Cryptographic primitives

Blockchain Course
Dimitris Grigoriou, Christos Nasikas, Dionysis Zindros

Hash functions

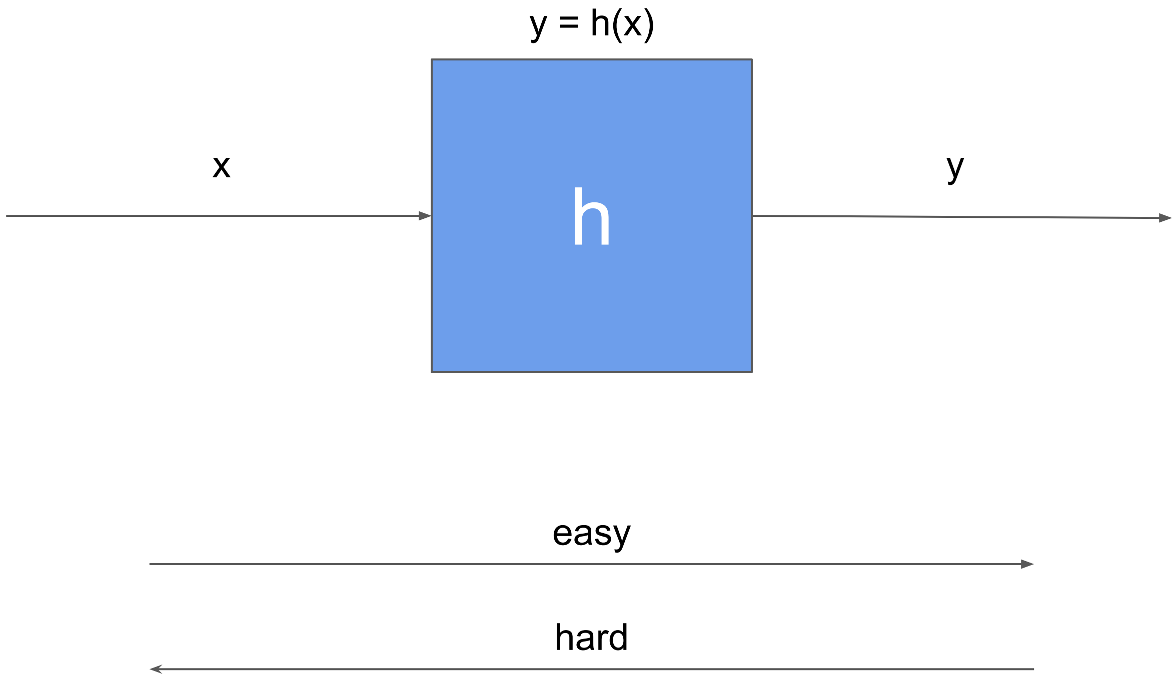
Hash functions

- Map data of arbitrary size to a string of a fixed length
- $h(x) = y$, where $h: \{0, 1\}^* \rightarrow \{0, 1\}^n$
- Various applications: hash tables, file integrity, cache mechanisms, identifiers, time-stamping

Cryptographic Hash Functions

- h(x) = y
- Easy to calculate y, given x
- Properties:
  - Collision resistance: Nobody can find different x, x’ that produce the same y
  - Hiding: Practically infeasible to find x from y
  - Puzzle-friendly: Even a small change to x affects significantly y

Cryptographic Hash Functions



sha2

- Descendant of sha1
- It is considered safe for cryptographic applications
- SHA224: $\{0,1\}^* \rightarrow \{0,1\}^{224}$
- SHA256: $\{0,1\}^* \rightarrow \{0,1\}^{256}$
- SHA384: $\{0,1\}^* \rightarrow \{0,1\}^{384}$
- SHA512: $\{0,1\}^* \rightarrow \{0,1\}^{512}$

Example of sha256

sha256('NBG')

=

79DE06AB4736A8430605FBE1CA7F69E5E9DBB792E977

4C583FFD06F3DFD1273

‘Reversing’ a hash

- Hashes are designed to be one-way, irreversible
- So, the reversing can be done only by **brute-force**
    - Try all possible strings of length 0, 1, 2, 3 etc…
    - Try words from a dictionary

Hash dictionary attack

```
for (word in dictionary):
    if(SHA256(word) == c):
        return word
```

Commitment schemes

Commitment schemes

- Alice wants to commit to a value “b”
- Alice and Bob do not trust each other
- **Binding**: Bob wants to know that Alice will not change her mind about b
- **Hiding**: Alice does not want to reveal her value beforehand

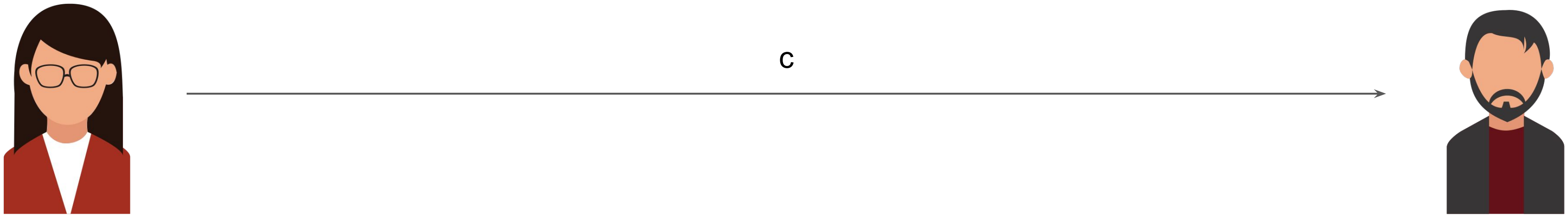
Commitment schemes: phases

- Commitment phase:
  - Alice chooses and commits to a value
  - Sends a secret to Bob
  - **Hiding:** Bob cannot use the secret to find the value of Alice
- Reveal phase:
  - Bob learns the initial value Alice chose
  - **Binding:** Bob confirms that Alice didn’t change her value using the secret

Commitment schemes: using a hash

- Commitment phase:
  - Alice computes **c = H(b)**
  - Send **c** to Bob
  - Bob cannot reverse **c** to get **b**

Commitment schemes: commitment phase



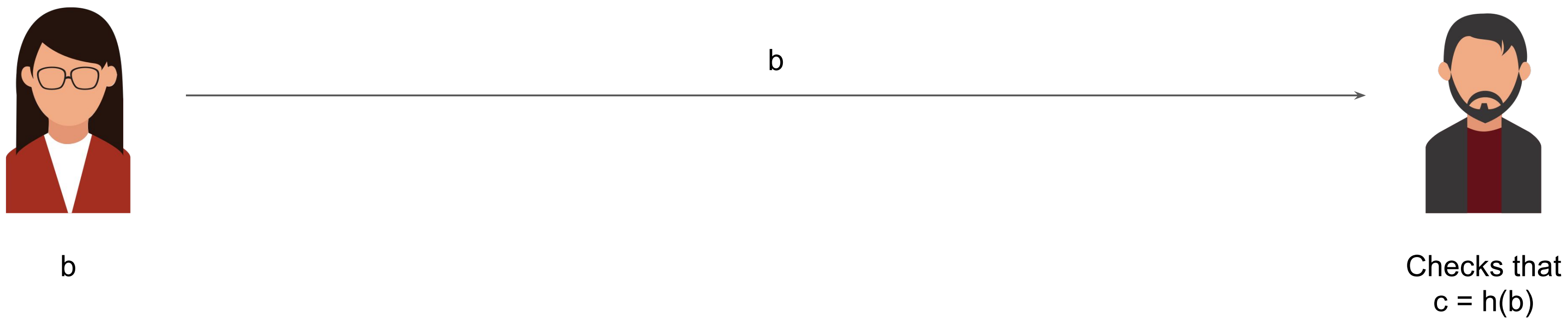
Computes $c = h(b)$

c

Commitment schemes: using a hash

- Reveal phase:
  - Alice sent **b** to Bob
  - Bob checks that H(b) = c

Reveal phase: reveal phase



Proof of work

- Cryptographic primitive that describes the proof of executed CPU cycles
- Alice wants to prove that Bob will dedicate his CPU power only for her
- How can she do that ?

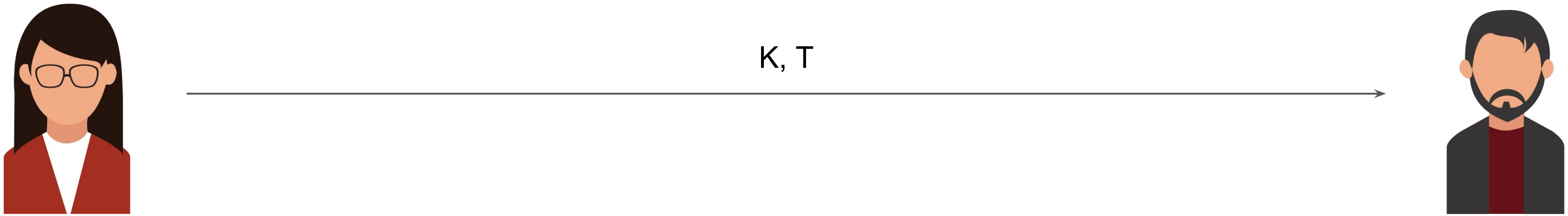
Proof of work

- Alice chooses a random *salt* K, big enough (i.e. 128 bits)
- Alice sends to Bob the salt K, and a target parameter T.
- The target parameter defines the amount of work to be done

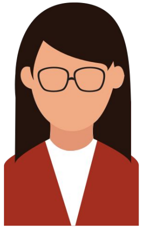
Proof of work

- Bob calculates x such that
    - $H(K \| x) < T$
- x is the *nonce*
- Bob sends x to Alice
- Alice validates that $H(K \| x) < T$

Proof of work: Request

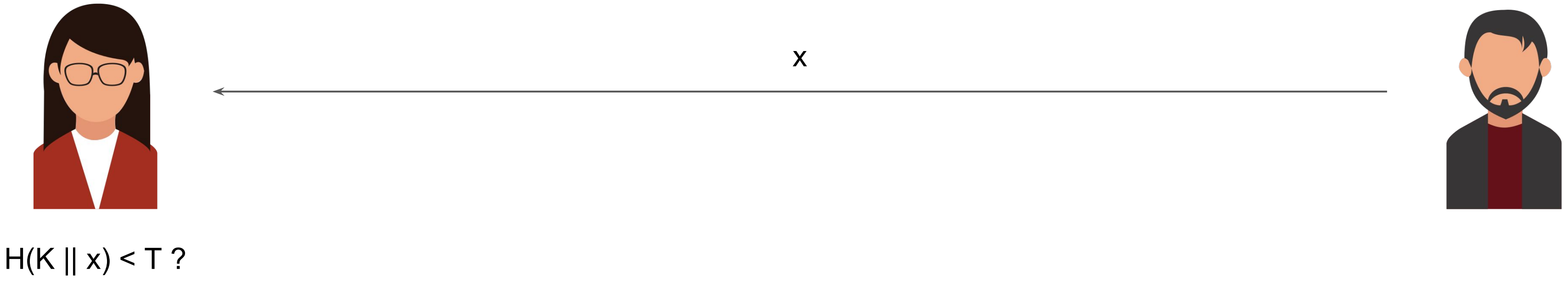


Proof of work: Work



$H(K \,||\, x) < T$?

Proof of work: Proof



Proof of work: Algorithm

```
while (H(K || x) >= T):
    x = rand()
return x
```

Proof of work

- Very important to blockchain consensus mechanisms
- More details in the following lectures...

Merkle trees

Merkle Trees

- A data structure: usually a binary tree
- Used for efficiently summarizing and verifying the integrity and validity of large sets of data
- Useful in peer-to-peer networks
  - Damaged / Altered blocks can be received and identified as such
  - Prevent malicious actors to send fake blocks
- Used in ipfs, git, BitTorrent, nosql databases, bitcoin, ethereum

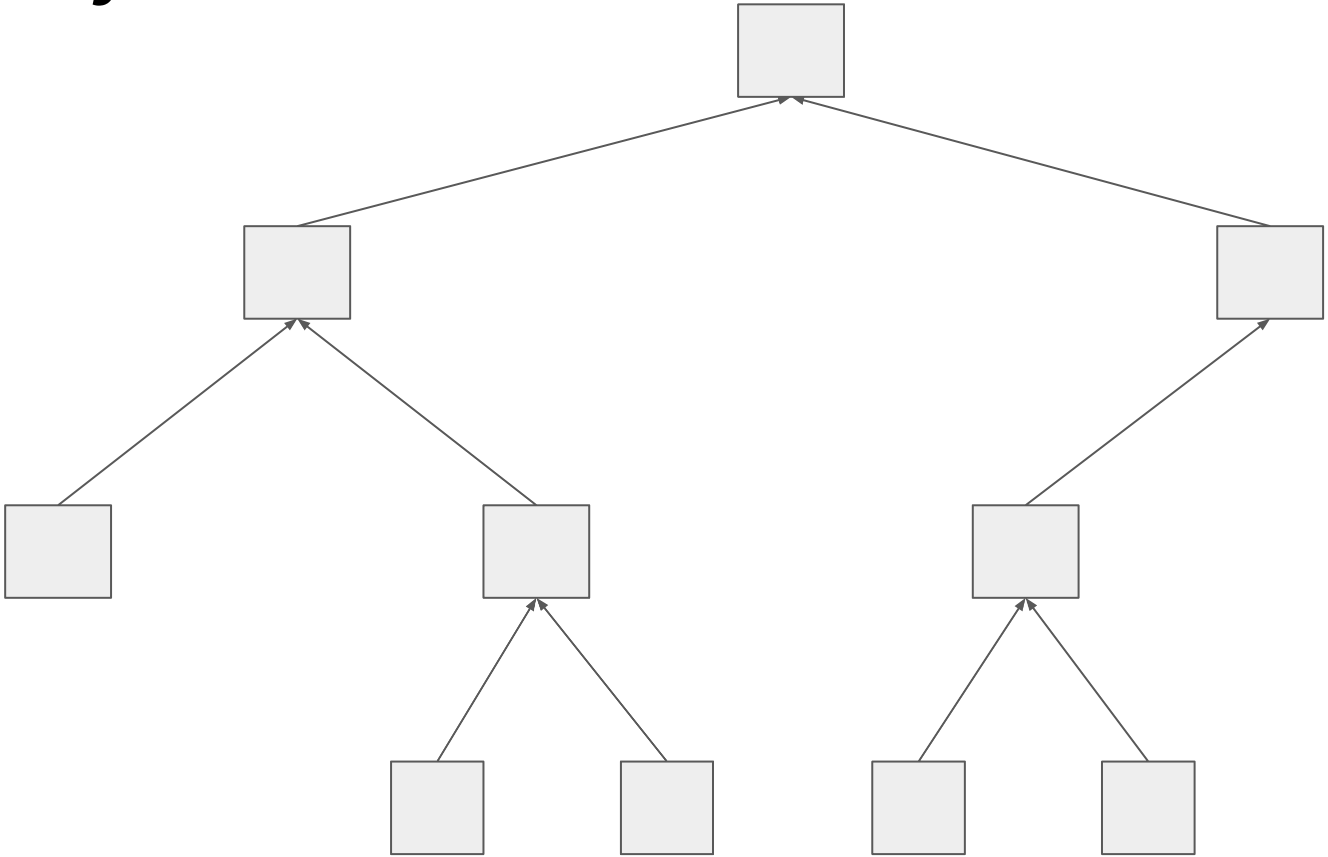
Merkle Trees

- Proofs for data integrity and validity
- Proofs require little memory and space
- Proofs require tiny amounts of information to be transmitted across networks

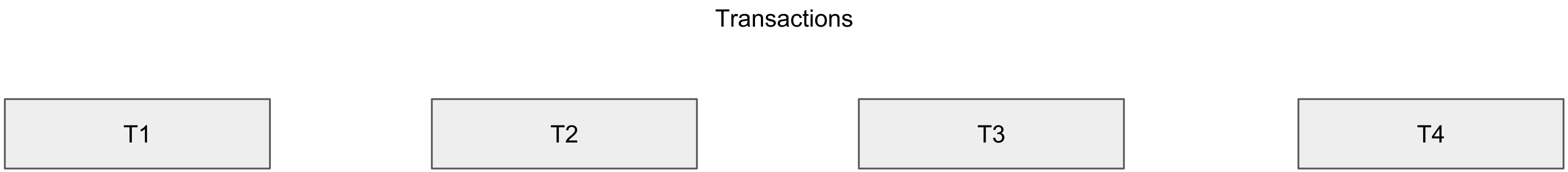
Merkle Trees

- Used for keeping a summary of all the information in a block
- Digital fingerprint of the entire set of information
- Information hasn't been altered or tampered within a block
- Fast proof of inclusion

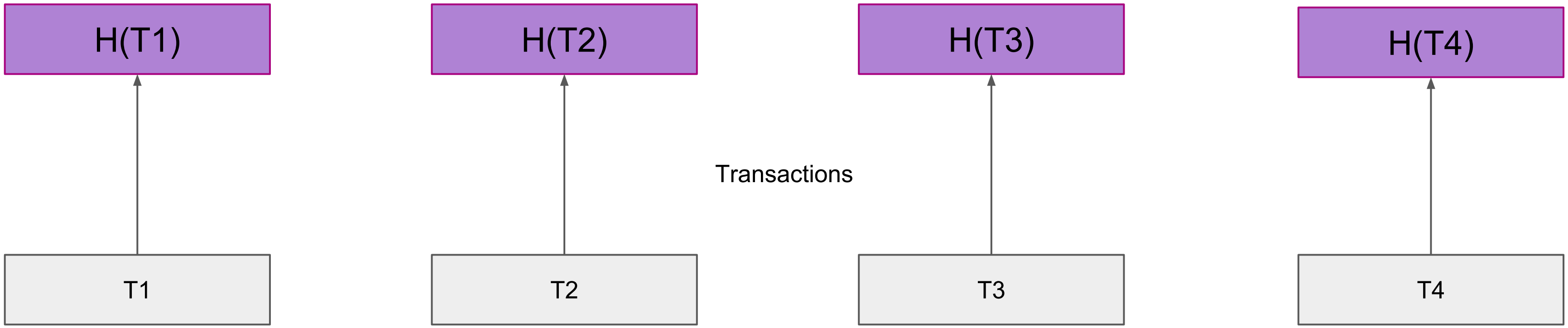
Binary tree



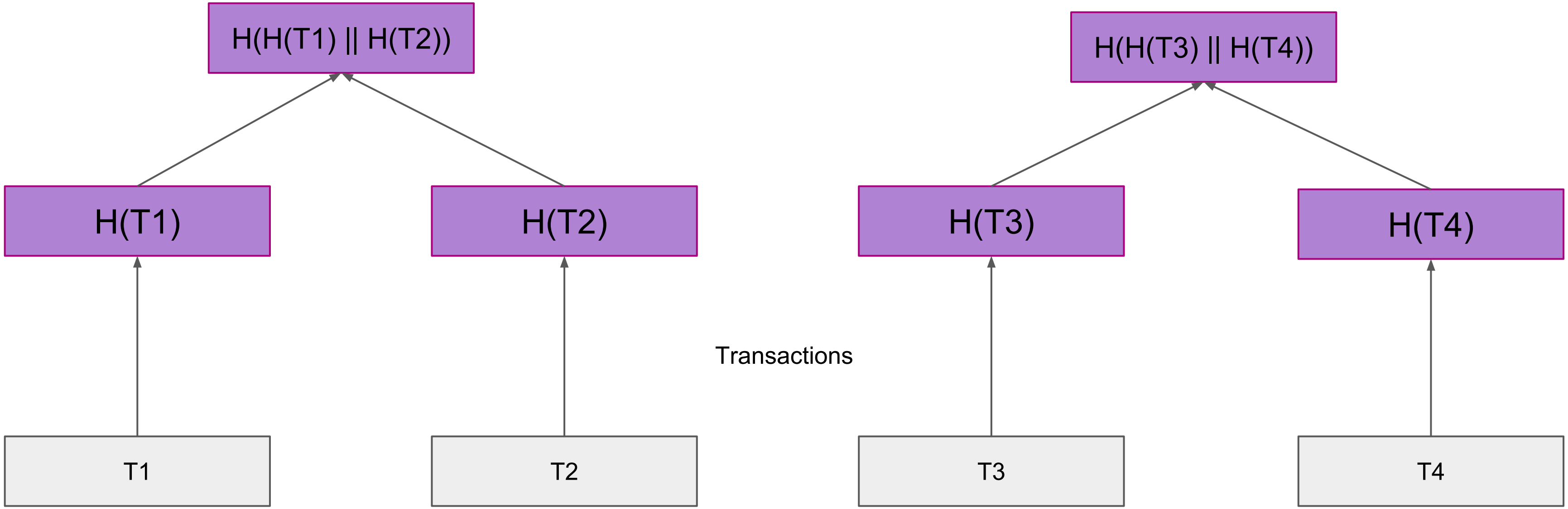
Merkle tree: construction



Merkle tree: construction

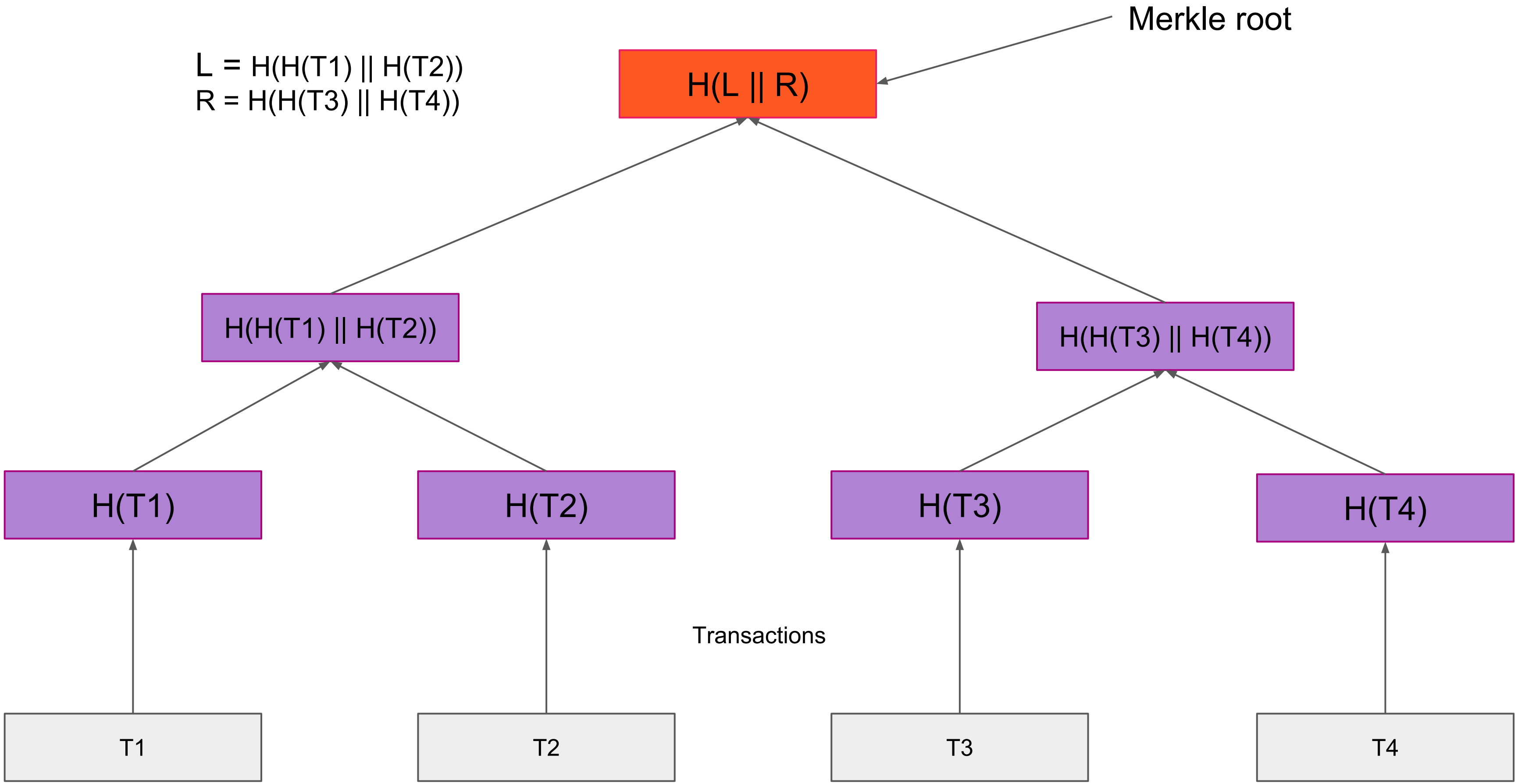


Merkle tree: construction

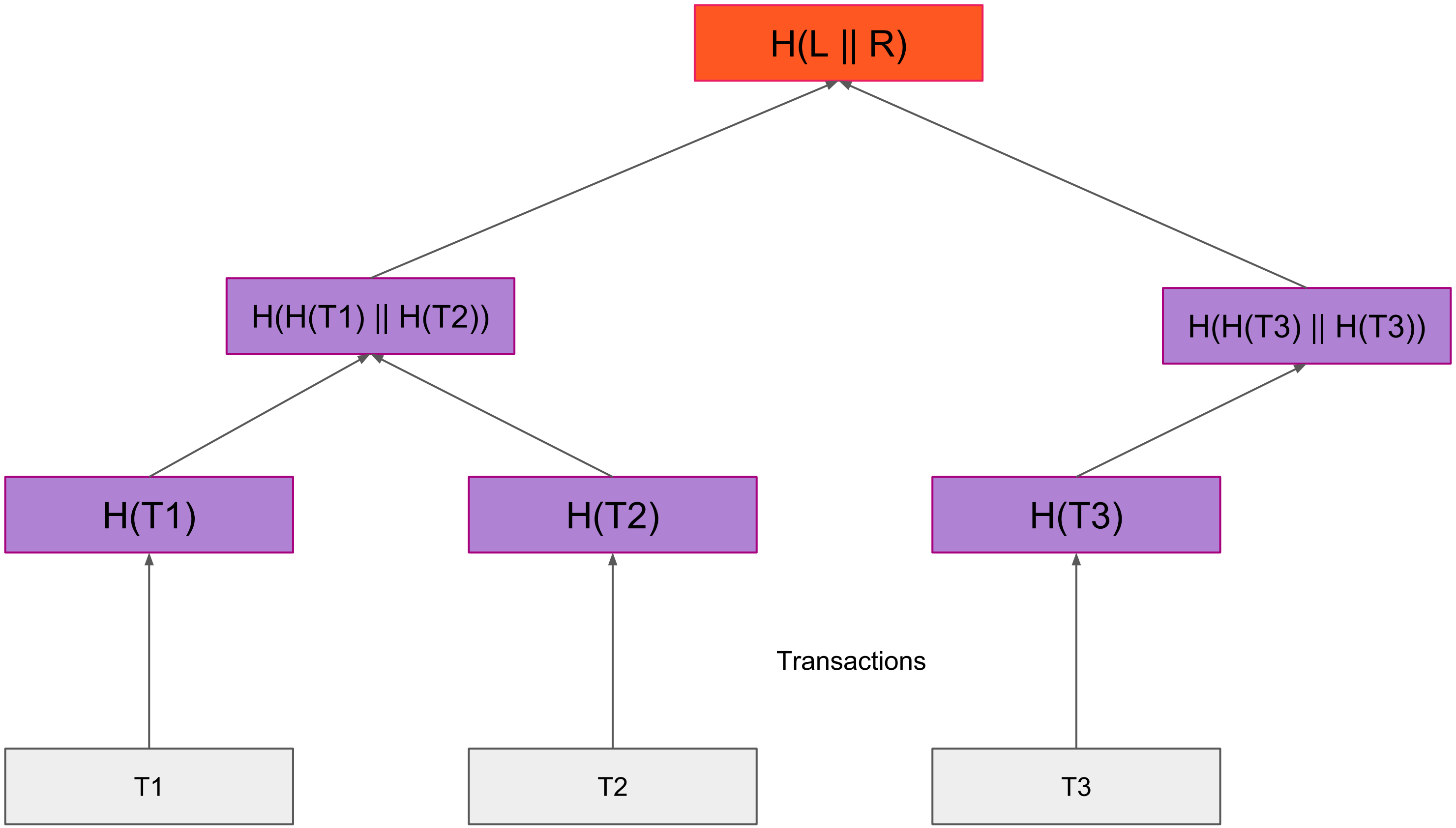


Merkle tree: construction

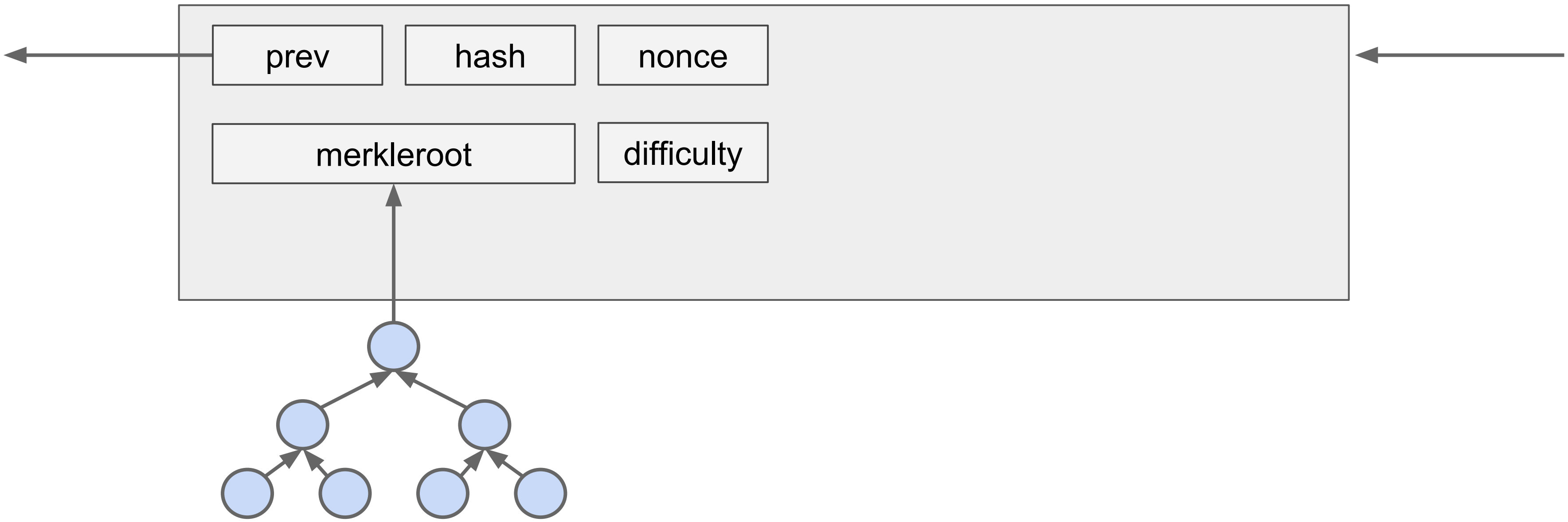
L = H(H(T1) || H(T2))
R = H(H(T3) || H(T4))



Merkle tree: construction



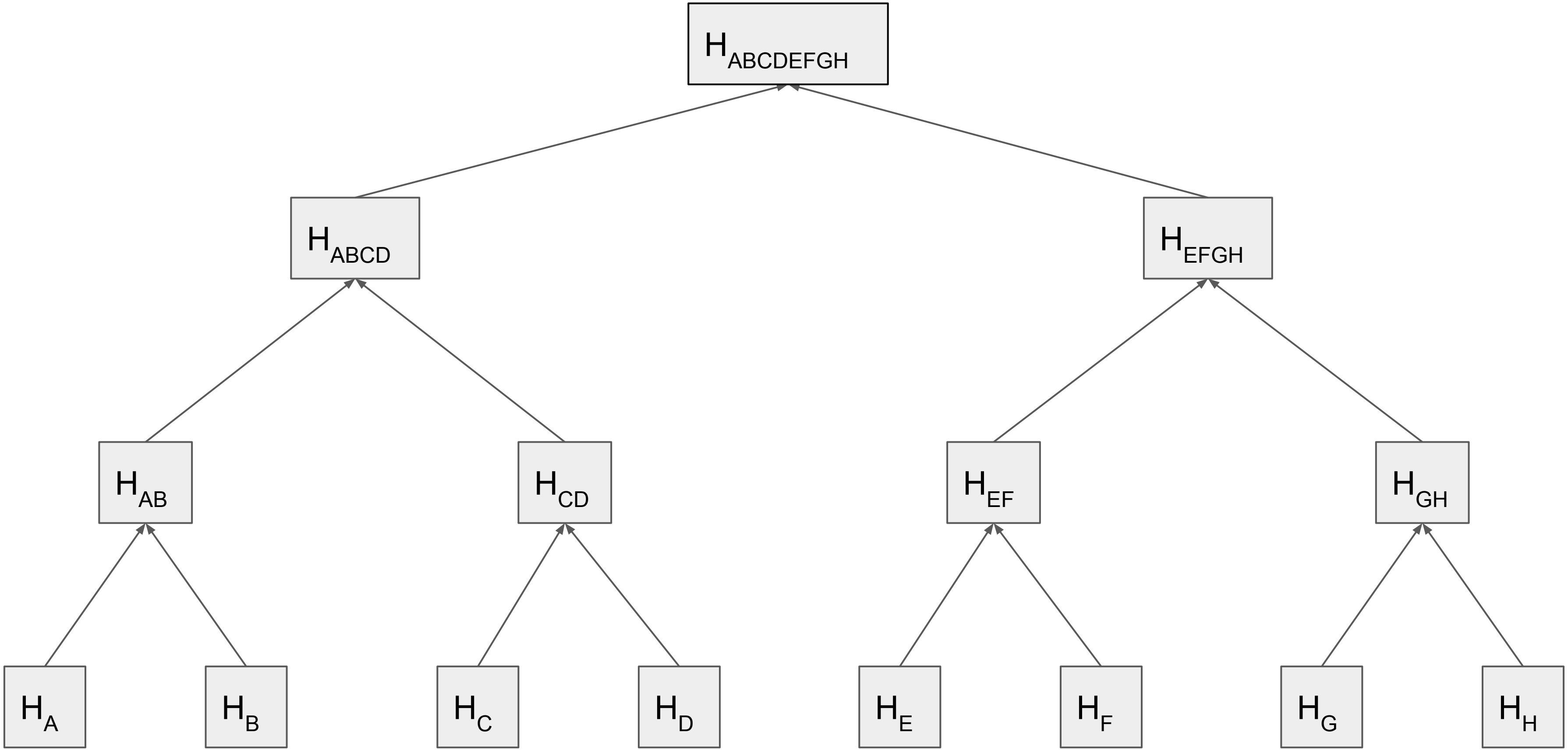
Merkle tree: bitcoin block



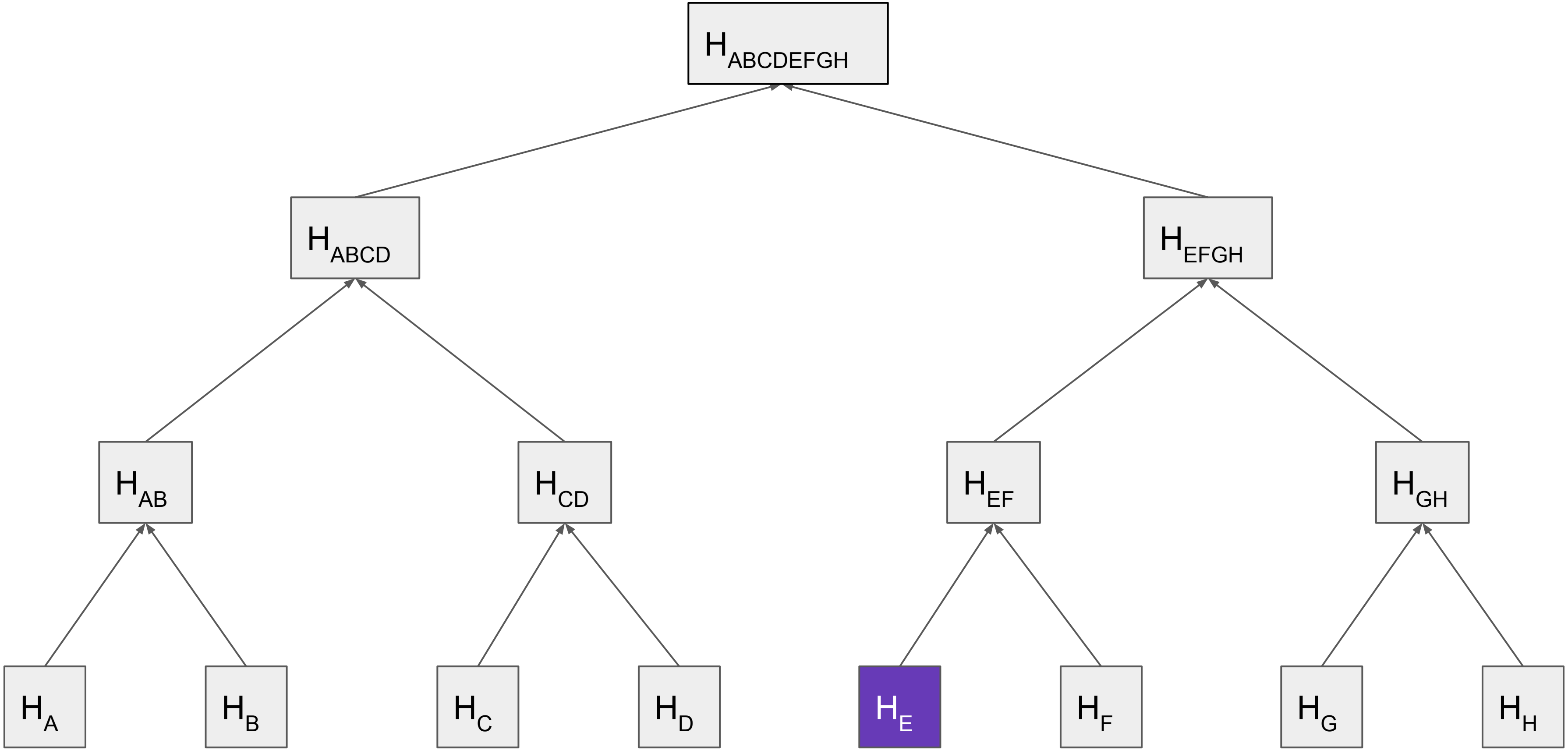
Merkle tree: proof of inclusion

- How do I prove that a transaction belongs to a certain tree?
- No need to download the entire tree

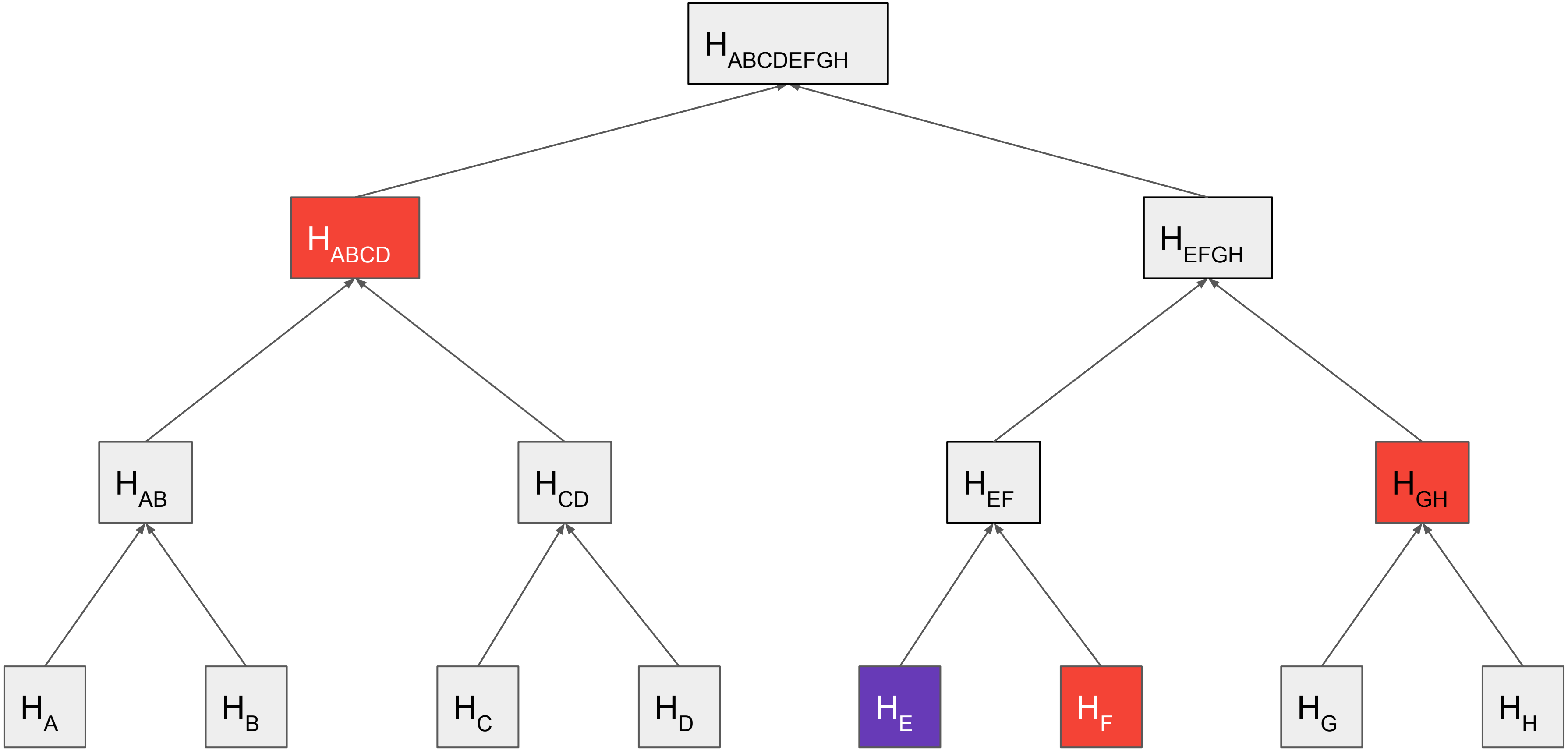
Merkle tree: proof of inclusion



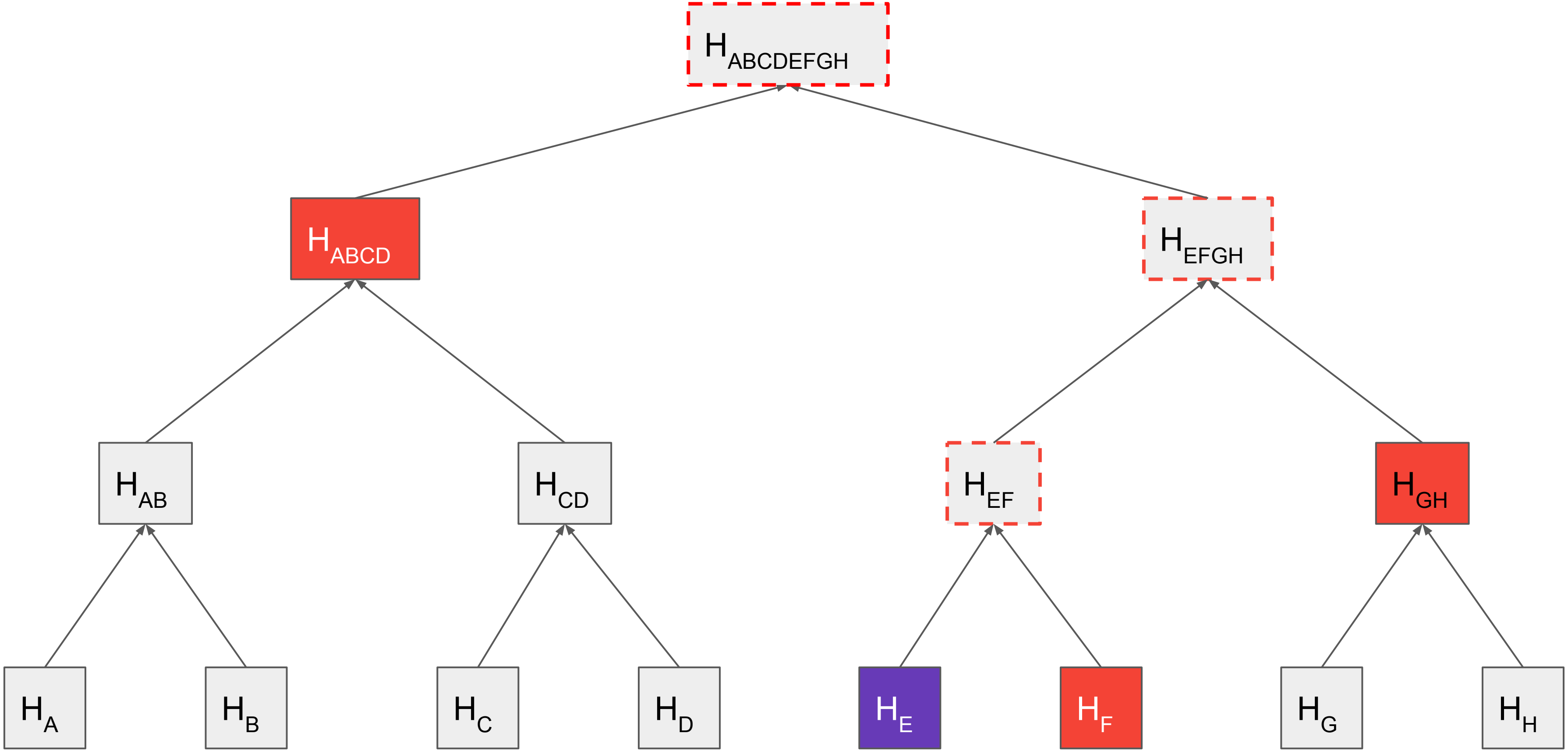
Merkle tree: proof of inclusion



Merkle tree: proof of inclusion



Merkle tree: proof of inclusion



Merkle tree: proof of inclusion

- As a client that needs to store space there is no need to store the entire tree, but only the header

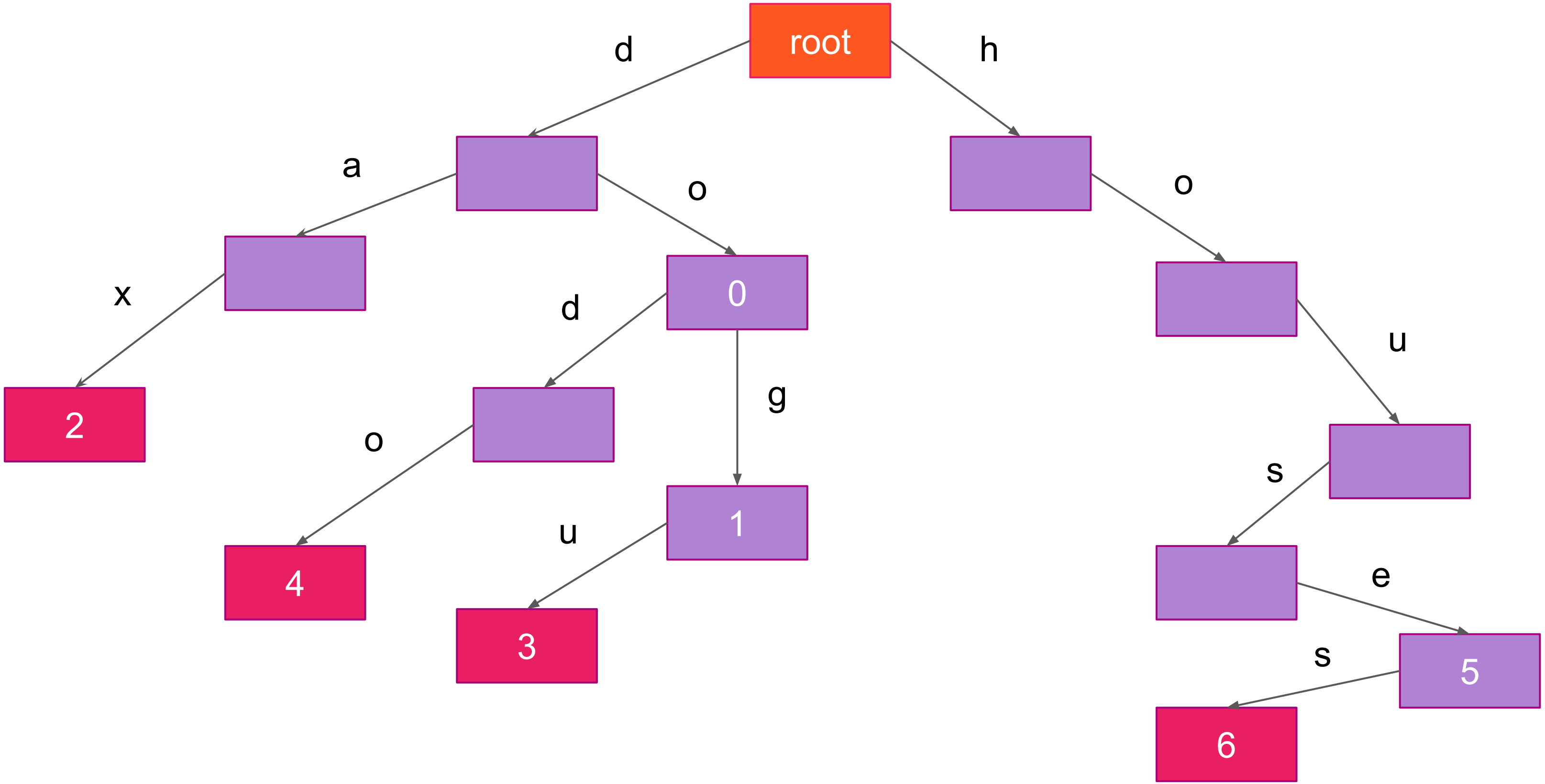
Tries

- Called also radix tree or prefix tree
- Search tree: ordered tree data structure
- Used to store a set or an associative array
- Keys usually are strings

Tries: example

{ **do**: 0, **dog**: 1, **dax**: 2, **dogu**: 3, **dodo**: 4, **house**: 5, **houses**: 6 }

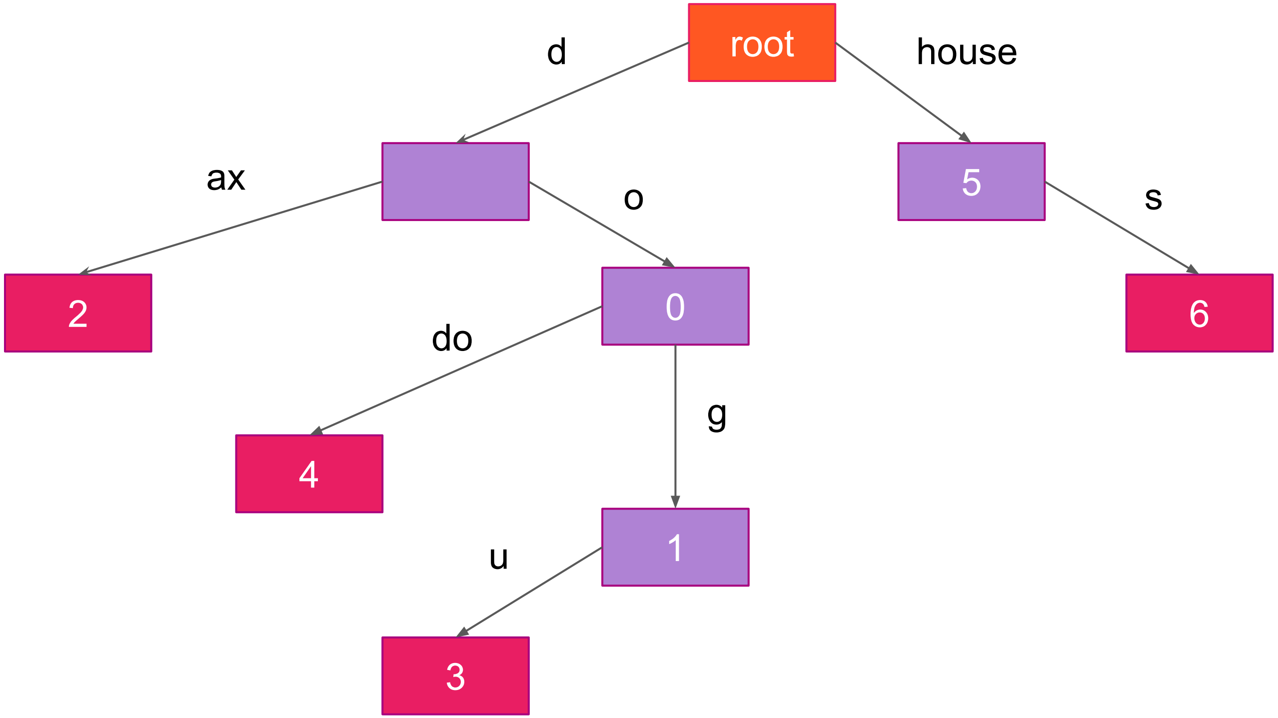
Tries



Patricia (or radix) trie

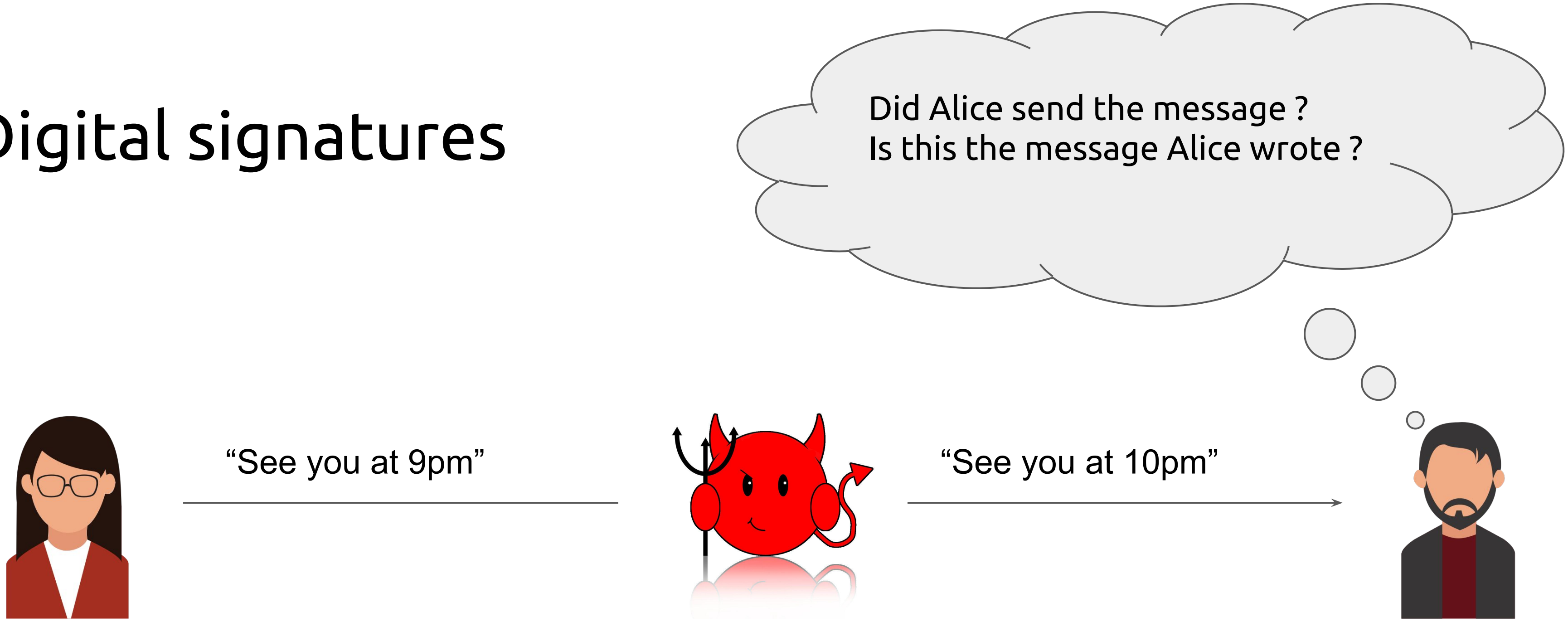
- Space-optimized trie
- Each node that is the only child is merged with its parent

Patricia trie



Digital signatures

Digital signatures



Digital signatures

- **Integrity**: The signed message was not *altered* in transit
- **Authentication**: The signature was created by a known sender
- **Non-repudiation**: The sender cannot deny having signed the message

Digital signatures

- The sender produce a key pair (vk, sk)
- sk: signing key
- vk: verification key

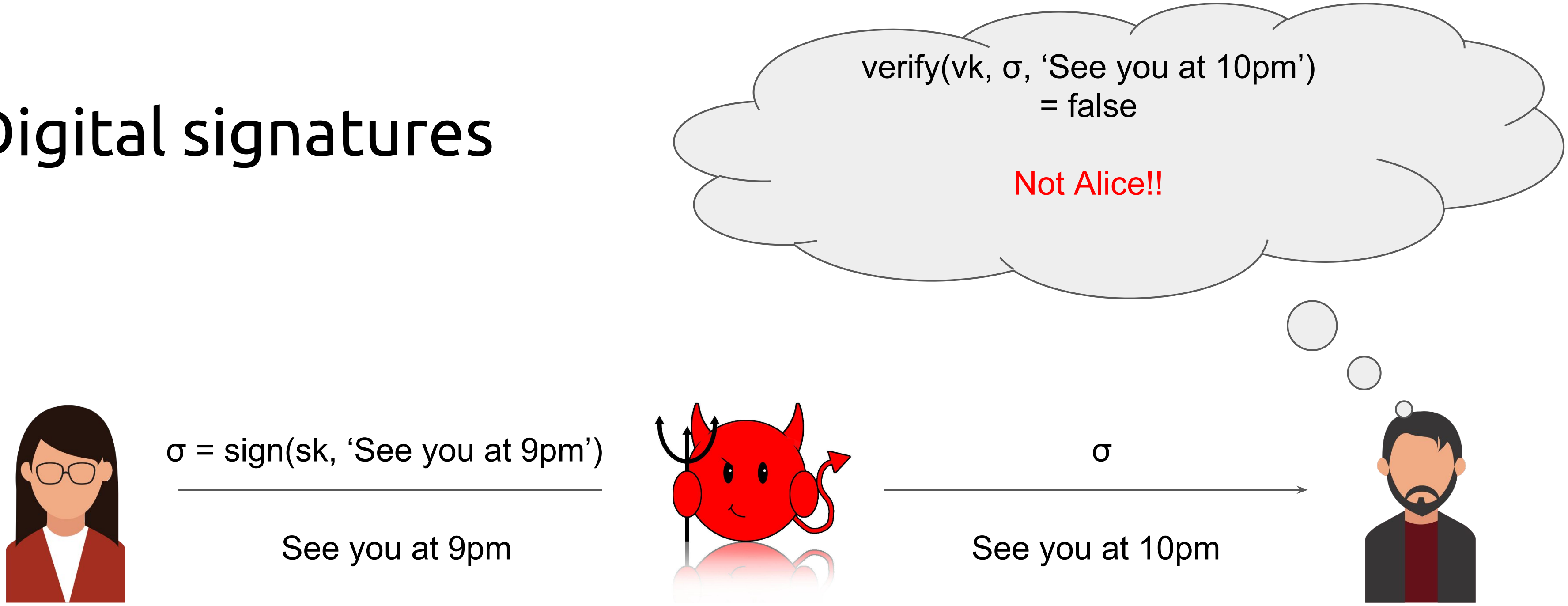
Digital signatures

- Verification key is **public**
  - publicly verifiable
  - transferable signatures
- Signing key is private

Digital signatures

- $\sigma = \text{sign}(sk, m)$
- $\text{verify}(vk, \sigma, m) = \{0, 1\}$

Digital signatures



Digital signatures

- Digital signatures are very important to blockchain systems
- A valid signature implies ownership
- More details in the following lectures...