COMP4137 Blockchain Technology and Applications
COMP7200 Blockchain Technology

Lecturer: Dr. Hong-Ning Dai (Henry)

Lecture 4

Bitcoin Basics

Cryptocurrency

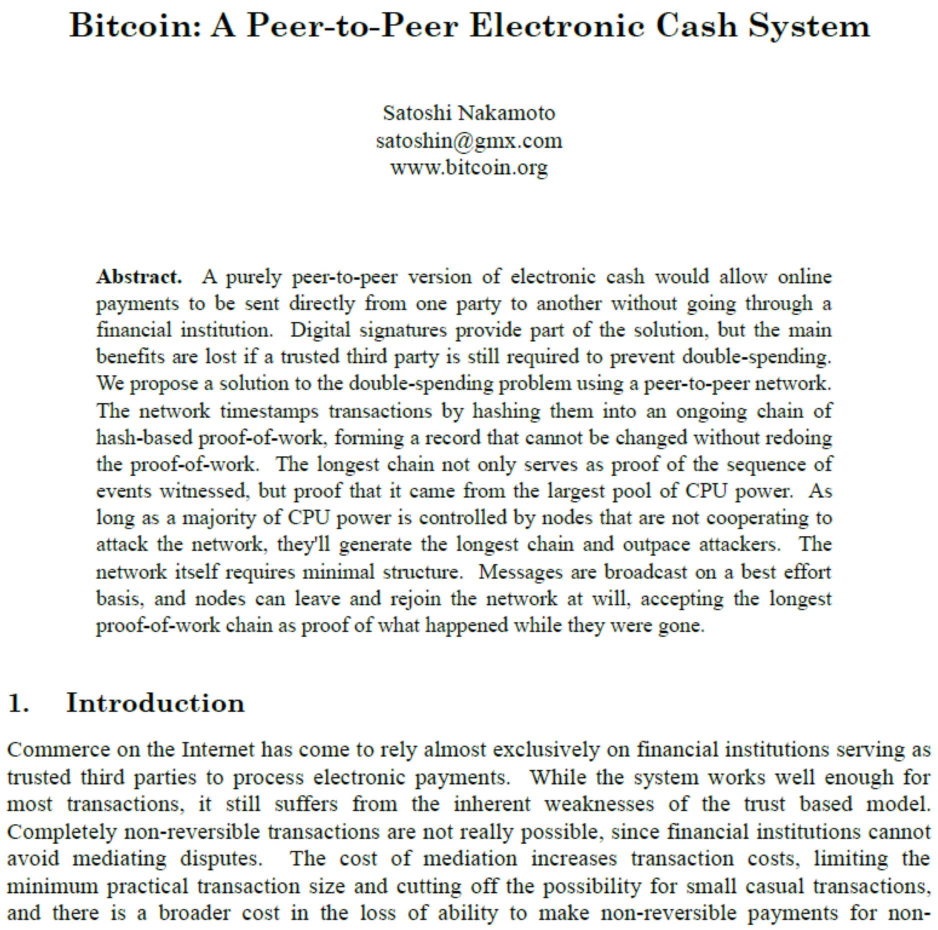
Bitcoin

Ethereum

Satoshi Nakamoto
中本聰

Vitalik Buterin

https://vitalik.ca/index.html



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

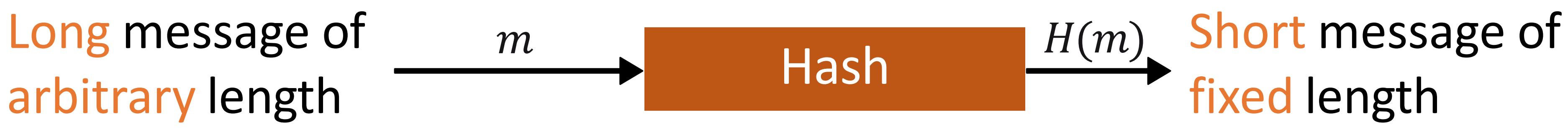
Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-

Outline

- Hash Function and Merkle Tree
- Elliptic Curve Cryptography (ECC)

Hash Functions

- A hash function maps/compresses messages of arbitrary lengths to an *m*-bit output
  - Output known as the fingerprint or the message digest
- What is an example of hash functions?
  - Given a hash function that maps Strings to integers in [0,2^{32}-1]
- A hash function is a many-to-one function, so collisions must happen.
- Hash functions are used in a number of data structures


Long message of arbitrary length
m
Hash
$H(m)$
Short message of fixed length

$$|H(m)| \ll |m|$$

$$|H(m)| = \{160, 256, 384, 512\}$$ (preferred 256 bits)

Hash Functions

- Currently popular Hash algorithms include MD5, SHA-1, and SHA-2.
- MD4 (RFC 1320) was designed by Ronald L. Rivest of MIT in 1990. MD is an acronym for Message Digest. Its output is 128 bits. MD4 has proven to be insufficiently secure.
- MD5 (RFC 1321) is an improved version of MD4 by Rivest in 1991. It still groups the inputs in 512 bits and the output is 128 bits. MD5 is more complex than MD4 and is slower and safer to calculate. MD5 has proven **not** to be "strong resistant collision".

Hash Functions

- Currently popular Hash algorithms include MD5, SHA-1, and SHA-2.
- SHA (Secure Hash Algorithm) is a family of Hash functions. The first algorithm was released in 1993 by NIST (National Institute of Standards and Technology).
- The well-known SHA-1 was introduced in 1995, and its output is a 160-bit hash value, so it is better against exhaustiveness. The SHA-1 design is based on the same principle as MD4 and mimics the algorithm. SHA-1 has been proven **not** to be "strong resistant collision".
- To improve security, NIST also designed SHA-224, SHA-256, SHA-384, and SHA-512 algorithms (collectively referred to as SHA-2), similar to the SHA-1 algorithm. SHA-3 related algorithms have also been proposed.

Security Requirements for Cryptographic Hash Functions

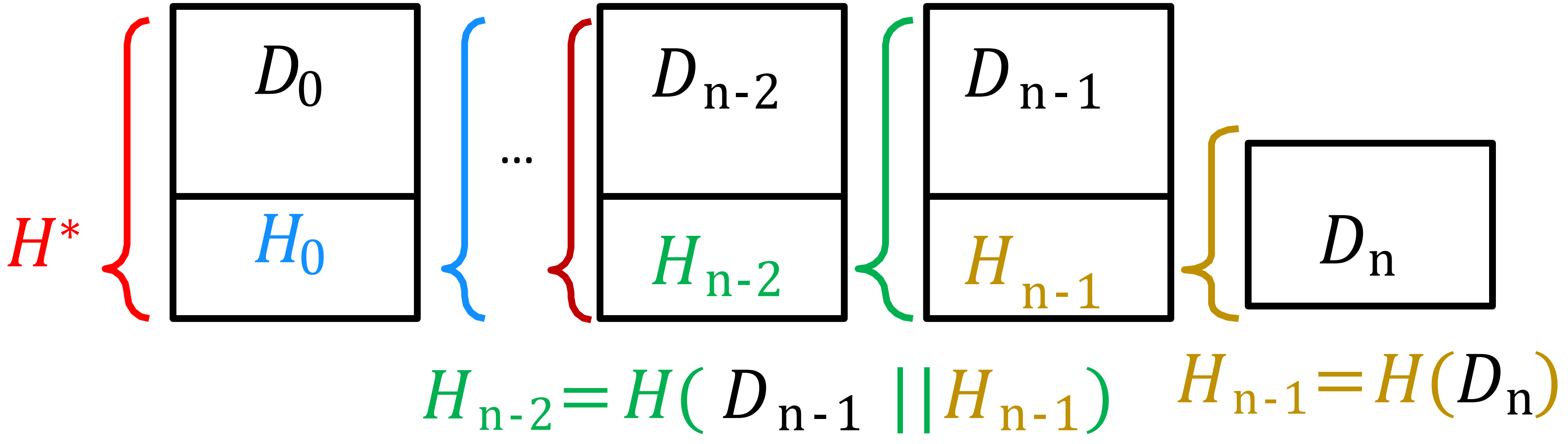
- Given a function H: $m \rightarrow H(m)$, then we say that H is:
- One-way property:
  - Given $H(m)$, it is computationally infeasible to find a value m

$$\begin{tikzcd} m \arrow[r, shift left, "\text{easy}"] & H(m) \arrow[l, shift left, "\times" description, "\text{hard}"] \end{tikzcd}$$

- Weak collision resistant:
  - Given an arbitrary m, it is computationally infeasible to find *some* m' s.t. $H(m') = H(m)$
- Strong collision resistant:
  - It is computationally infeasible to find any two distinct values m_1,m_2, s.t. $H(m_1) = H(m_2)$

Chained Hash

- More general construction than one-way hash chains
- Useful for authenticating a sequence of data values $D_0, D_1, \ldots, D_n$
- H^* authenticates the entire chain



Merkle Tree

- Introduced by Ralph Merkle, 1979
  - "Classic" cryptographic construction
  - Involves combining hash functions on binary tree structure

- An authentication scheme
  - Using only one-way hash function as building blocks

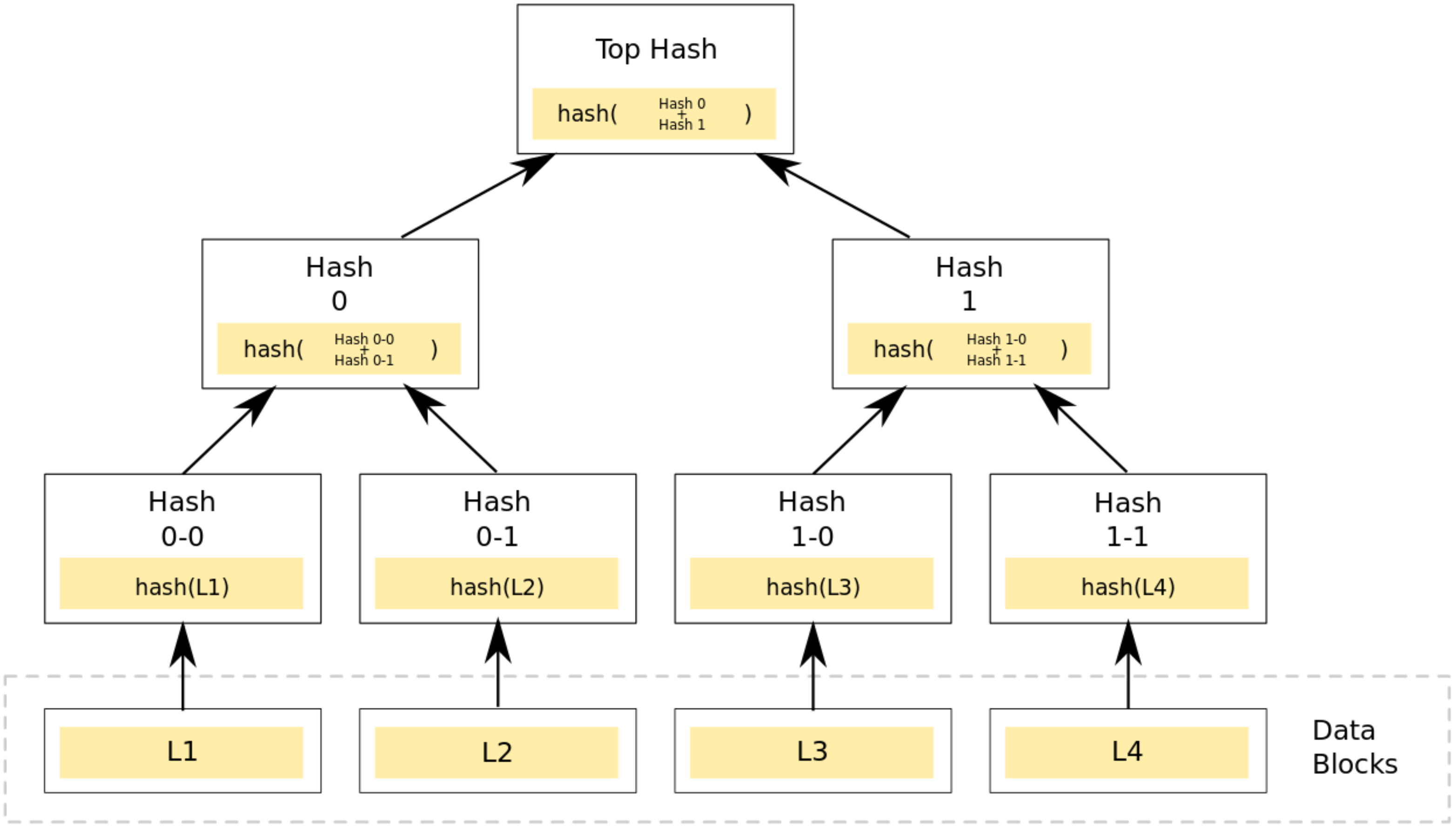
- An efficient data structure with many practical applications
  - Blockchain, privacy protection, integrity check, quick search, …

Merkle Tree Data Structure

- A binary tree over data values for authentication purpose
- Verifier stores the root as the commitment of the Merkle tree

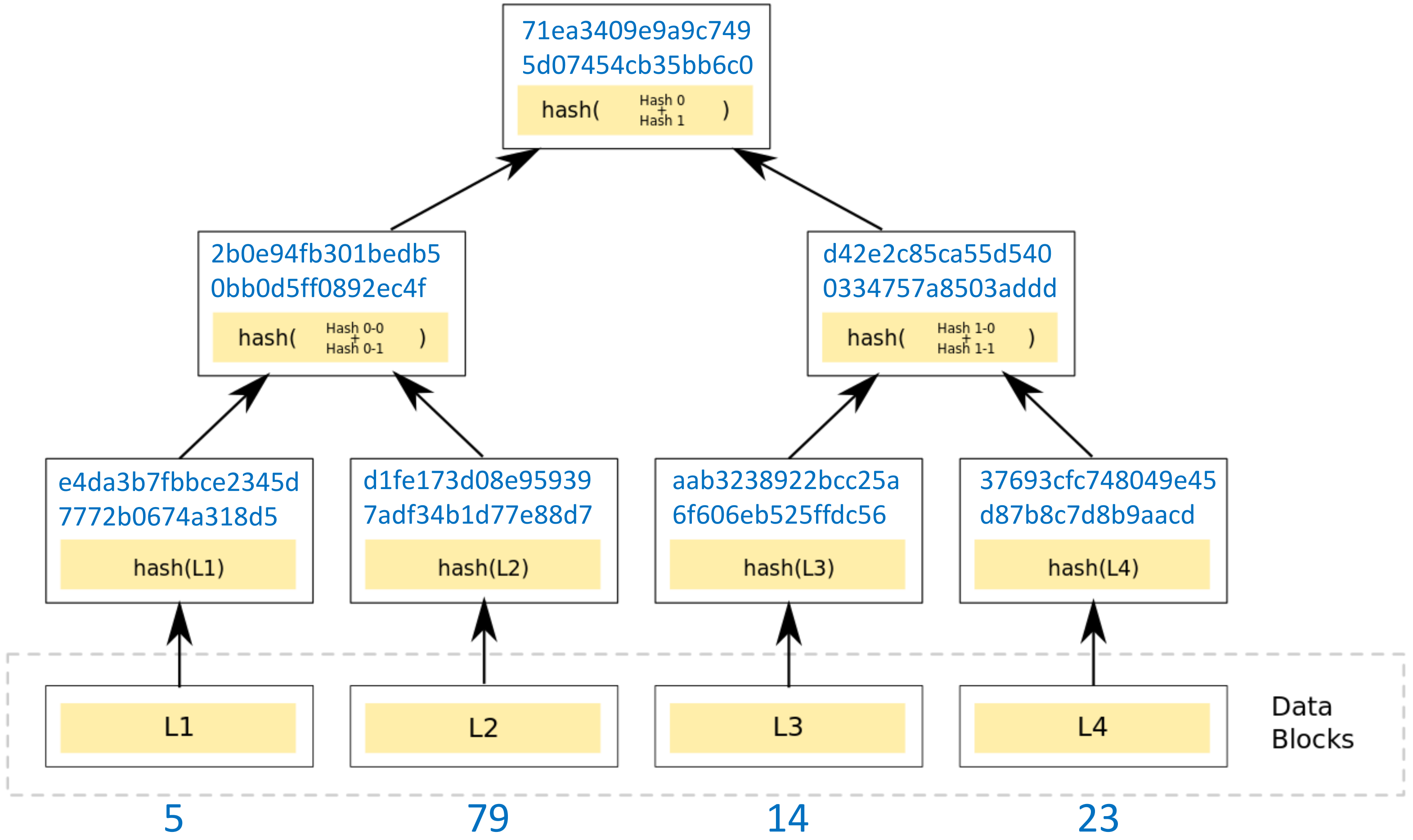
Merkle Tree Data Structure

- Binary tree, nodes are assigned values (e.g. 160 bits)
- Extra, secret values associated to each leaf



Example

- Binary tree, nodes are assigned values (e.g. 160 bits)
- Extra, secret values associated to each leaf



Setup

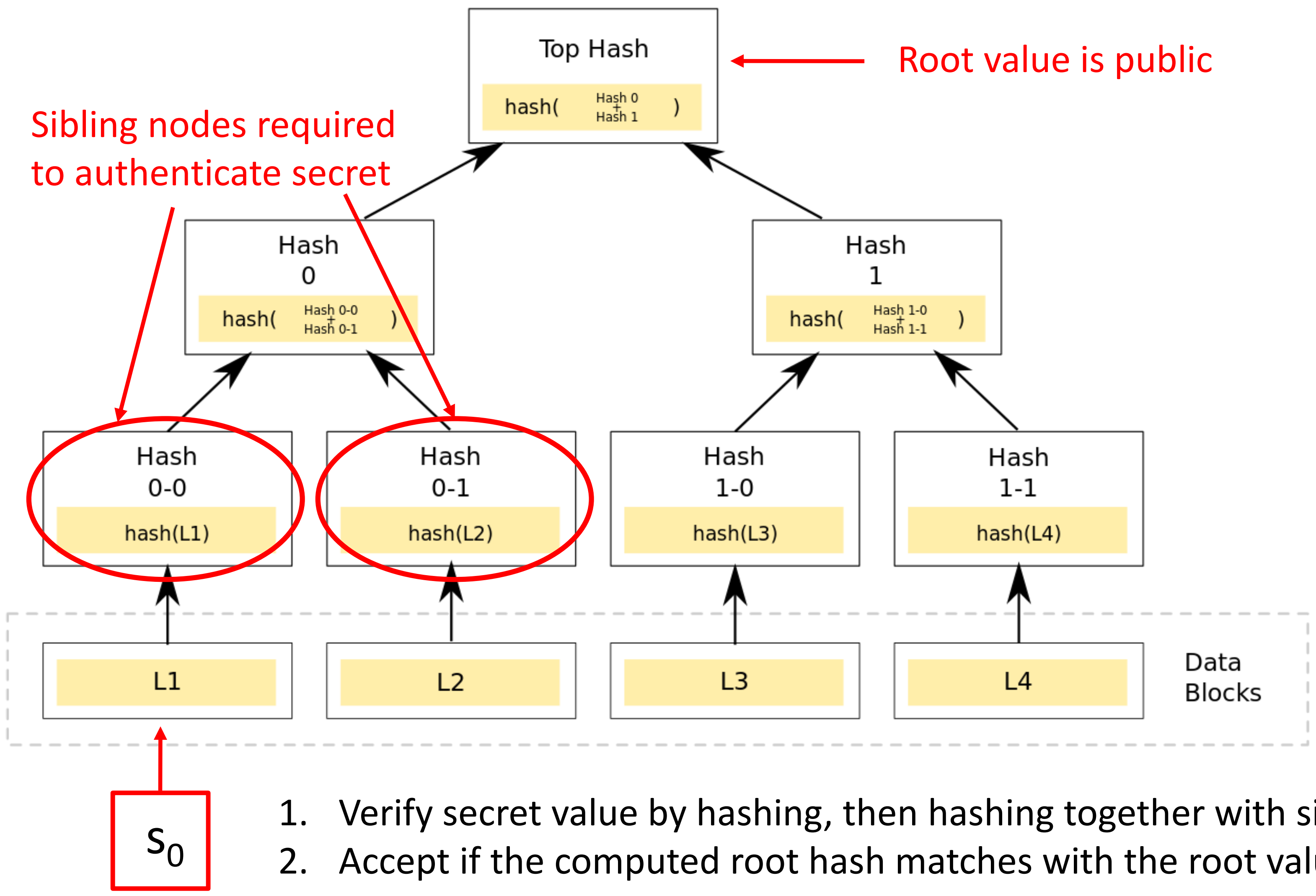
- Computing the tree and root hash
  - Prepare leaf secrets s_i
  - Use hash function to get leaf / interior node values
  - Generate root hash P
- Complexity analysis
  - Tree of height H has $N = 2^H$ leaf nodes
  - Nodes at height H will depend on 2^H leaf values
  - Obtaining P requires calculating all N leaf values plus 2^H-1 more hash function evaluations

Authenticating A Secret

- Prover wishes to reveal s_i to identify herself
  - Prover sends i, s_i
  - Additional data required: "**sibling node**" values

- Verifier checks s_i against the public root hash P
  - Hash first s_i
  - Hash result together with its sibling in tree
  - Repeat, moving up tree
  - Check result with root

Sibling Node Values Required



1. Verify secret value by hashing, then hashing together with sibling, etc.
2. Accept if the computed root hash matches with the root value

Public Key Infrastructure Scenario

- Alice and Bob have public key certificates issued by certificate authority (CA).
- Alice wishes to perform a transaction with Bob and sends him her public key certificate.
- Bob, concerned that Alice's private key may have been compromised, sends a request to CA that contains Alice's certificate serial number.

Public Key Infrastructure Scenario

- Receiving the query, CA checks the revocation status of Alice's certificate.
  - CA maintains the certificate database.
  - The database is the only trusted location where a compromise to Alice's certificate would be recorded.
- CA confirms that Alice's certificate is still OK
  - returns a signed, successful 'response' to Bob.
- Bob verifies the signed response using CA's public key.

A Large Amount of enquiries?

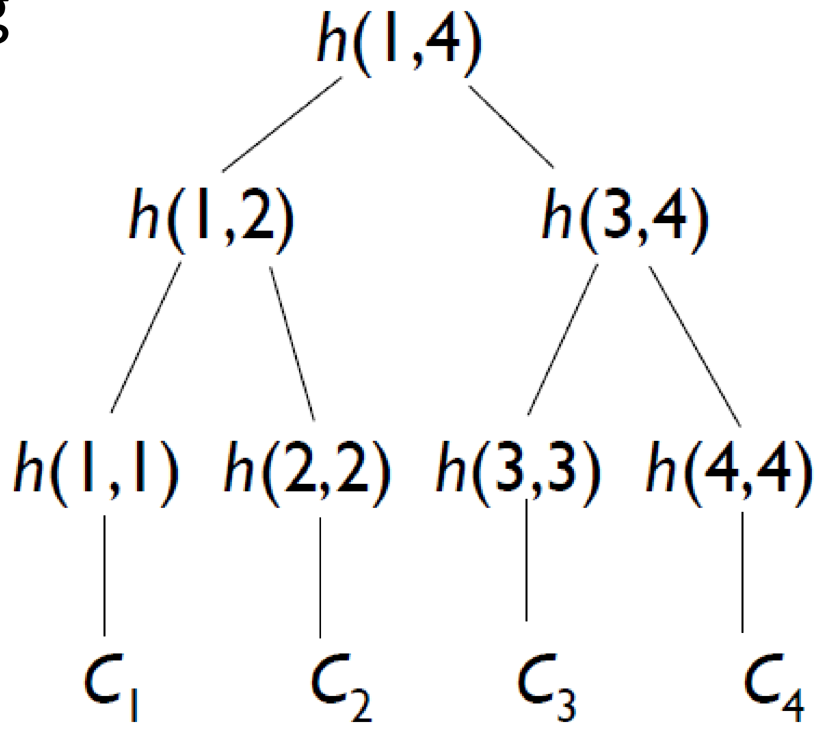
- When Alice has a large number of certificates that require CA to respond
  - A naive approach requires CA to sign each response
  - Bob needs to verify the CA's signatures in turn

Low Efficiency!

- Merkle tree can improve efficiency and only requires one signature on the root hash
  - Batch Verification: utilize the root hash to quickly verify the integrity of a batch of certificates

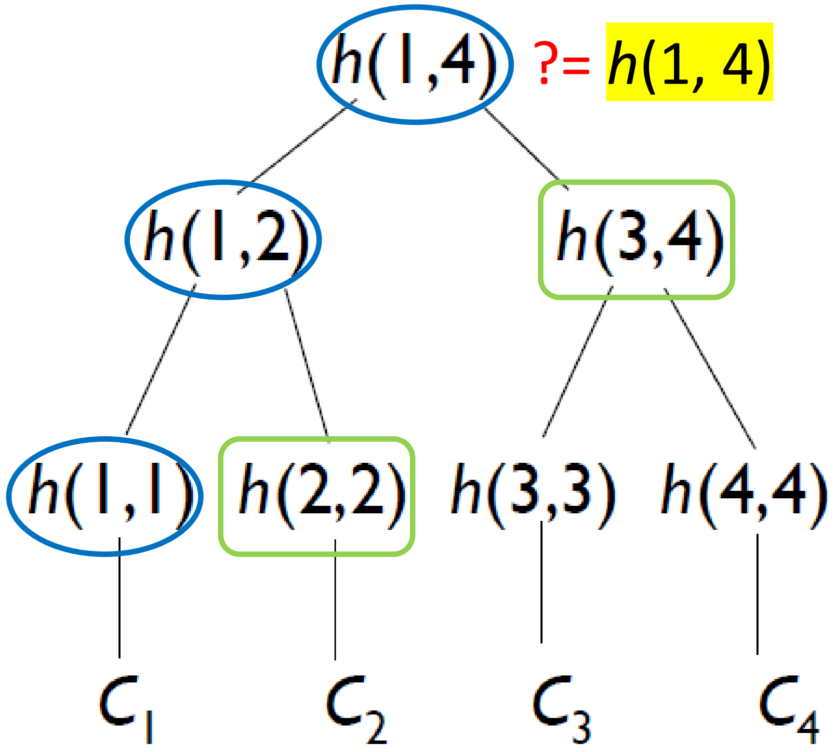
Merkle Tree Scheme

- Construct Merkle hash tree by computing hashes recursively
  - *h* is hash function
  - C_i is certificate *i*
- Root hash (i.e., *h*(1,4)) is published
  - Root hash is signed by CA to ensure the value's integrity



Validation

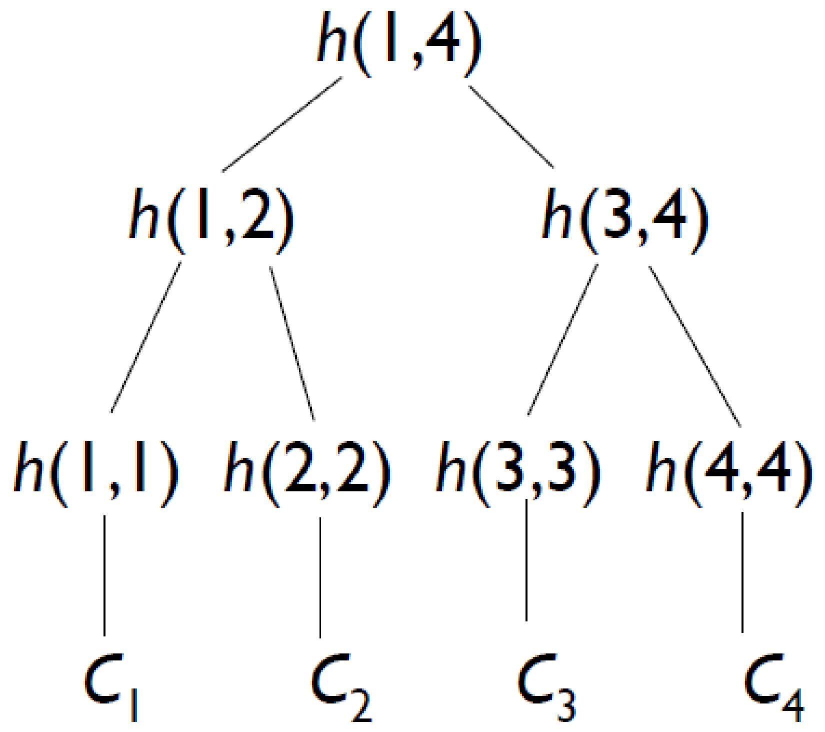
- To validate C_1:
  - Compute $h(1, 1)$
  - Obtain $h(2, 2)$
  - Compute $h(1, 2)$
  - Obtain $h(3, 4)$
  - Compute $h(1,4)$
  - Compare to known $h(1, 4)$
- Need to know siblings of nodes on path from C_1 to the root
  - The proof from CA consists of these hashes (in rectangles on the left)



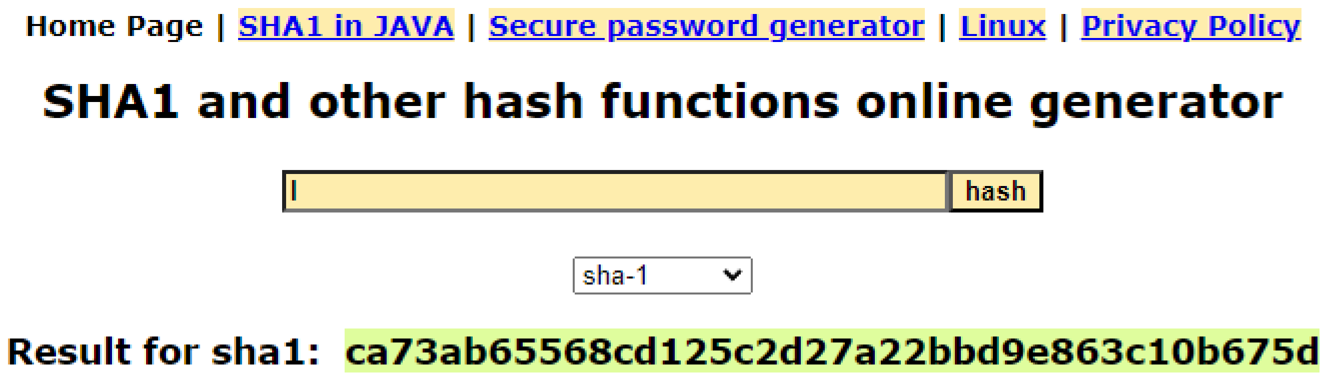
Example

- C_1: I
- C_2: Love
- C_3: E-Payment and
- C_4: Cryptocurrency
- Hash Function: SHA-1
- http://www.sha1-online.com/

- h(1,4)?



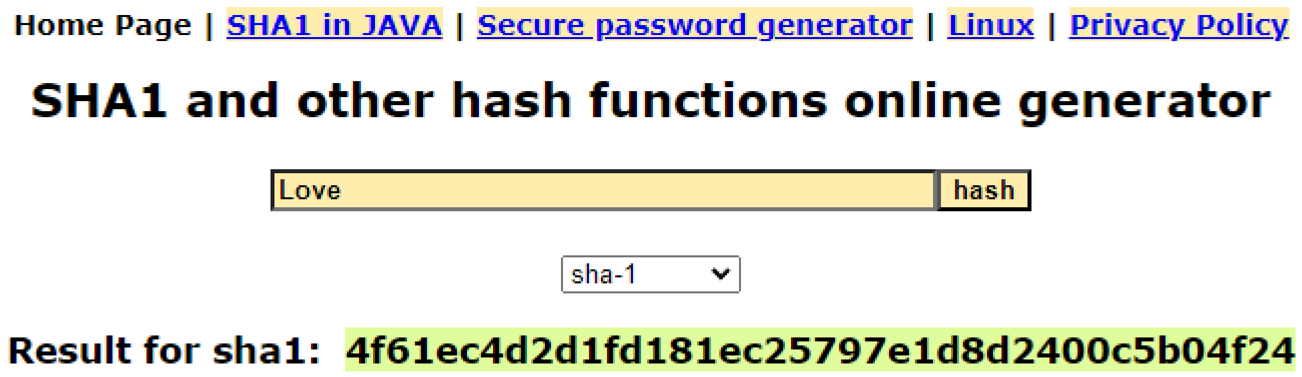
H(1,1) & H(2,2)


Home Page | SHA1 in JAVA | Secure password generator | Linux | Privacy Policy

SHA1 and other hash functions online generator

I

Result for sha1: ca73ab65568cd125c2d27a22bbd9e863c10b675d


Home Page | SHA1 in JAVA | Secure password generator | Linux | Privacy Policy

SHA1 and other hash functions online generator

Love

Result for sha1: 4f61ec4d2d1fd181ec25797e1d8d2400c5b04f24

Example

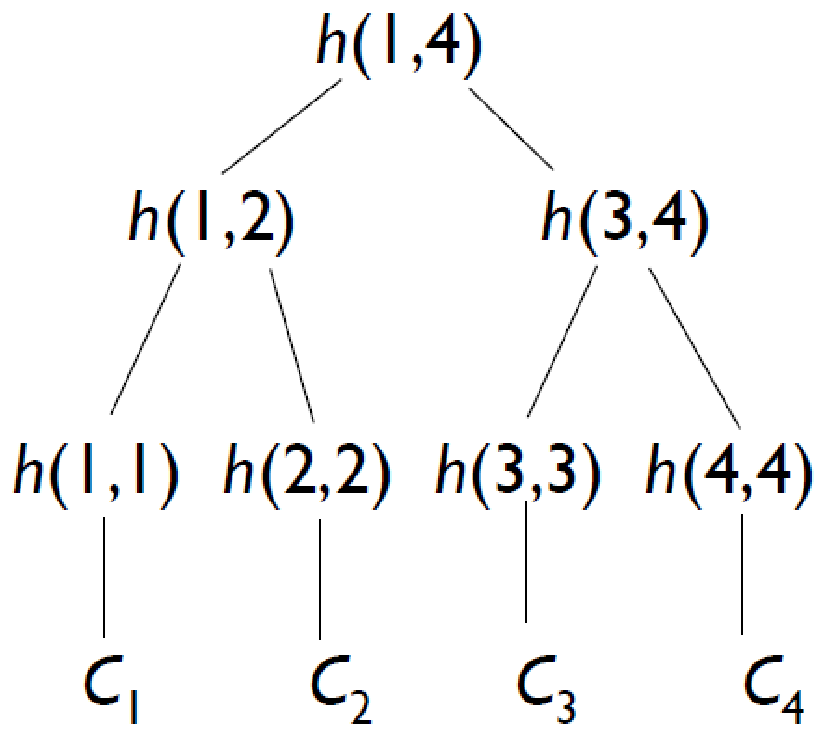
- h(1,4)?
- h(1,1)=ca73ab65568cd125c2d27a22bbd9e863c10b675d
- h(2,2)=4f61ec4d2d1fd181ec25797e1d8d2400c5b04f24
- h(3,3)=ea049bd6063f621ec4d1ce6fa70a3c9382c59364
- h(4,4)=3f9b3fd3271d34c9b292f15cd52d0e3b71499428
- h(1,2)=eb42ab5639880ef199cbf4e86d9ec4fad2891023
- h(3,4)=362121ce0c2a5d3a5a37463fbb0f17eb7edc47c5
- h(1,4)=7039ef1cb6974943aab0c1d79b016c4410377bbe

h(1,2)

=hash(h(1,1)||h(2,2))

=hash(ca73ab65568cd125c2d27a22bbd9e863c10b675d4f61ec4d2d1fd181ec25797e1d8d2400c5b04f24)

=eb42ab5639880ef199cbf4e86d9ec4fad2891023



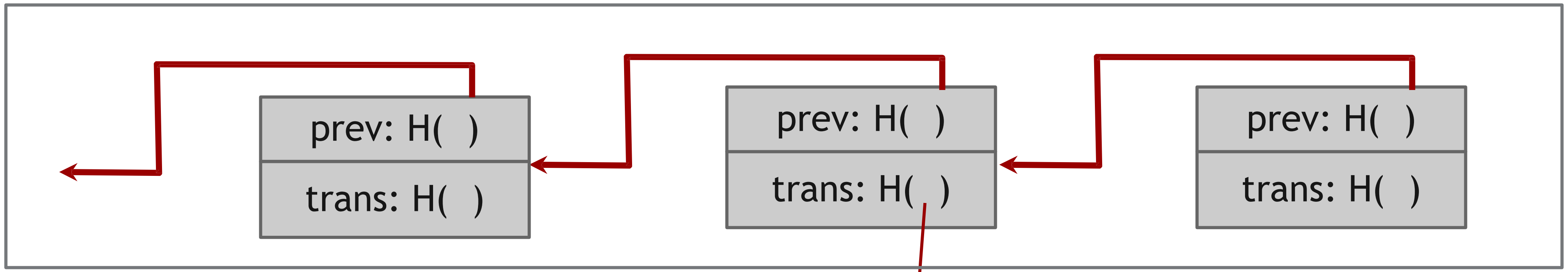
Merkle Tree in Blockchain

- We build a Merkle Tree for many transactions, and store the root in the block header.
  - Can easily check whether transactions stored in a block have been modified
  - Enable efficient membership proofs for transactions in a block, which are necessary for Simple Payment Verification (SPV) nodes that only store block headers and not block contents.

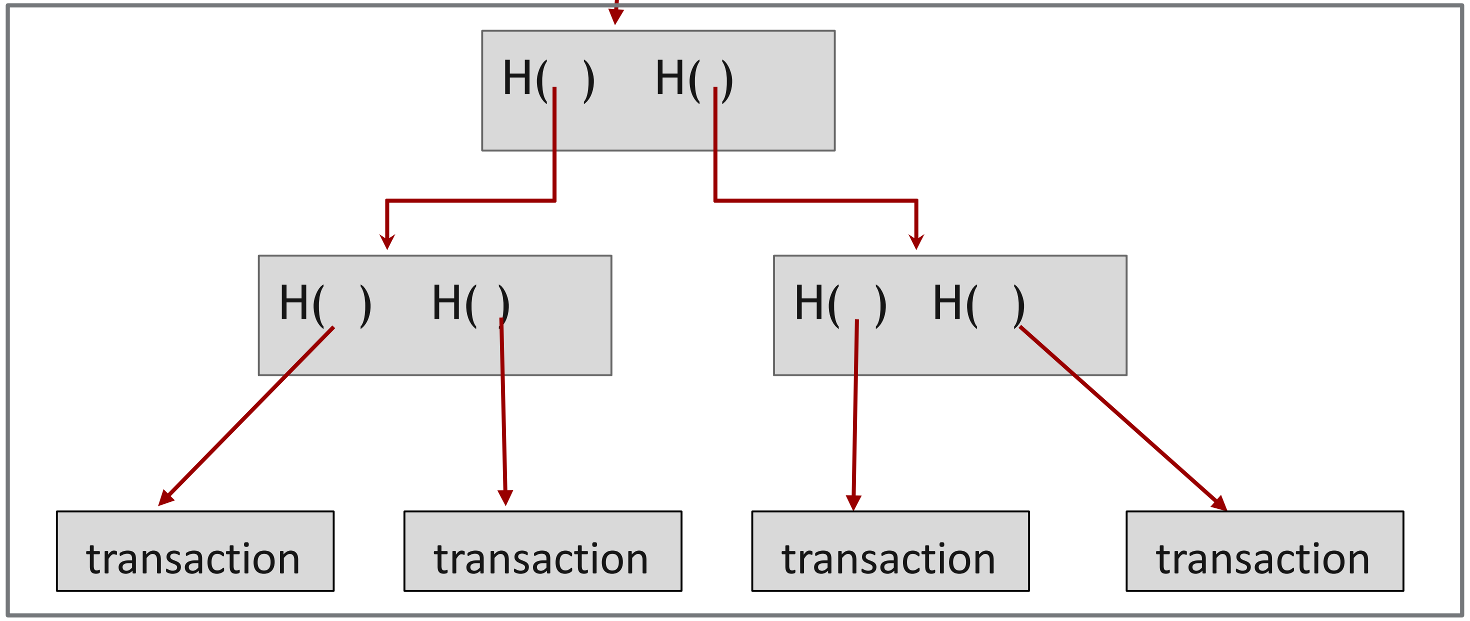
- Just concatenate all TXs and store the hash in header?
  - Not efficient for membership proof – O(n) complexity
  - Efficient with Merkle tree – O(log(n)) complexity

Merkle Hash Tree in Blockchain

Hash chain of blocks



Hash tree (Merkle tree) of transactions in each block



Outline

- Merkle Tree
- Elliptic Curve Cryptography (ECC)