Tr

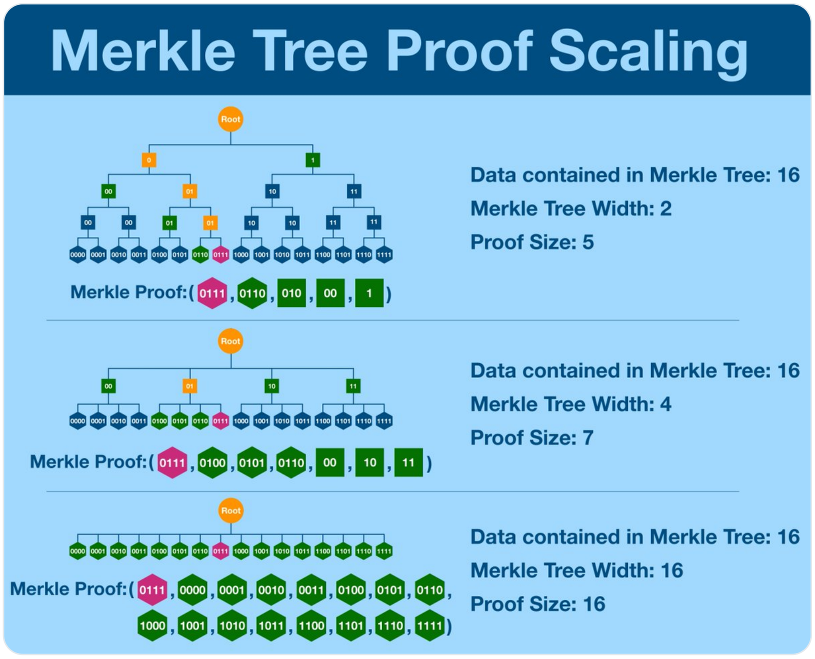
Haym @SalomonCrypto

Oct 29 • 19 tweets • SalomonCrypto/status/1586391080727064577

(1/18) The Problem with Merkle Trees

At the heart of @Bitcoin, @ethereum and many blockchain computers is the Merkle Tree. While this data structure has served us well, it is not perfect... and if you look ahead, you can see impending problems.

Let's talk Merkle proof scaling.



Merkle Tree Proof Scaling
Root
Data contained in Merkle Tree: 16
Merkle Tree Width: 2
Proof Size: 5
Merkle Proof:(0111, 0110, 010, 00, 1)
Data contained in Merkle Tree: 16
Merkle Tree Width: 4
Proof Size: 7
Merkle Proof:(0111, 0100, 0101, 0110, 00, 10, 11)
Data contained in Merkle Tree: 16
Merkle Tree Width: 16
Proof Size: 16
Merkle Proof:(0111, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111)

(2/18) A Merkle Tree is a data structure that allows the efficient verification of the contents of a large dataset.

Check out this thread for a deeper dive, but we'll quickly review the basics.

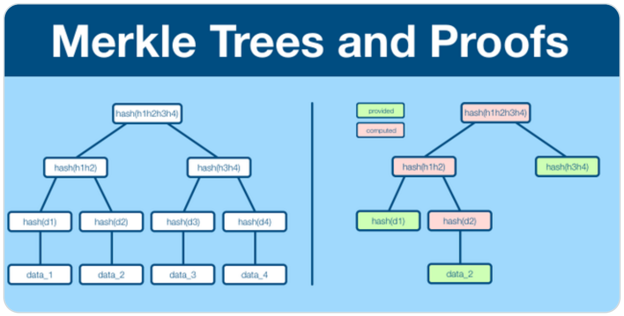
(also note the typo below... Markle tree HAHA)

Haym
@SalomonCrypto · **Follow**

(1/13) Computer Science 201: Merkle Trees and Merkle Proofs

If you want to understand @Bitcoin, @ethereum and blockchain technology, you need to learn:

- How a Merkle trees expresses a large dataset
- How a Merkle proof works
- Why a Markle tree is so efficient



10:17 PM · Sep 7, 2022

Read the full conversation on Twitter

440 Reply Copy link

Read 15 replies

(3/18) Merkle Trees are made by successive rounds of hashing (applying a hash function to a set of data).

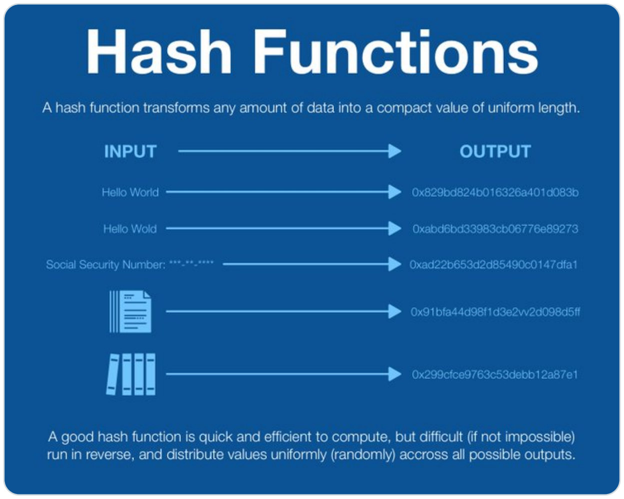
A hash function transforms data of arbitrary contents (and length) into a unique, compact string. Think of a hash like a unique serial number for a set of data.

Haym
@SalomonCrypto · **Follow**

(1/7) Computer Science 101: Hash Functions

What is a hash function? What are the characteristics of a good hash function? Where do hash functions appear and why do I hear about them all the time?

If you want to understand the fundamental tool of crypto, this guide is for you!


Hash Functions
INPUT
OUTPUT
Hello World
0x829bd824b016326a401d083b
Hello Wold
0xabd6bd33983cb06776e89273
Social Security Number: ***-**-****
0xad22b653d2d85490c0147dfa1
0x91bfa44d98f1d3e2vv2d098d5ff
0x299cfce9763c53debb12a87e1

A hash function transforms any amount of data into a compact value of uniform length.

A good hash function is quick and efficient to compute, but difficult (if not impossible) run in reverse, and distribute values uniformly (randomly) accross all possible outputs.

3:54 PM · Sep 7, 2022

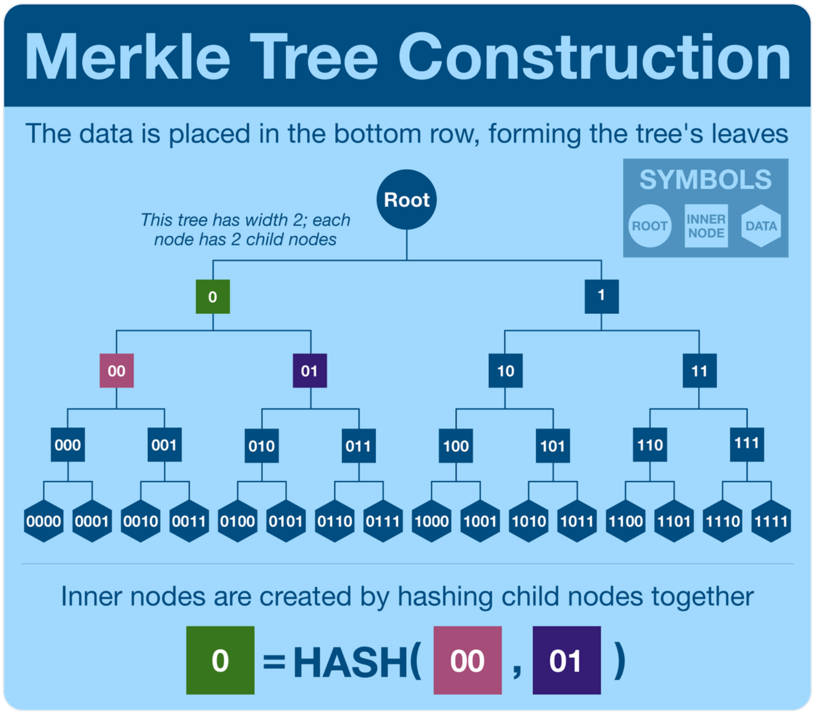
Read the full conversation on Twitter

123 Reply Copy link

Read 1 reply

(4/18) We begin by placing our data into the bottom row and picking a tree width (number of children per node). To move up a level, we simply select [width] number of nodes and hash them together.

This continues until a single hash is reached. We call this hash the Merkle root.



(5/18) Remember, hashing creates an identifier for whatever was put into the hash function. This method will generate a unique string for each node, irrevocably tied to the underlying dataset.

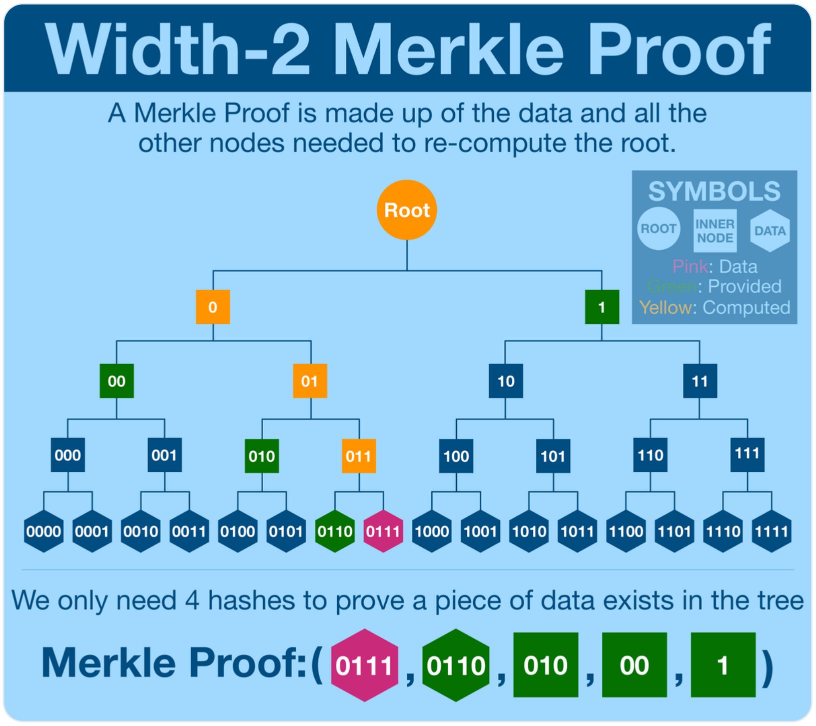
Any changes to the data will require updating the tree, all the way up to the root.

(6/18) Once we have our Merkle root, we are in business. The root is a lightweight string that can be used to verify any single piece of data in the associated Merkle Tree.

Your tree can have billions of items, yet our Merkle root can verify even a single datum spec.

(7/18) We call the trick a Merkle proof; rather than transfer the entire dataset, a user can send a single data point AND the hashes that make up the intermediate nodes of the Merkle Tree.

Using just the branches and the data point, a verifier can rebuild the Merkle root.



(8/18) If we didn't use computer science, (basically) the only way to verify a piece of data is in the dataset is to pass the entire dataset.

The Merkle Tree is a huge improvement, drastically reducing the amount of data transmission needed to verify the point.

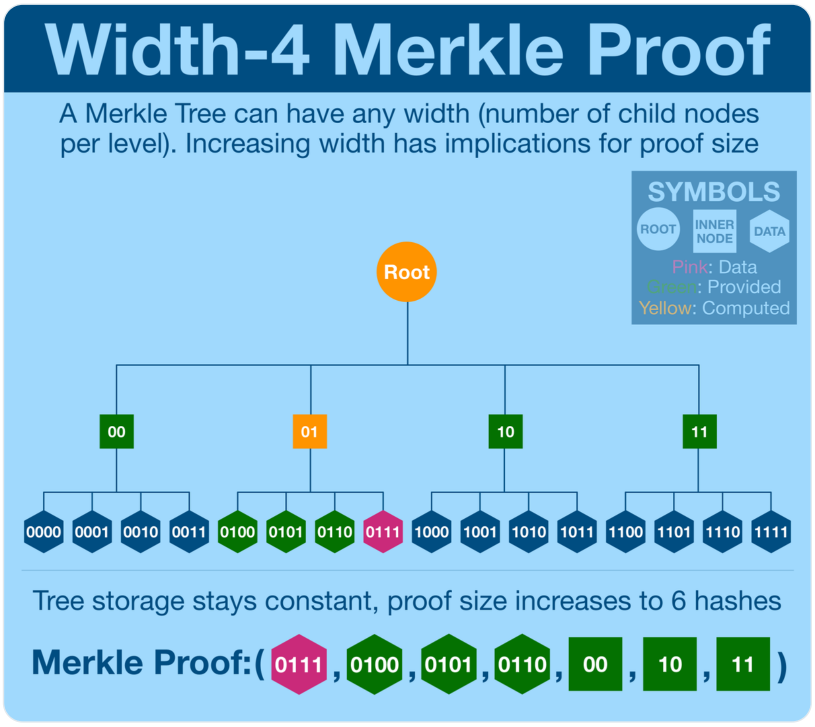
(9/18) Now, you might look at the graphic above and notice something: most of the components of the Merkle proofs are inner nodes (squares).

What if we could reduce the number of inner nodes by increasing the width (and therefore decreasing the height) of the tree?

(10/18) Let's take a look at a wider tree, one with width 4.

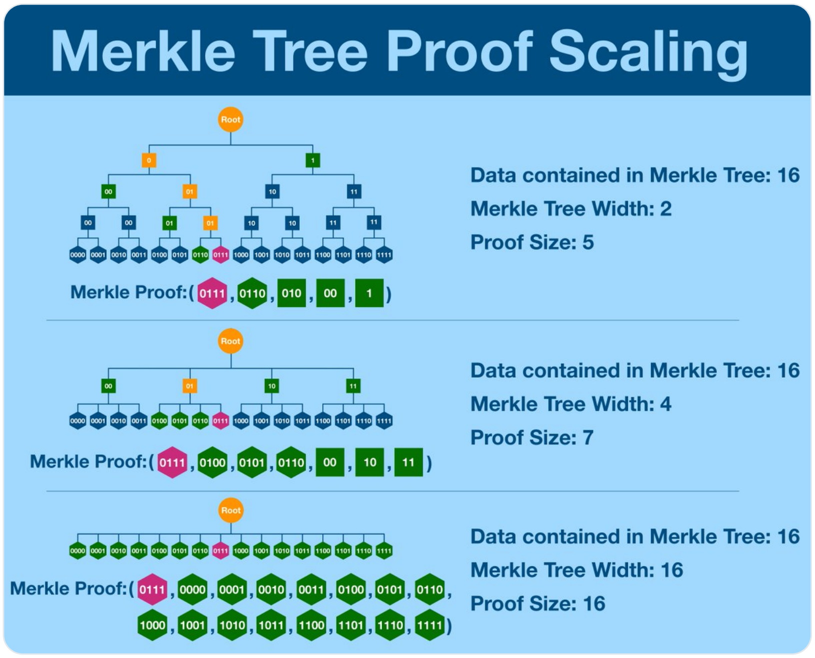
Turns out, the Merkle proof is actually LARGER. Yes, there are less levels we need to traverse before we reach the root, but each level requires so much more data.

Maybe a width of 4 is especially bad?


Width-4 Merkle Proof
A Merkle Tree can have any width (number of child nodes per level). Increasing width has implications for proof size
SYMBOLS
ROOT
INNER NODE
DATA
Pink: Data
Green: Provided
Yellow: Computed
Root
00
01
10
11
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
Tree storage stays constant, proof size increases to 6 hashes
Merkle Proof:(0111 , 0100 , 0101 , 0110 , 00 , 10 , 11)

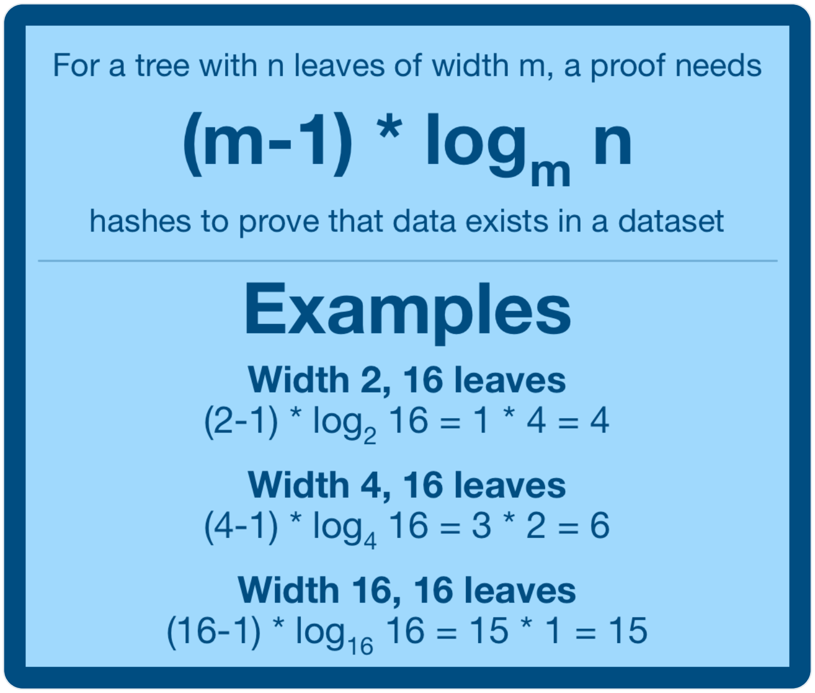
(11/18) Unfortunately, this is not the case. The wider a Merkle Tree gets, the less efficient the Merkle proofs are. In fact, a Merkle Tree is optimal when width = 2.

Take a look at the last one, where width = 16. That's what @ethereum uses.


Merkle Tree Proof Scaling
Merkle Proof:(0111 , 0110 , 010 , 00 , 1)
Data contained in Merkle Tree: 16
Merkle Tree Width: 2
Proof Size: 5
Merkle Proof:(0111 , 0100 , 0101 , 0110 , 00 , 10 , 11)
Data contained in Merkle Tree: 16
Merkle Tree Width: 4
Proof Size: 7
Merkle Proof:(0111 , 0000 , 0001 , 0010 , 0011 , 0100 , 0101 , 0110 , 1000 , 1001 , 1010 , 1011 , 1100 , 1101 , 1110 , 1111)
Data contained in Merkle Tree: 16
Merkle Tree Width: 16
Proof Size: 16

(12/18) Here's the formula for calculating how many components a Merkle proof will need.

Once again, the end result is that Merkle Trees are optimal when width = 2.



For a tree with n leaves of width m, a proof needs

$$(m-1) * \log_m n$$

hashes to prove that data exists in a dataset

Examples

Width 2, 16 leaves
$(2-1) * \log_2 16 = 1 * 4 = 4$

Width 4, 16 leaves
$(4-1) * \log_4 16 = 3 * 2 = 6$

Width 16, 16 leaves
$(16-1) * \log_{16} 16 = 15 * 1 = 15$

(13/18) Merkle trees are incredibly powerful and versatile data structures that will continue to see use long after you and I are gone, but they are not perfect for all applications.

This is especially true as the tree grows; proof size increases with tree/dataset size.

(14/18) And this is ultimately the problem we find ourselves up against. @ethereum uses Merkle trees to store the state of the EVM. Every account, balance, asset, protocol, etc is stored inside of Merkle tree, updated with every block, every 12 seconds.

(15/18) Today, the tree (technically multiple) is MASSIVE, but we are still in @ethereum's infancy. Yes, the Merkle tree still is much more efficient that transferring the entire data set, but the Merkle proofs are getting so large we are running up against the same barriers.

(16/18) As previously mentioned, today @ethereum uses Merkle trees with a width of 16. We can make some (significant) improvements just by transitioning to the optimal Merkle structure.

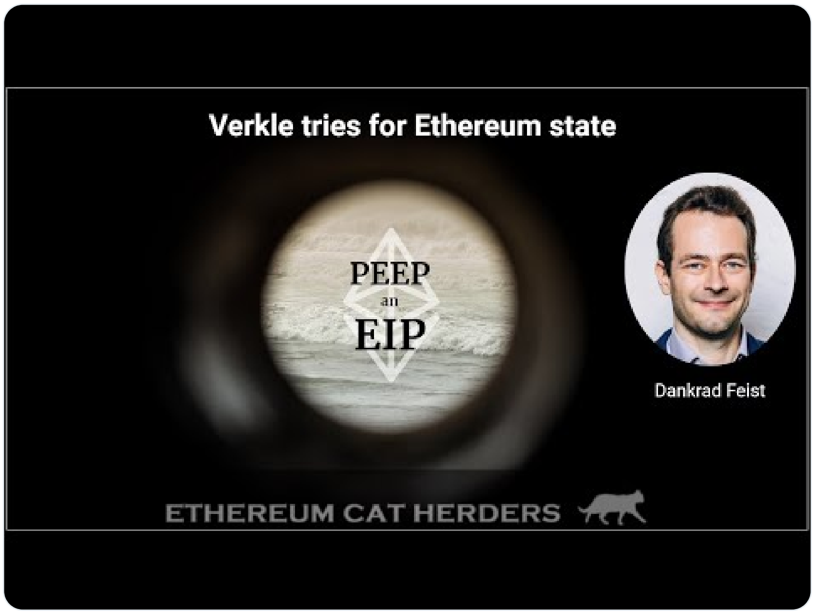
But that's just a band-aid for a systemic issue, for the tree will still inexhaustibly grow.

(17/18) So, dear reader, here's where I am going to leave you off: with a huge freaking problem. The data structure at the core of @ethereum is literally not scalable, it's only going to work as long as Ethereum remains (relatively) unused.

But don't worry, we'll be back.

(18/18) We haven't spent all this time just figuring out what was wrong with Merkle Trees.

We've also been working on a replacement!



https://www.youtube.com/embed/RGJOQHzg3UQ

Like what you read? Help me spread the word by retweeting the thread (linked below).

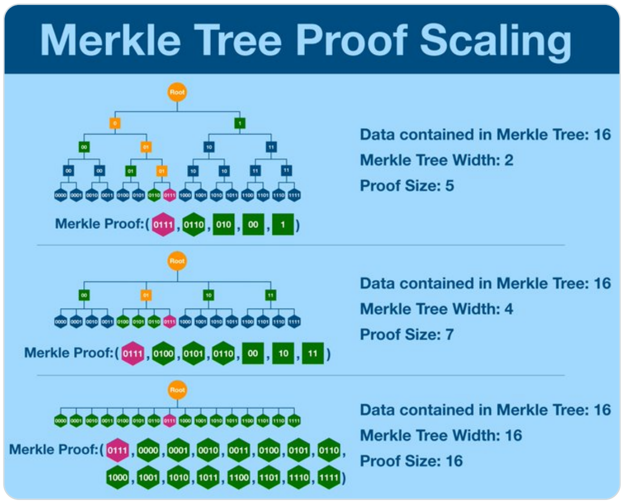
Follow me for more explainers and as much alpha as I can possibly serve.

Haym
@SalomonCrypto · **Follow**

(1/18) The Problem with Merkle Trees

At the heart of @Bitcoin, @ethereum and many blockchain computers is the Merkle Tree. While this data structure has served us well, it is not perfect... and if you look ahead, you can see impending problems.

Let's talk Merkle proof scaling.


Merkle Tree Proof Scaling
Root
Merkle Proof:(0111 , 0110 , 010 , 00 , 1)
Data contained in Merkle Tree: 16
Merkle Tree Width: 2
Proof Size: 5
Merkle Proof:(0111 , 0100 , 0101 , 0110 , 00 , 10 , 11)
Data contained in Merkle Tree: 16
Merkle Tree Width: 4
Proof Size: 7
Merkle Proof:(0111 , 0000 , 0001 , 0010 , 0011 , 0100 , 0101 , 0110 , 1000 , 1001 , 1010 , 1011 , 1100 , 1101 , 1110 , 1111)
Data contained in Merkle Tree: 16
Merkle Tree Width: 16
Proof Size: 16

4:14 PM · Oct 29, 2022

Read the full conversation on Twitter

34 Reply Copy link

Read 3 replies

• • •