CSE 3400/CSE 5850 - Introduction to Cryptography & Cybersecurity / Introduction to Cybersecurity

Lecture 7
Hash Functions – Part II

Ghada Almashaqbeh

UConn

Adapted from textbook slides

Outline

- Hash based MACs.
- Accumulators.
  - Merkle-Damgard.
  - Merkle trees.
  - Blockchains.

Hash based MAC

- Hash-based MAC is often faster than block cipher-based MACs.
- How? Heuristic constructions:

Prepend Key: $MAC_k^{PK}(m) = h(k \mathbin{+\!\!+} m)$

Append Key: $MAC_k^{AK}(m) = h(m \mathbin{+\!\!+} k)$

Message-in-the-Middle: $MAC_k^{MitM}(m) = h(k \mathbin{+\!\!+} m \mathbin{+\!\!+} k)$

- Are these secure assuming CRHF? OWF? Both?
  - No.
- But: all are 'secure in the random oracle model': when the hash function is assumed to behave like a random function.

Hash-based MAC: HMAC

- HMAC uses an unkeyed hash function h:

$$HMAC_k(x)=h(k \oplus opad \,||\, h(k \oplus ipad \,||\, x))$$

  - *opad, ipad:* fixed sequences (of 36x, 5Cx resp.)
- It is a secure MAC under 'reasonable assumptions' [beyond our scope]
- Widely deployed
- More results, more exposure ➔ confidence!
- Hash functions are useful for MACs in another way:
  - Hash then MAC for efficiency.

Accumulators

- Generalization of collision-resistant hash
  - Input is a **sequence (ordered list)** of messages
  - Output is n-bit **digest**, denoted Δ
- Collision resistance accumulator means that it is hard to find two different message lists that have the same digest.

Accumulator Components

- Digest function Δ: $\{m_i \epsilon \{0,1\}^*\} \rightarrow \{0,1\}^{\boldsymbol{n}}$
  - Also called accumulate function.
  - Collision-resistance requirement
- Validation of Inclusion: PoI and $VerPoI$
  - PoI function: compute Proof of Inclusion
  - $VerPoI$ function: verify PoI
  - Optional, also Proof-of-Non-Inclusion (PoNI)
- Extending the Sequence: Extend function with optional PoC and $VerPoC$
  - PoC: Proof of Consistency (from old digest to new)
  - $VerPoC$ function: verify PoC

Correctness and Security for PoI and PoC

- Correctness means that on input a valid PoI, VerPoI will output 1.
  - Same for PoC.
- For PoI: security means that a PPT adversary cannot forge a valid PoI for a message that is no the hashed list.
- For PoC: security means that a PPT adversary cannot forge a valid PoC for an invalid digest extension.

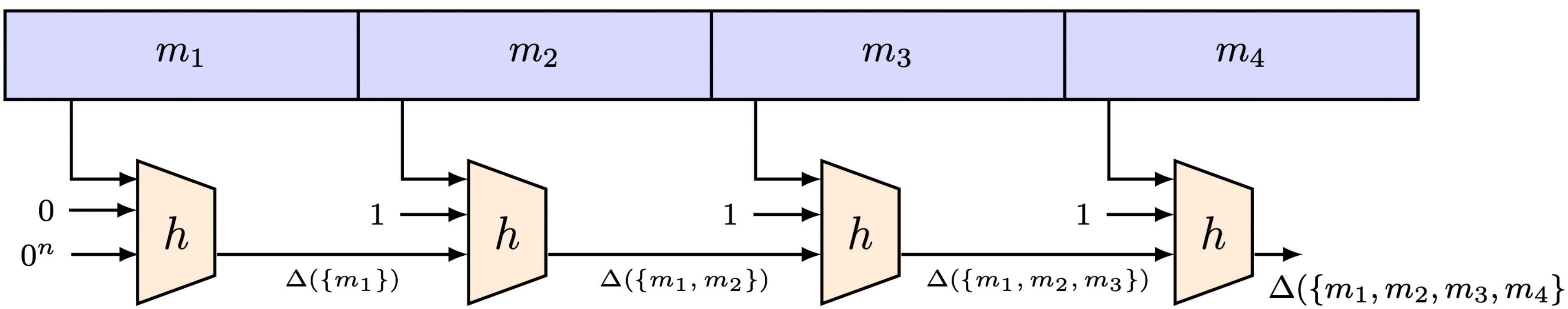
We will Study Three Accumulator Types

- Merkle-Damgard accumulators.
- Merkle trees.
- Blockchains

The Merkle-Damgard Accumulator

- Idea: hash iteratively, message by message:

  $\Delta(m_1, \ldots, m_l) = h(\Delta(m_1, \ldots, m_{l-1})||1||m_l)$; $\Delta(m_1) = h(0^{n+1}||m_1)$
- If h is a CRHF, then Δ is a collision-resistant digest
  - Proof… (out of scope, but you can see details in textbook)



Merkle-Damgard Length-Padding

- Aka Merkle - Damgard Strengthening
- Let $pad(x)=1||0^k||\mathrm{bin}\,(|x|)$; $x'=x||pad(x)$
  - Where $bin\,(|x|)$ is the n–bit binary representation of $|x|$
- For $i=1, \ldots, l$, where $l = |x'|/n$, and let x'_i *is the* i^{th} n-bit block of x'.
- Apply the construction in the prior slide to obtain the digest of x'

This is just a high level idea, care needed to avoid collisions

The Digest-Chain Extend Function

- Beyond digest and collision resistance: sequence-related integrity mechanisms
- For digest-chain, the **extend function:**
  - Input: digest and 'next' sequence
  - Output: digest (of entire sequence)
  - Correctness requirement:

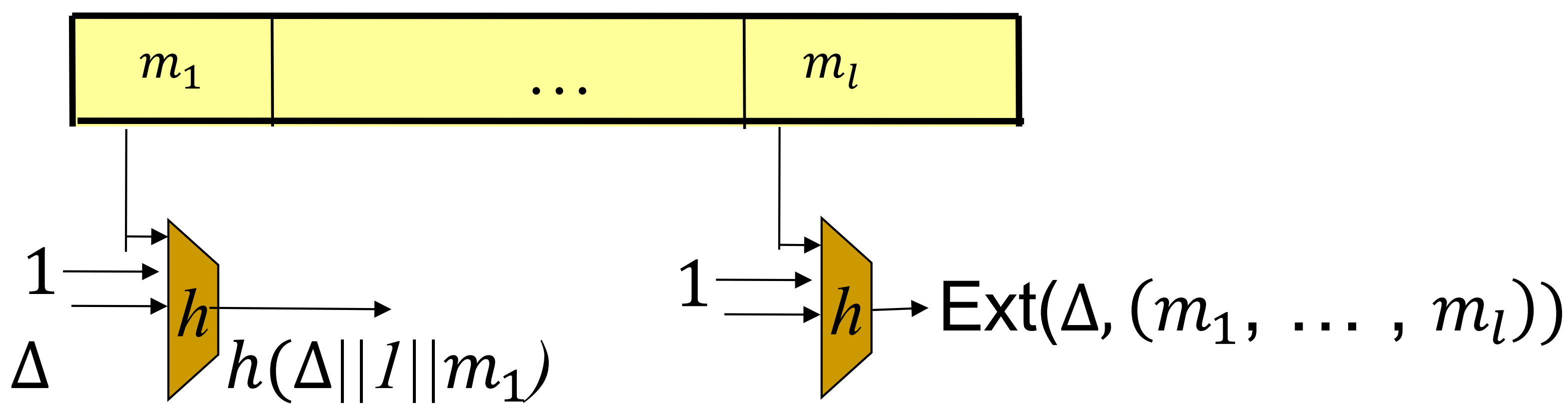
$$Extend(\Delta_l, M_{l+1,l'}) = \Delta(M_l \mathbin{+\!\!\!+} M_{l+1,l'})$$

The Merkle-Damgard Extend Function

- We can define Extend for Merkle-Damgard:
  - Idea: Just continue last digest!

$$\mathcal{MD}^h.Extend\,(\Delta, \{m_1, \ldots, m_l\}) \equiv \begin{cases} \text{Let } \Delta_1 \leftarrow h(\Delta \mathbin{+\!\!+} 1 \mathbin{+\!\!+} m_1) \\ \text{For } l = 1\text{: } \Delta_1 \\ \text{For } l > 1\text{:} \\ \qquad \mathcal{MD}^h.Extend\,(\Delta_1, \{m_2, \ldots, m_l\}) \end{cases}$$

- Not secure to be used to construct a MAC!

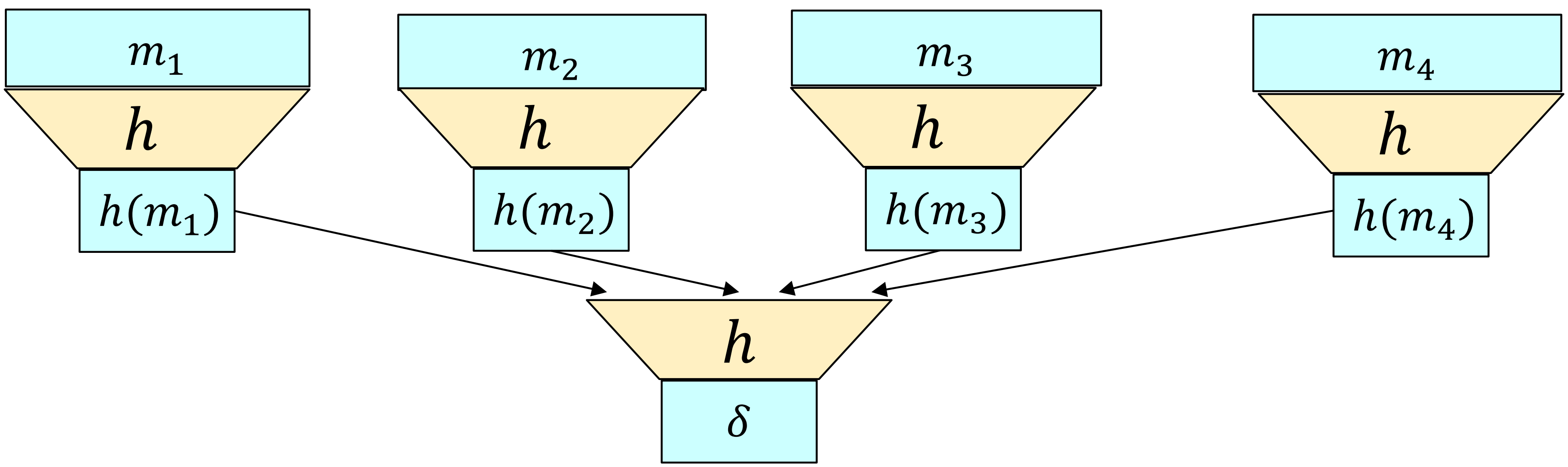


Two-layered Merkle Tree

- Short digest validates integrity of large object
  - Often, object consists of multiple 'files'
- Merkle tree : integrity for many 'messages'
  - Hash each 'message' in block, then hash-of-hashes

$$\delta = h(h(m_1)||h(m_2)||h(m_3)||h(m_4))$$

  - Validate each 'message' independently
    - Advantages: **efficiency** (computation, communication) and **privacy**



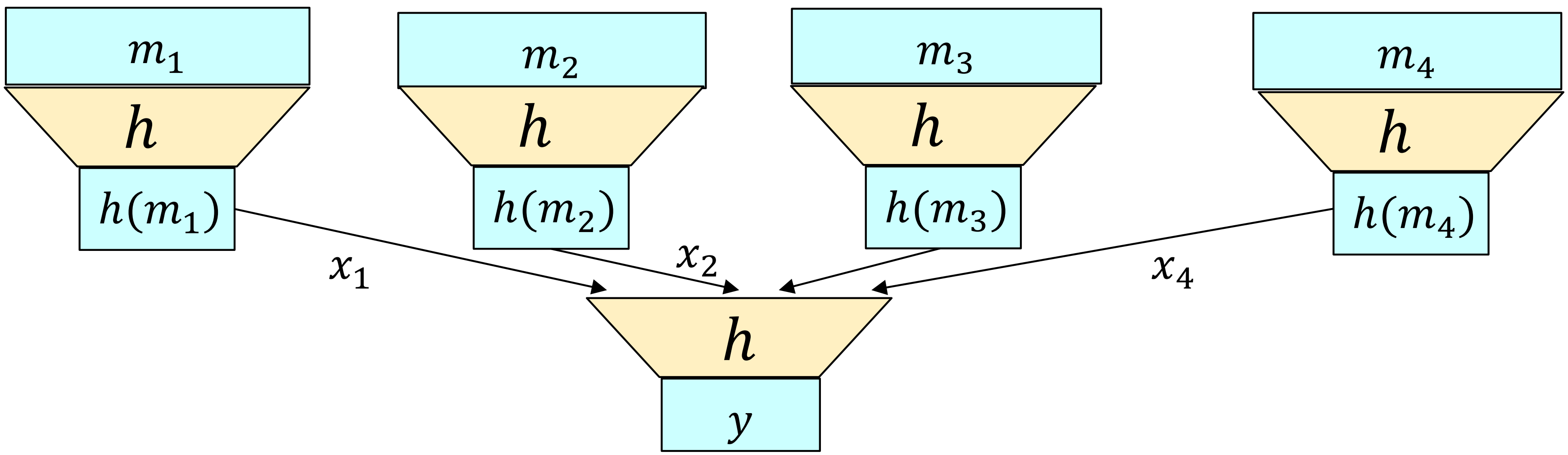
Two-layered Merkle tree

- Hash each item in block separately:

$$x_1 = h(m_1), x_2 = h(m_2), \quad \ldots$$

- Digest is hash of hashes:

$$y = \Delta(m_1, m_2, \ldots) = h(x_1 || x_2 || \ldots)$$



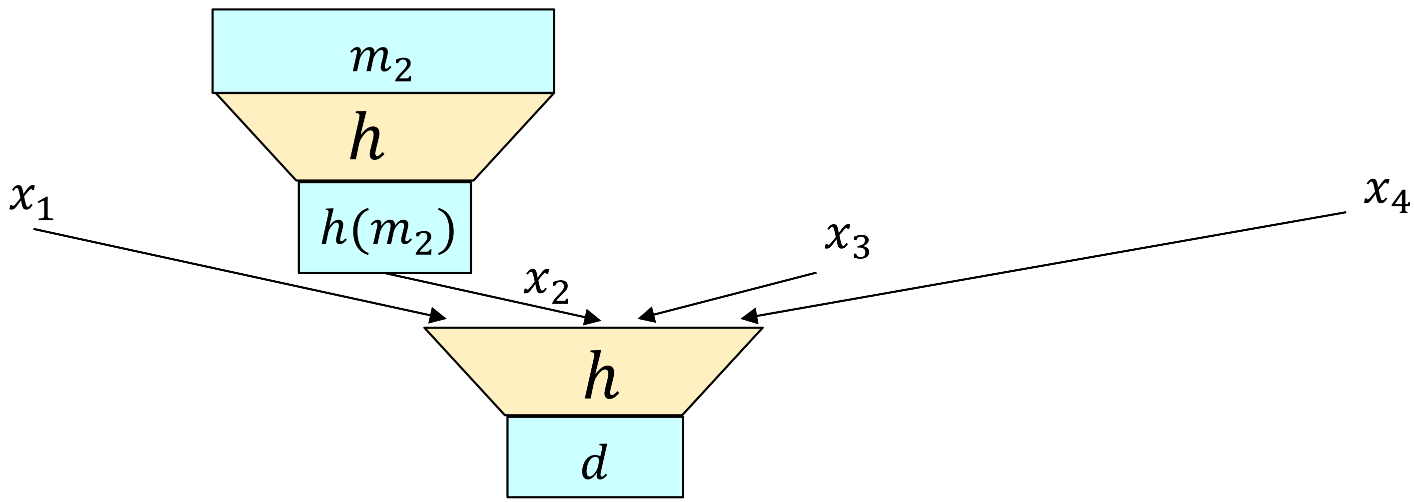
Allows each user to receive, validate only required items. How?

To verify inclusion of m_2 ...

$$2l\mathcal{MT}.\Delta(m_1, \ldots, m_l) \equiv h\left[h(m_1) \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} h(m_l)\right]$$

$$2l\mathcal{MT}.PoI((m_1, \ldots, m_l), j) \equiv \{h(m_i)\}_{i=1}^{l}$$

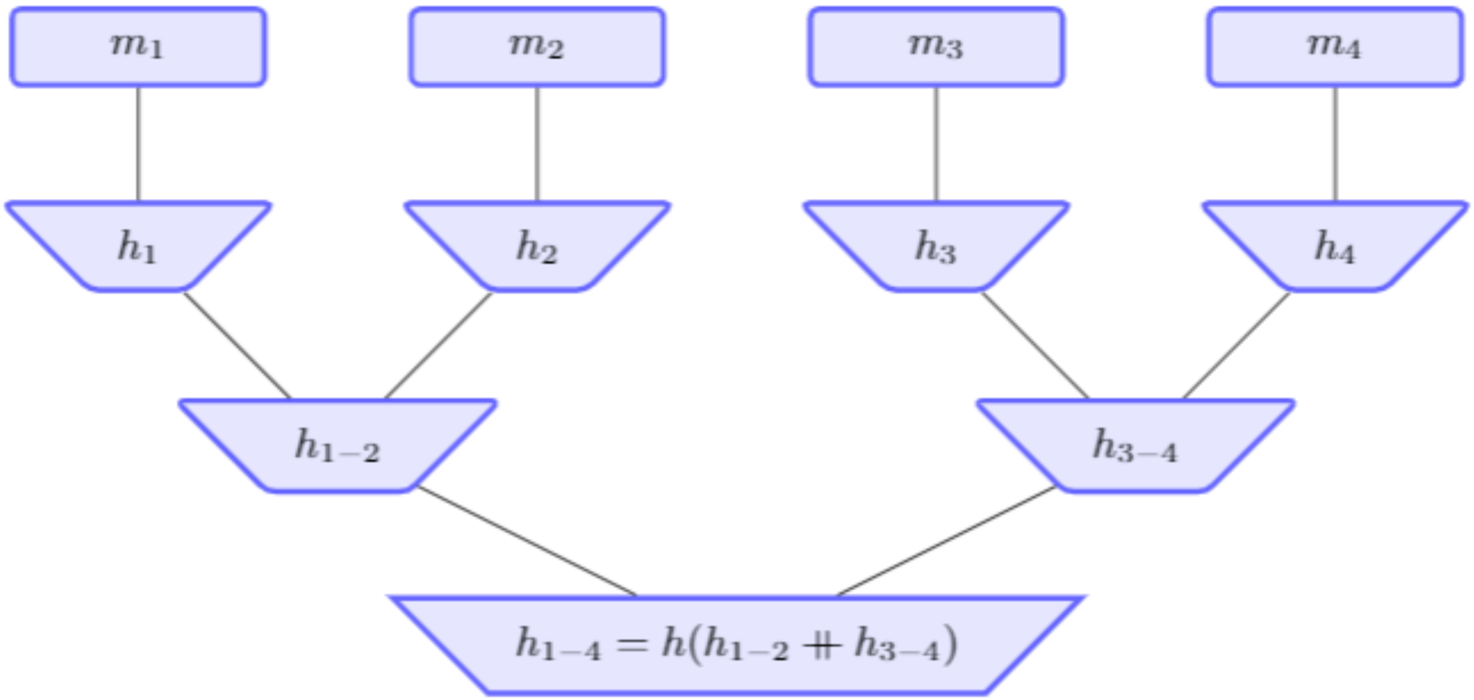
$$2l\mathcal{MT}.VerPoI(d, m, i, \{x_i\}_{i=1}^{l}) \equiv \left[\begin{array}{c} \textsc{True} \textit{ if } x_i = h(m), \textit{ and} \\ d = h(x_1 \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} x_l) \end{array}\right]$$



Receive and validate only m_2. Other hashes still required, though.

The Merkle Tree Construction

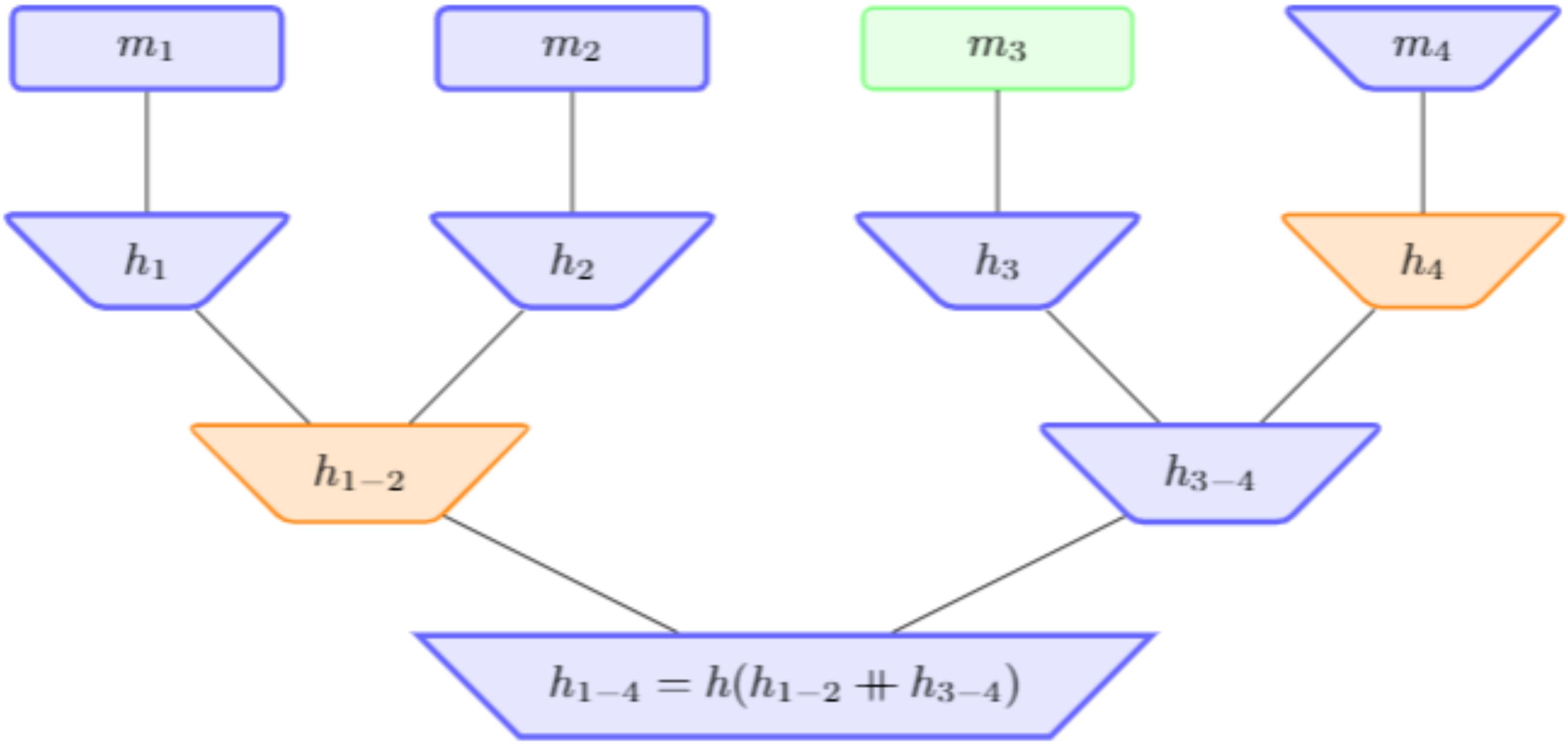
- Reduce length of 'proofs' – send less hashes of 'other msgs'



$$\mathcal{MT}.\Delta(M) \equiv \begin{cases} \text{If } L = 0: & h(m_1) \\ \text{Else} & h\left(\mathcal{MT}.\Delta\left(m_1, \ldots, m_{2^{L-1}}\right) + \right. \\ & \left. + \mathcal{MT}.\Delta\left(m_{2^{L-1}+1}, \ldots, m_{2^L}\right)\right) \end{cases}$$

Merkle Tree: Proof of Inclusion (PoI)

- To prove inclusion of m_3 , send also 'proofs': h_{1-2}, h_4



Blockchains

- Separate slide set.

Covered Material From the Textbook

- Chapter 3: Sections 3.7, 3.8, and 3.9
  - Only the material that corresponds to what we covered in class
- Chapter 4: Section 4.4.5

Thank You!