Blockchain and Distributed Ledger technologies

SAPIENZA
UNIVERSITÀ DI ROMA

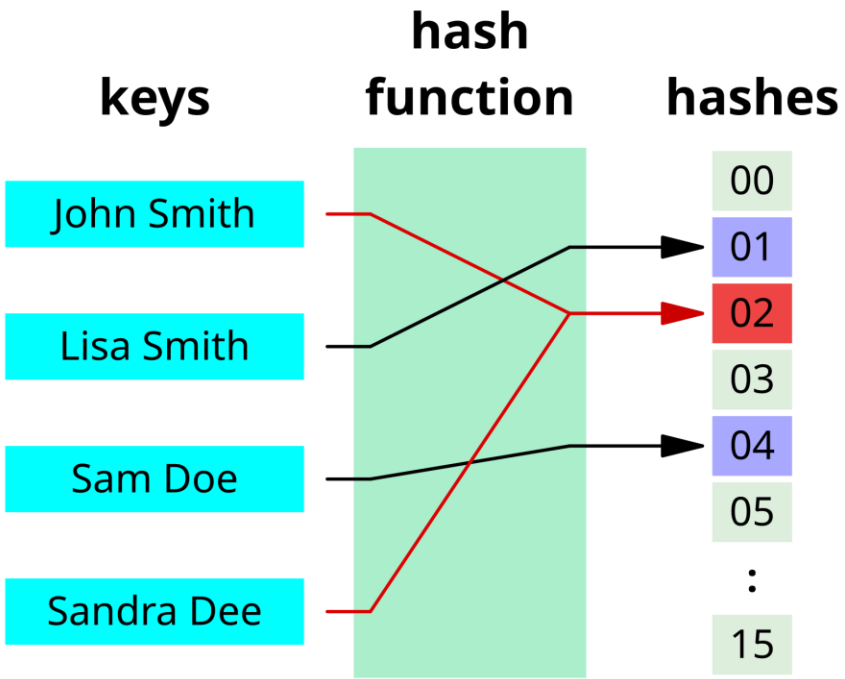
Massimo La Morgia
massimo.lamorgia@uniroma1.it

Hash

Definition

A hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values.

Hashing is a method of applying a cryptographic hash function to data, which calculates a relatively unique output (called a message digest, or just digest) for an input of nearly any size



Images from Wikipedia and definition from NISTIR 8202 and "Handbook of Applied Cryptography" (Menez, Van Oorschot, Vanstone)

Hash

H(“Bitcoin”) = b4056df6691f8dc72e56302ddad345d65fead3ead9299609a826e2344eb63aa4

H(”bitcoin”) = 6b88c087247aa2f07ee1c5956b8e1a9f4c7f892a70e324f1bb3d161e05ca107b

H(“1”) = 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b

H(“2”) = d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35

H() is the SHA-256

Hash - properties

Hash function general properties

- Its input can be any string of any size.
- It produces a fixed size output.
- It is efficiently computable.

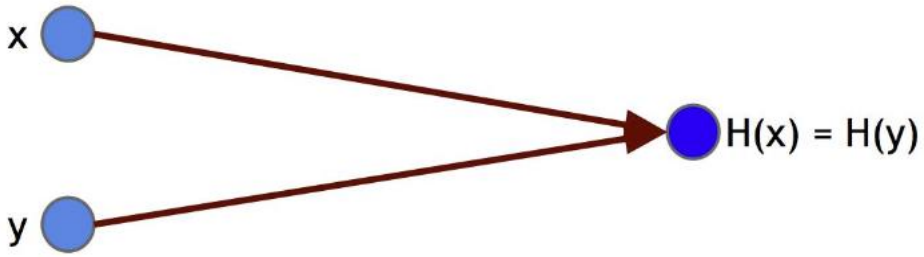
cryptographic hash functions properties

- Collision-resistance
- Hiding
- Puzzle-friendliness

Hash – Collision-resistance

Collision-resistance: *A hash function H is said to be collision resistant if it is infeasible to find two values, x and y , such that x ≠ y , yet H(x) = H(y) .*

This properties want that **nobody** can find a case in which two different input values for the hash function provide the same value as a output.

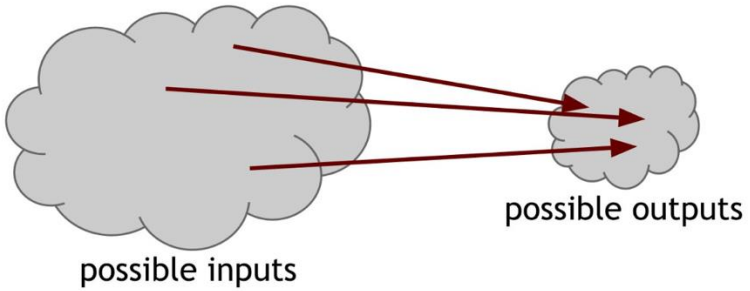


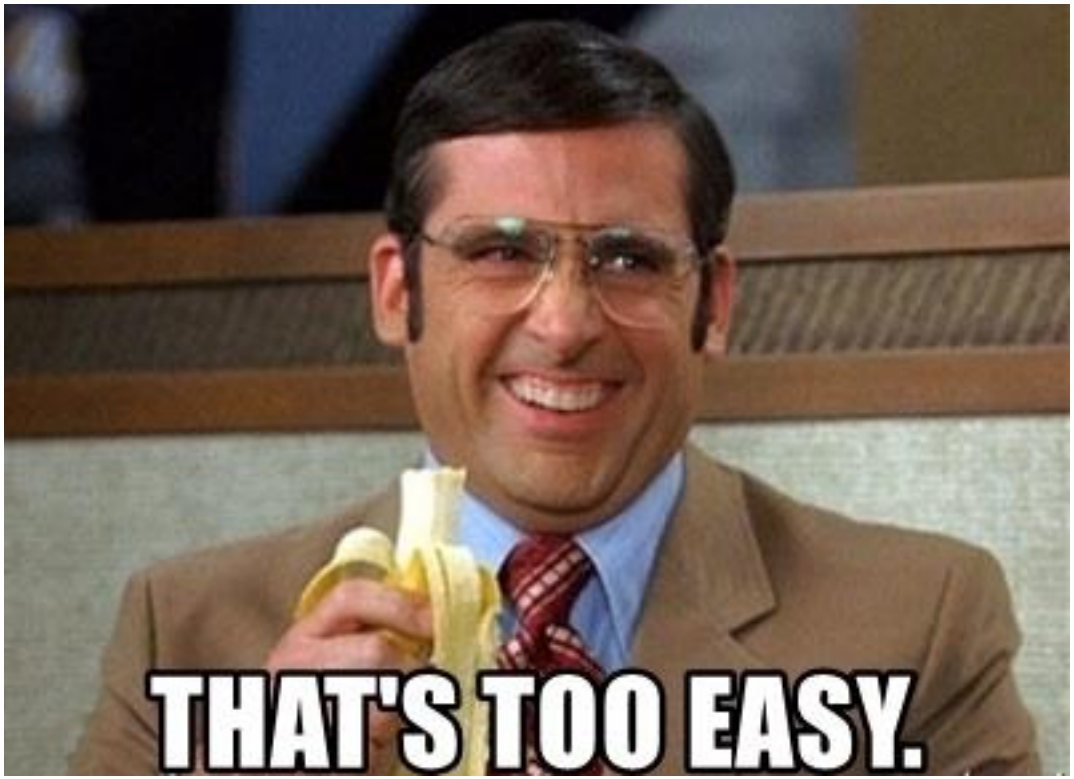
The case we want to avoid

Note that: **nobody** can find a collision is very different from says that **no collision exist.**

Hash – finding a collision

1. Consider an hash function with 256-bit output size.
2. Now pick $2^{256} + 1$ input.
3. Computes the hash of each of them.
4. For sure you have found a collision.





Hash – finding a collision

2^{256}=115792089237316195423570985008687907853269984665640564039457584007913129639936

if a computer calculates 10,000 hashes per second (H/s), it would take more than 10^{27} years (octillion) to calculate 2^{128} hashes!

A modern computer (ASIC) calculates 234 TH/s.

So it needs 46077254411749 years.

https://hashrateindex.com/blog/top-10-bitcoin-mining-asic-machines-of-2025-2/

Hash – finding a collision

Let's go by chance:

You have $\frac{1}{2^{256}}$ probability to find a collision (remember that 2^{256} is a 78 digits number).

Probability to win at a "superenalotto" lottery: $\frac{1}{622614630}$

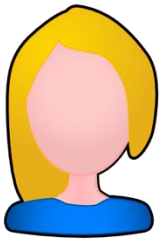
Probability to win at Scratch & win 6'000'000€ spending 25€: $\frac{1}{10080000}$

https://www.grattaevinci.com/probabilita-di-vincita

Applications: Message digest

Alice has a very large file and uploads it to a cloud storage service.

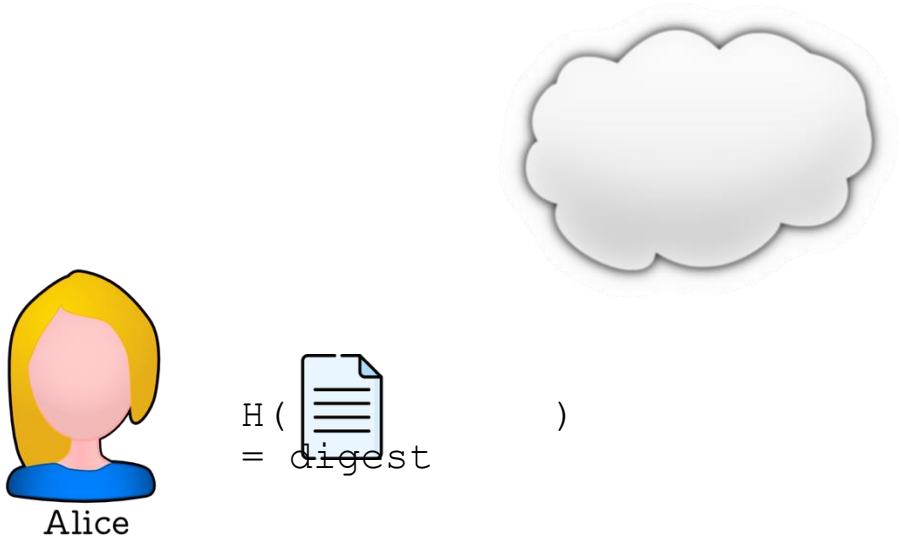
Applications: Message digest



Alice

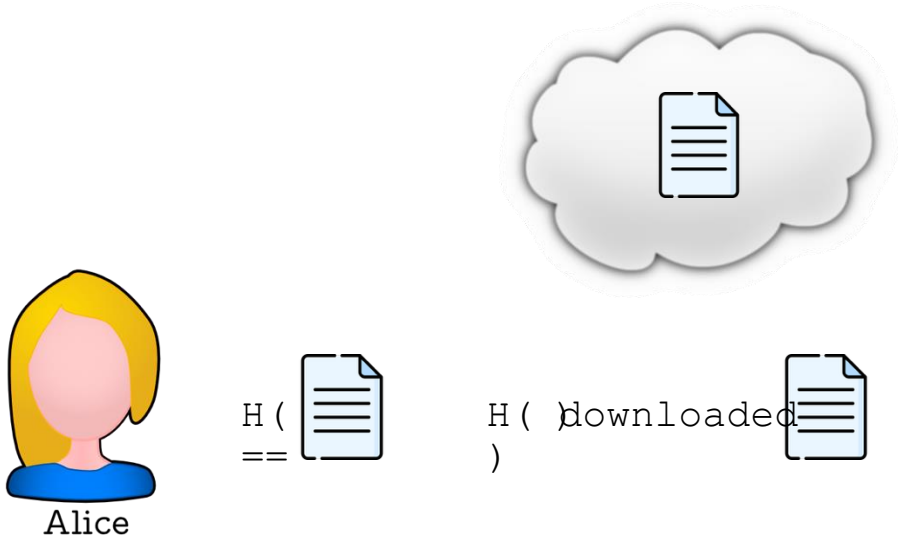
How can Alice be sure that the service provider does not modify her file?

Applications: Message digest



Before uploading the file, Alice computes the hash of the file (digest) and stores it.

Applications: Message digest



Now, when Alice needs the file, she can download it, compute the hash of the downloaded file, and check whether the two digests match.

If they match, Alice can be sure that the file has not been modified.

Hash – Hiding

Hiding: *A hash function H is hiding if: when a secret value r is chosen from a probability distribution that has high min-entropy , then given H(r ‖ x) it is infeasible to find x .*

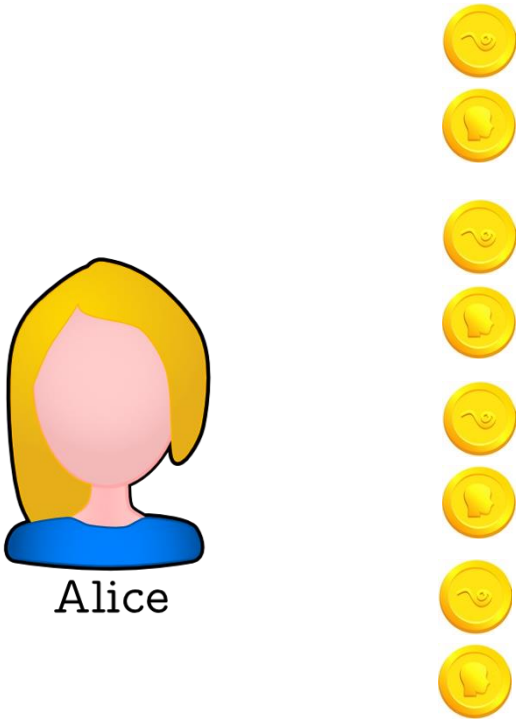
r ‖ x = is r concatenated x

In information-theory, min-entropy is a measure of how predictable an outcome is, and high min-entropy captures the intuitive idea that the distribution (i.e., random variable) is very spread out.

In other words: If we're given the output of the hash function H(r||x) = y , there's no feasible way to figure out what the input, x , was.

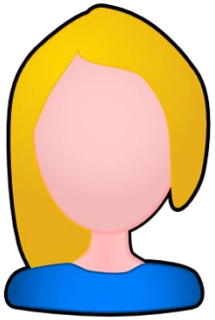
.

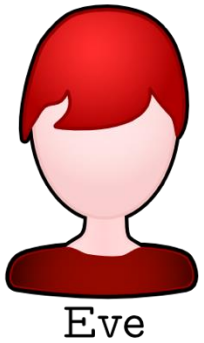
Hiding: heads or tails



Alice is playing at heads or tails.

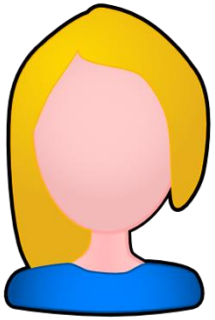
Hiding: heads or tails





Eve has to guess Alice's output of the coinflip.

Hiding: heads or tails



Alice

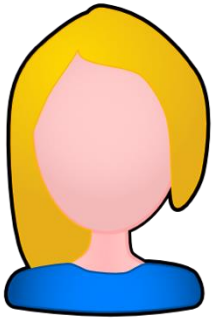
H( ) = a

H( ) = b

Eve

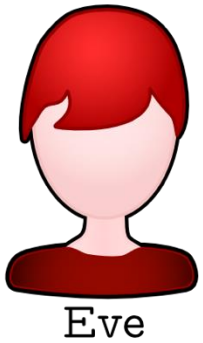
Alice reveals only the hash of the output.

Hiding: heads or tails



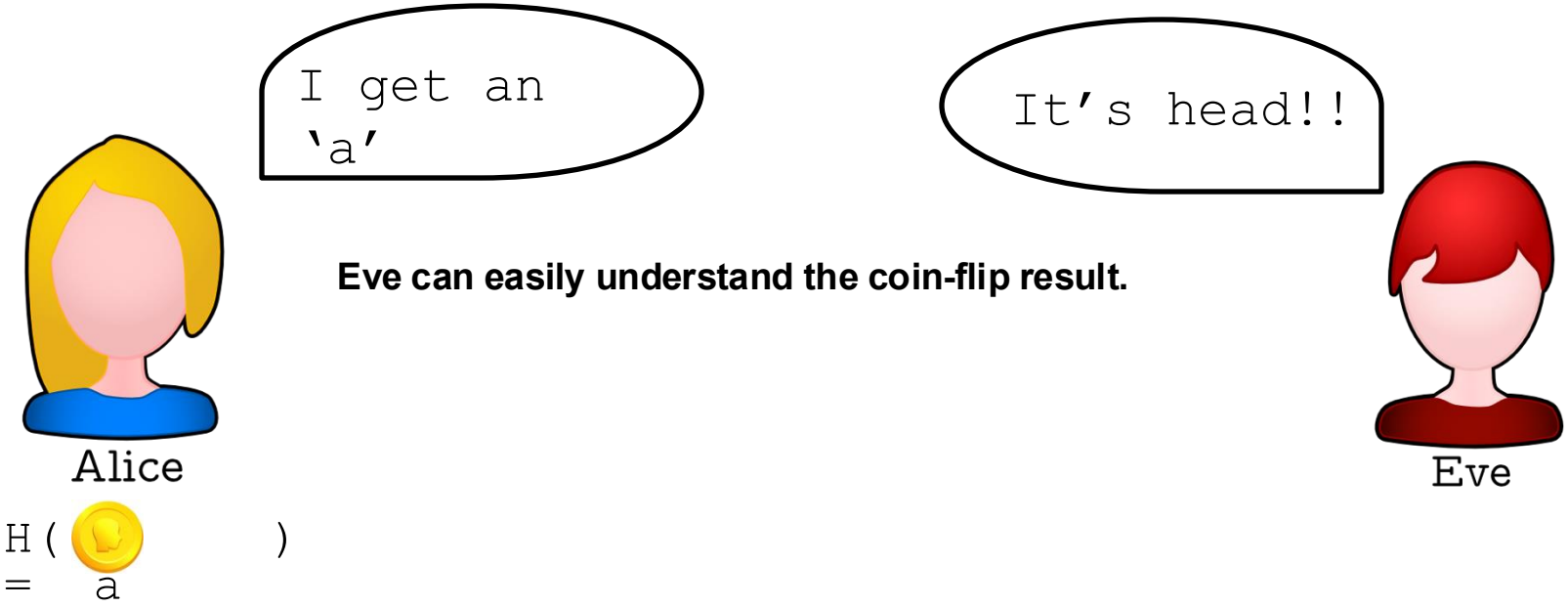
H( ) = a

H( ) = b



If Eve knows the hash function and the set of possible outputs, she can precompute every one of them.

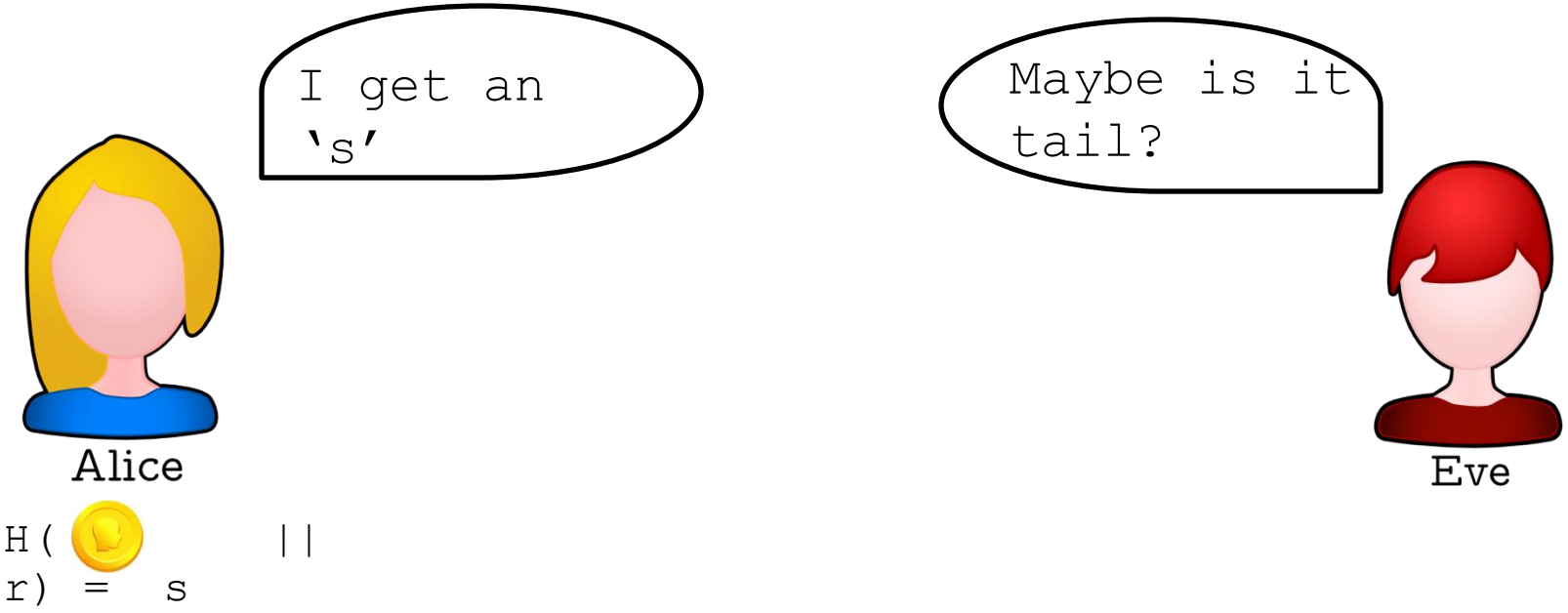
Hiding: heads or tails



Eve can easily understand the coin-flip result.

Eve can easily understand the coin-flip result.

Hiding: heads or tails



It is possible to hide an input that's not spread out by concatenating it with another input that is spread out.

Applications: Commitments

Goal: To allow one party to fix a value in a way that is both binding (the value cannot be changed later) and hiding (the value is not revealed until it is intentionally opened).

How the protocol works: A commitment protocol aims to enable a party to commit to a value in such a way that the value remains hidden until the party chooses to reveal it. At the same time, the protocol must be binding, ensuring that the committer cannot change the value after the commitment is made, and hiding, guaranteeing that no information about the value is leaked before it is opened. To achieve this, the protocol generates a verifiable proof of commitment that convinces the verifier of its validity without revealing the underlying value.

Use case

Cheat-Proof Result Prediction Scheme:

- I want to predict the result of a football match, but I do not want to reveal my prediction before the match begins.
- You need assurance that I cannot change my prediction after the match has ended.
- To achieve this, I generate a commitment by scrambling my prediction with a cryptographic method and share only the commitment with you.
- After the match, I reveal both my original prediction and the method I used to generate the commitment.
- You can then verify that my revealed prediction matches the original commitment, proving that I did not cheat.

Applications: Commitments

A commitment scheme consists of two functions:

The commitment function

com := commit(msg, nonce)

The commit function takes a message and secret random value, called a **nonce**, as input and returns a commitment.

The verify function

verify(com, msg, nonce)

The verify function takes a commitment, nonce, and message as input.
It returns true if com == commit(msg , nonce) and false otherwise.

We require that the following two security properties hold:

1. **Hiding** : Given com , it is infeasible to find msg

2. **Binding** : It is infeasible to find two pairs (msg, nonce) and (msg', nonce') such that msg ≠msg' and commit (msg, nonce) == commit(msg' , nonce')

Hash – Puzzle-friendliness

Puzzle-friendliness: *A hash function H is said to be **puzzle-friendly** if for every possible n-bit output value y , if k is chosen from a distribution with high min-entropy, then it is infeasible to find x such that **H(k ‖ x) = y** in time significantly less than* 2^n.

Recall that k ‖ x = is k concatenated x

In information-theory, min-entropy is a measure of how predictable an outcome is, and high min-entropy captures the intuitive idea that the distribution (i.e., random variable) is very spread out.

In other words: if someone wants to target the hash function to come out to some particular output value y , it's very difficult to find another value that hits exactly that target (y).

.

Hash – Puzzle-friendliness

A search puzzle consists of:

- a hash function, H.
- a value, id (which we call the puzzle-ID), chosen from a high min-entropy distribution
- and a target set Y.

A solution to this puzzle is a value, x , such that $H(id \| x) \in Y$.

If a search puzzle is puzzle-friendly, this implies that there's no solving strategy for this puzzle which is much better than just trying random values of x .

Application: Hashcash - Anti spam filter

- The sender generates a cryptographic puzzle (e.g., finding a hash with a certain number of leading zeros).
- The email client computes the solution before sending the email.
- The recipient's server quickly verifies the solution.
- If valid, the email is accepted; otherwise, it's rejected as spam.

Hash – Sha256

Sha-256 is the hash function primary used in Bitcoin.

As underlying hash function sha-256 uses an hash function called **compression function.**

Input size 768 bit

Output size 256 bit

Hash – Merkle Demagger

Sha-256 is the hash function primary used in Bitcoin.

As underlying hash function sha-256 uses an hash function called **compression function.**

The compression function has a input size of 768 bits and produce 256-bit output.

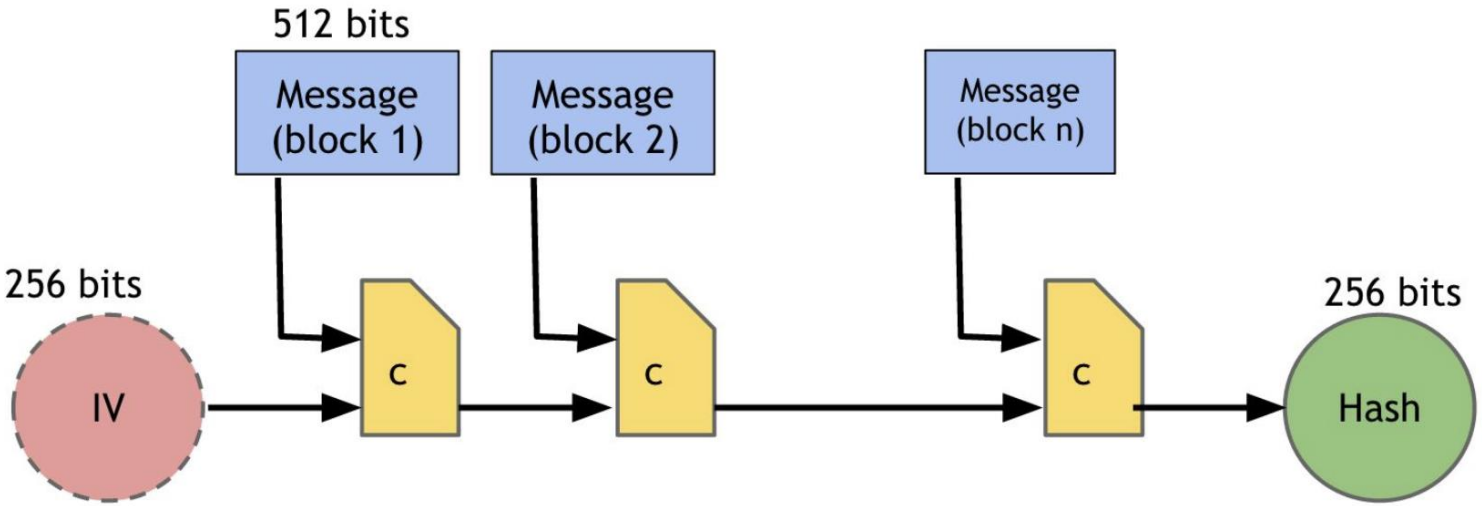
SHA-256 uses the Merkle-Damgard transform to turn a fixed-length collision-resistant compression function into a hash function that accepts arbitrary-length inputs.

Input length = m

Output length = n

An input of any length is divided in blocks of length **m-n.**

An initialization vector (IV) with length n is initialized.



Hash – Hashpointer

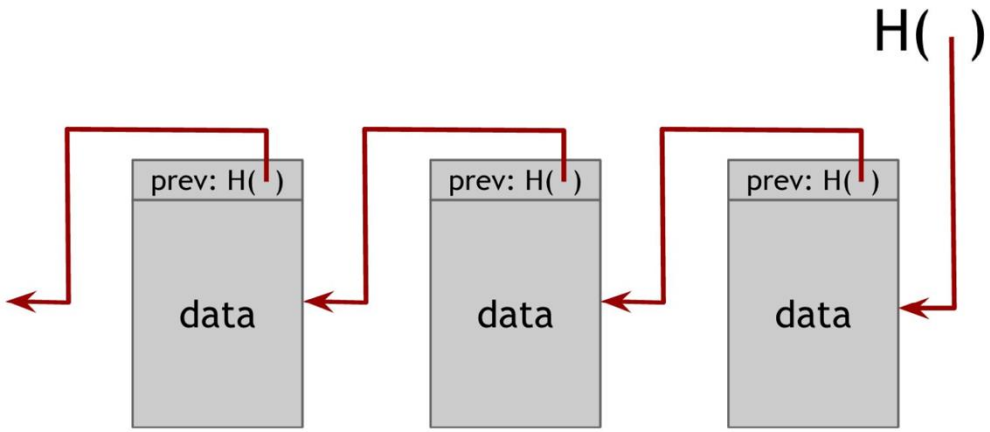
A hash pointer is a pointer to where some information is stored together with a cryptographic hash of the information.

Whereas a regular pointer gives you a way to retrieve the information, a hash pointer also gives you a way to verify that the information **hasn't changed.**

Hashpointer – Blockchain

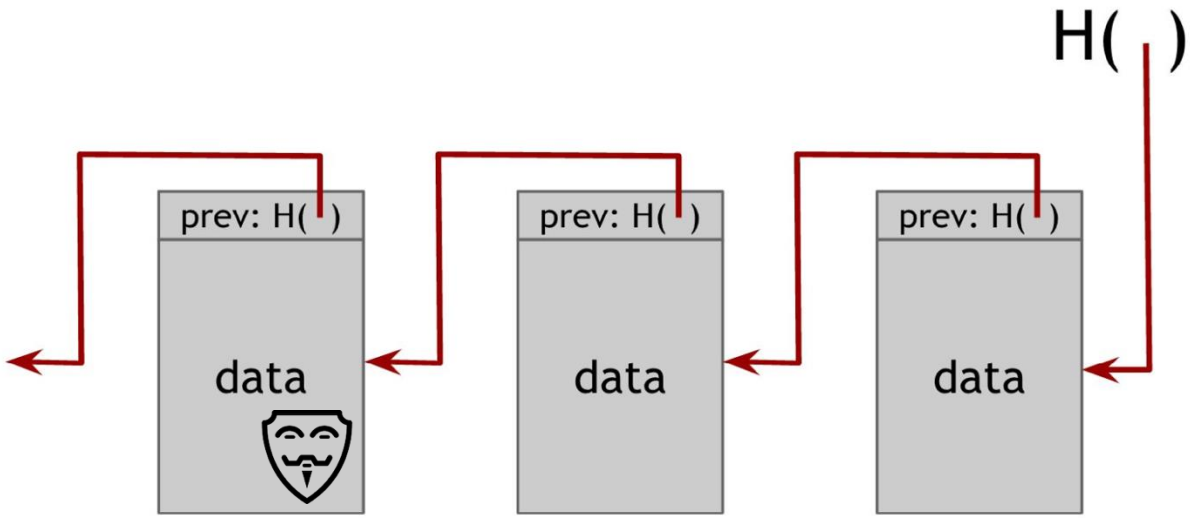
A blockchain is a linked list that is built with hash pointers instead of pointers.

A blockchain is a log data structure that stores a bunch of data, and allows us to append data onto the end of the log. But if somebody alters data that is earlier in the log, we're going to detect it.



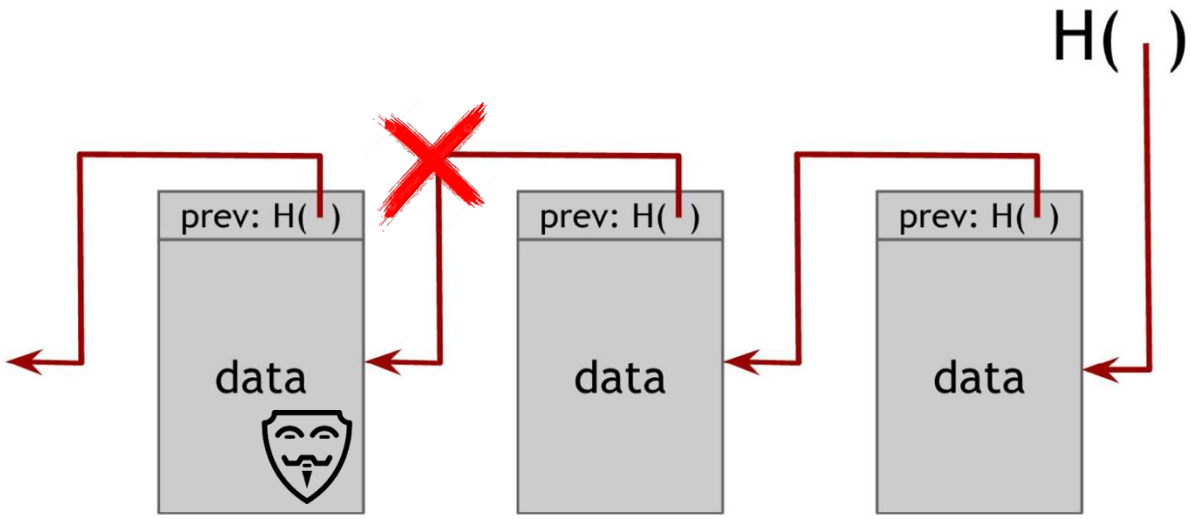
Hashpointer – Blockchain

Indeed, if someone tamper one of the blocks….



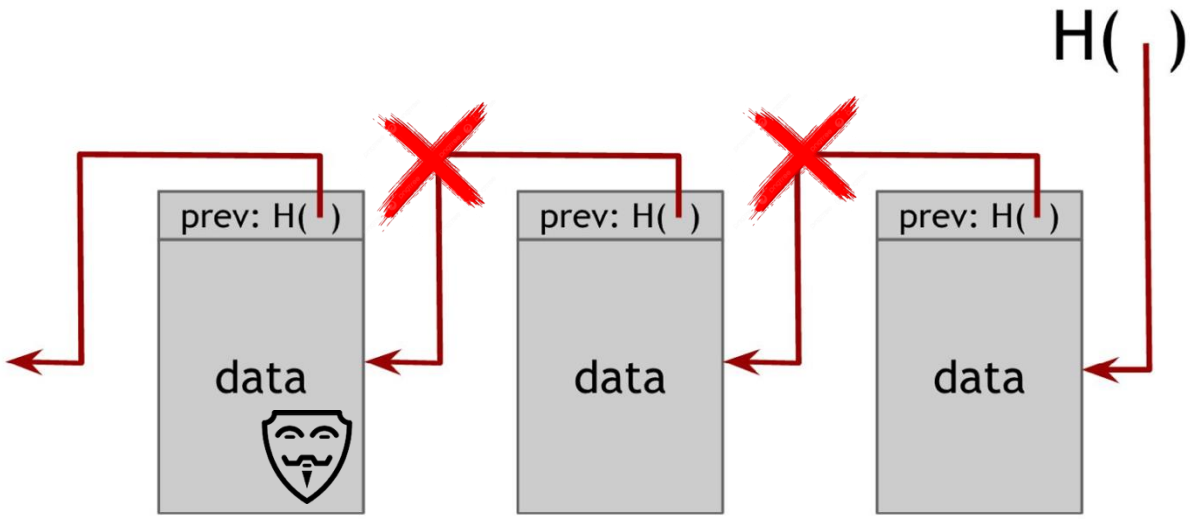
Hashpointer – Blockchain

The hash-pointer linking to that block is not more matching up…



Hashpointer – Blockchain

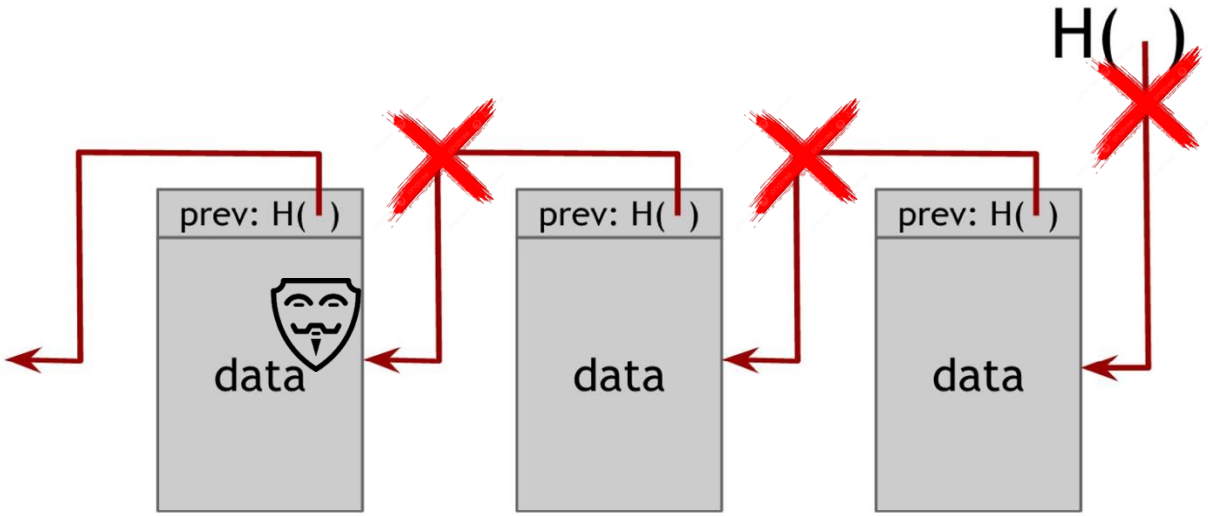
And the error propagates to all the subsequent blocks and hashpointers…



Hashpointer – Blockchain

…Until the head of the blockchain is reached.

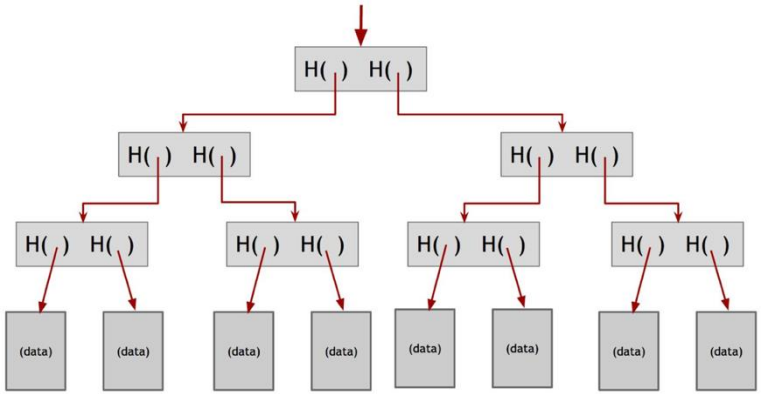
Thus, by just remembering the single hash pointer of the head of the blockchain, we've essentially remembered a tamper-evident hash of the entire list.



Hashpointer – MerkleTree

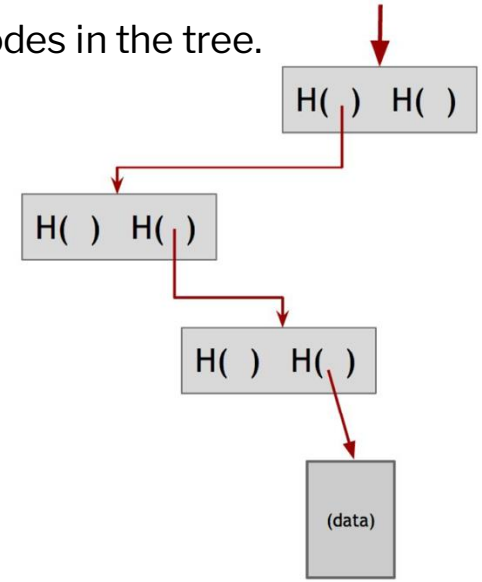
Data are in the leaves of a binary tree.

Nodes are made of pair of two hashpointers, one for each block of data



Proof of membership

If we want to prove that a data is in the Merkle Tree and there are n nodes in the tree.
We need to verify only about log(n) items.



(Binary) Merkle trees and Merkle roots in Bitcoin

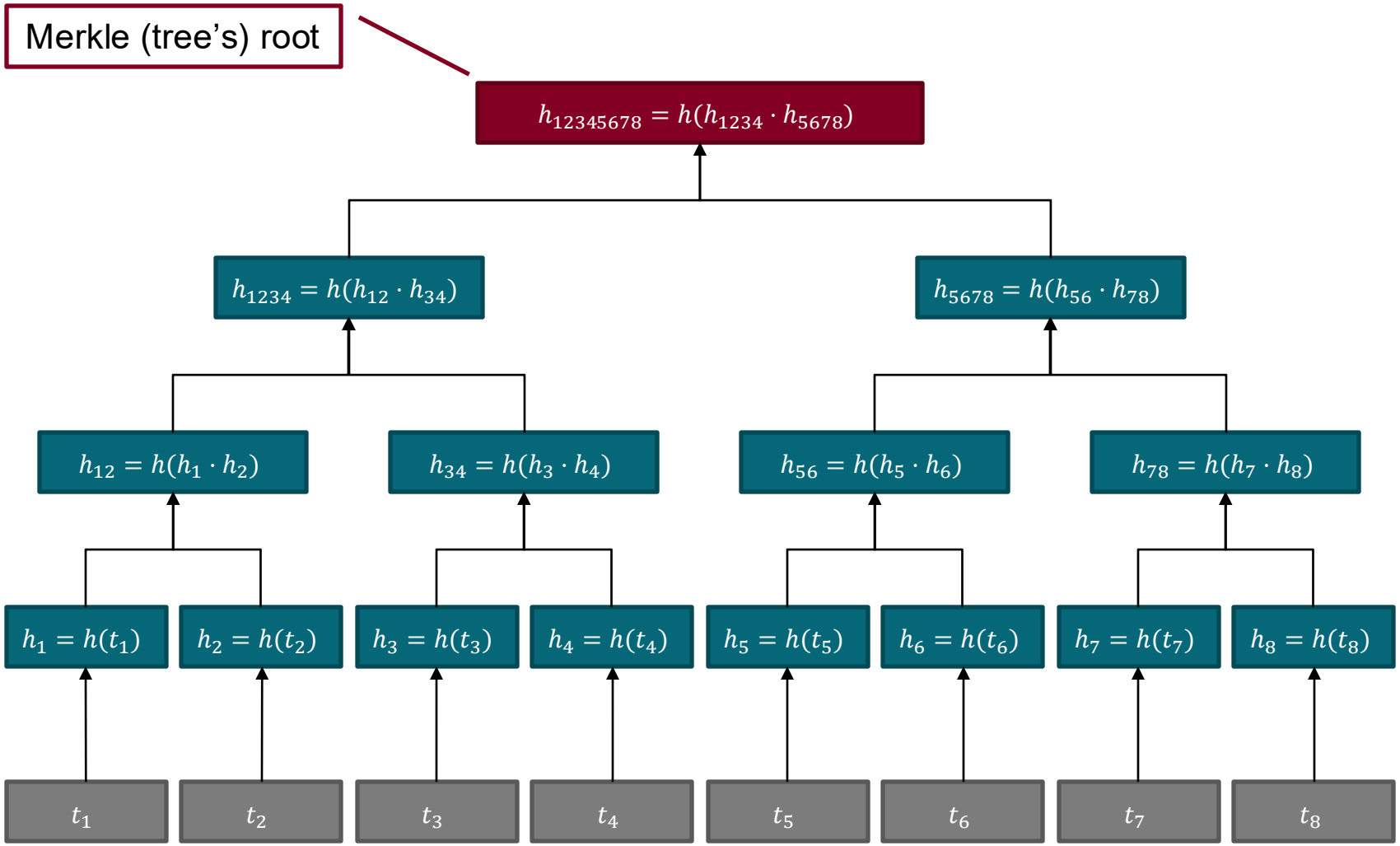


Image curtesy of Prof. Di Ciccio

Merkle proof of membership

- Suppose a light node wants to verify t_2
- A full archival node is requested to send only hashes: h_1, h_{34}, h_{5678}

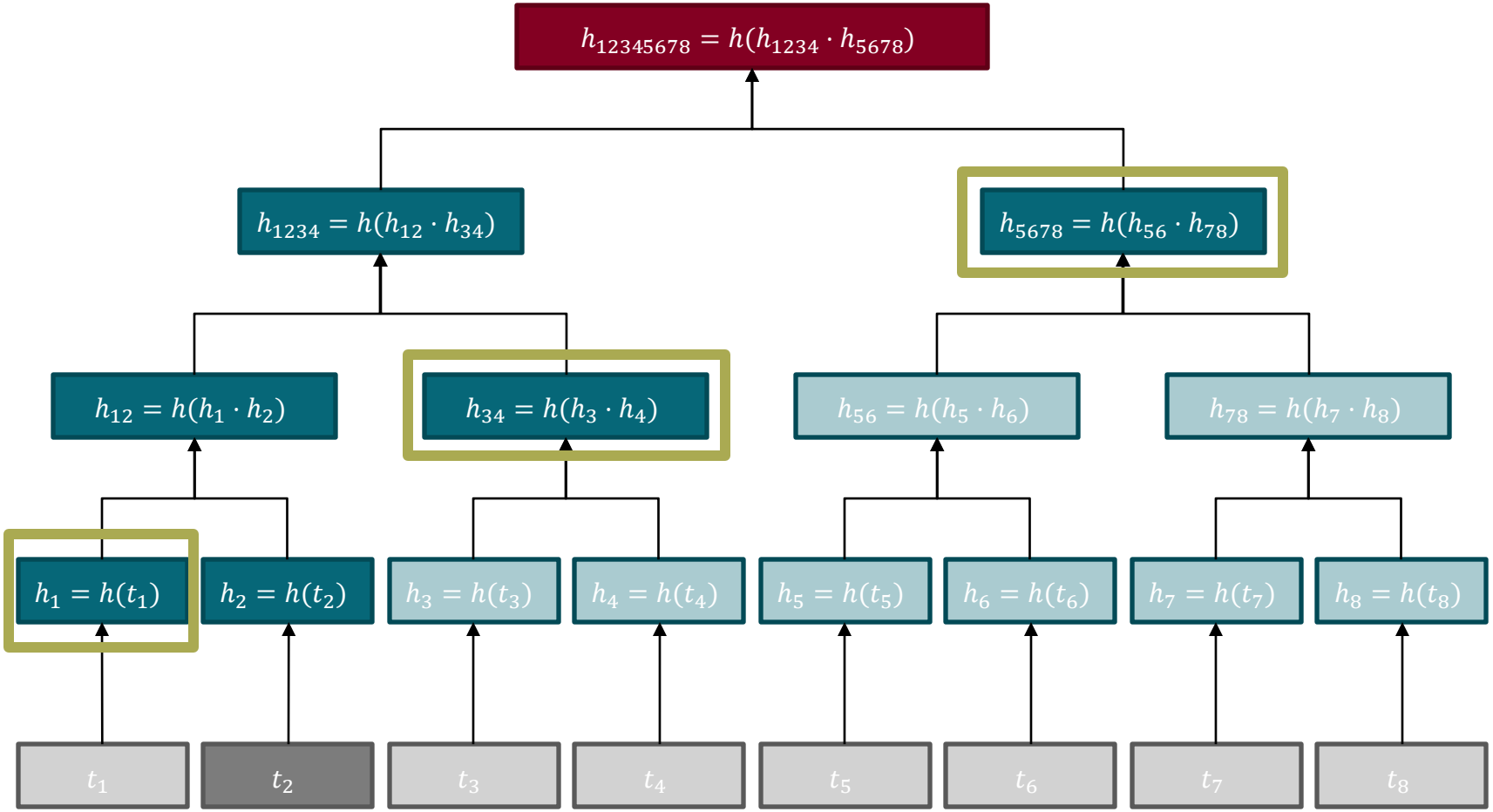


Image curtesy of Prof. Di Ciccio

Merkle proof of non membership

- A sorted merkel tree (a Merkle tree where the blocks at the bottom are ordered with some sorting function)
- Following the path made from the item before and the item after where the item we are looking for would be it is possible to show that the item does not exists in the merkletree.

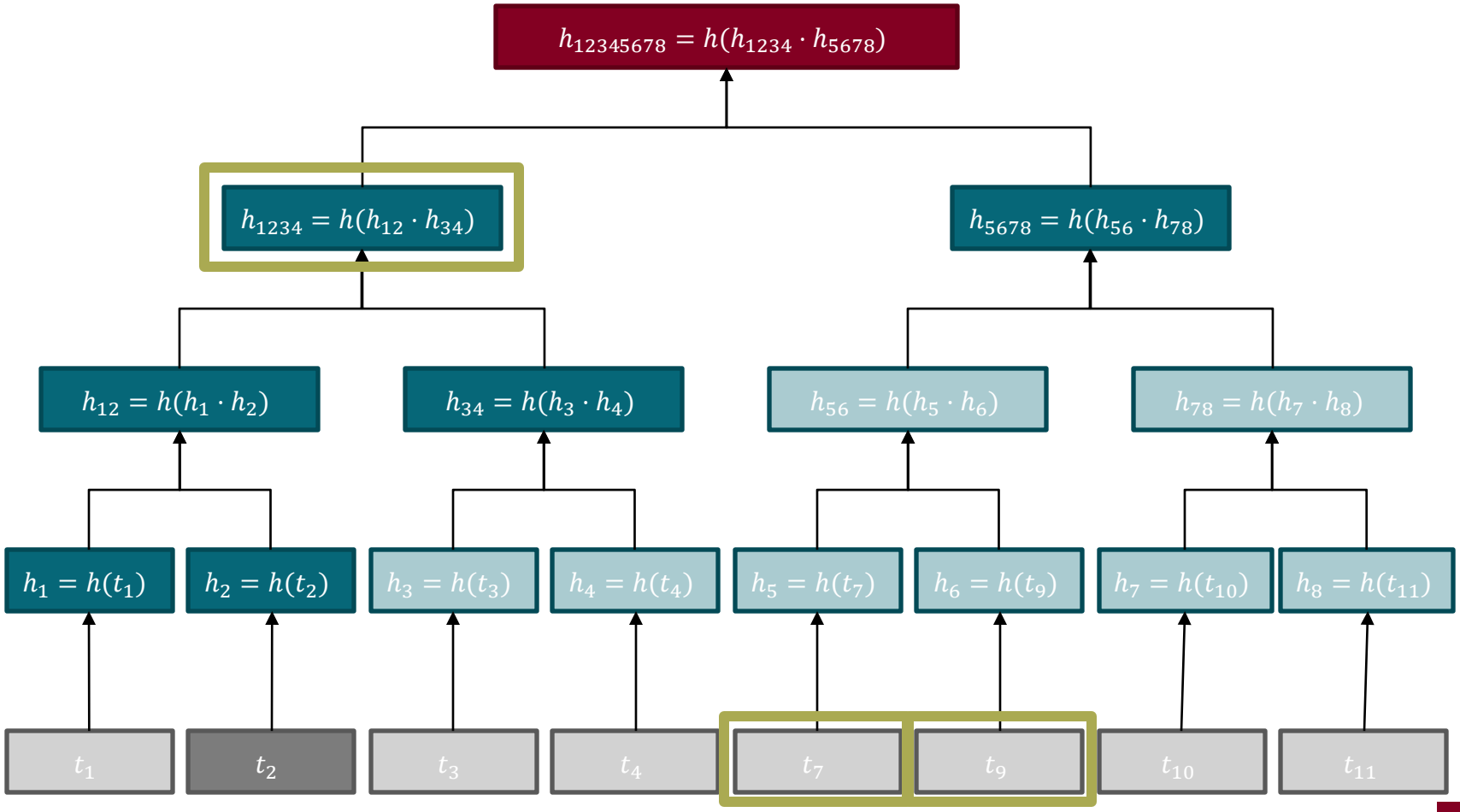


Image curtesy of Prof. Di Ciccio

What if we do not have 2^n transactions?

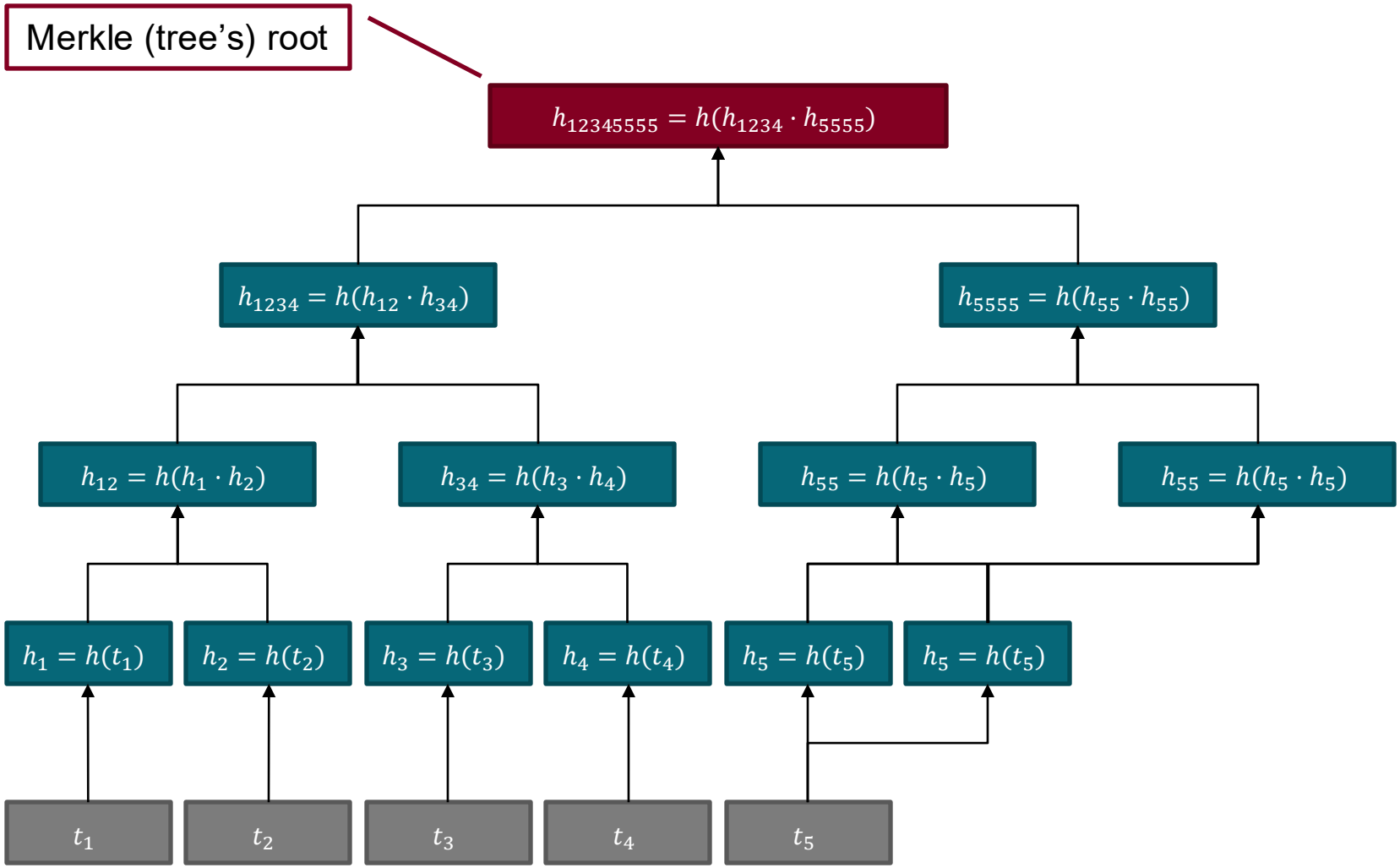


Image curtesy of Prof. Di Ciccio

Hashpointer – MerkleTree

We can store the root to be sure that the underlaying data did not change.