Blockchain Principles and Applications

Amir Mahdi Sadeghzadeh, PhD

Data and Network Security Lab (DNSL)
Trustworthy and Secure AI Lab (TSAIL)

Most slides have been adapted from ECE/COS 470, Pramod Viswanath, Princeton 2024 and CS251, Dan Boneh, Stanford 2023.

Basic Cryptographic Primitives

1. Cryptographic Hash Functions

2. Hash Accumulators
   Merkle trees

Centralized Blockchain

3. Digital Signatures

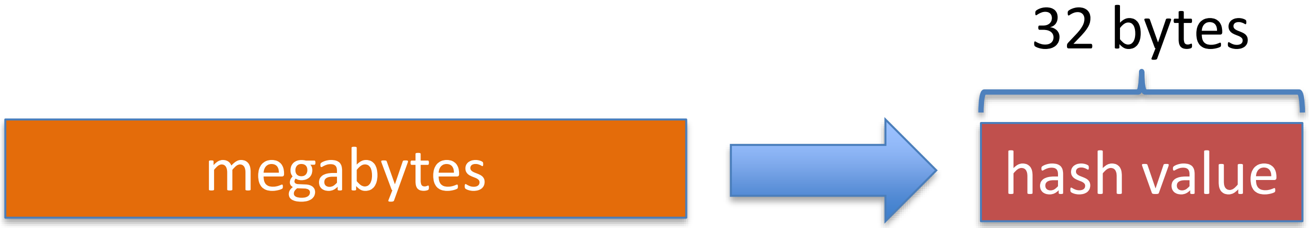
Decentralized Blockchain

Cryptography Background

(1) cryptographic hash functions

An efficiently computable function $H\colon\ M \rightarrow T$

where $|M| \gg |T|$



$T = \{0,1\}^{256}$

Collision resistance

Def: a **collision** for $H: M \to T$ is pair $x \neq y \in M$ s.t. $\boxed{H(x) = H(y)}$

$|M| \gg |T|$ implies that many collisions exist

Def: a function $H: M \to T$ is **collision resistant** if it is "hard" to find even a single collision for H (we say H is a CRF)

Example: **SHA256**: $\{x : \text{len}(x) < 2^{64} \text{ bytes}\} \to \{0,1\}^{256}$

(output is 32 bytes)

Hash Functions

Defining Properties:

1. Arbitrary sized inputs

2. Fixed size deterministic output

3. Efficiently computable

4. Minimize collisions

Canonical application:

Hash Tables

Store and retrieve data records

Application: committing to data on a blockchain

Alice has a large file m. She posts $h = H(m)$ (32 bytes)

Bob reads h. Later he learns m' s.t. $H(m') = h$

H is a CRF $\Rightarrow$ Bob is convinced that $m' = m$
(otherwise, m and m' are a collision for H)

We say that $h = H(m)$ is a **binding commitment** to m

(note: not hiding, h may leak information about m)

Committing to a list (of transactions)

Alice has $S = (m_1, m_2, \ldots, m_n)$

32 bytes

Goal:

- Alice posts a <u>short</u> binding commitment to S, $h = \text{commit}(S)$
- Bob reads h. Given $(m_i, \text{ proof } \pi_i)$ can check that $S[i] = m_i$

  Bob runs $\text{verify}(h, i, m_i, \pi_i) \rightarrow$ accept/reject

security: adv. cannot find (S, i, m, π) s.t. $m \neq S[i]$ and

$\text{verify}(h, i, m, \pi) = \text{accept}$ where $h = \text{commit}(S)$

Cryptographic Hash Function

Extra Property:

Specialized one way function

Canonical application:

Puzzle generation
mining process

Hash(nonce, block-hash) < Threshold

Another application: proof of work

Goal: computational problem that

- takes time $\Omega(D)$ to solve, but
- solution takes time O(1) to verify

(D is called the **difficulty**)

How? $H: X \times Y \rightarrow \{0,1,2, \dots, 2^n - 1\}$ e.g. $n = 256$

- puzzle: input $x \in X$, output $y \in Y$ s.t. $H(x, y) < 2^n / D$
- verify(x, y): accept if $\boxed{H(x, y) < 2^n / D}$

Another application: proof of work

<u>**Thm**</u>: if H is a "random function" then the best algorithm requires D evaluations of H in expectation.

Note: this is a parallel algorithm

$\Rightarrow$ the more machines I have, the faster I solve the puzzle.

Proof of work is used in some consensus protocols (e.g., Bitcoin)

Bitcoin uses $H(x, y) = \mathrm{SHA256}(\mathrm{SHA256}(x.y))$

Hash Pointer

Hash of the information acts as pointer to location of information

Regular pointer: retrieve information

Hash pointer: retrieve information and verify the information has not changed

Regular pointers can be used to build data structures: linked lists, binary trees.

Hash pointers can also be used to build related data structures. Crucially useful for blockchains. In fact, **blockchain itself is a hash pointer-based data structure.**

Blockchain: a linked list via hash pointers

Block: Header + Data

Header: Pointer to previous block
= hash of the previous block

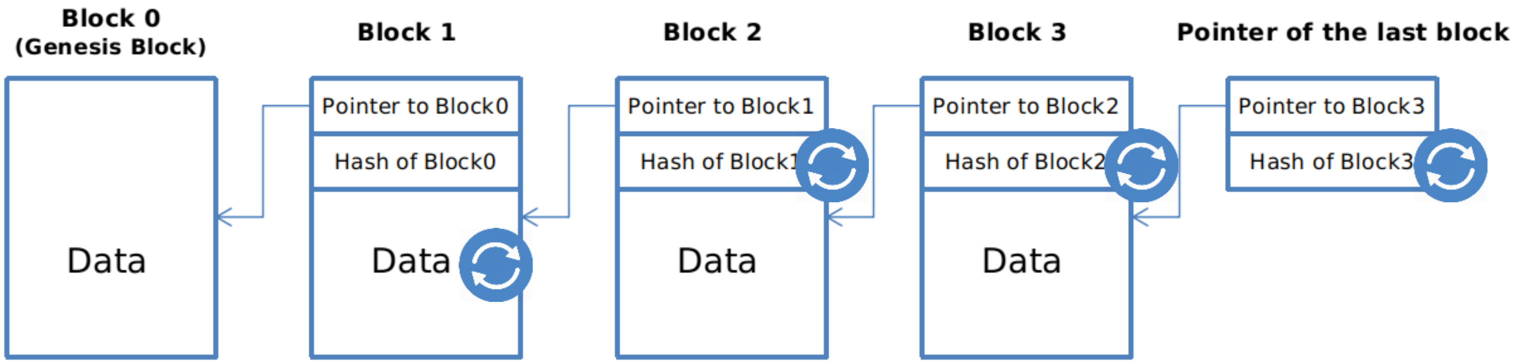
Data: information specific to the block

Application: tamper evident information log

Head of the chain being known is enough to find tamper evidence in any internal block

Hence the phrase: **block chain**
Or simply: **blockchain**

Blockchain: a linked list via hash pointers



Allows the creation of a tamper-evident information log

How about searching for specific data elements?

Merkle Tree

Binary tree of hash pointers

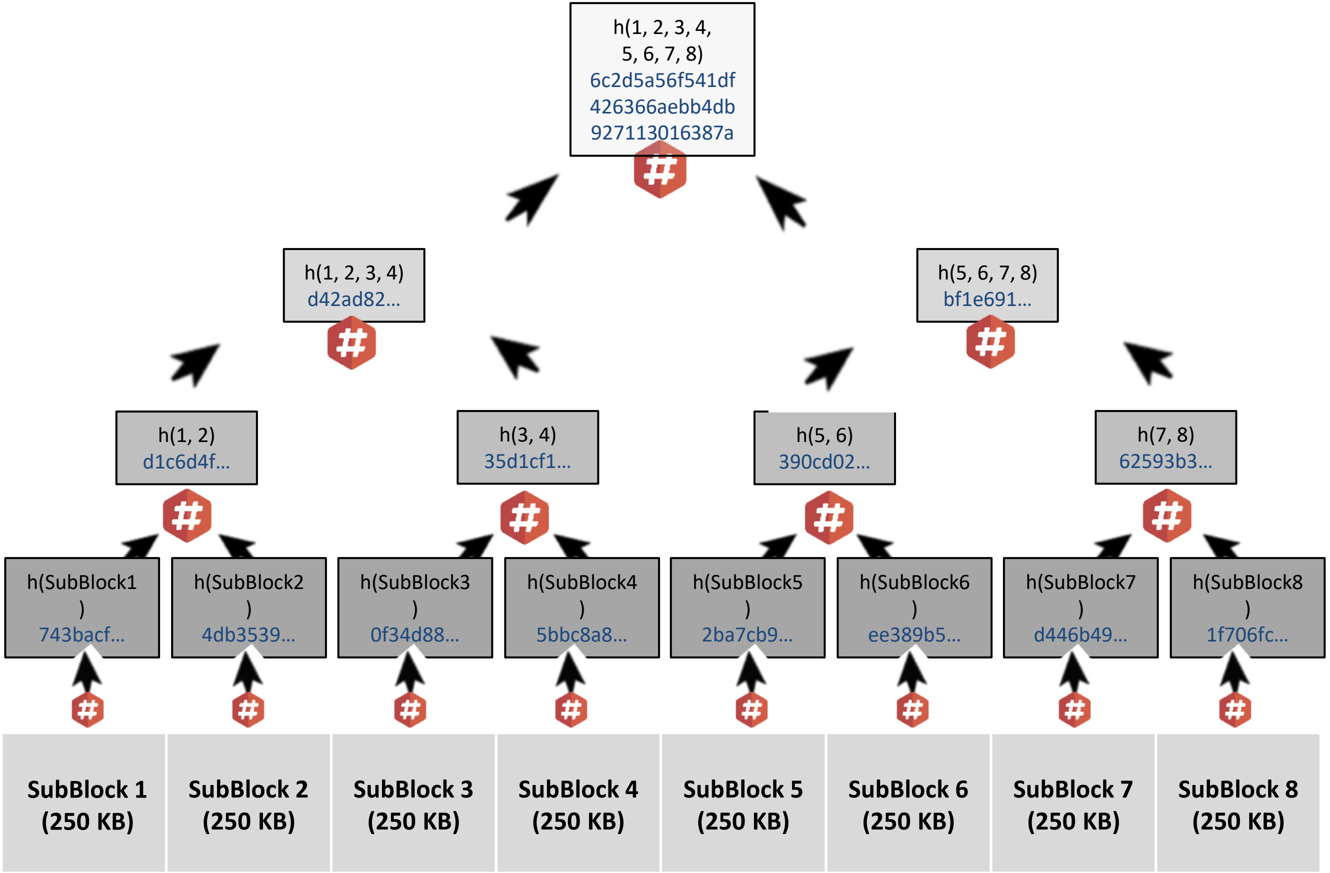
Retain only the root of the tree

Tamper of any data in the bottom of the tree is evident

Proof of Membership

Proof of Non-membership

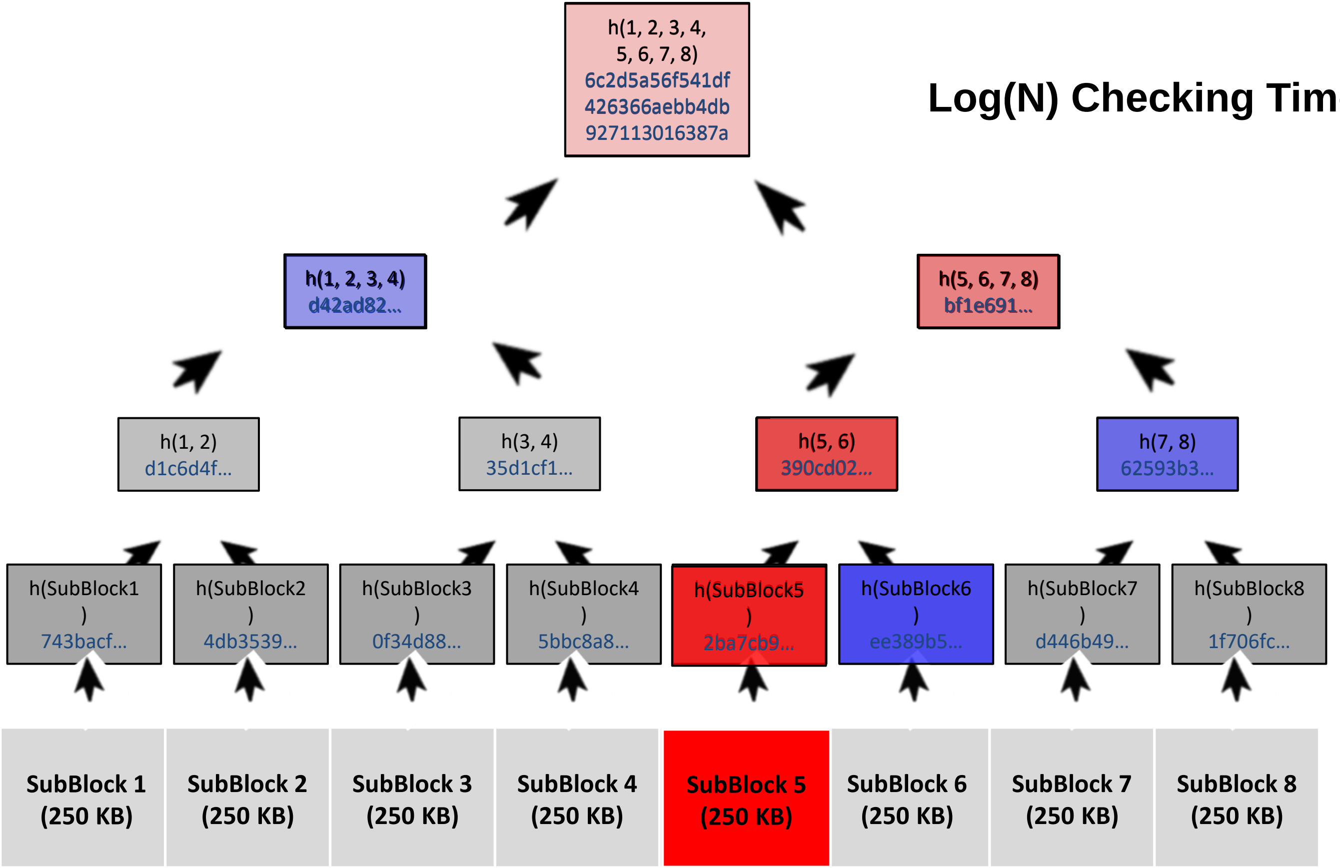
Merkle Tree



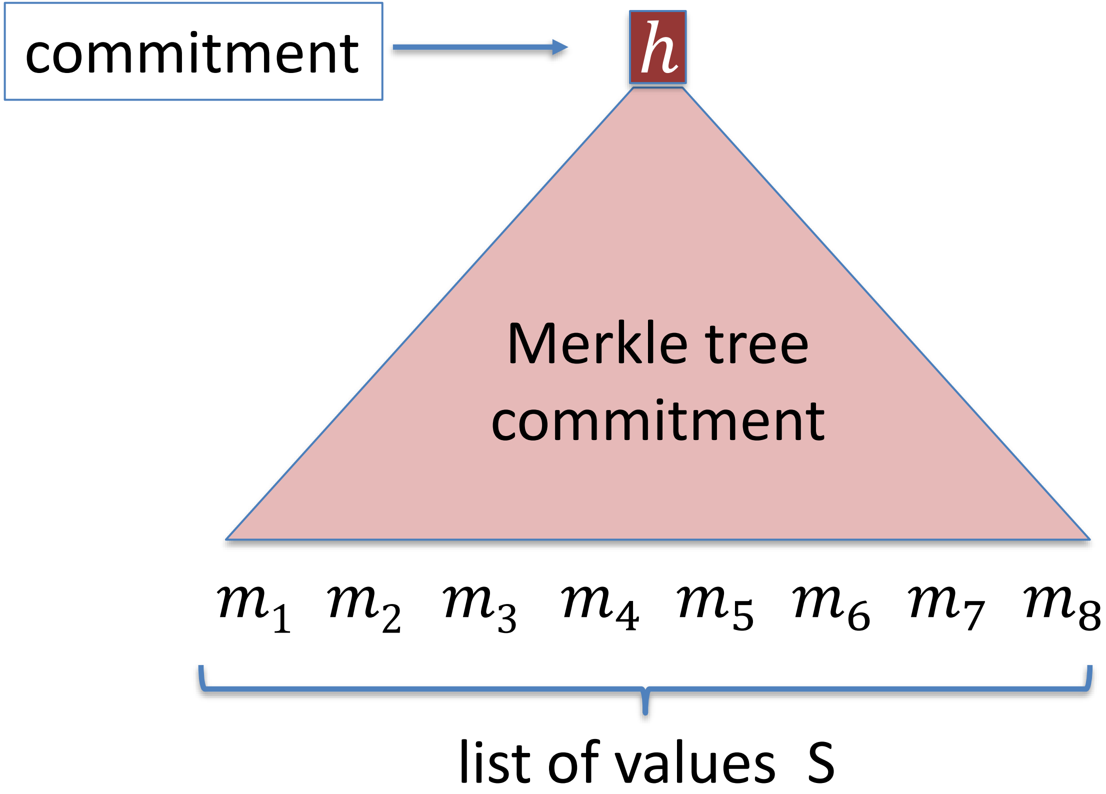
h(1, 2, 3, 4,
5, 6, 7, 8)
6c2d5a56f541df
426366aebb4db
927113016387a
h(1, 2, 3, 4)
d42ad82...
h(5, 6, 7, 8)
bf1e691...
h(1, 2)
d1c6d4f...
h(3, 4)
35d1cf1...
h(5, 6)
390cd02...
h(7, 8)
62593b3...
h(SubBlock1
)
743bacf...
h(SubBlock2
)
4db3539...
h(SubBlock3
)
0f34d88...
h(SubBlock4
)
5bbc8a8...
h(SubBlock5
)
2ba7cb9...
h(SubBlock6
)
ee389b5...
h(SubBlock7
)
d446b49...
h(SubBlock8
)
1f706fc...
SubBlock 1
(250 KB)
SubBlock 2
(250 KB)
SubBlock 3
(250 KB)
SubBlock 4
(250 KB)
SubBlock 5
(250 KB)
SubBlock 6
(250 KB)
SubBlock 7
(250 KB)
SubBlock 8
(250 KB)

Proof of Membership

Log(N) Checking Time

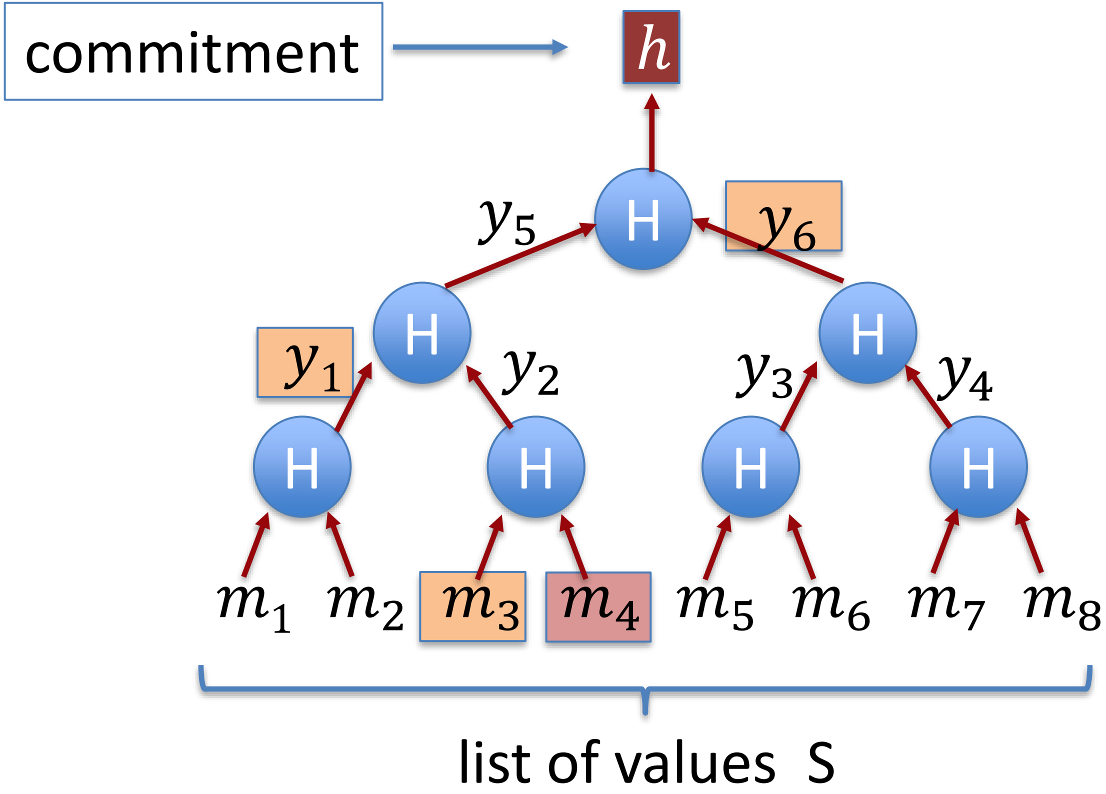


Merkle tree (Merkle 1989)



Goal:
- commit to list S of size n
- Later prove $S[i] = m_i$

Merkle tree (Merkle 1989) [simplified]



Goal:

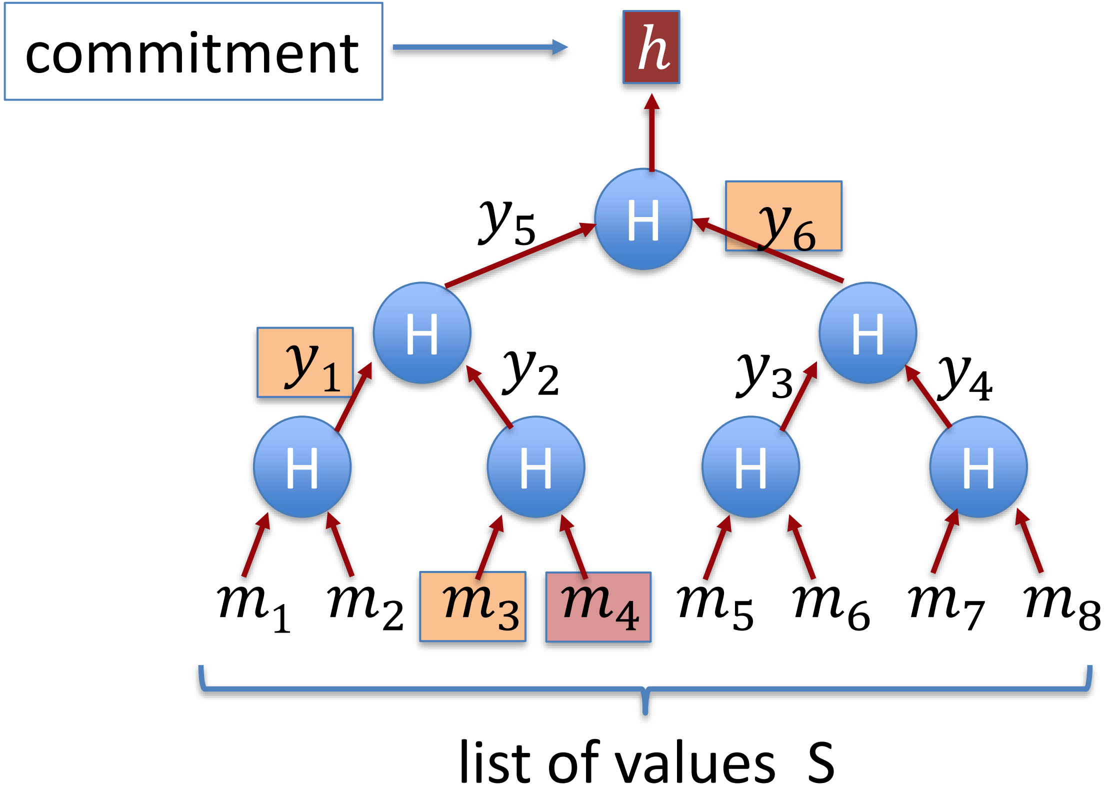
- commit to list S of size n
- Later prove $S[i] = m_i$

To prove $S[4] = m_4$,

proof π = (m_3, y_1, y_6)

length of proof: $\log_2 n$

Merkle tree (Merkle 1989) [simplified]



To prove $S[4] = m_4$,

proof $\pi = (m_3, y_1, y_6)$

Bob does:

$y_2 \leftarrow H(m_3, m_4)$
$y_5 \leftarrow H(y_1, y_2)$
$h' \leftarrow H(y_5, y_6)$
accept if $h = h'$

Merkle tree (Merkle 1989)

Thm: For a given n: if H is a CRF then

adv. cannot find (S, i, m, π) s.t. $|\mathrm{S}| = \mathrm{n}$, $m \neq S[i]$,

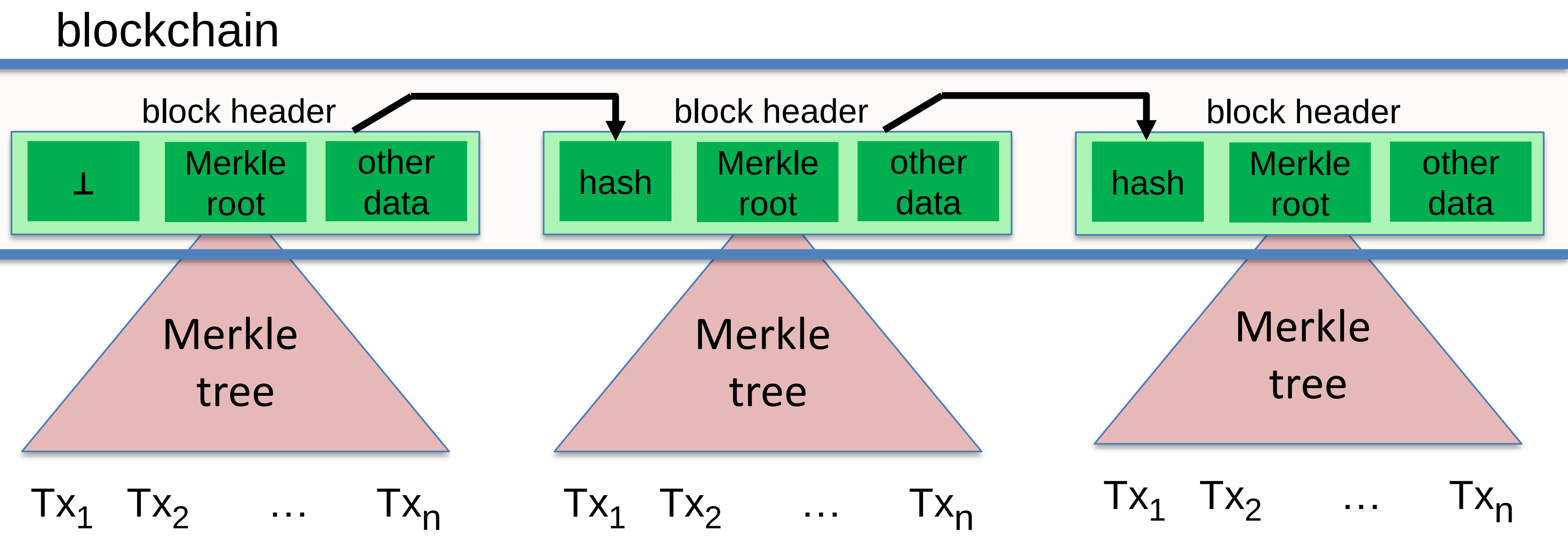
$h = \mathrm{commit}(S)$, and $\mathrm{verify}(h, i, m, \pi) = \mathrm{accept}$

(to prove, prove the contra-positive)

How is this useful? To post a block of transactions S on chain suffices to only write commit(S) to chain. Keeps chain small.

$\Rightarrow$ Later, can prove contents of every Tx.

Abstract block chain



Merkle proofs are used to prove that a Tx is "on the block chain"

Blockchain with Merkle Trees

Block: Header + Data

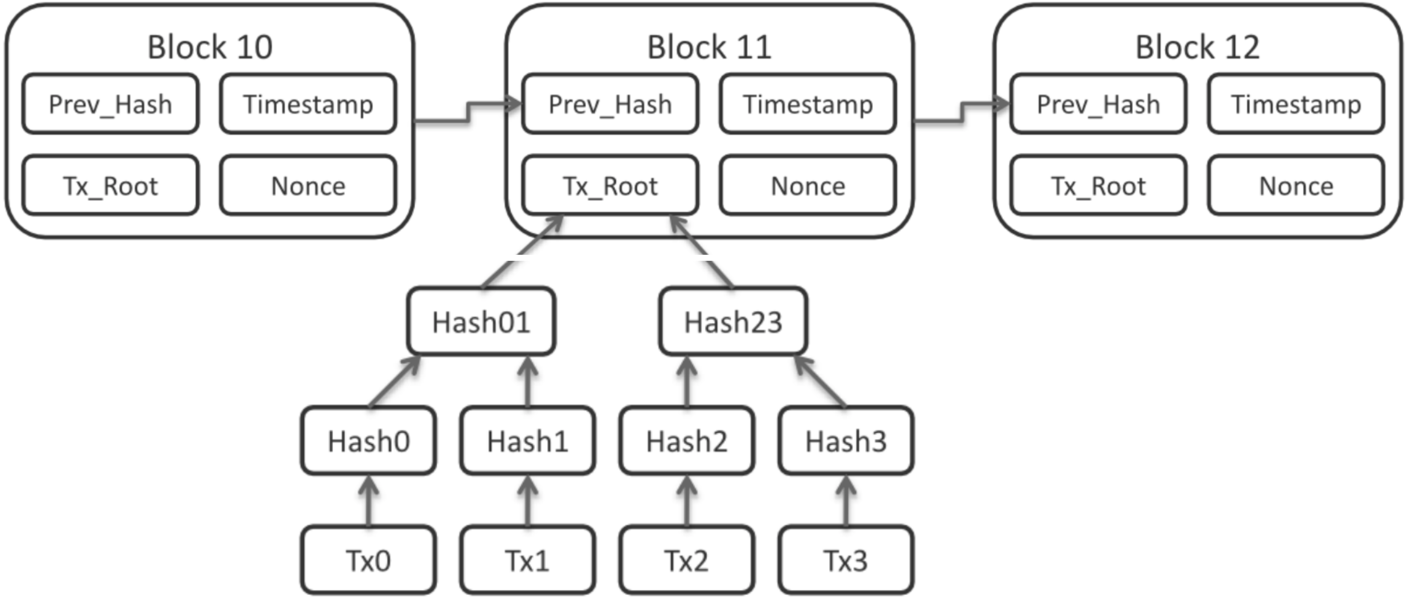
Header: Pointer to previous block
= hash of the previous block header and Merkle root of data of previous block

Data: information specific to the block

Application: Centralized tamper evident information log with efficient proof of membership of any data entry

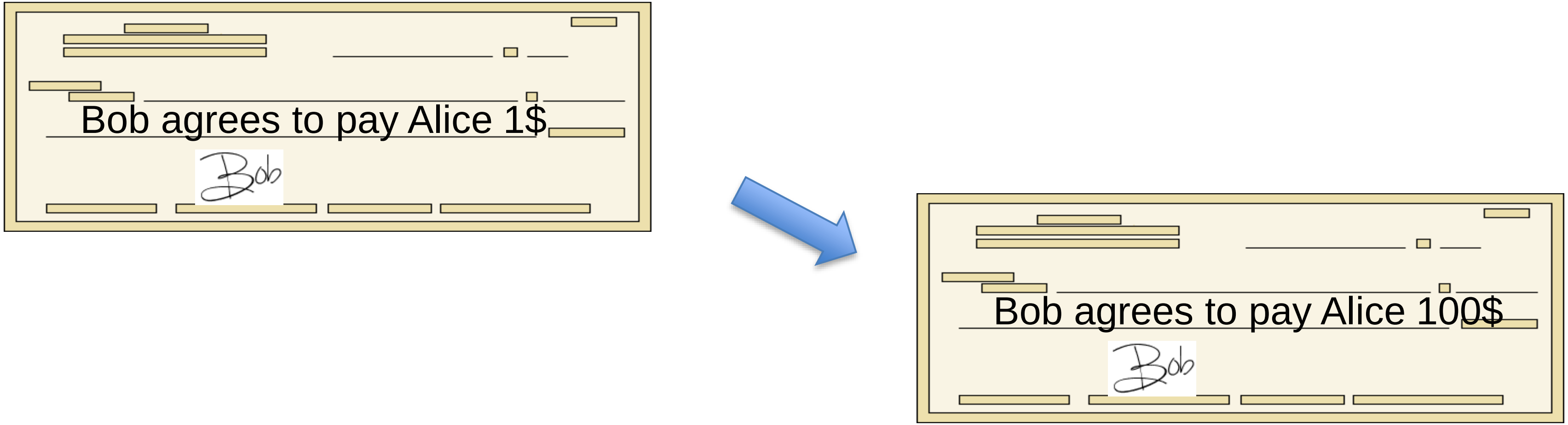
Head of the chain being known is enough to find tamper evidence in any internal block

Blockchain with Merkle Trees



Signatures

Physical signatures: bind transaction to author

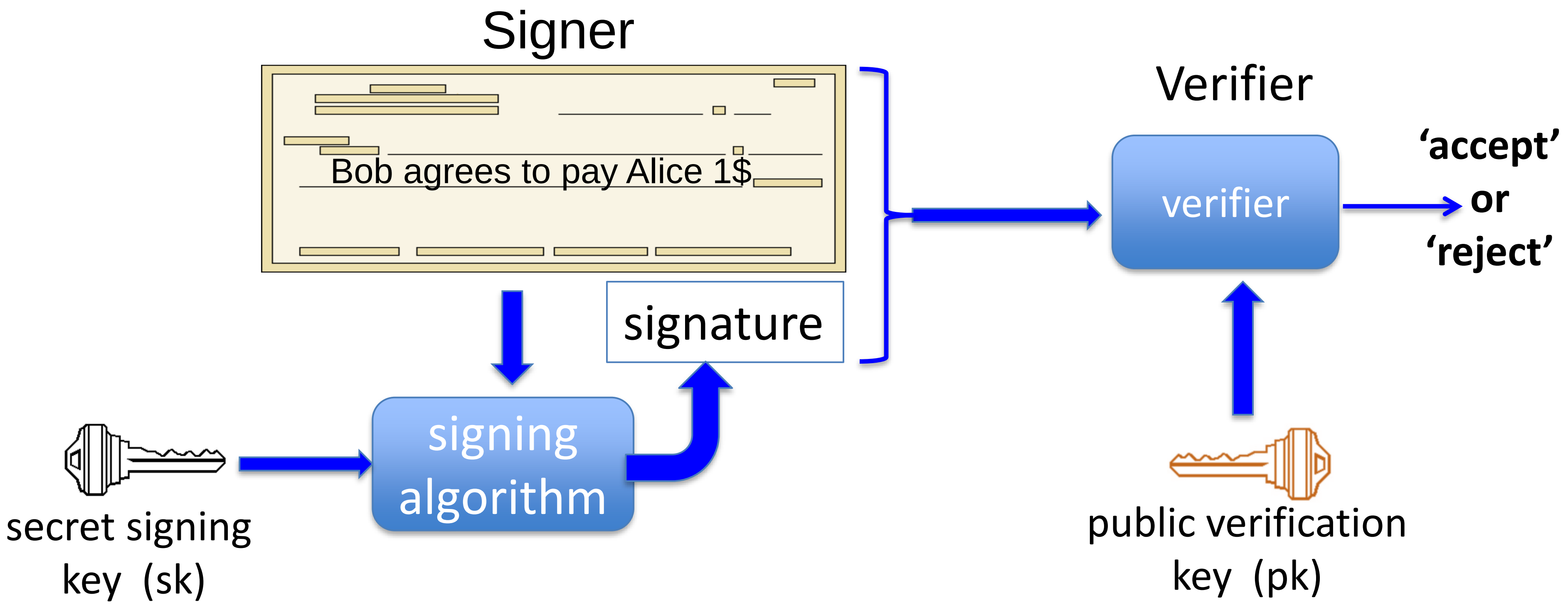


Problem in the digital world:

anyone can copy Bob's signature from one doc to another

Digital signatures

Solution: make signature depend on document



Digital signatures: syntax

Def: a signature scheme is a triple of algorithms:

- **Gen**(): outputs a key pair (pk, sk)
- **Sign**(sk, msg) outputs sig. σ
- **Verify**(pk, msg, σ) outputs 'accept' or 'reject'

Secure signatures: (informal)

Adversary who sees signatures **on many messages** of his choice, cannot forge a signature on a new message.

Families of signature schemes

1. RSA signatures (old … not used in blockchains):
   - long sigs and public keys (≥256 bytes), fast to verify

2. Discrete-log signatures: Schnorr and ECDSA (Bitcoin, Ethereum)
   - short sigs (48 or 64 bytes) and public key (32 bytes)

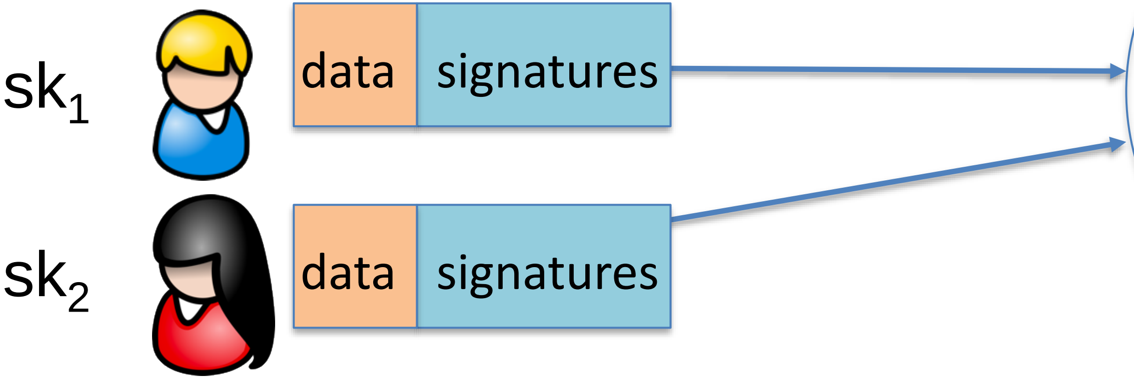
3. BLS signatures: 48 bytes, aggregatable, easy threshold
   (Ethereum 2.0, Chia, Dfinity)

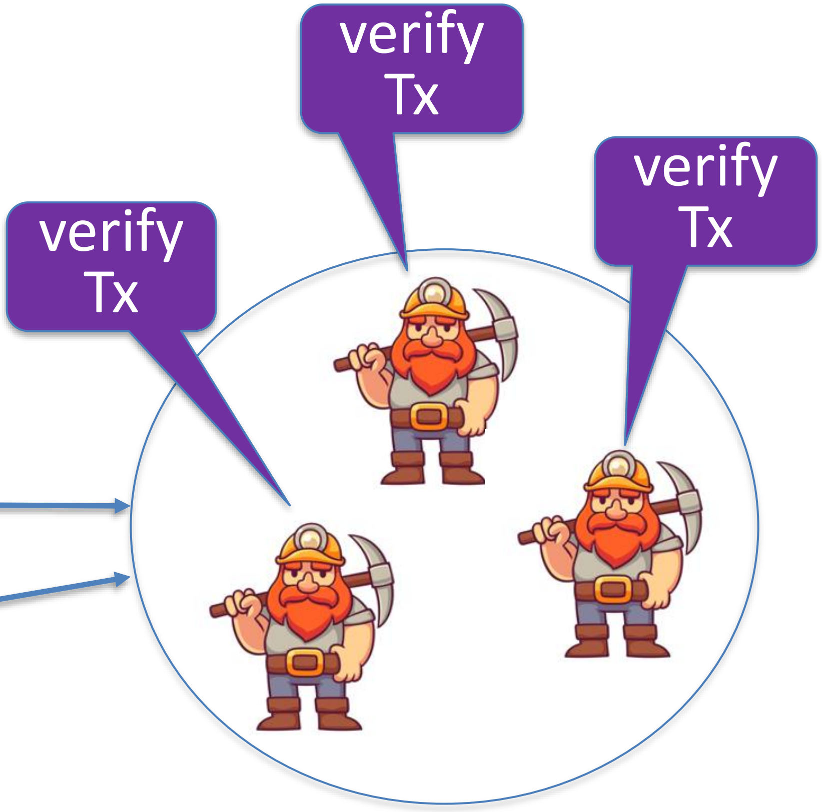
4. Post-quantum signatures: long (≥600 bytes)

Signatures on the blockchain

Signatures are used everywhere:

- ensure Tx authorization,
- governance votes,
- consensus protocol votes.





Decentralizing the Blockchain

Digital Signatures

Decentralized Identity Management

Elements of a cryptocurrency

Digital Signatures

Key generation

(secretkey, publickey) = *Generatekeys*(keysize)

Randomized function

Signature

Sig = *sign*(secretkey, message)

Verification

verify(publickey, Sig, message)

Unforgeable Signatures

Unforgeable

Computationally hard to generate a verifiable signature without knowing the secret key

ECDSA

Elliptic Curve Digital Signature Algorithms

Cryptographicaly secure against an adaptive adversary

Signatures in Practice

Elliptic Curve Digital Signature Algorithm
(ECDSA)
Standard part of crypto libraries

Public key: 512 bits

Secret key: 256 bits

Message: 256 bits
Note: can sign hash of message

Signature: 512 bits

Decentralized Identity Management

Public keys are your identity
- *address* in Bitcoin terminology

Can create multiple identities
- (publickey, secretkey) pairs
- publish publickey
- sign using secretkey

Can create oneself
- verifiable by others

Resources

- ECE/COS 470, Pramod Viswanath, Princeton 2024
- CS251, Dan Boneh, Stanford 2023