

Crypto Background

Blockchains and Cryptocurrencies (Spring 2026)

Course logistics

- Assignment 1a comes out today:
It will be posted on the Github page, and
Dorian (the TA) will make a Piazza
announcement
- If you want to add the course, ask at EOC

News?

January 28, 2026 1:16 PM 2 min read

N

Fidelity Launches FIDD Stablecoin To Challenge Circle's USDC, Tether's USDT

by Parshwa Turakhiya Benzinga Staff Writer

Follow



N

Hacker steals \$282 million crypto from a victim in social-engineering attack

A sophisticated social-engineering attack led to the theft of more than \$282 million in BTC and LTC, with the funds rapidly laundered through monero.

By [Oliver Knight](#) | Edited by [Jesse Hamilton](#)

Updated Jan 16, 2026, 4:59 p.m. Published Jan 16, 2026, 1:56 p.m.



Make us preferred on Google



The first private stablecoin on Aleo.

Control what you share on your terms.

MINT USDCX ON ALEO HERE ➔

Live

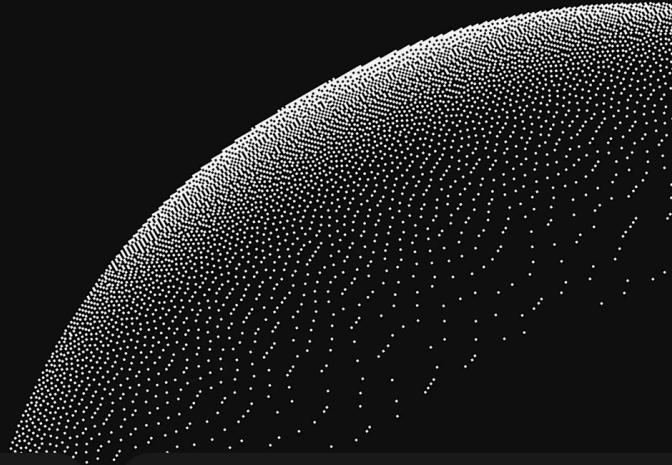
AVAILABLE ON MAINNET NOW

First

PRIVATE AND PROGRAMMABLE STABLECOIN

Trusted

FAST. SECURE.



Review: cash problems

- **Double spending**

- To capture double spending you need an online (networked) party that must be trusted

- **Authentication / Authentication**

- How do I prove that I am the owner of currency & thus authorized to transact with it?

- **Origin/Issuance**

- How is new currency created?

This lecture

We need cryptography to build cryptocurrency

Crypto background

hash functions

random oracle model

digital signatures

... and applications

Cryptographic Hash Functions

Question:

- Let's imagine that two computers each have a “ledger” of transactions
- How do they quickly verify that their ledgers are the same?

Hash function

- takes a string of arbitrary length as input
- fixed-size output (i.e., hash function
“compresses” the input)
- efficiently computable

Security properties:

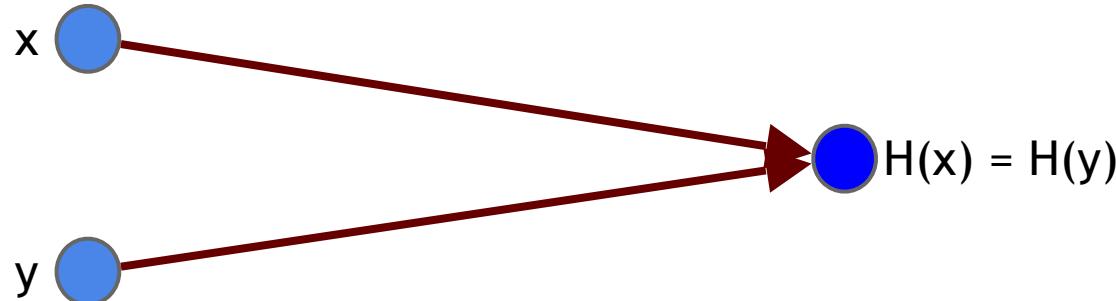
- Collision resistance
- Preimage resistance (one-way)

Property 1: Collision resistance

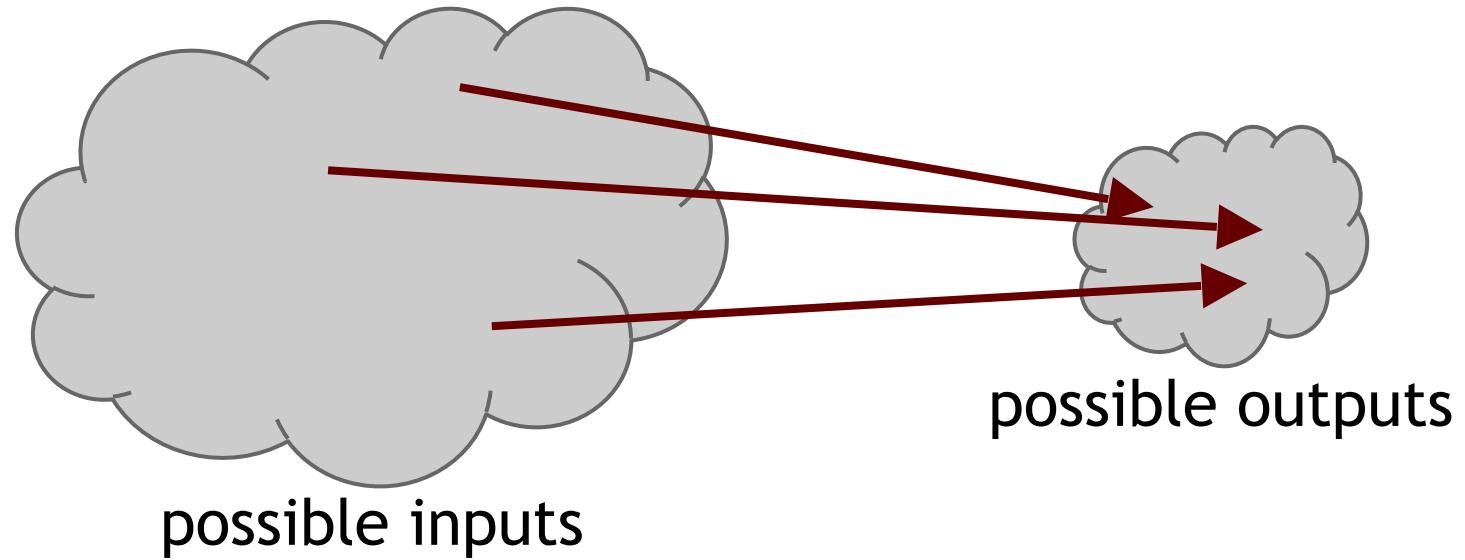
What's a collision?

Property 1: Collision resistance

Do collisions exist in common hash functions?



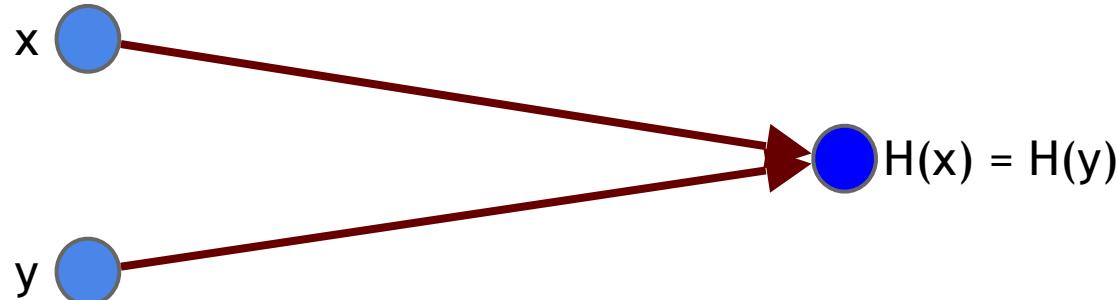
Collisions do exist ...



... but can a real-world adversary find them?

Property 1: Collision resistance

No efficient adversary can find x and y such that $x \neq y$ and $H(x)=H(y)$



How to find a collision (for 256 bit output)

- try 2^{130} randomly chosen inputs
- 99.8% chance that two of them will collide

This works no matter what H is, but it takes too long to matter

- If a computer calculates 10,000 hashes/sec, it would take 10^{27} years to compute 2^{128} hashes

How to find a collision (for 256 bit output)

- try 2^{130} randomly chosen inputs
- 99.8% chance that two of them will collide

This works, but it takes too long

Q: How many hashes/sec does the Bitcoin network compute?

- If a computer calculates 10,000 hashes/sec, it would take 10^{27} years to compute 2^{128} hashes

Is there a faster way to find collisions?

- For some possible H 's, yes.
- For others (like SHA-256), we don't know of one.

Provably secure collision-resistant hash functions can be constructed based on “hard” number-theoretic problems.

Defining Collision Resistance

- Real-world adversaries
 - In practice, everyone has bounded resources
 - Therefore, reasonable to model a real-world adversary as such an entity
 - However, we do not make any assumptions about the adversarial strategy. He can use its (bounded) resources in any possible way

Cryptographic adversary: A probabilistic polynomial-time (PPT) algorithm

Defining Collision Resistance...

- Collision Resistance (informal): A hash function H is collision-resistant if for all PPT adversaries A ,

$\Pr[A \text{ outputs } x, y \text{ s.t. } x \neq y \text{ and } H(x) = H(y)]$
= “very small”

Defining Collision Resistance...

- Collision Resistance (informal): A hash function H is collision-resistant if for all PPT adversaries A ,
$$\Pr[A \text{ outputs } x, y \text{ s.t. } x \neq y \text{ and } H(x) = H(y)] \\ = \text{“very small”}$$
- “Very small” captured via a function that tends to 0.
Formal definition: Modern Cryptography

Application: Hash as message digest

If we know $H(x) = H(y)$, and H is collision resistant
it's safe to assume that $x = y$.

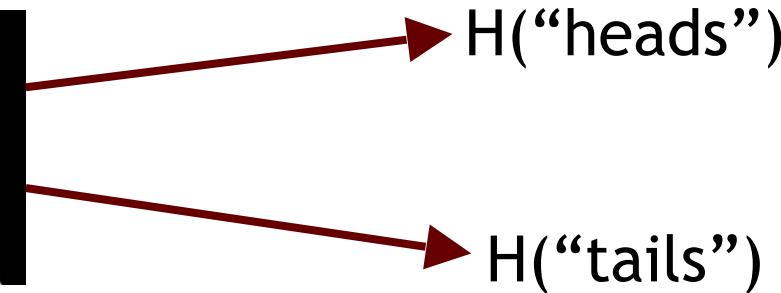
To recognize a file that we saw before,
just remember its hash.

Useful because the hash is small.

Property 2: Pre-image Resistance

Intuition: Given $H(x)$, no efficient adversary can find x , except with very small probability

Problem: What if input space of x is very small, or some inputs are much more likely than others?



easy to find x !

Property 2: Preimage

This definition is useless in this setting. How can we specify a meaningful version of the definition?

Intuition: Given $H(x)$,
except with very small probability
Efficient adversary can find x ,

Problem: What if input space of x is very small, or some inputs are much more likely than others?



$H(\text{"heads"})$

$H(\text{"tails"})$

easy to find x !

Defining Preimage Resistance

- Preimage Resistance: A hash function H is preimage-resistant if for all PPT adversaries A ,

$$\Pr[x \leftarrow \{0,1\}^k, A(H(x)) \text{ outputs } x' \text{ s.t. } H(x')=H(x)] = \text{small}$$


x is drawn from uniform distribution over $\{0,1\}^k$ for some sufficiently large k

Preimage Resistance (contd.)

- If x is drawn from the uniform distribution, then inverting $H(x)$ is hard
- But what if x is drawn from low-entropy distribution?
- Can append a random string r to x and then compute $H(r \mid x)$ to prevent enumeration attacks

Theorem: Collision resistance implies preimage resistance if the hash function is sufficiently compressing

Application: Commitment

Want to “seal a value in an envelope”, and
“open the envelope” later.

Commit to a value, reveal it later.

Commitment Schemes

$(com, key) := \text{commit}(msg)$

$\text{match} := \text{verify}(com, key, msg)$

To seal msg in envelope:

$(com, key) := \text{commit}(msg)$ -- then publish com

To open envelope:

publish key, msg

anyone can use $\text{verify}()$ to check validity

Commitment Schemes

$(com, key) \leftarrow \text{commit}(msg)$

$match \leftarrow \text{verify}(com, key, msg)$

Security properties:

- Hiding: Given com , no PPT adversary can find* msg
- Binding: No PPT adversary can find* $msg \neq msg'$ such that $\text{verify}(\text{commit}(msg), msg') == \text{true}$

* Except with very small probability

Commitment Schemes

$\text{commit}(\text{msg}) \rightarrow (\text{H}(\text{key} \mid \text{msg}), \text{key})$

where key is a random 256-bit value

$\text{verify}(\text{com}, \text{key}, \text{msg}) \rightarrow (\text{H}(\text{key} \mid \text{msg}) == \text{com})$

Security properties:

- Hiding: If H is a *random oracle*, given $\text{H}(\text{key} \mid \text{msg})$, hard to find msg .
- Binding: Collision-resistance \rightarrow Hard to find $\text{msg} \neq \text{msg}'$ such that $\text{H}(\text{key} \mid \text{msg}) == \text{H}(\text{key} \mid \text{msg}')$

Random Oracle (RO)

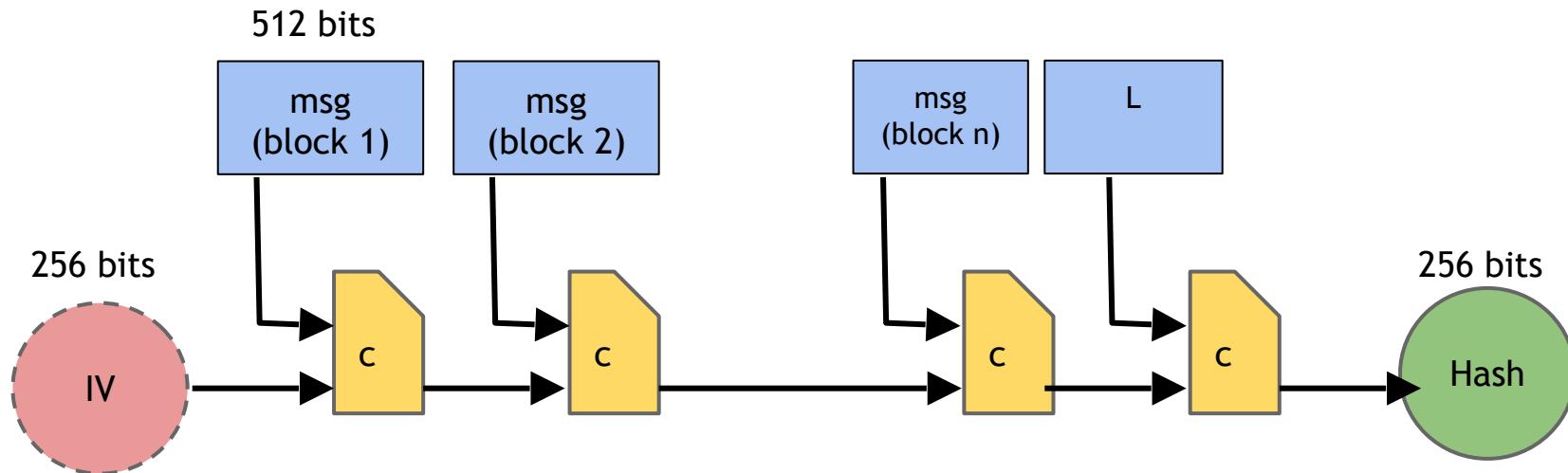
- Imagine an elf in a box with an infinite writing scroll
- Upon receiving an input x , the elf checks the scroll if there is an entry y corresponding to x . If yes, it returns y .
- Otherwise, elf chooses a random value y (from the output space) and returns it. It adds an entry (x,y) to the scroll.

Random Oracle (RO)

- In practice-oriented provable security, hash functions are often modeled as a random oracle
- Each party (including adversary) is given black-box access to the random oracle. They can query the random oracle any polynomial number of times
- By definition, the answers of random oracle answers are unpredictable
- Random oracle captures many security properties such as one-wayness, collision-resistance .

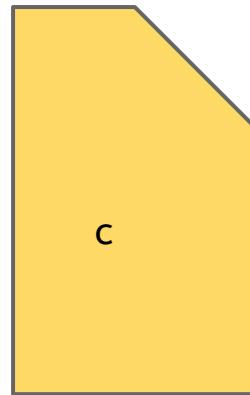
SHA-256 hash function

Suppose msg is of length L s.t. L is a multiple of 512 (pad with 0s otherwise)



Theorem [Merkle-Damgård]: If c is collision-resistant, then SHA-256 is collision-resistant.

SHA-256 hash function



Q: What the heck is inside of c?

Theorem [Merkle-Damgard]: If c is collision-resistant, then SHA-256 is collision-resistant.

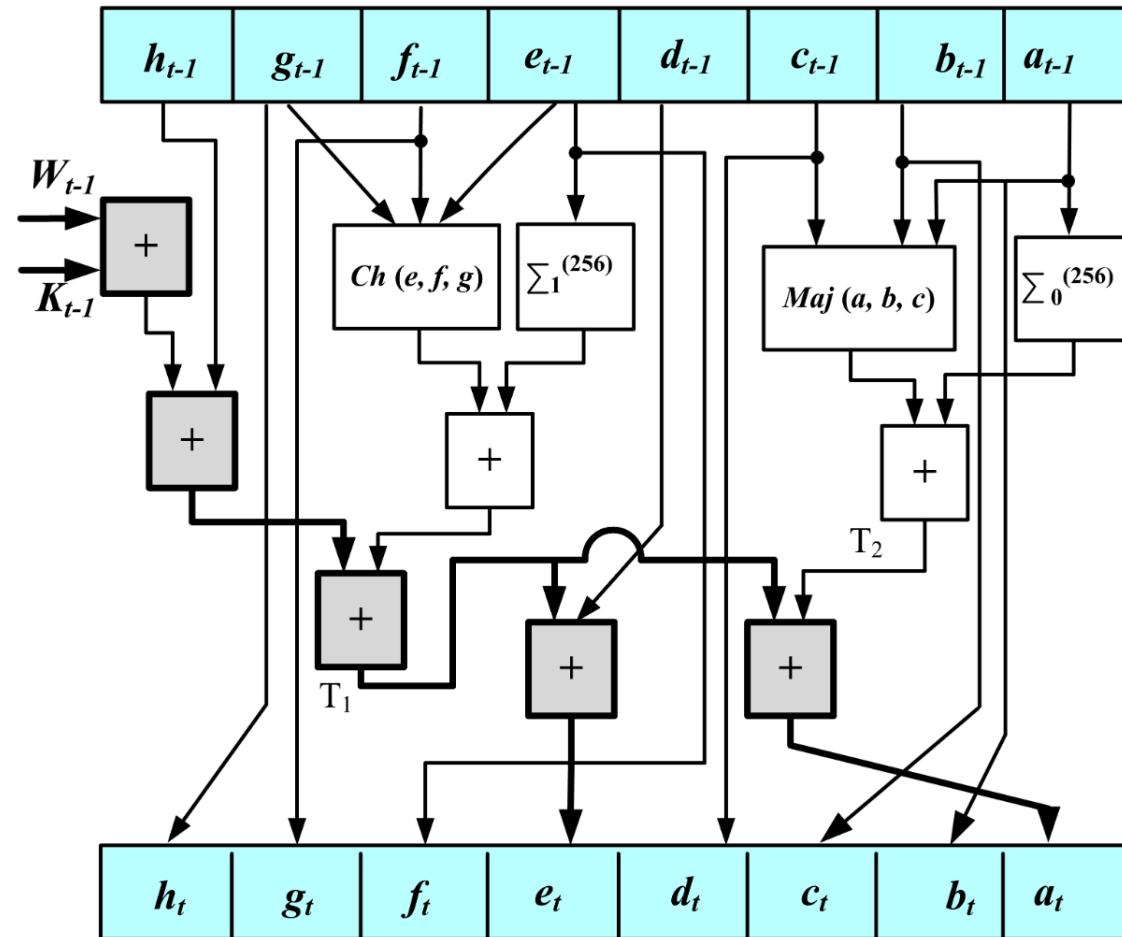


Fig. 3. SHA-256 hash function. Base transformation round.

Reminder: hash functions

- Take as input an arbitrary-length string
- Output a (shorter) fixed-size string

Cryptographic hash function security:

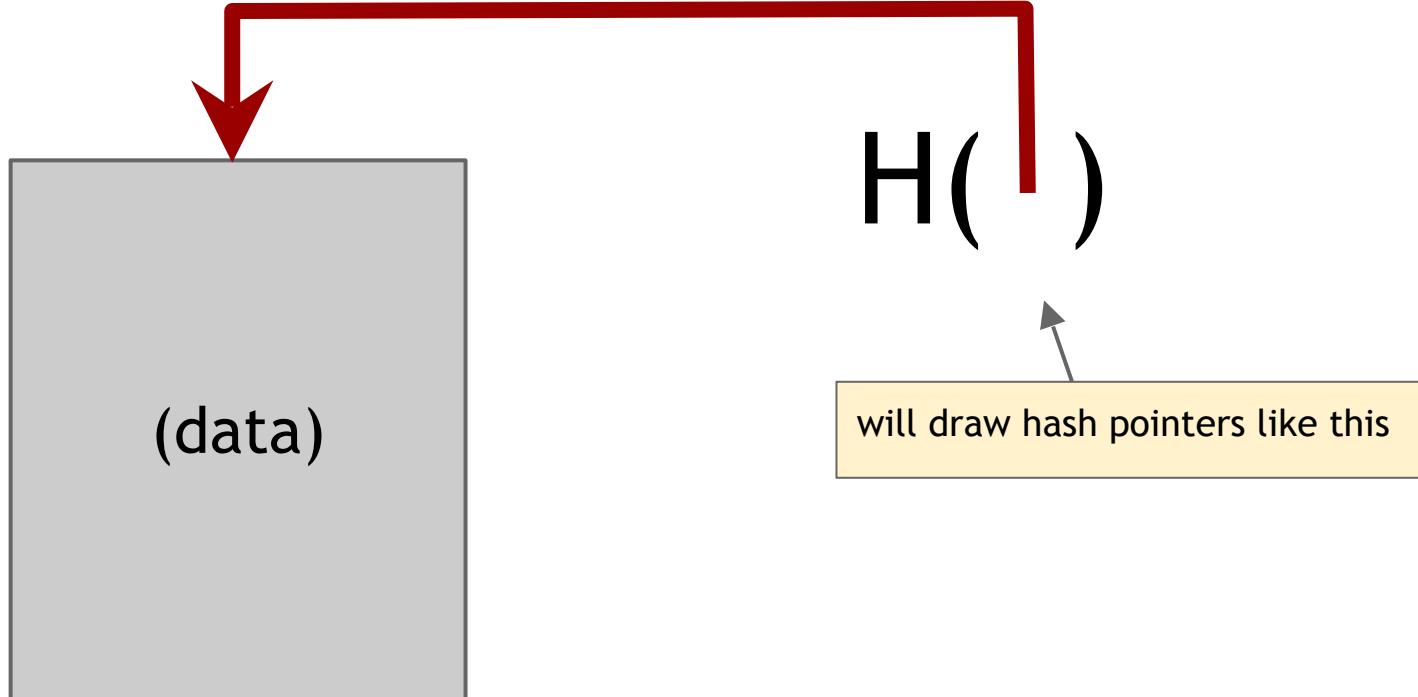
- Collision-resistant
- Pre-image resistant
- “Random oracle”-like (for some cases)

Hash pointer

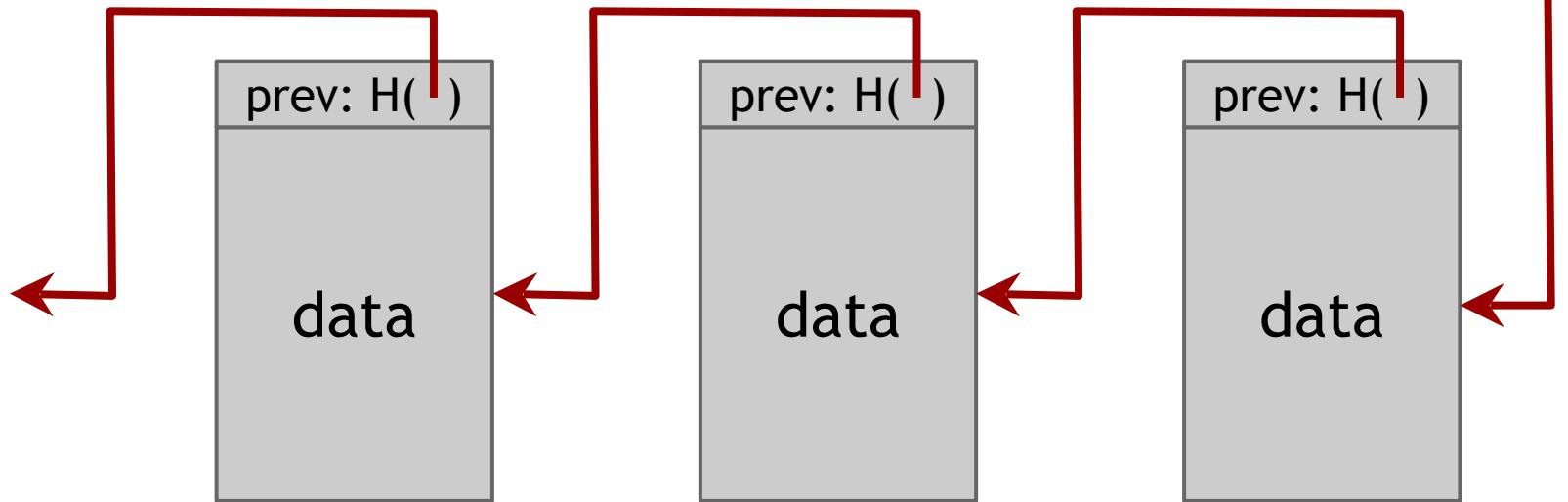
- pointer to where some info is stored, *and*
- cryptographic hash of the info

If we have a hash pointer, we can

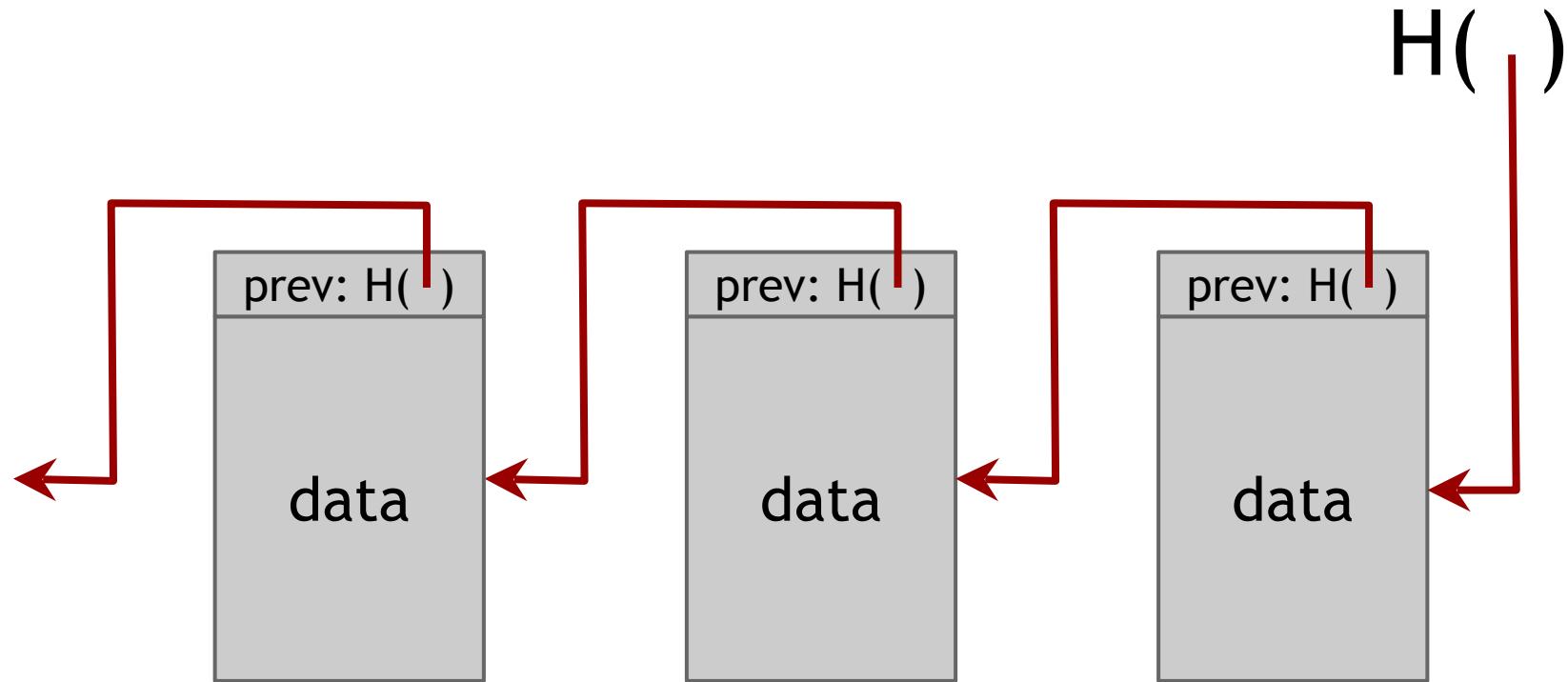
- ask to get the info back, *and*
- verify that it hasn't changed



Building data structures with hash pointers

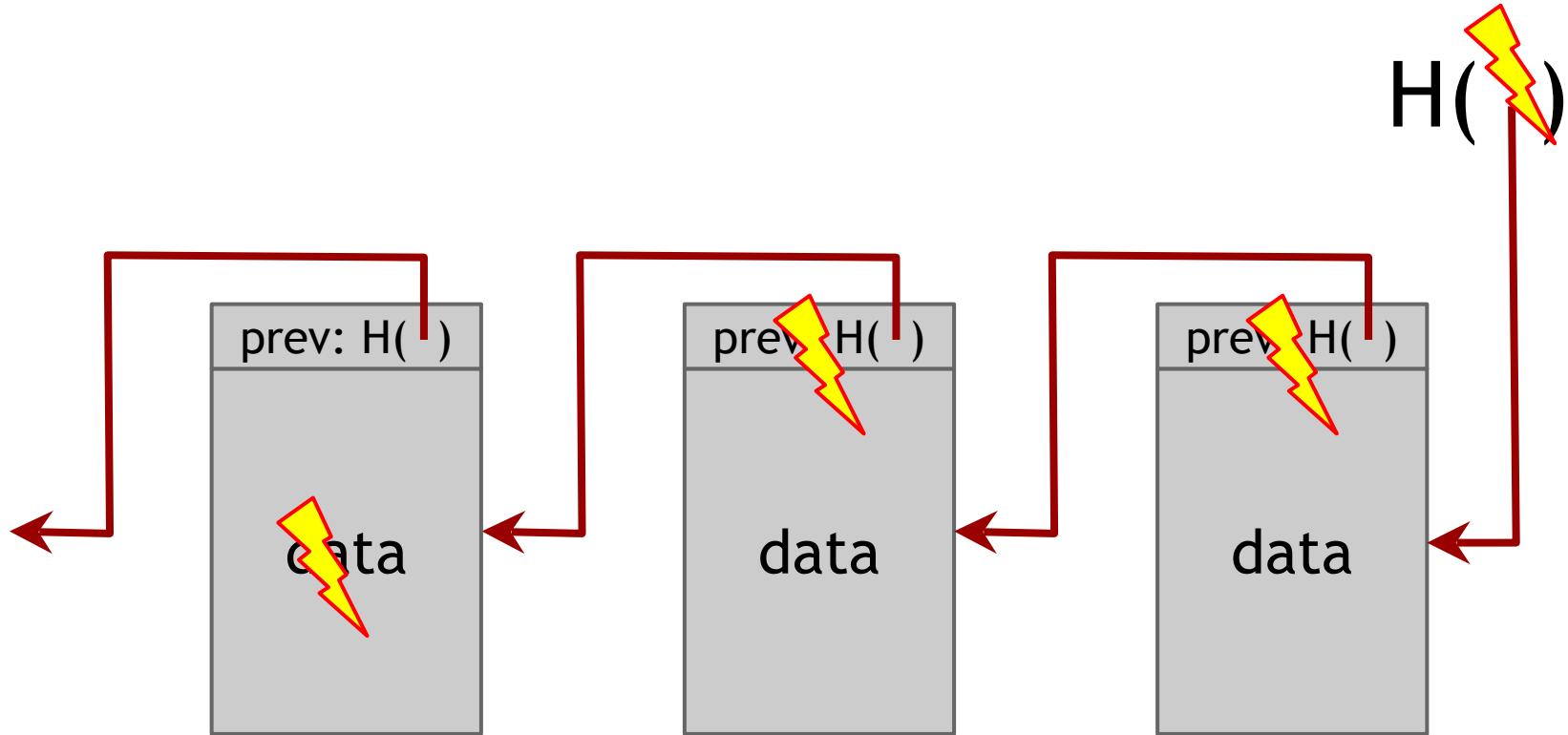
$H()$ 

Linked list with hash pointers = “Blockchain”



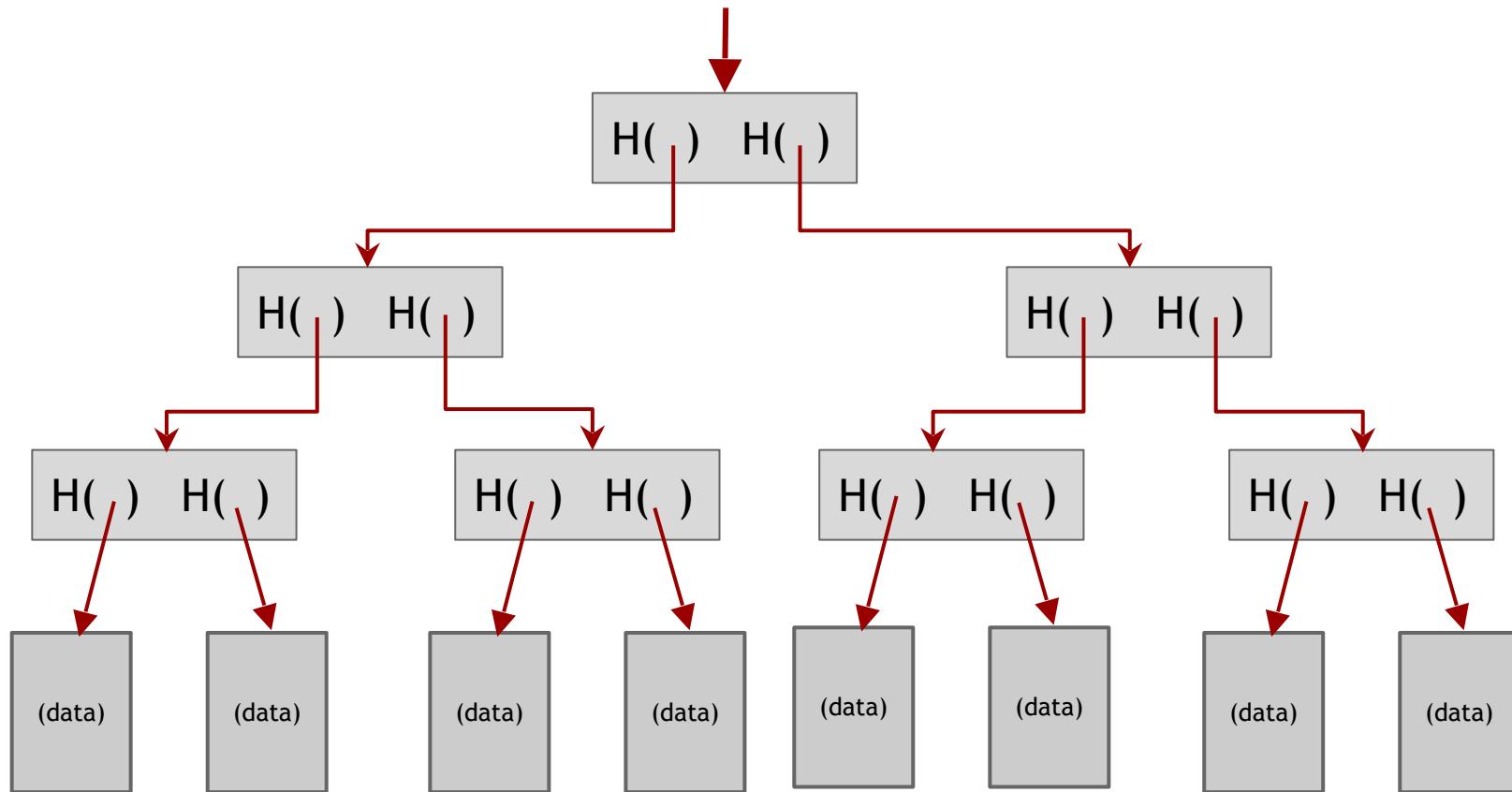
use case: tamper-evident log

detecting tampering

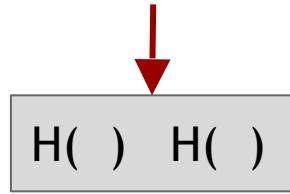


use case: tamper-evident log

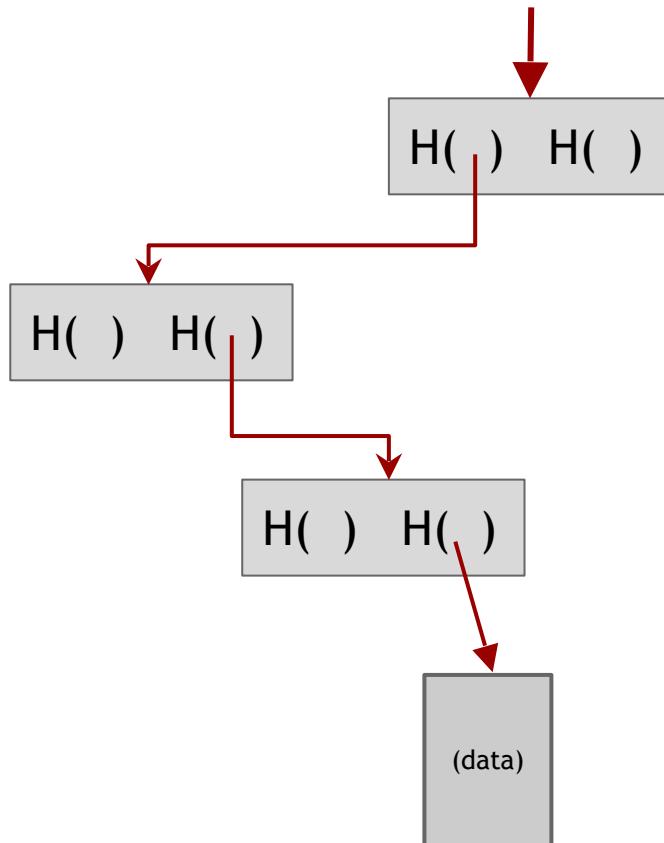
binary tree with hash pointers = “Merkle tree”



proving membership in a Merkle tree



proving membership in a Merkle tree



show $O(\log n)$ items