfer of value in bitcoin is actioned through transactions. More specifically, a bitcoin user can spend one or more of their UTXOs by creating a transaction and adding one or more of their UTXOs as the transaction's input.

This model of UTXO makes Bitcoin different from Ethereum. Let's see some examples to understand the difference.

Firstly, bitcoin UTXOs cannot be partially spent. If a bitcoin user spends 0.5 bitcoin (using their only UTXO which is worth 1 bitcoin) they have to deliberately self–address (send themselves) 0.5 bitcoin in return change. If they don't send themselves change, they will lose the 0.5 bitcoin change to the bitcoin miner who mines their transaction.

Secondly, at the most fundamental level, bitcoin does not maintain user account balances. With bitcoin, a user simply holds the private keys to one or more UTXO at any given point in time. ***Digital wallets make it seem like the bitcoin blockchain automatically stores and organizes user account balances and so forth. This is not the case.***



A visualization of how wallets work in bitcoin

The UTXO system in bitcoin works well, in part, due to the fact that digital wallets are able to facilitate most of the tasks associated with transactions. Including but not limited to:

a) handling UTXOs

b) storing keys

c) setting transaction fees

c) setting transaction fees

d) providing return change addresses

e) aggregating UTXOs (to show available, pending and total balances)

One analogy for the transactions in the UTXO model is paper bills (banknotes). Each account keeps track of how much money it has by adding up the number of bills (UTXOs) in the purse (associated with this address/wallet). When we want to spend money, we use one or more bills (existing UTXOs), enough to cover the cost and maybe receive some change back (new UTXO). Each bill can only be spent once since, once spent, the UTXO is removed from the pool.

To summarize, we know that:

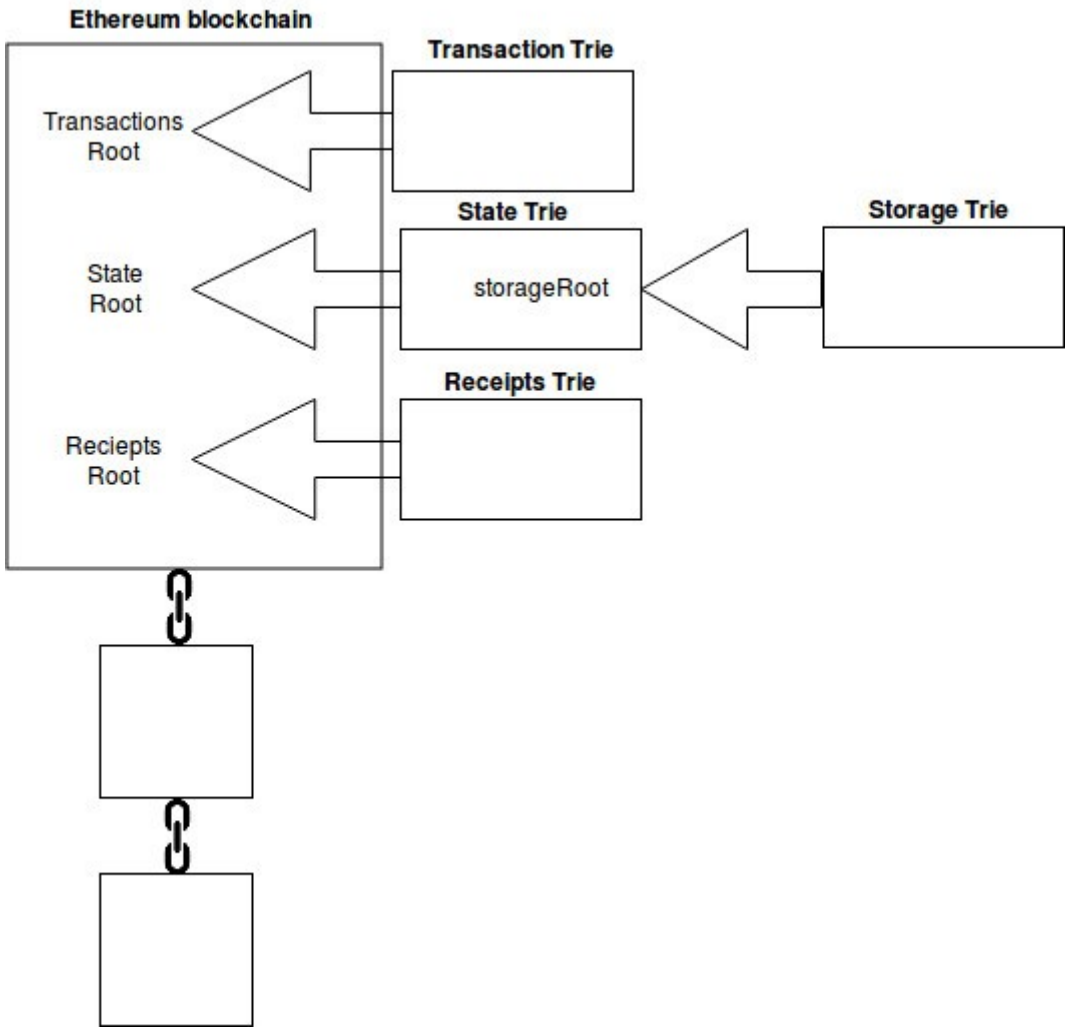
- the bitcoin blockchain does not hold account balances
- bitcoin wallets hold keys to UTXOs
- if included in a transaction, an entire UTXO is spent (in some cases partially received back as “change” in the form of a brand new UTXO)

Ethereum

In contrast to the information above, the Ethereum world state is able to manage account balances, and more. The state of Ethereum is not an abstract concept. It is part of Ethereum’s base layer protocol. As the yellow paper mentions, Ethereum is a transaction–based “state” machine; a technology on which all transaction–based state machine concepts may be built.

Let’s start at the beginning. As with all other blockchains, the Ethereum blockchain begins life at its own genesis block. From this point (genesis state at block 0) onward, activities such as transactions, contracts, and mining will continually change the state of the Ethereum blockchain. In Ethereum, an example of this would be an account balance (stored in the state trie) which changes every time a transaction, in relation to that account, takes place.

Importantly, data such as account balances are not stored directly in the blocks of the Ethereum blockchain. Only the root node hashes of the transaction trie, state trie and receipts trie are stored directly in the blockchain. This is illustrated in the diagram below.



Source

You will also notice, from the above diagram, that the root node hash of the storage trie (where all of the smart contract data is kept) actually points to the state trie, which in turn points to the blockchain. We will zoom in and cover all of this in more detail soon.

There are two vastly different types of data in Ethereum; permanent data and ephemeral data. An example of permanent data would be a transaction. Once a transaction has been fully confirmed, it is recorded in the transaction trie; it is never altered. An example of ephemeral data would be the balance of a particular Ethereum account address. The balance of an account address is stored in the state trie and is altered whenever transactions against that particular account occur. It makes sense

that permanent data, like mined transactions, and ephemeral data, like account balances, should be stored separately. Ethereum uses trie data structures to manage data.

The record-keeping for Ethereum is just like that in a bank. An analogy is using an ATM/debit card. The bank tracks how much money each debit card has, and when we need to spend money, the bank checks its record to make sure we have enough balance before approving the transaction.

A Comparison between UTXO and Account approach

The benefits of the UTXO Model:

- ***Scalability***—Since it is possible to process multiple UTXOs at the same time, it enables parallel transactions and encourages scalability innovation.
- ***Privacy***—Even Bitcoin is not a completely anonymous system, but UTXO provides a higher level of privacy, as long as the users use new addresses for each transaction. If there is a need for enhanced privacy, more complex schemes, such as ring signatures, can be considered.

The benefits of the Account/Balance Model:

- ***Simplicity***—Ethereum opted for a more intuitive model for the benefit of developers of complex smart contracts, especially those that require state information or involve multiple parties. An example is a smart contract that keeps track of states to perform different tasks based on them. UTXO's stateless model would force transactions to include state information, and this unnecessarily complicates the design of the contracts.
- ***Efficiency***—In addition to simplicity, the Account/Balance Model is more efficient, as each transaction only needs to validate that the sending account has enough balance to pay for the transaction.

One drawback for the Account/Balance Model is the exposure to double spending attacks. An incrementing nonce can be implemented to counteract this type of attack.

In Ethereum, every account has a public viewable nonce and every time a transaction is made, the nonce is increased by one. This can prevent the same transaction being submitted more than once. (Note, this nonce is different from the Ethereum proof of work nonce, which is a random value.)

Like most things in computer architecture, both models have trade-offs. Some blockchains, notably Hyperledger, adopt UTXO because they can benefit from the innovation derived from the Bitcoin blockchain. We will look into more technologies that are built on top of these two record-keeping models.

A closer look at the trie structure in Ethereum

Let's look at the state, storage and transaction tries in a bit more depth.

State trie—the one and only

There is one, and one only, global state trie in Ethereum.

This global state trie is constantly updated.

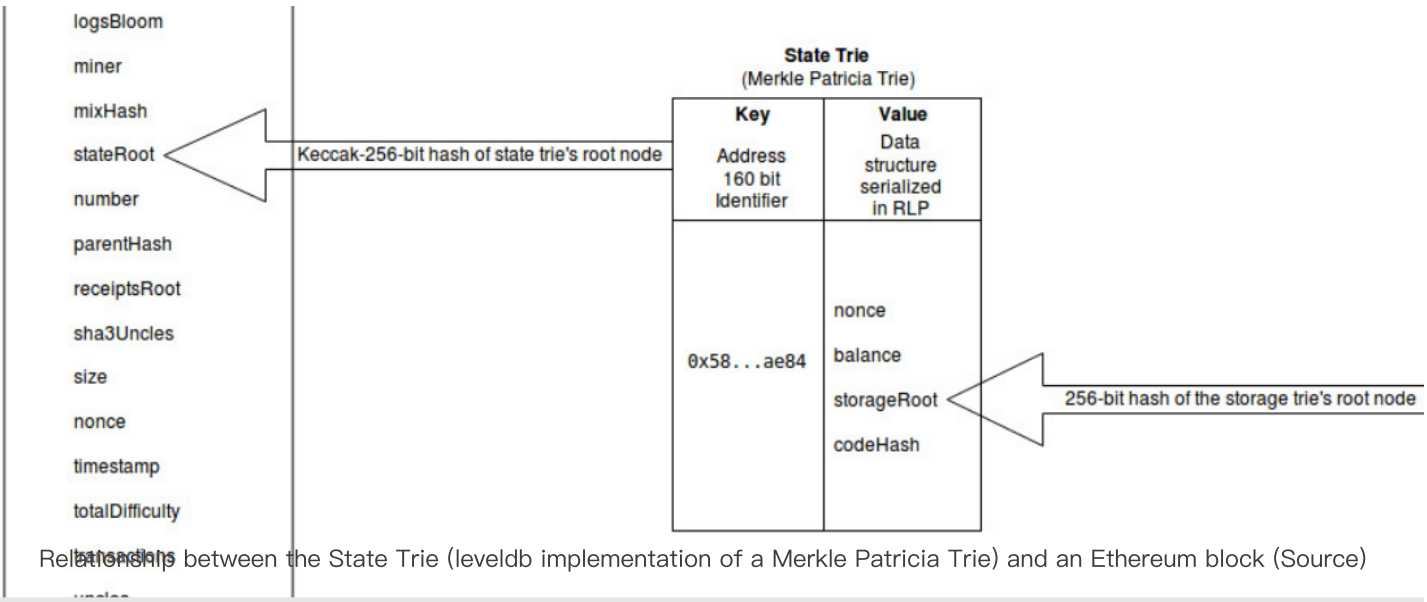
The state trie contains a key and value pair for every account which exists on the Ethereum network.

The "key" is a single 160 bit identifier (the address of an Ethereum account).

The "value" in the global state trie is created by encoding the following account details of an Ethereum account (using the Recursive-Length Prefix encoding (RLP) method):
- nonce
- balance
- storageRoot
- codeHash

The state trie's root node (a hash of the entire state trie at a given point in time) is

The state trie's root node (a hash of the entire state trie at a given point in time) is used as a secure and unique identifier for the state trie; the state trie's root node is cryptographically dependent on all internal state trie data.

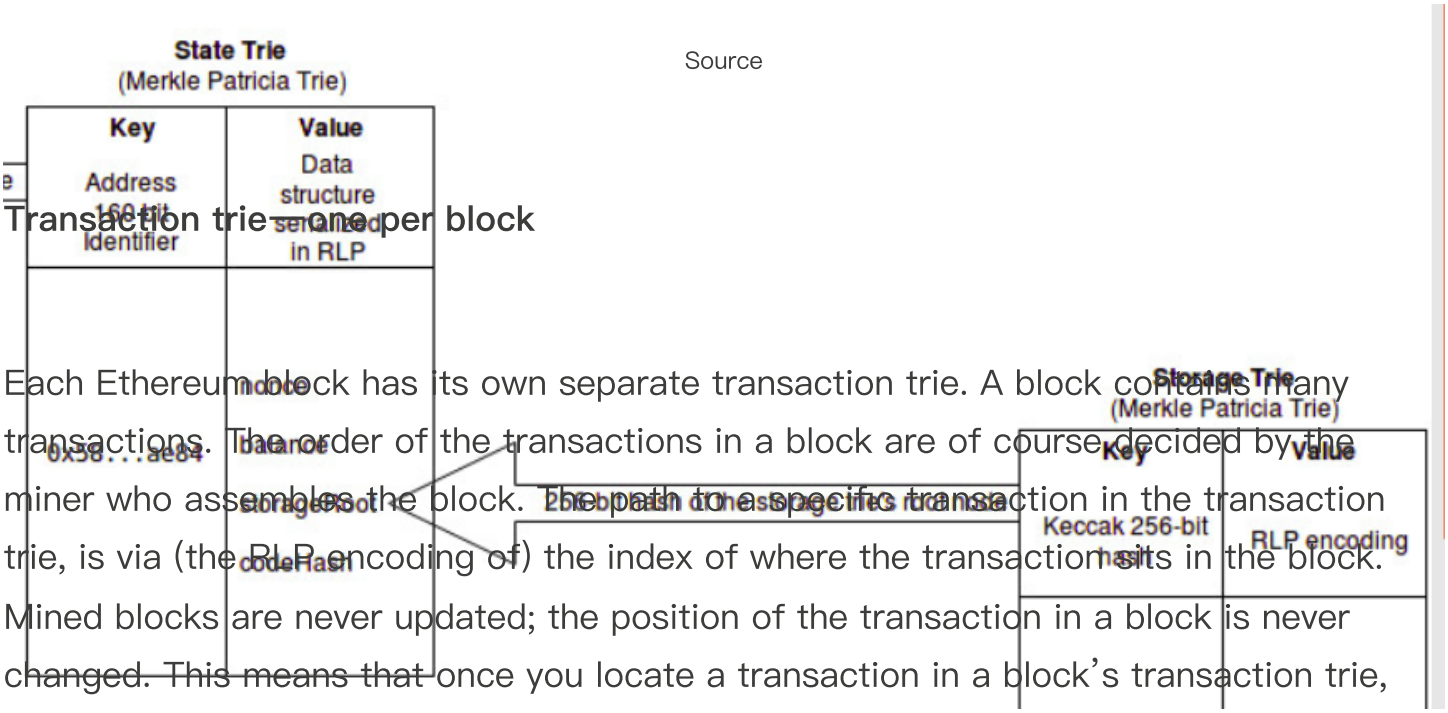


Relationship between the State Trie (leveldb implementation of a Merkle Patricia Trie) and an Ethereum block (Source)

```
size: 533,
stateRoot: "0x8c77785e3e9171715dd34117b047dffe44575c32ede59bde39fbf5dc074f2976",
timestamp: 1517107395,
totalDifficulty: 3021121,
transactions: [],
transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
uncles: []
```

State Trie—Keccak-256-bit hash of the state trie's root node stored as the "stateRoot" value in a given block. stateRoot: '0x8c77785e3e9171715dd34117b047dffe44575c32ede59bde39fbf5dc074f2976' (Source)

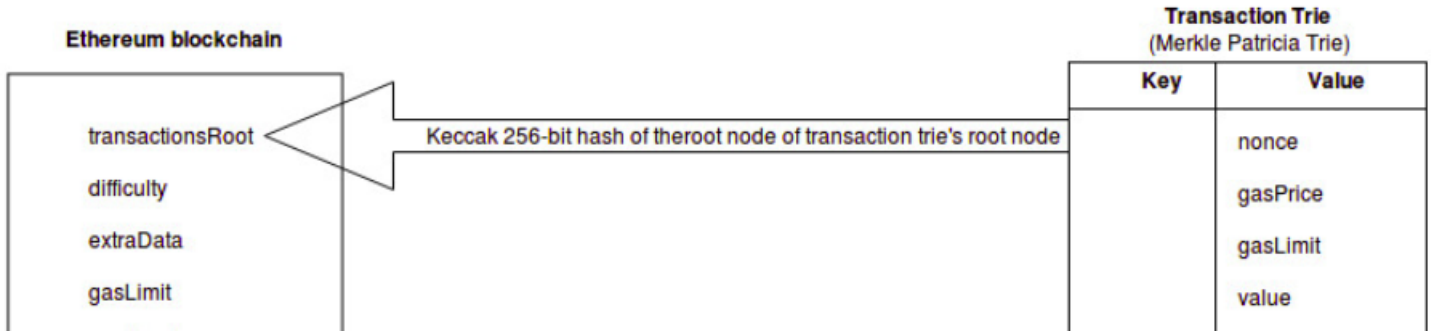
own storage trie. A 256-bit hash of the storage trie's root node is stored as the storageRoot value in the global state trie (which we just discussed).



Source

Transaction trie — one per block

Each Ethereum block has its own separate transaction trie. A block contains many transactions. The order of the transactions in a block are of course decided by the miner who assembles the block. The path to a specific transaction in the transaction trie, is via (the RLP-encoding of) the index of where the transaction sits in the block. Mined blocks are never updated; the position of the transaction in a block is never changed. This means that once you locate a transaction in a block's transaction trie, you can return to the same path over and over to retrieve the same result.



Source

Concrete examples of tries in Ethereum

The main Ethereum clients use two different database software solutions to store their tries. Ethereum's Rust client Parity uses rocksdb. Whereas Ethereum's Go, C++ and Python clients all use leveldb.

Ethereum and rocksdb

Rocksdb is out of scope for this post. This may be covered at a later date, but for now, let's explore how 3 out of the 4 major Ethereum clients utilize leveldb.

Ethereum and leveldb

LevelDB is an open source Google key–value storage library which provides, amongst other things, forward and backward iterations over data, ordered mapping from string keys to string values, custom comparison functions and automatic compression. The data is automatically compressed using "Snappy" an open source Google compression/decompression library. Whilst Snappy does not aim for maximum compression, it aims for very high speeds. Leveldb is an important storage and retrieval mechanism which manages the state of the Ethereum network. As such, leveldb is a dependency for the most popular Ethereum clients (nodes) such as go–ethereum, cpp–ethereum and pyethereum.

Whilst the implementation of the trie data structure can be done on disk (using database software such as leveldb) it is important to note that there is a difference between traversing a trie and simply looking at the flat key/value database.

To learn more, we have to access the data in leveldb using the appropriate Patricia trie libraries. To do this we will need an Ethereum installation.

Here is a easy to follow tutorial for setting up your own Ethereum private network.

How To: Create Your Own Private Ethereum Blockchain
*Dev highlights of this week*medium.com

Once you have set up your Ethereum private network, you will be able to execute

transactions and explore how Ethereum's "state" responds to network activities such as transactions, contracts and mining. If you are not in a position to install an Ethereum private network, that's OK. We will provide our code examples and screen captures from our Ethereum private network.

Analysing the Ethereum database

As we mentioned previously there are many Merkle Patricia Tries (referenced in **each** block) within the Ethereum blockchain:

- State Trie
- Storage Trie
- Transaction Trie
- Receipts Trie

*The following sections assume that you have **installed and configured your very own Ethereum private network,** or that you are happy to just follow along as we run some software and talk to Ethereum's leveldb database.*

To reference a particular Merkle Patricia Trie in a particular block we need to obtain its root hash, as a reference. The following commands allow us to obtain the root hashes of the state, transaction and receipt tries in the genesis block.

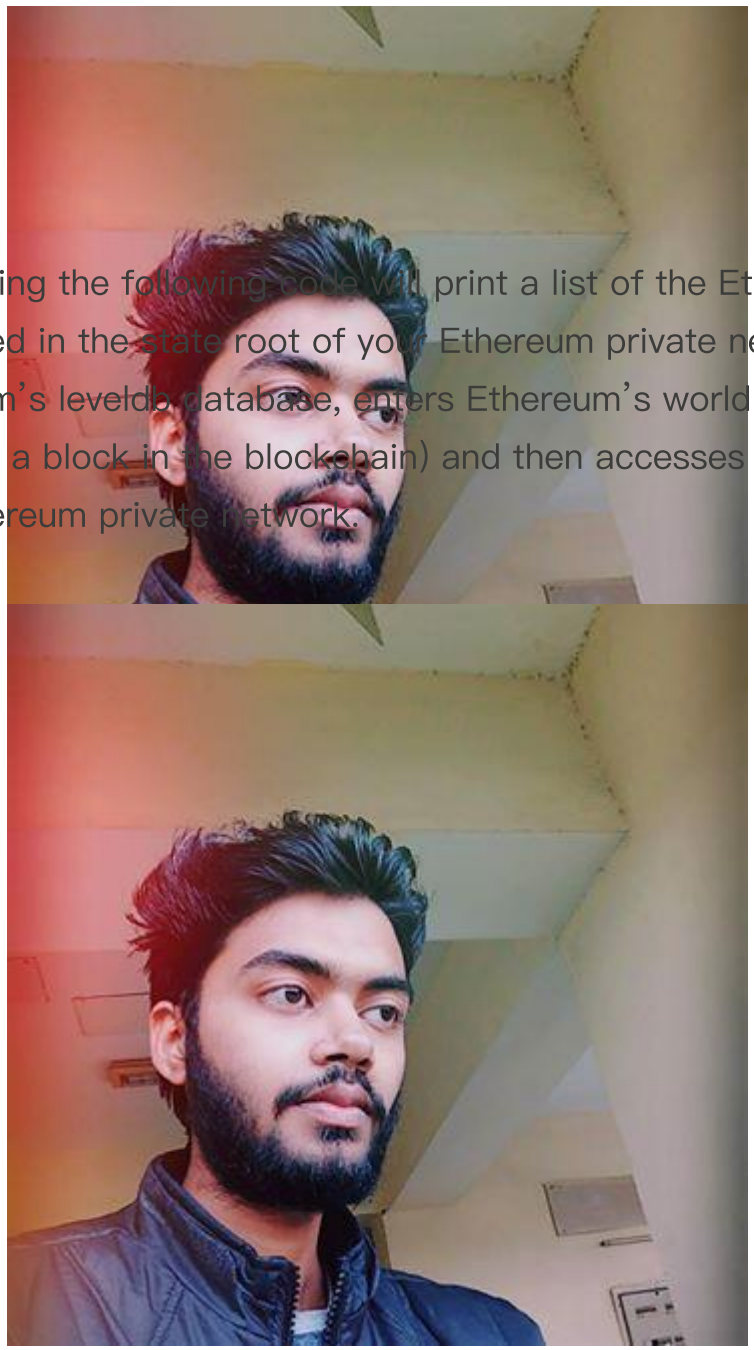
```
> web3.eth.getBlock(0).stateRoot
"0xe4034299877a3fd13eb3da17503af458dd0741ec3b617f2eaf2338ca945edb0e2"
> web3.eth.getBlock(0).transactionsRoot
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421"
> web3.eth.getBlock(0).receiptsRoot
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421"
>
```

block), please use the following command.

Installing npm, node, level and ethereumjs

We will be using a combination of nodejs, level and ethereumjs (which implements Ethereum's VM in Javascript) to inspect the leveldb database. The following commands will further prepare our environment.

From this point, running the following code will print a list of the Ethereum account keys (which are stored in the state root of your Ethereum private network). The code connects to Ethereum's leveldb database, enters Ethereum's world state (using a stateRoot value from a block in the blockchain) and then accesses the keys to all accounts on the Ethereum private network.



```
<Buffer 15 f5 e0 eb 04 db 31 de 72 ff b4 b9 64 0f c9 12 49 af 60 74 d9 8d a1 e1 1f 50 d2 a3 37 55 39 05>
<Buffer be 13 87 9f 13 52 0d 22 33 92 ef 63 74 24 42 b4 56 0c be b7 3f 1d 7e 20 80 96 5f 91 de a5 25 fd>
<Buffer 31 98 3a 89 3e 98 1c b4 1a 9f 3e 49 7e a1 fa 5e 1e 4d 60 fe 18 41 f4 7b 35 af e2 f2 da 85 d1 38>
```

Output for the above code

Interestingly, accounts in Ethereum are only added to the state trie once a transaction has taken place (in relation to that specific account). For

*example, just creating a new account using "geth account new" will not include that account in the state trie; even after many blocks have been mined. However, if a successful transaction (one which costs gas **and** is included in a mined block) is recorded against that account, then and only then will that account appear in the state trie. This is clever logic which protects against malicious attackers continuously creating new accounts and bloating the state trie.*

Decoding the data

You will have noticed that querying leveldb returns encoded results. This is due to the fact that Ethereum uses its own specially "Modified Merkle Patricia Trie" implementation when interacting with leveldb. The Ethereum Wiki provides information in relation to the design and implementation of both Ethereum's Modified Merkle Patricia Trie and Recursive Length Prefix (RLP) encoding. In short, Ethereum have extended on the trie data structures. For example the Modified Merkle Patricia contains a method which can shortcut the descent (down the trie) through the use of an "extension" node.

In Ethereum, a single Modified Merkle Patricia trie node is either:

- an empty string (referred to as NULL)
- an array which contains 17 items (referred to as a branch)
- an array which contains 2 items (referred to as a leaf)
- an array which contains 2 items (referred to as an extension)

As Ethereum's tries are designed and constructed with rigid rules, the best way to inspection them is through the use of computer code. The following example uses ethereumjs. The ethereumjs repositories are easy to install and use; they will be perfect for us to quickly peer into Ethereum leveldb database.

The code below (when provided with a particular block's stateRoot as well as an

The code below (when provided with a particular block's stateRoot as well as an Ethereum account address) will return that account's correct balance in a human readable form.

```
Account Address: 0xccc6b46fa5606826ce8c18fece6f519064e6130b
Balance: 300000
```

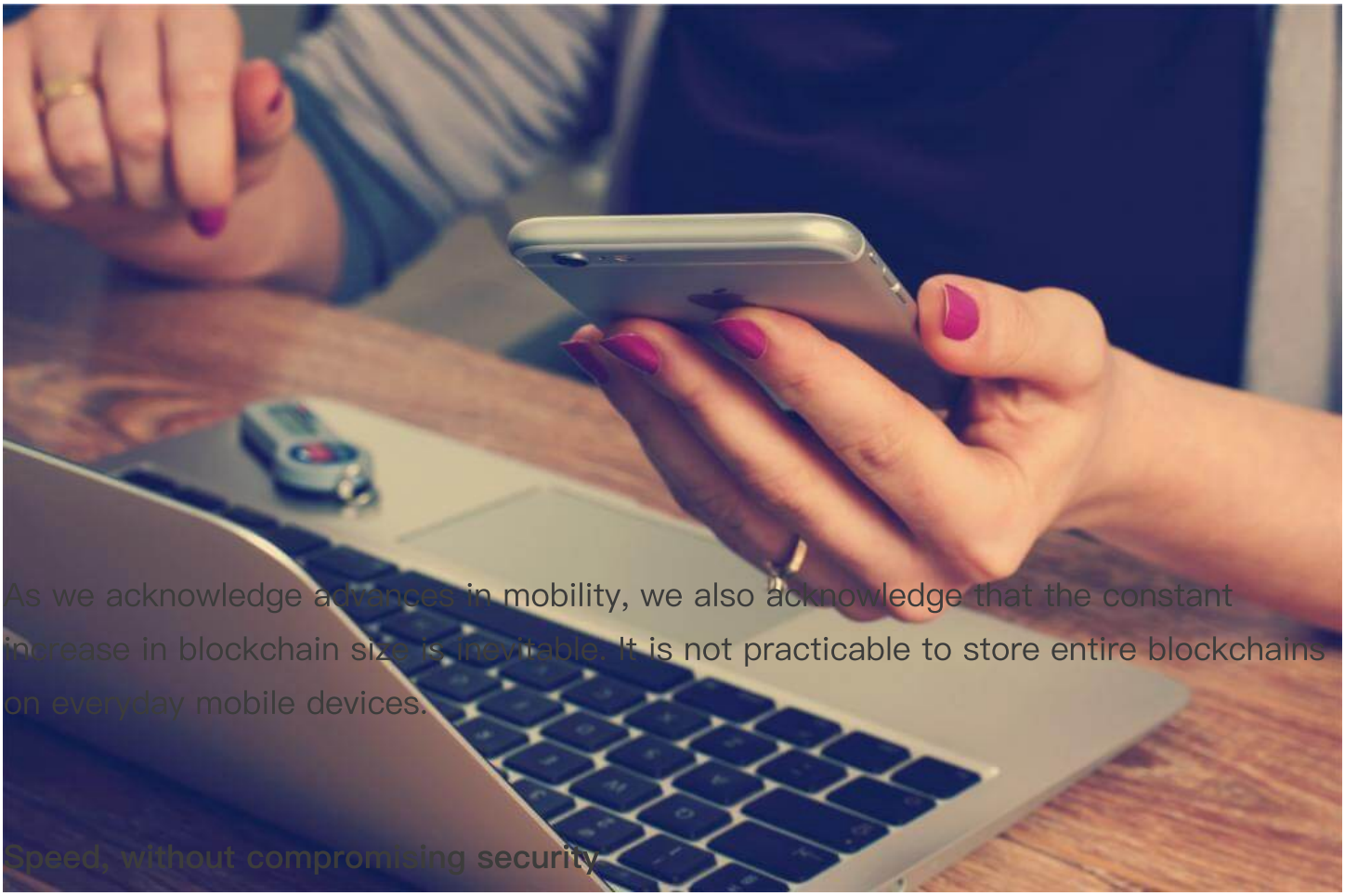
Output from the following code (account balance for address `0xccc6b46fa5606826ce8c18fece6f519064e6130b`)

Conclusion

We have demonstrated above that Ethereum has the ability to manage its "state". This clever upfront design has many advantages.

Mobility

Given that mobile devices and Internet of Things (IoT) devices are now ubiquitous, the future of e-commerce depends on safe, robust and fast mobile applications.



The design of Ethereum's world state and its use of the Modified Merkle Patricia Trie provides many opportunities in this space. Every function (put, update and delete) performed on a trie in Ethereum utilizes a deterministic cryptographic hash. Further, the unique cryptographic hash of a trie's root node can be used as evidence that the trie has not been tampered with.

For example any changes to a trie's data, at any level (such as increasing an accounts balance in the leveldb database) will completely change the root hash. This cryptographic feature provides an opportunity for light clients (devices which do not store the entire blockchain) to quickly and reliably query the blockchain i.e. does account "0x … 4857" have enough funds to complete this purchase at block height "5044866"?

“The size complexity of a Merkle proof is logarithmic in the quantity of data stored. This means that, even if the entire state tree is a few gigabytes in size, if a node receives a state root from a trusted source that node has the ability to know with full certainty the validity of any information with the tree by only downloading a few kilobytes of data in a proof.”

Spend limits

An interesting idea, mentioned in the Ethereum white paper is the notion of a savings account. In this scenario two users (perhaps a husband and wife, or business partners) can each withdraw 1% of the accounts total balance per day. This idea is only mentioned in the “further applications” section of the white paper, but it sparks interest because of the fact that, in theory, it could be implemented as part of Ethereum’s base protocol layer (as apposed to having to be written as part of a second layer solution or third–party wallet). You may recall our discussion about bitcoin UTXOs at the start of this article. UTXOs are blind to blockchain data, and as we discussed, the bitcoin blockchain does not actually store a users account balance. For this reason the base protocol layer of bitcoin is far less likely (or perhaps unable to) implement any sort of daily spend limits.

Consumer confidence

As work continues in this space we will see a lot of development in light clients. More specifically, safe, robust and fast mobile applications, which can interact with blockchain technologies.

Thanks for reading ;)

Learned something? Press and hold the 👏 to say "thanks!" and help others find this article.

Hold down the clap button if you liked the content! It helps me gain exposure .

Want to learn more? Checkout my previous articles.

HackPedia: 16 Solidity Hacks/Vulnerabilities, their Fixes and Real World Examples
*A Complete List of all Solidity Hacks/Vulnerabilities, their Fixes and Real World Hack Examples*hackernoon.com

ConsensusPedia: An Encyclopedia of 30 Consensus Algorithms

*A complete list of all consensus algorithms.*hackernoon.com

ContractPedia: An Encyclopedia of 40 Smart Contract Platforms

*A Complete List of all Smart Contract supportive Platforms*hackernoon.com

Difference between SideChains and State Channels

*A complete comparison of the two scaling methods.*hackernoon.com

Clap 50 times and follow me on Twitter: @vasa_develop

7

by Vaibhav Saini @vasa.

Entrepreneur | Co-founder @tbc_inc, an MIT CIC incubated startup | Speaker |https://vaibhavsaini.com

This article was featured in...

Horizen | Coindesk | Stackexchange | Lucassaldanha | Learnblockchain | Dzone | Yahoo

Related Stories

Subject Matter

Smart Contract Versioning by @vasa

#ethereum

The Future of the Creator Economy by @kadeemclarke

#creator-economy

How to Build an Ethereum Transaction App with React and Solidity: Part 2 by @daltonic

#react

Interest in Crypto-based Derivatives Decreased, New Data Indicates by @Ishan Pandey

#crypto-trading

Buying and Using an ENS Domain: A Guide, A Cautionary Tale, and a 1000-ft Review of Ethereum's Web3 by @utsavjaiswal

#ethereum

What is the Difference Between Uniswap V1, V2, and V3? by @blockscribers

#uniswap

TAGS

#ethereum #how-ethereum-works #storage-in-ethereum #ethereum-data-storage #ethereum-data