Lecture #11: Merkle and Merkle-Patricia Trees

COMS 4995-001: The Science of Blockchains

URL: https://timroughgarden.org/s25/

Tim Roughgarden

Goals for Lecture #11

1. Querying a commitment.
   – e.g., verifiable reporting of whether a given tx included in a block
2. Merkle trees (e.g., for transactions).
   – commitment to a list, easy to reveal any given list element
3. Merkle-Patricia trees (e.g., for blockchain state).
   – as used in Ethereum
4. Merkle proofs for state transitions/“statelessness.”
   – e.g., verifiable reporting of the result of simulating a transaction

Recap from Lecture #10

1. Can use a collision-resistant hash function (like SHA-256) to give short names to objects (unique for all practical purposes).

2. Practitioners treat cryptographic hash functions like SHA-256 as random functions even though they are not.
   - more predictable than random functions (e.g., length-extension attack), but doesn’t necessarily contradict collision-resistance

3. Output of a cryptographic hash function can be viewed as a binding (and hiding) commitment to a particular input.
   - infeasible to find a second input that would yield the same commitment

Revealing a Committed Value

Example: in Tendermint, validator hears about a blockhash h(B) before hearing the block B.

- e.g., as the predecessor in the current leader's proposal
- e.g., in "up-to-date" messages from other validators

Revealing a Committed Value

Example: in Tendermint, validator hears about a blockhash h(B) before hearing the block B.

- e.g., as the predecessor in the current leader's proposal
- e.g., in "up-to-date" messages from other validators

Solution: ask others for block, check that hash matches h(B).

Revealing a Committed Value

Example: in Tendermint, validator hears about a blockhash h(B) before hearing the block B.

- e.g., as the predecessor in the current leader's proposal
- e.g., in "up-to-date" messages from other validators

Solution: ask others for block, check that hash matches h(B).

Properties:

- no false negatives: if sent actual block B, hash will match h(B)

Revealing a Committed Value

Example: in Tendermint, validator hears about a blockhash h(B) before hearing the block B.

- e.g., as the predecessor in the current leader’s proposal
- e.g., in “up-to-date” messages from other validators

Solution: ask others for block, check that hash matches h(B).

Properties:

- no false negatives: if sent actual block B, hash will match h(B)
- no false positives (unless find collision of h): if sent a block B’ $\neq$ B, hash h(B’) won’t match h(B)

Partial Reveal of a Committed Value

Easier (?) problem: for blockhash h(B) and tx t, is $t \in B$?

- e.g., wallet software tracking blockhashes but not blocks

Partial Reveal of a Committed Value

Easier (?) problem: for blockhash h(B) and tx t, is $t \in B$?

– e.g., wallet software tracking blockhashes but not blocks

Simple solution: ask for B, check that hash = h(B), check if $t \in B$.

– as before, no false negatives or (by collision-resistance) false positives

– might be downloading 4000 transactions to check for one

Partial Reveal of a Committed Value

Easier (?) problem: for blockhash h(B) and tx t, is $t \in B$?

- e.g., wallet software tracking blockhashes but not blocks

Simple solution: ask for B, check that hash = h(B), check if $t \in B$.

- as before, no false negatives or (by collision-resistance) false positives
- might be downloading 4000 transactions to check for one

Question: how to check with less communication?

Partial Reveal of a Committed Value

Easier (?) problem: for blockhash h(B) and tx t, is $t \in B$?

- e.g., wallet software tracking blockhashes but not blocks

Simple solution: ask for B, check that hash = h(B), check if $t \in B$.

- as before, no false negatives or (by collision-resistance) false positives
- might be downloading 4000 transactions to check for one

Question: how to check with less communication?

Idea: add more structure to the commitment.

- will use hierarchy of hashes, not just a single hash

Merkle Trees

Example application: committing to txs in each Bitcoin block.

Assume: n pieces of data $x_1, x_2, \ldots, x_n$, n a power of 2. h = hash fn.

Merkle Trees

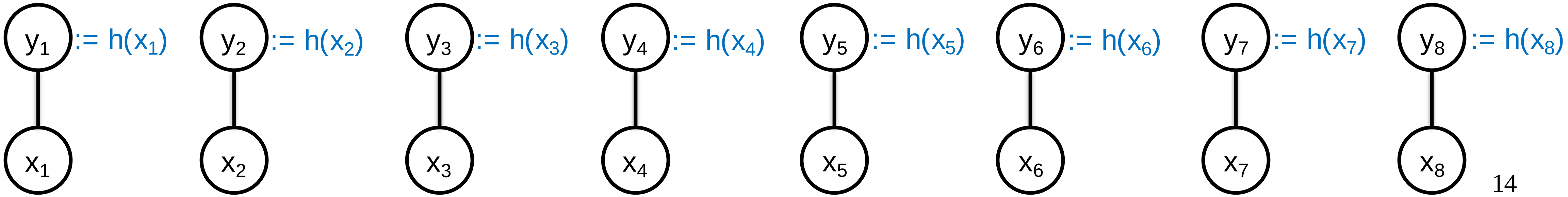
Example application: committing to txs in each Bitcoin block.

Assume: n pieces of data $x_1, x_2, \ldots, x_n$, n a power of 2. h = hash fn.

Merkle Trees

Example application: committing to txs in each Bitcoin block.

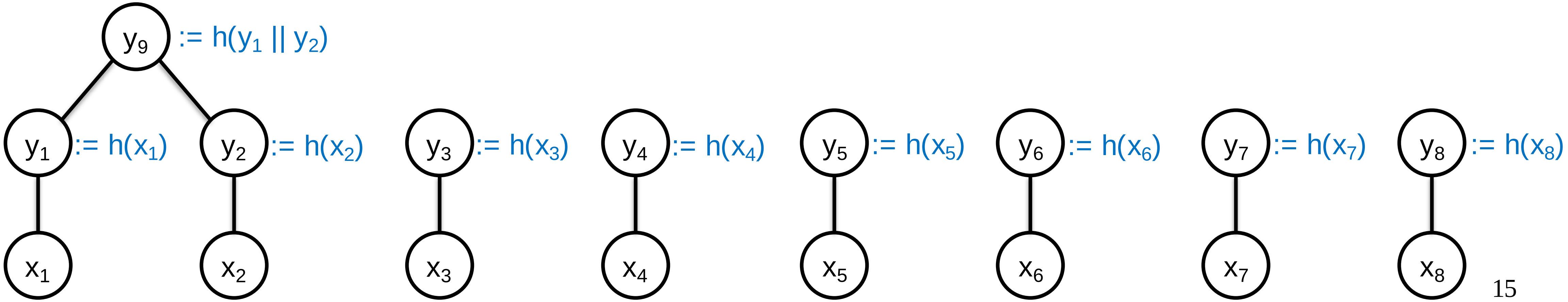
Assume: n pieces of data $x_1,x_2,\ldots,x_n$, n a power of 2. h = hash fn.



Merkle Trees

Example application: committing to txs in each Bitcoin block.

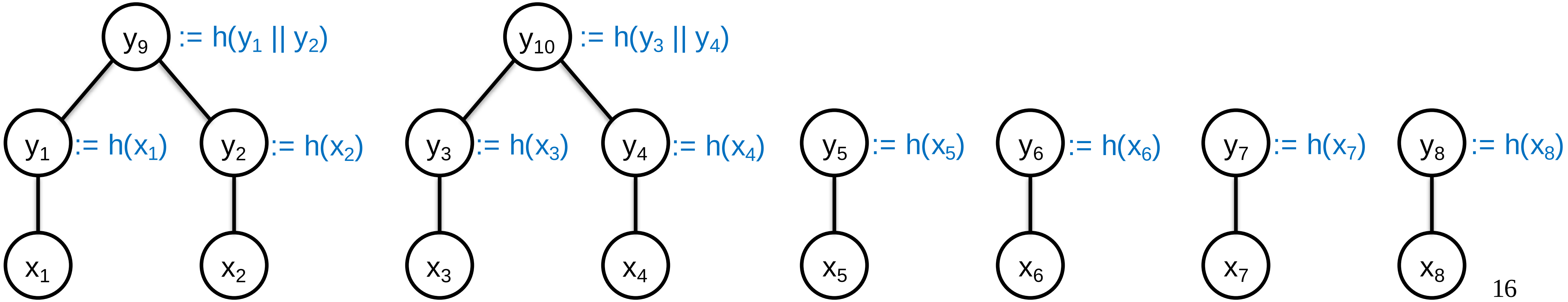
Assume: n pieces of data $x_1,x_2,\ldots,x_n$, n a power of 2. h = hash fn.



Merkle Trees

Example application: committing to txs in each Bitcoin block.

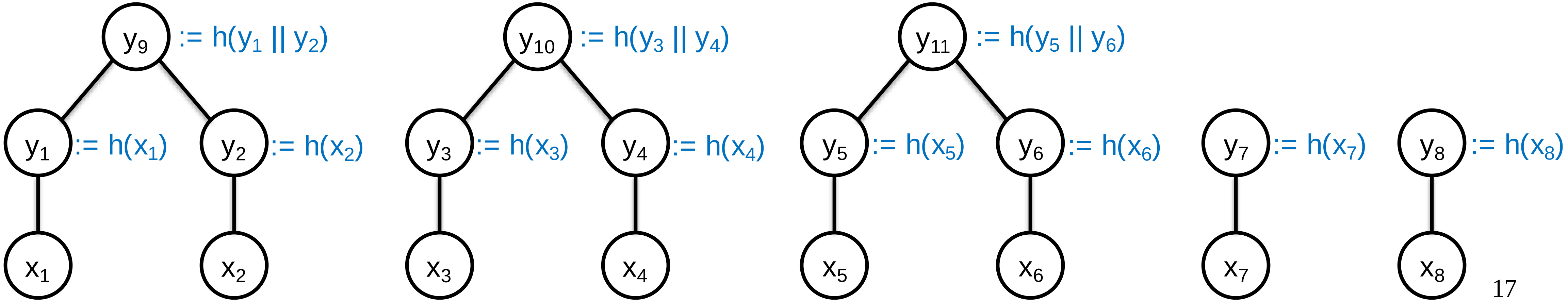
Assume: n pieces of data $x_1, x_2, \ldots, x_n$, n a power of 2. h = hash fn.



Merkle Trees

Example application: committing to txs in each Bitcoin block.

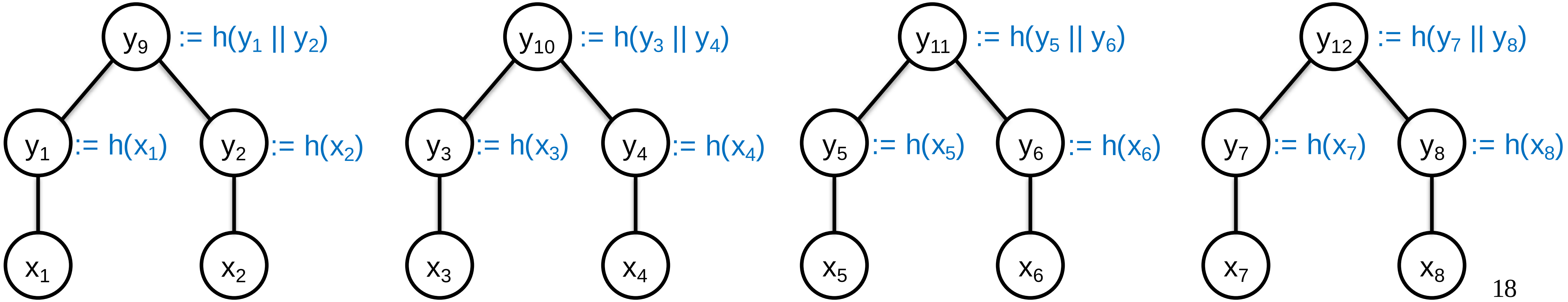
Assume: n pieces of data $x_1, x_2, \ldots, x_n$, n a power of 2. h = hash fn.



Merkle Trees

Example application: committing to txs in each Bitcoin block.

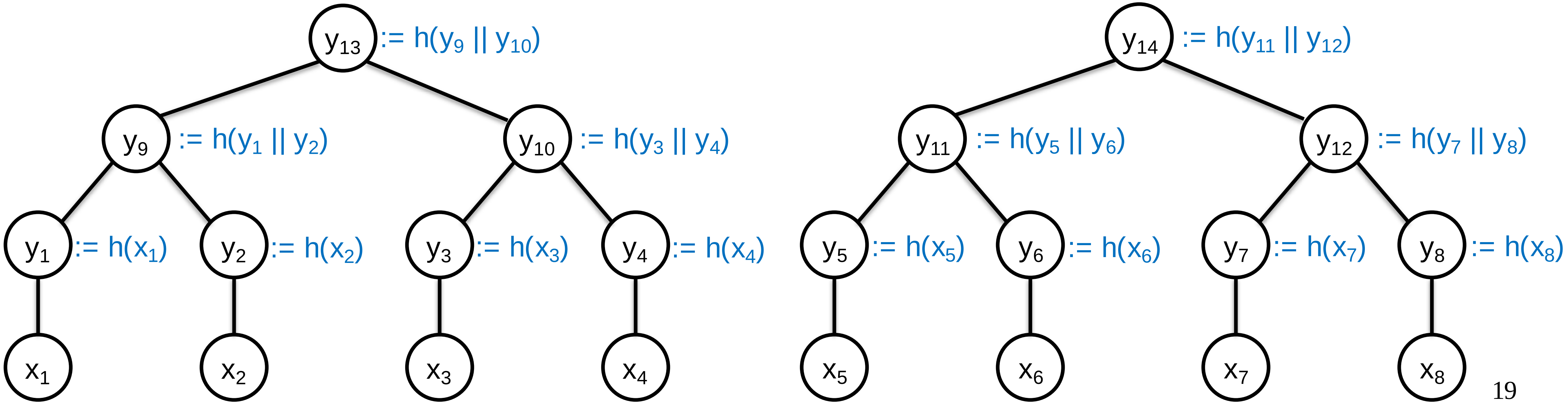
Assume: n pieces of data $x_1, x_2, \ldots, x_n$, n a power of 2. h = hash fn.



Merkle Trees

Example application: committing to txs in each Bitcoin block.

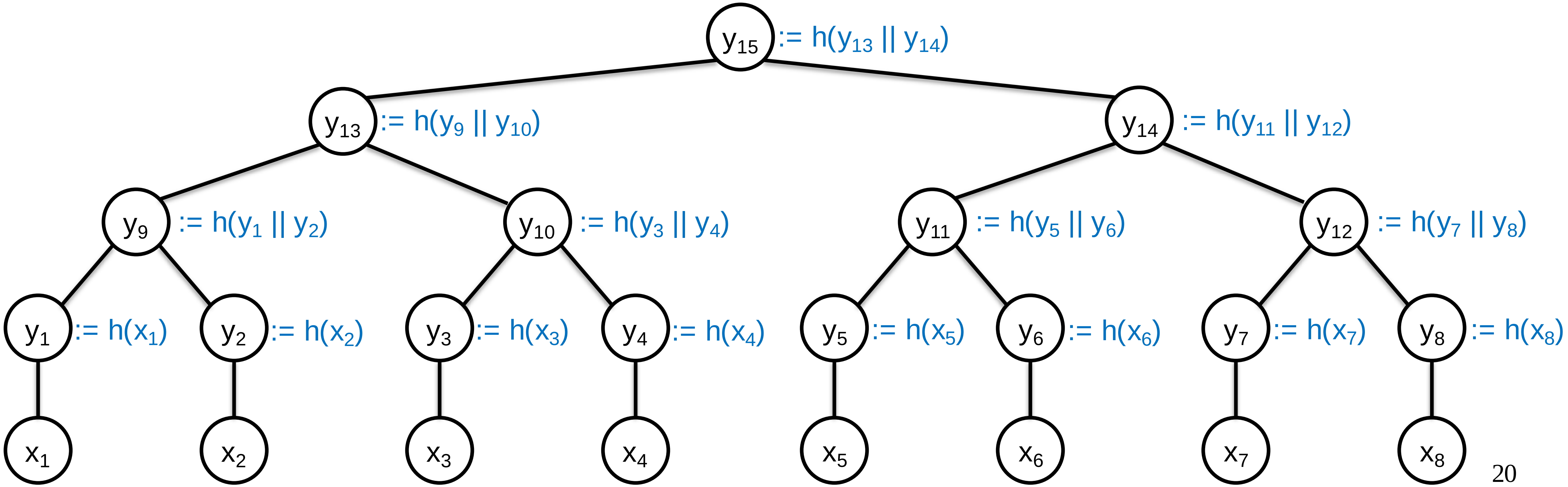
Assume: n pieces of data $x_1,x_2,\ldots,x_n$, n a power of 2. h = hash fn.



Merkle Trees

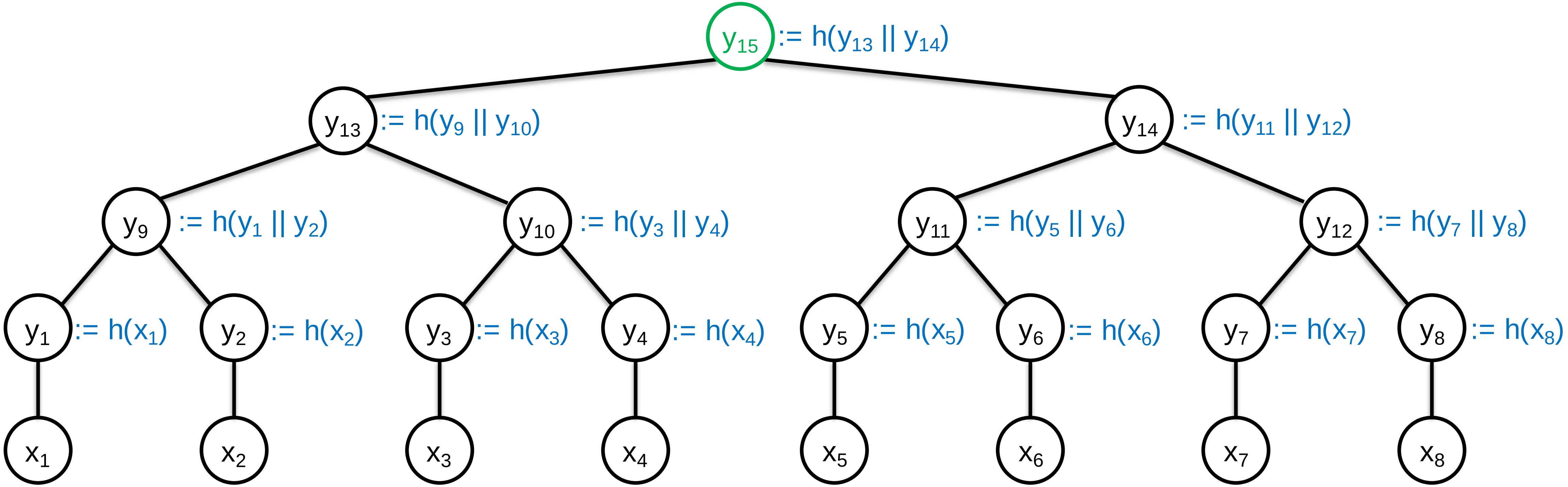
Example application: committing to txs in each Bitcoin block.

Assume: n pieces of data $x_1,x_2,\ldots,x_n$, n a power of 2. h = hash fn.



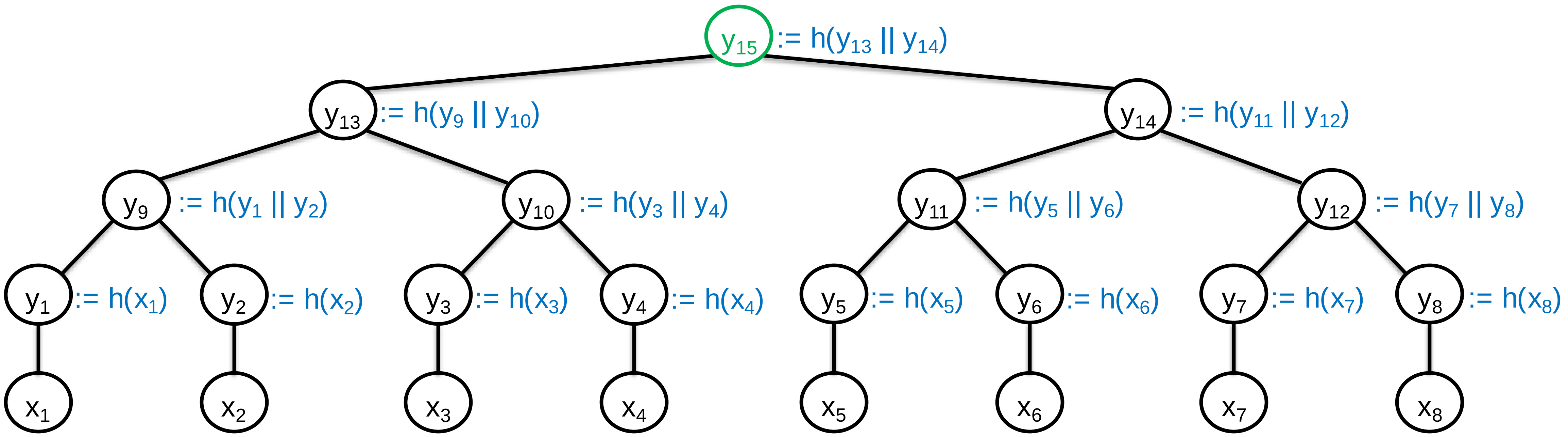
Merkle Trees

Assume: n pieces of data $x_1,x_2,\ldots,x_n$, n a power of 2. h = hash fn.

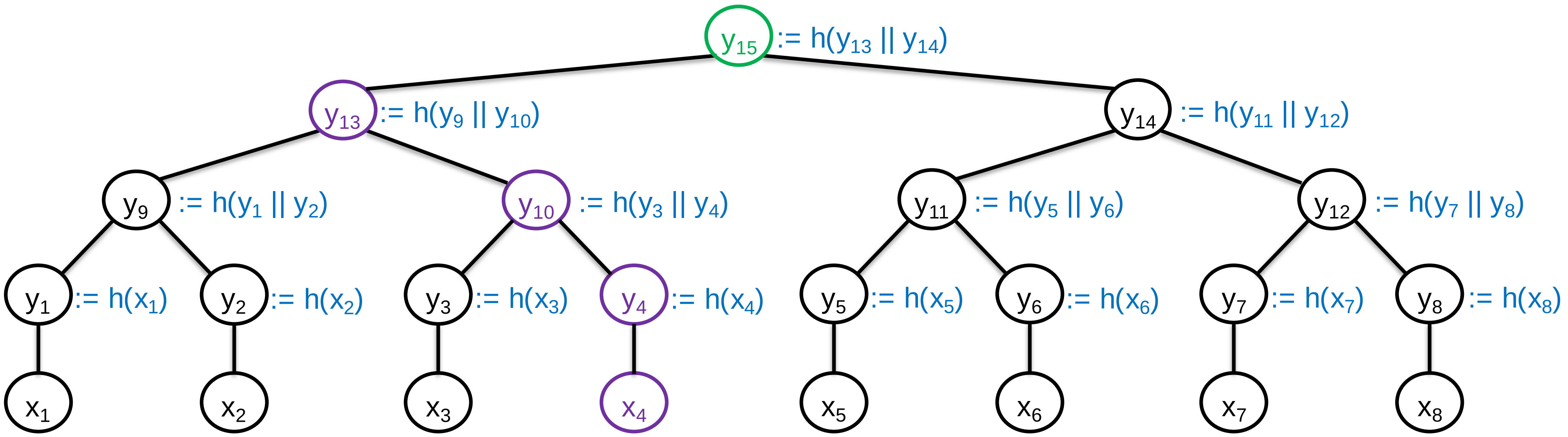


Final commitment: the Merkle root (i.e., y_{2n-1}).

Merkle Proofs

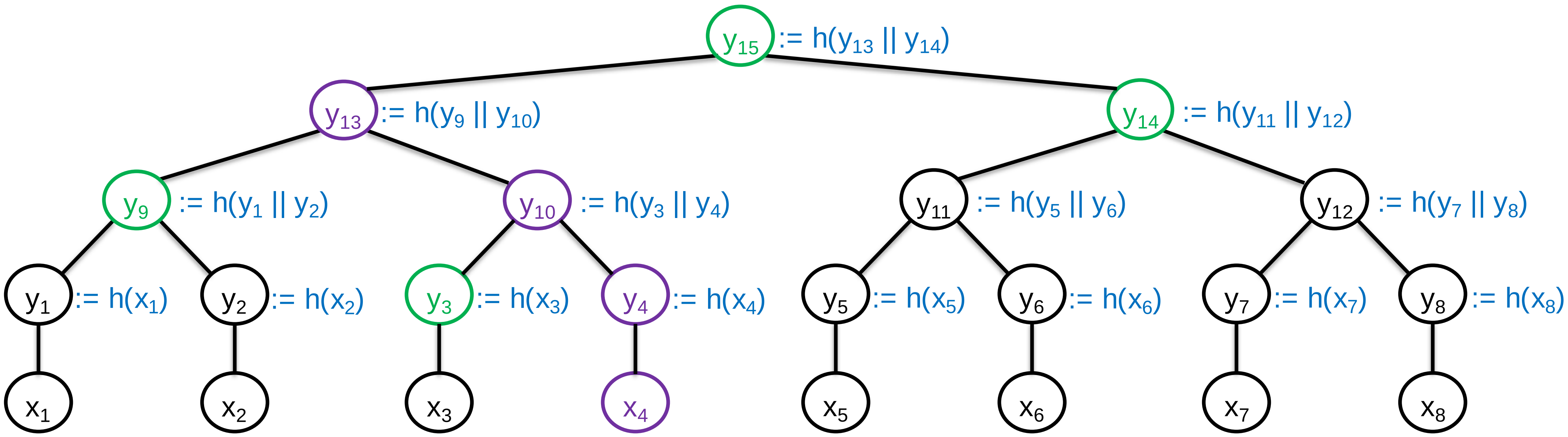


Merkle Proofs



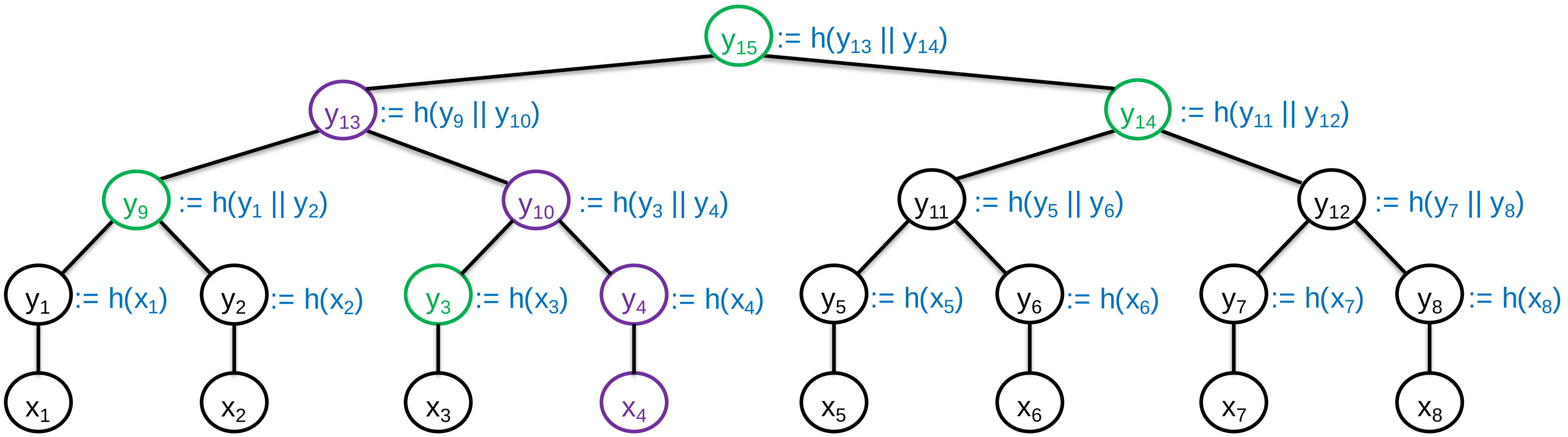
To prove that $t = x_4$:

Merkle Proofs



To prove that $t = x_4$: exhibit siblings along root-leaf path: y_3, y_9, y_{14}.

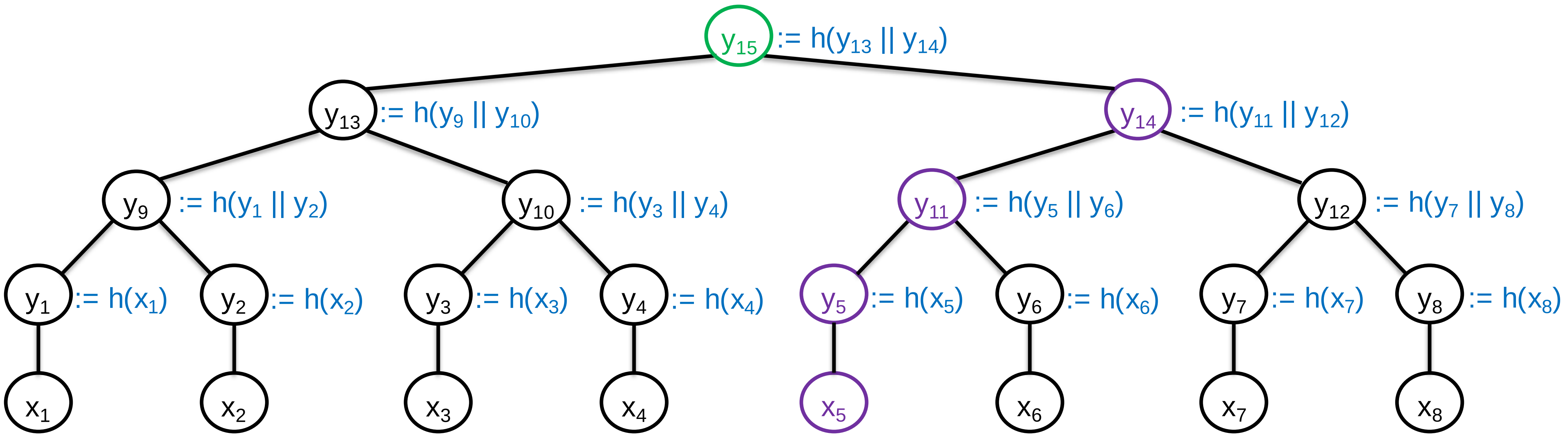
Merkle Proofs



To prove that $t = x_4$: exhibit siblings along root-leaf path: y_3, y_9, y_{14}.

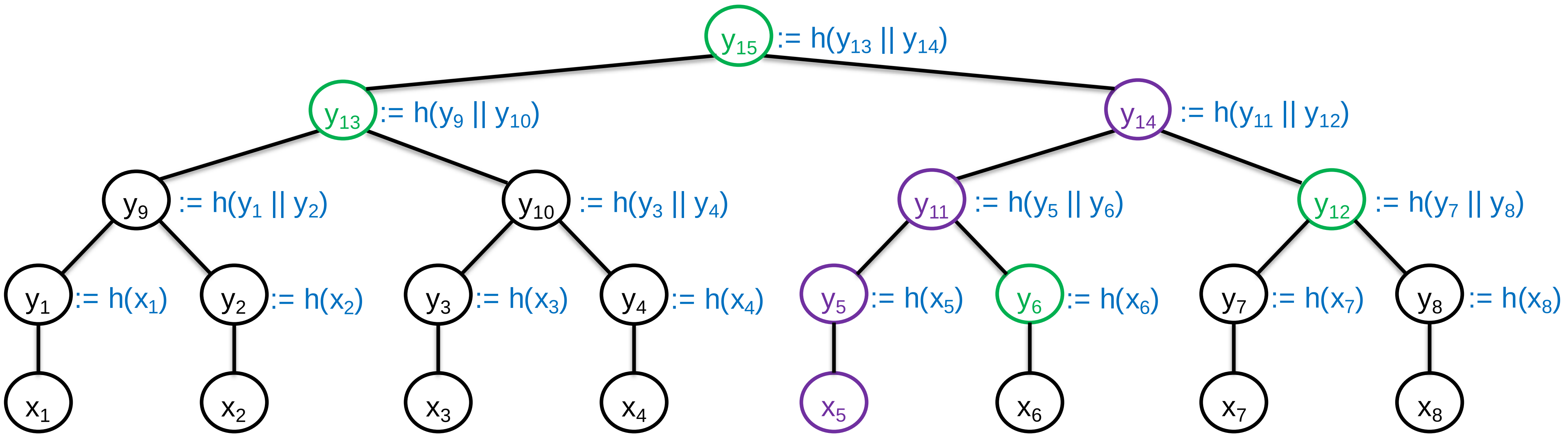
To verify: compute $z_4=h(t)$, $z_{10}=h(y_3 \| z_4)$, $z_{13}=h(y_9 \| z_{10})$, $z_{15}=h(z_{13} \| y_{14})$, check that $z_{15}=y_{15}$.

Merkle Proofs



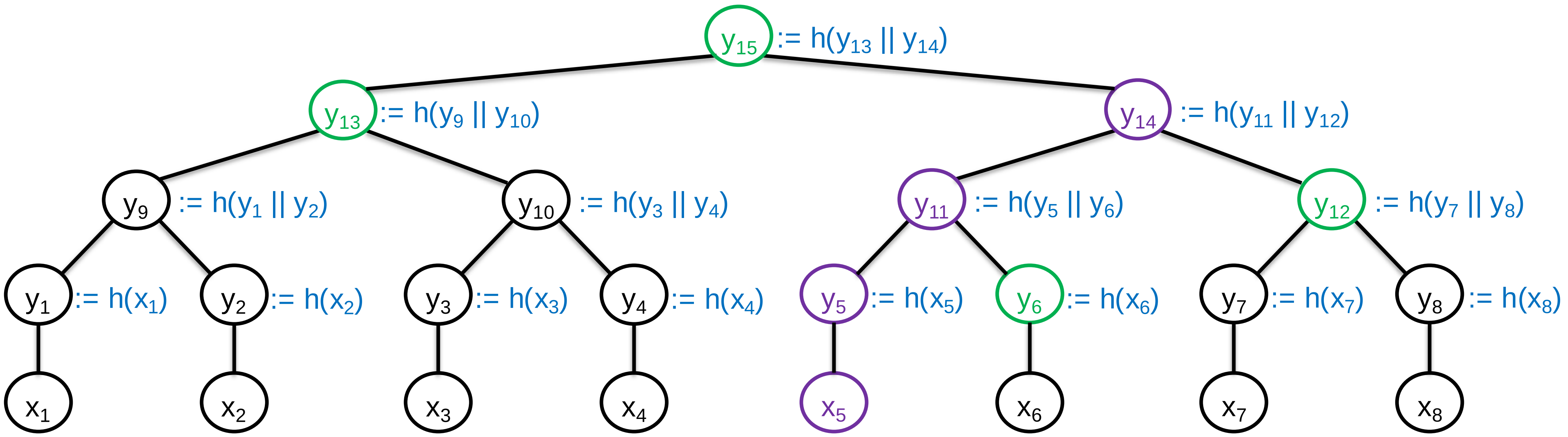
To prove that $t = x_5$:

Merkle Proofs



To prove that $t = x_5$: exhibit siblings along root-leaf path: y_6, y_{12}, y_{13}.

Merkle Proofs



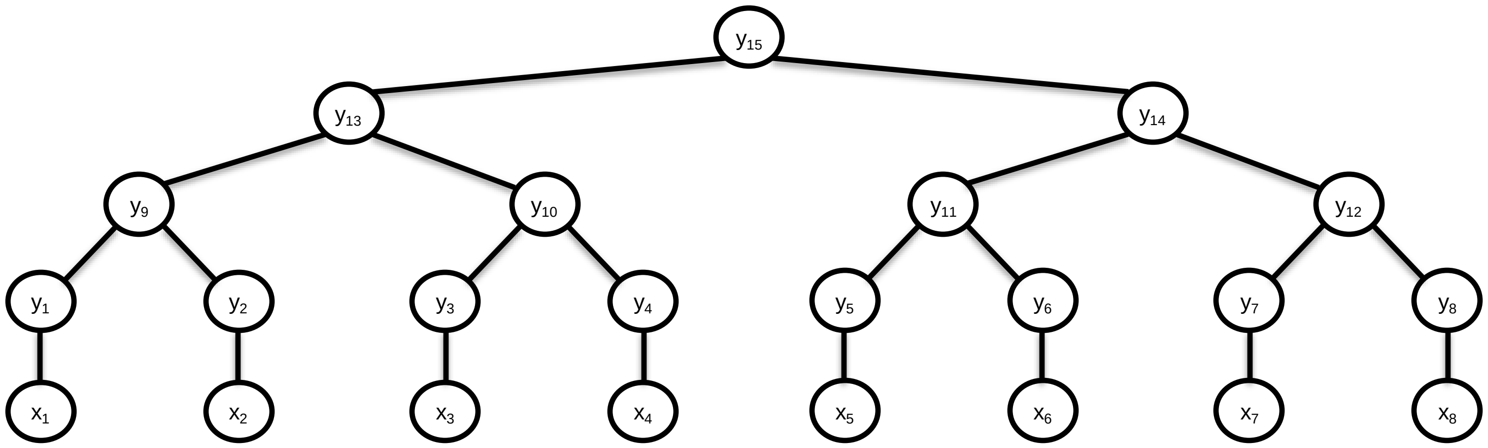
To prove that $t = x_5$: exhibit siblings along root-leaf path: y_6, y_{12}, y_{13}.

To verify: compute $z_5=h(t)$, $z_{11}=h(z_5 \,||\, y_6)$, $z_{14}=h(z_{11} \,||\, y_{12})$,
$z_{15}=h(y_{13} \,||\, z_{14})$, check that $z_{15}=y_{15}$.

Merkle Proofs: Time and Space

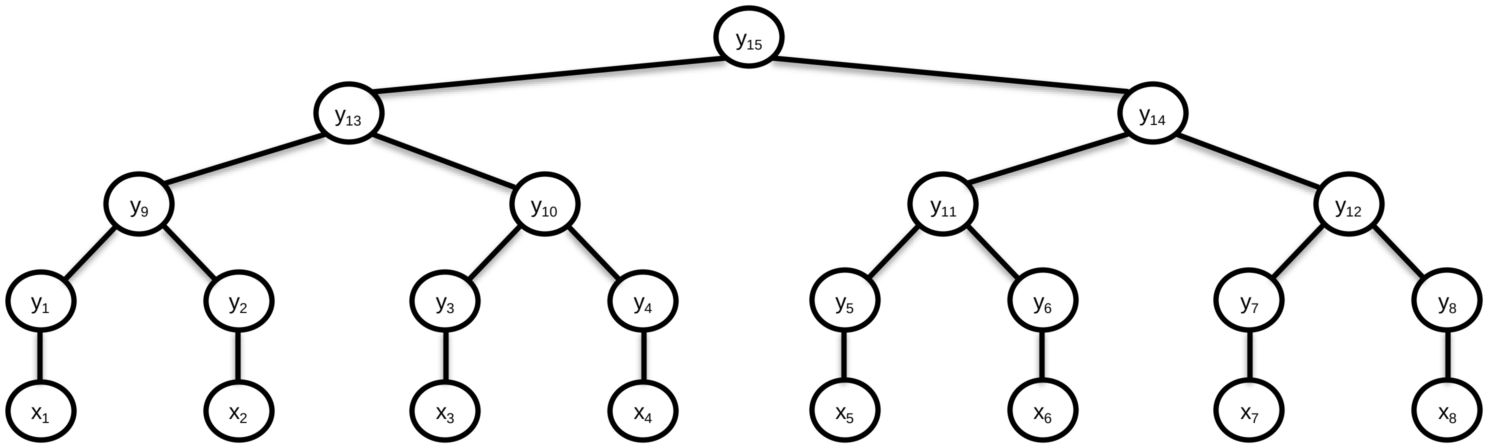
In general: O(n) space and hashes to construct Merkle tree.

- commitment size = O(1) [256-bit Merkle root]
- Merkle proofs: O(log n) space, O(log n) time to construct, O(log n) hashes to verify



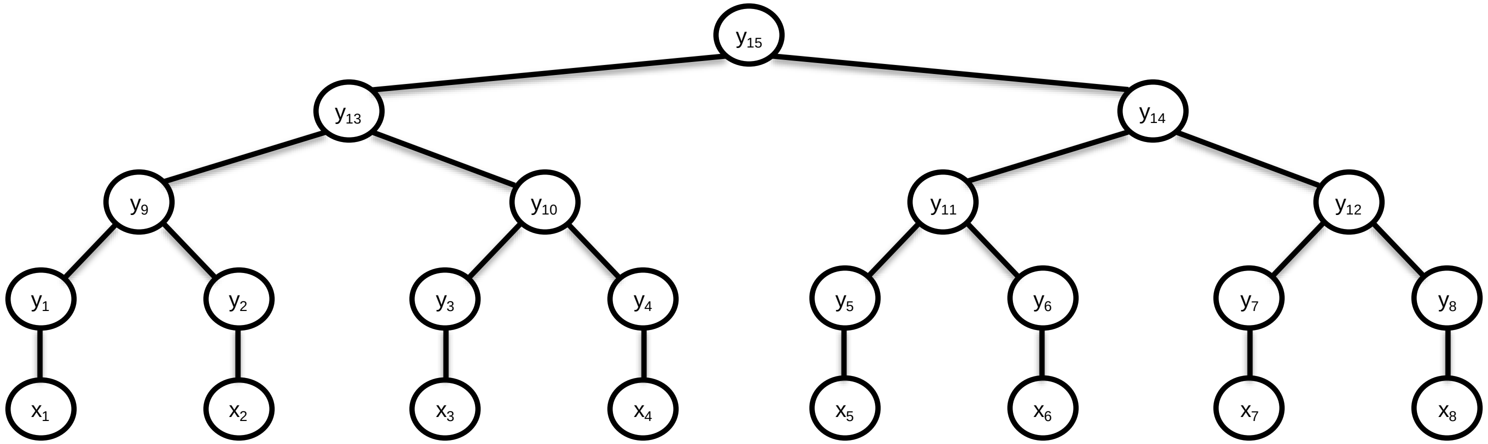
Merkle Proofs: Correctness

- no false negatives: if $t \in \{x_1, x_2, \ldots, x_n\}$, can construct a Merkle proof guaranteed to pass the test



Merkle Proofs: Correctness

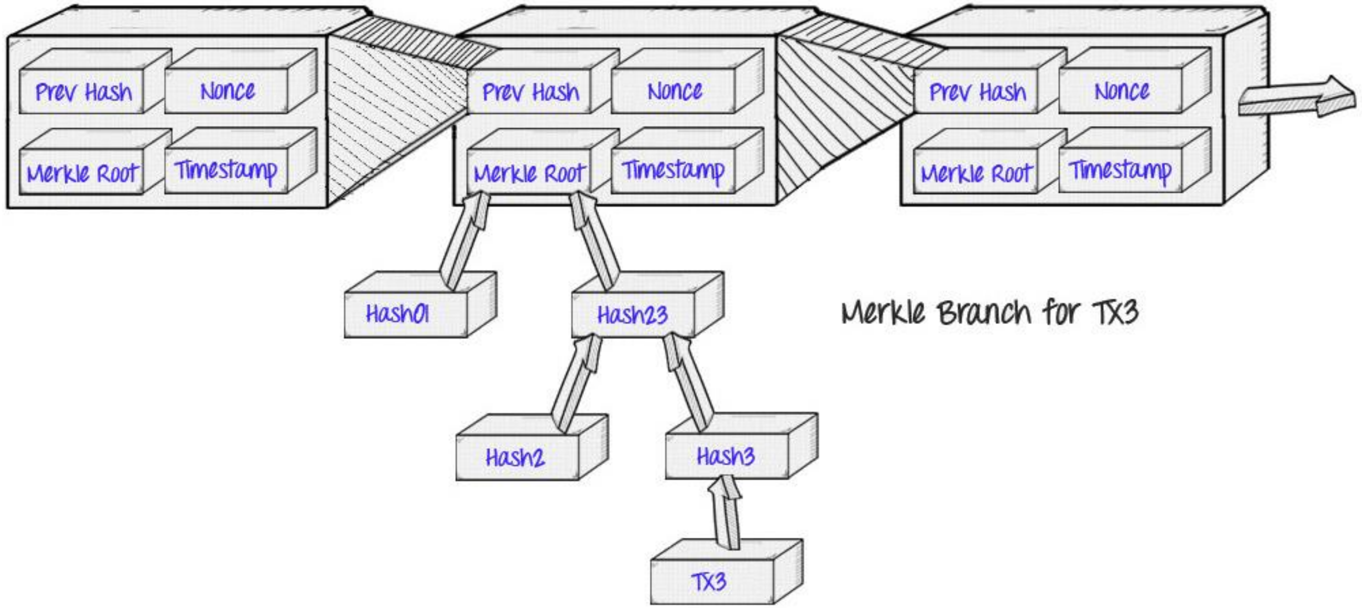
- no false negatives: if $t \in \{x_1, x_2, \dots, x_n\}$, can construct a Merkle proof guaranteed to pass the test
- no false positives (unless find collision of h): if $t \notin \{x_1, x_2, \dots, x_n\}$, infeasible to find false Merkle proof that passes the test
  – somewhere along path, need to find false sibling hash giving the correct parent hash



Merkle Trees in Bitcoin

In Bitcoin: each block includes Merkle root of its txs (as metadata).

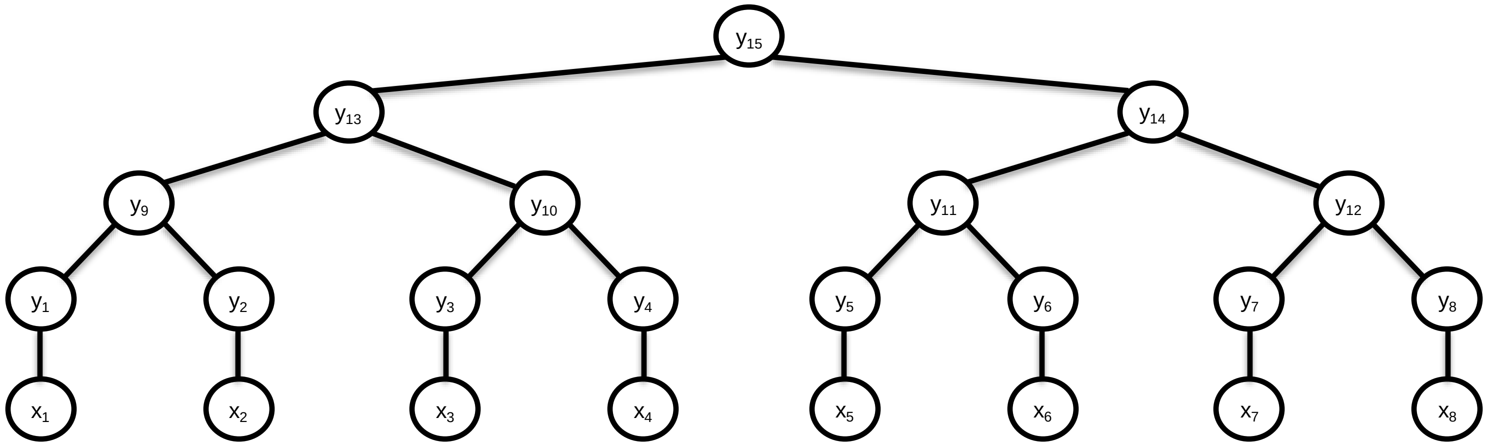
- block name = hash of its metadata ("block header"), not of entire block
- block name depends on each of its txs via Merkle root in block header



From "Merkling in Ethereum" by Vitalik Buterin https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/

Proof of Non-Membership

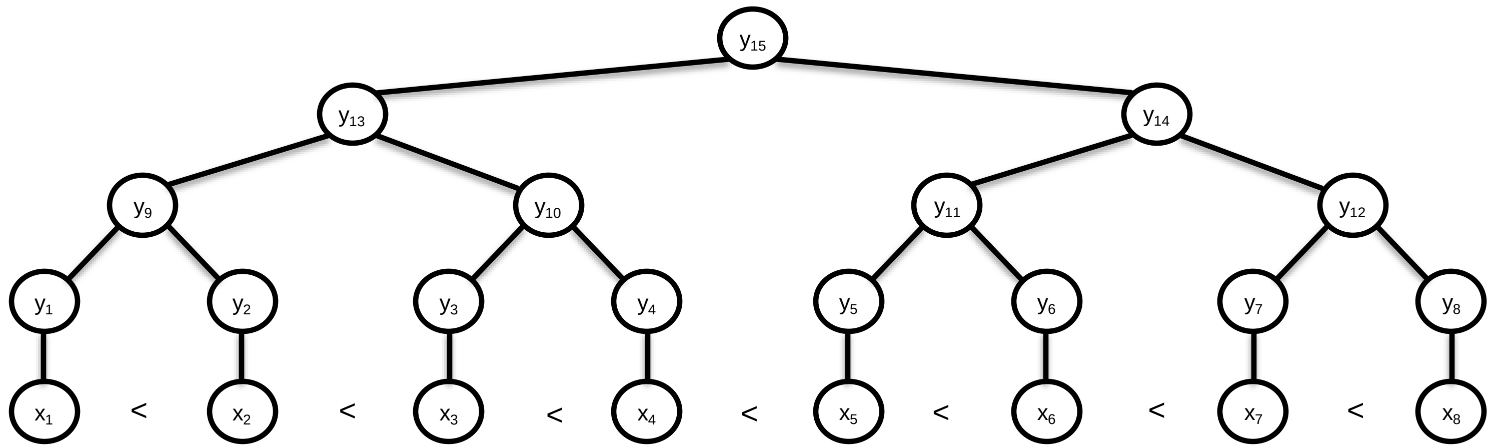
Issue: how to verify assertion that $t \notin \{x_1, x_2, \dots, x_n\}$?



Proof of Non-Membership

Issue: how to verify assertion that $t \notin \{x_1, x_2, \ldots, x_n\}$?

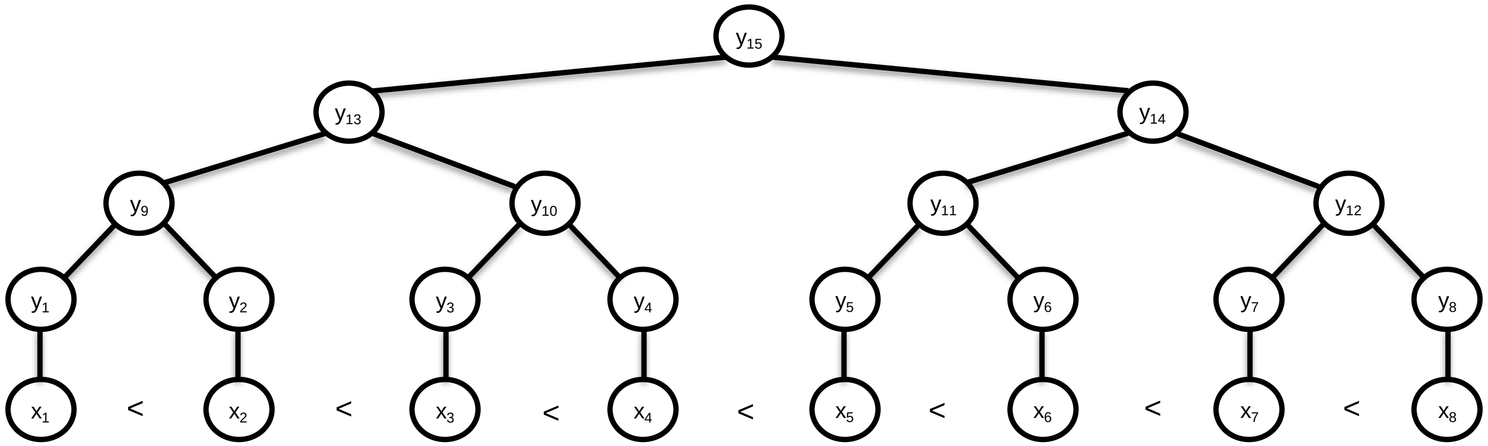
- modified construction: arrange leaves in sorted order



Proof of Non-Membership

Issue: how to verify assertion that $t \notin \{x_1, x_2, \ldots, x_n\}$?

- modified construction: arrange leaves in sorted order
- to prove non-membership:
  - let i be such that $x_i < t < x_{i+1}$
  - prove membership of x_i (at position i) and x_{i+1} (at position i+1)



(Modified) Merkle-Patricia Trees

Application: committing to states of all accounts in Ethereum.

- example: want short proof of some account's ETH balance

(Modified) Merkle-Patricia Trees

Application: committing to states of all accounts in Ethereum.

– example: want short proof of some account's ETH balance

Note: accounts naturally indexed by ID (rather than position).

(Modified) Merkle-Patricia Trees

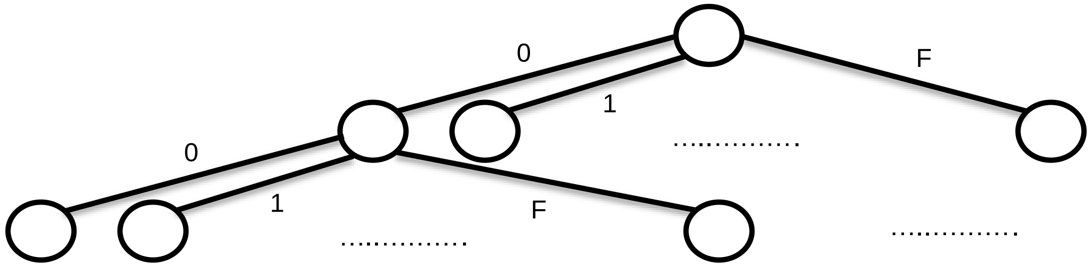
Application: committing to states of all accounts in Ethereum.

– example: want short proof of some account's ETH balance

Note: accounts naturally indexed by ID (rather than position).

First cut: radix tree, branching factor = 16 (hexadecimal), 40 levels.

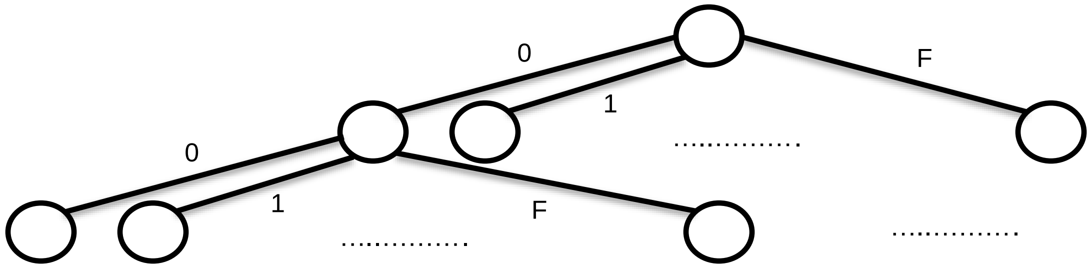
– leaves correspond to accounts (labels on root-leaf path ⬄ account ID)



(Modified) Merkle-Patricia Trees

First cut: radix tree, branching factor = 16 (hexadecimal), 40 levels.

– leaves correspond to accounts (labels on root-leaf path ⇔ account ID)

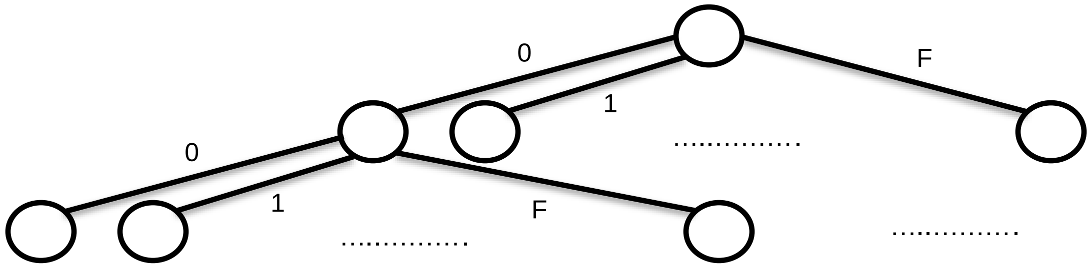


Optimizations:

(Modified) Merkle-Patricia Trees

First cut: radix tree, branching factor = 16 (hexadecimal), 40 levels.

– leaves correspond to accounts (labels on root-leaf path ⇔ account ID)



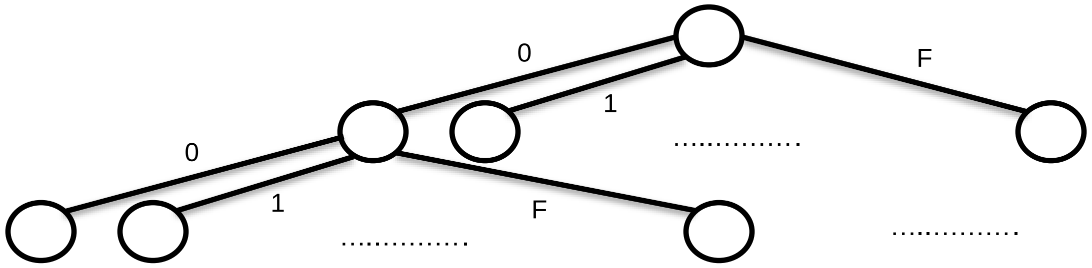
Optimizations:

– introduce nodes and edges to tree only as needed

(Modified) Merkle-Patricia Trees

First cut: radix tree, branching factor = 16 (hexadecimal), 40 levels.

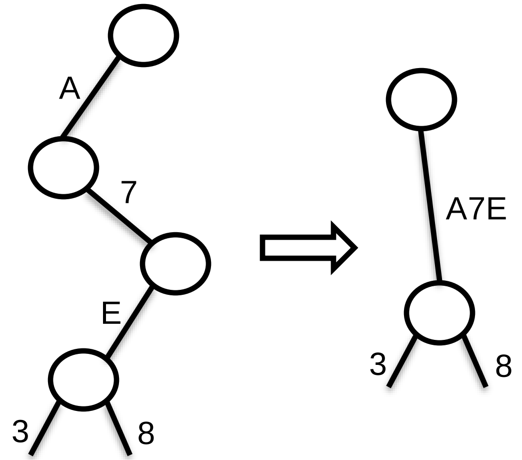
– leaves correspond to accounts (labels on root-leaf path ⇔ account ID)



Optimizations:

– introduce nodes and edges to tree only as needed

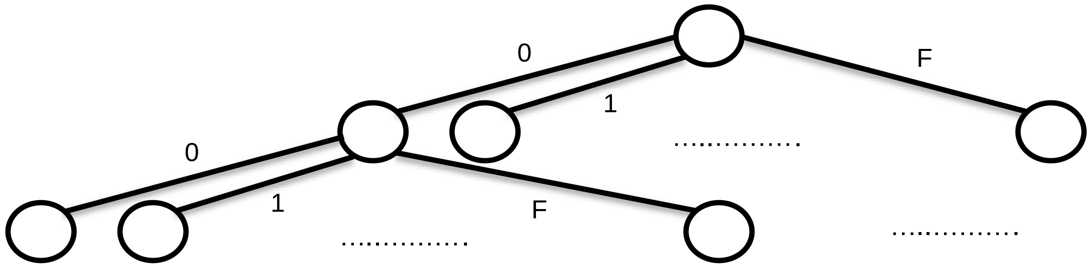
– contract paths of nodes with only one child



(Modified) Merkle-Patricia Trees

First cut: radix tree, branching factor = 16 (hexadecimal), 40 levels.

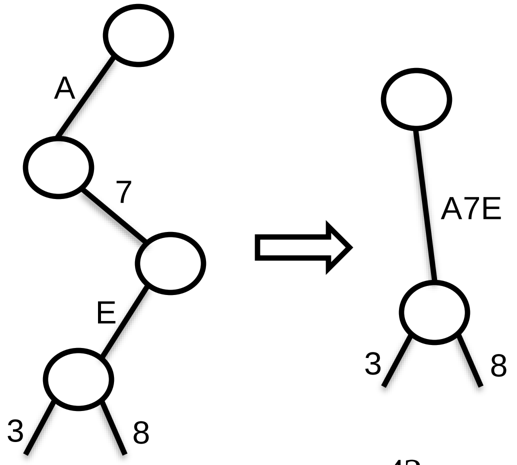
– leaves correspond to accounts (labels on root-leaf path ⇔ account ID)



Optimizations:

– introduce nodes and edges to tree only as needed

– contract paths of nodes with only one child



Merkle proofs: must supply all (≤ 15) sibling hashes.

Merkle-Patricia Trees in Ethereum

Primary use: state of Ethereum blockchain (leaves = accounts).

- even with optimizations, size = 100s of GBs (at least)
- every block includes commitment to "state root" (post-execution)

Merkle-Patricia Trees in Ethereum

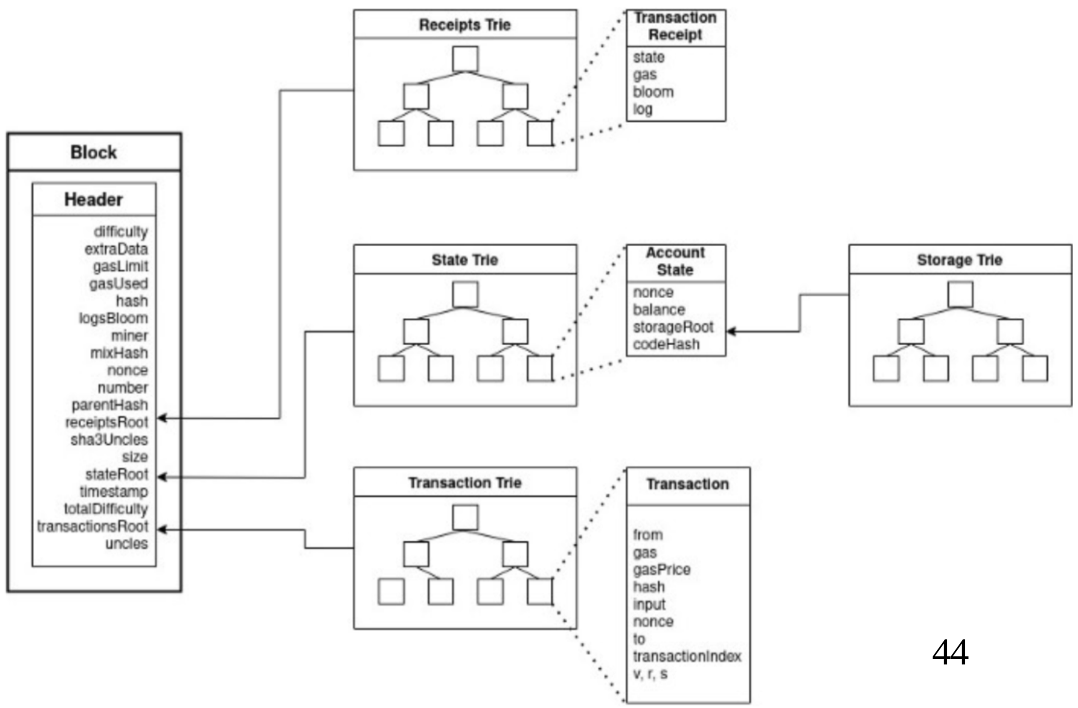
Primary use: state of Ethereum blockchain (leaves = accounts).

– even with optimizations, size = 100s of GBs (at least)

– every block includes commitment to “state root” (post-execution)

Additional uses:

- storage in each account
  – leaves = memory locations
- transactions in each block
  – leaves = transactions
- tx receipts in each block



Proving a State Transition

Basic query to MPT: current balance of my account?

– answer by supplying Merkle proof w.r.t. current state tree

Proving a State Transition

Basic query to MPT: current balance of my account?

- answer by supplying Merkle proof w.r.t. current state tree

Advanced query: how would balance change after executing tx t?

Proving a State Transition

Basic query to MPT: current balance of my account?

– answer by supplying Merkle proof w.r.t. current state tree

Advanced query: how would balance change after executing tx t?

– answer by supplying Merkle proofs for every read/write to the state tree required to carry out the computation

Proving a State Transition

Basic query to MPT: current balance of my account?

- answer by supplying Merkle proof w.r.t. current state tree

Advanced query: how would balance change after executing tx t?

- answer by supplying Merkle proofs for every read/write to the state tree required to carry out the computation
- first, supply bytecode (tx only specifies account + data, not bytecode) and Merkle proof that it's the correct bytecode (check against state root)

Proving a State Transition

Basic query to MPT: current balance of my account?

– answer by supplying Merkle proof w.r.t. current state tree

Advanced query: how would balance change after executing tx t?

– answer by supplying Merkle proofs for every read/write to the state tree required to carry out the computation

– first, supply bytecode (tx only specifies account + data, not bytecode) and Merkle proof that it's the correct bytecode (check against state root)

– simulate computation, whenever state is accessed, supply corresponding value and Merkle proof that it's the correct value

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

– in Bitcoin, need to know current UTXOs to assess block validity
– in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

- in principle, could include the entire state in the block

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

- in principle, could include the entire state in the block
- slightly less crazy: include all Merkle proofs needed to assess validity

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

- in principle, could include the entire state in the block
- slightly less crazy: include all Merkle proofs needed to assess validity
- one possible future: Verkle trees (using KZG commitments)

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

- in principle, could include the entire state in the block
- slightly less crazy: include all Merkle proofs needed to assess validity
- one possible future: Verkle trees (using KZG commitments)
- another: each block includes SNARK proving its correctness