CSE 3400 - Introduction to Computer & Network Security
(aka: Introduction to Cybersecurity)

Lecture 7
Hash Functions – Part II

Ghada Almashaqbeh

UConn

From Textbook Slides by Prof. Amir Herzberg

UConn

Outline

- Hash based MACs.
- Domain extension.
- Merkle digest and Merkle trees.
- Blockchains.

Hash based MAC

- ## Hash-based MAC is often faster than block-cipher MAC

- ## 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 'secure in random oracle model'

Hash-based MAC: HMAC

- HMAC uses only the unkeyed hash function h:

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

   - *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.

Digest Schemes

- **Generalization of collision-resistant hash**
  - Input is a **sequence** of messages
  - Output is n-bit **digest**, denoted Δ
- **Three types of schemes:**
  - Digest functions (or accumulators)
  - Merkle Digest (and Merkle trees)
  - Blockchains
- **In other textbooks, this is referred to as Domain Extension.**

Digest Functions

- ## Generalization of collision-resistant hash
  - ### Input is a **sequence** of messages
 - ### Output is n-bit **digest**, denoted Δ

Definition A digest function Δ *is an efficiently computable function (in PPT) that maps blocks (finite sequences of binary strings) to n-bit binary string i.e.,* $\Delta : (\{0,1\}^*)^* \to \{0,1\}^n$, *where n is the* security parameter.

Digest function Δ *is* collision resistant *if the* digest collision-resistance advantage $\varepsilon_{\mathcal{A},\Delta}^{DCR}(n)$ *is negligible (in n), for every efficient adversary* $\mathcal{A} \in PPT$, *where:*

$$\varepsilon_{\mathcal{A},\Delta}^{DCR}(n) \equiv \Pr\left((B, B') \leftarrow \mathcal{A}(1^n) \ s.t. \ B \neq B' \wedge \Delta(B) = \Delta(B')\right)$$
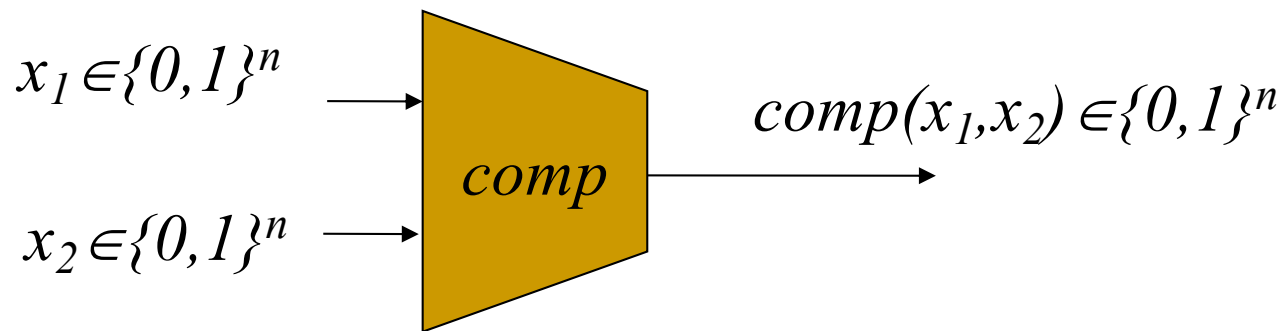
The Merkle-Damgard Digest Function

- The Merkle-Damgard construction of:
  - Collision-Resistant Digest function from CRHF
  - VIL CRHF from compression function (FIL CRHF): $|m_i| = n$

- Idea: hash iteratively, message by message:

$$\Delta(m_1, \dots, m_l) = h(\Delta(m_1, \dots, m_{l-1})||1||m_l) \; ; \; \Delta(m_1) = h(0^{n+1}||m_1)$$

- Lemma 4.2: if h is a CRHF, then Δ is a collision-resistant digest

- Proof... (see details in textbook)
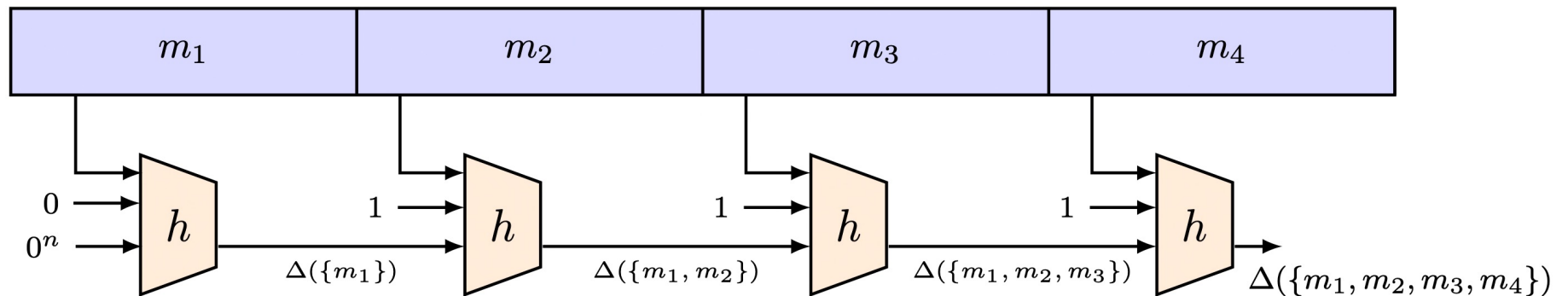
VIL CRHF from FIL CRHF

- **Recall**: design and cryptanalyze simple (FIL) function, use it to construct strong (VIL) function

- Build VIL CRHF $\{0,1\}^* \rightarrow \{0,1\}^n$ from FIL CRHF (aka <u>compression function</u>) $comp: \{0,1\}^m \rightarrow \{0,1\}^n$

 - E.g. $m=2n$, i.e. $comp: \{0,1\}^{2n} \rightarrow \{0,1\}^n$

$x_1 \in \{0,1\}^n$ → *comp* → $comp(x_1, x_2) \in \{0,1\}^n$

$x_2 \in \{0,1\}^n$ →

  - The Merkle-Damgard constructs a CRHF from a compression function

  - Requires `MD-strengthening' extension (next slide)

Merkle-Damgard Length-Padding

- Aka Merkle - Damgard Strengthening
- Let $pad(x)=1||0^k||\text{bin}(|x|)$; $x'=x||pad(x)$
  - Where $bin(|x|)$ is the L–bit binary representation of $|x|$
  - And: $|x|+|pad(x)|\equiv 0 \bmod L$
  - Simplify: assume $|x|\equiv 0 \bmod L$, $|pad(x)|=L$
- Let $y_0=IV$ be some fixed L bits (IV=Initialization Value)
- For $i=1,..|x'|/L$ let $y_i=h(x'[i] || y_{i-1})$
- Output $h^{MD}(x)=y_{l+1}$

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

$x[1]$	$x[2]$	…	$x[l]||10^k$	$bin(|x|)$

$IV \longrightarrow h \to h \to \quad \to h \to h \to h^{MD}(x)=y_{l+1}=h(|x| || y_l)$
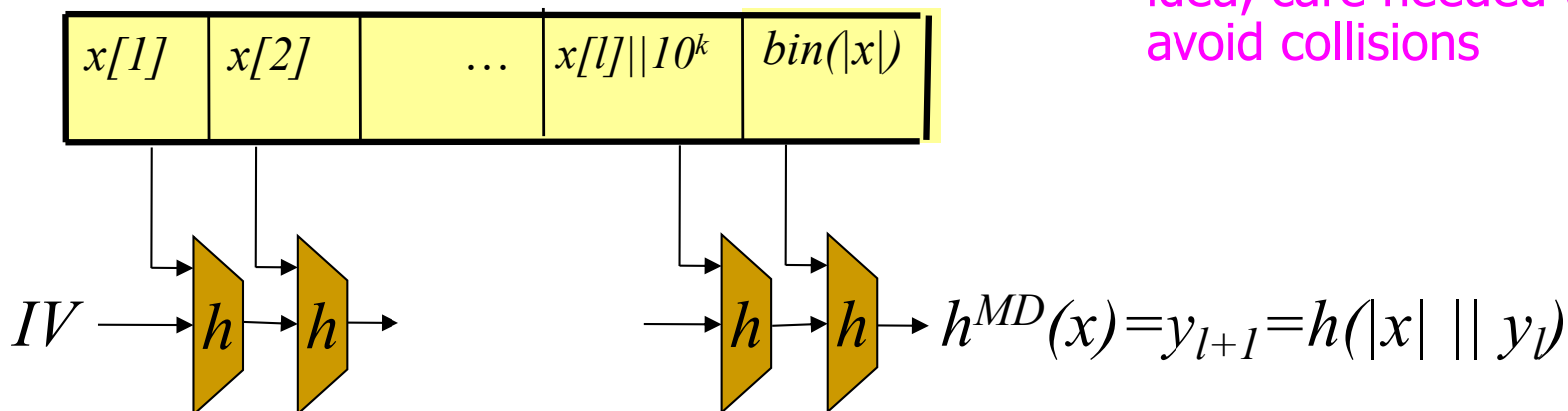
The Digest-Chain Extend Function

- 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:

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

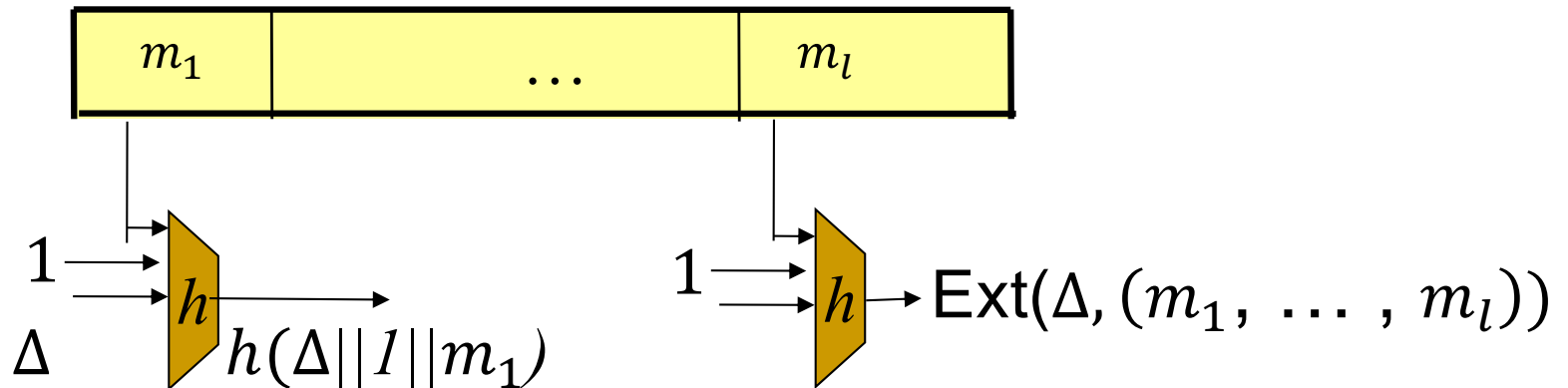Use to (1) extend chain, (2) validate new digest (with new seq.), or (3) use digest to validate a message

The Merkle-Damgard Extend Function

- We can define Extend for Merkle-Damgard:

  - Idea: Just continue last digest!

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

- Not secure to be used to construct a MAC!

Merkle Digest Schemes

- **Digest function $\Delta: \{m_i \epsilon \{0,1\}^*\} \to \{0,1\}^n$**
  - Collision-resistance requirement

- **Validation of Inclusion: PoI and $VerPoI$**
  - PoI function: compute Proof of Inclusion
  - $VerPoI$ function: verify PoI
  - Both: mandatory and optimized
  - Optional, also Proof-of-Non-Inclusion (PoNI)

- **Extending the Sequence: PoC and $VerPoC$**
  - PoC: Proof of Consistency (from old digest to new)
  - $VerPoC$ function: verify PoC
  - Optional

Merkle digest scheme: definition

Definition (Merkle digest scheme)**.** *A Merkle digest scheme $\mathcal{M}$ is a tuple of three PPT functions $(\mathcal{M}.\Delta, \mathcal{M}.PoI, \mathcal{M}.VerPoI)$, where:*

$\mathcal{M}.\Delta$ *is the* Merkle tree digest *function, whose input is a sequence of messages $B = \{m_i \in \{0,1\}^*\}_i$ and whose output is an n-bit* digest*: $\mathcal{M}.\Delta : (\{0,1\}^*)^* \to \{0,1\}^n$.*

$\mathcal{M}.PoI$ *is the* Proof-of-Inclusion *function, whose input is a sequence of messages $B = \{m_i \in \{0,1\}^*\}_i$, an integer $i \in [1, |B|]$ (the index of one message in B), and whose output is a* Proof-of-Inclusion *(PoI): $\mathcal{M}.PoI : (\{0,1\}^*)^* \times \mathbb{N} \to \{0,1\}^*$.*

$\mathcal{M}.VerPoI$ *is the* Verify-Proof-of-Inclusion *predicate, whose inputs are digest $d \in \{0,1\}^n$, message $m \in \{0,1\}^*$, index $i \in \mathbb{N}$, proof $p \in \{0,1\}^*$, and whose output is a bit (1 for 'true' or 0 for 'false'): $\mathcal{M}.VerPoI : \{0,1\}^n \times \{0,1\}^* \times \mathbb{N} \times \{0,1\}^* \to \{0,1\}$.*

Merkle digest: correctness and security

- Correctness means that on input a valid PoI, VerPoI will output 1.

- Security means that a PPT adversary cannot find collisions against the digest functions, and cannot forge a valid PoI.

Proof of Consistency (PoC)

- ## A Merkle digest scheme supports PoC if it has two more functions:

$M.PoC(B_C, B_N)$ *is the* Extend and Proof-of-Consistency *function PoC, whose input are two sequences, B_C and B_N, and whose output $p_{CN} = M.PoC(B_C, B_N)$ is a binary string which we call the* Proof-of-Consistency *from $\Delta_C \equiv M.\Delta(B_C)$ to $\Delta_{CN} \equiv M.\Delta(B_{CN})$.*

$M.VerPoC(\Delta_C, \Delta_{CN}, l_C, l_N, p) \in \{\textbf{True}, \textbf{False}\}$ *is the* Verify-Proof-of-Consistency *predicate, whose inputs are the two digests Δ_C, Δ_{CN}, the numbers of entries (l_C and l_N), and a string (PoC) p.*

- ## Correctness and security of PoC:

  - Correctness: A valid generated PoC will cause VerPoC to output 1.

  - Security: a PPT adversary cannot forge a valid PoC.

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
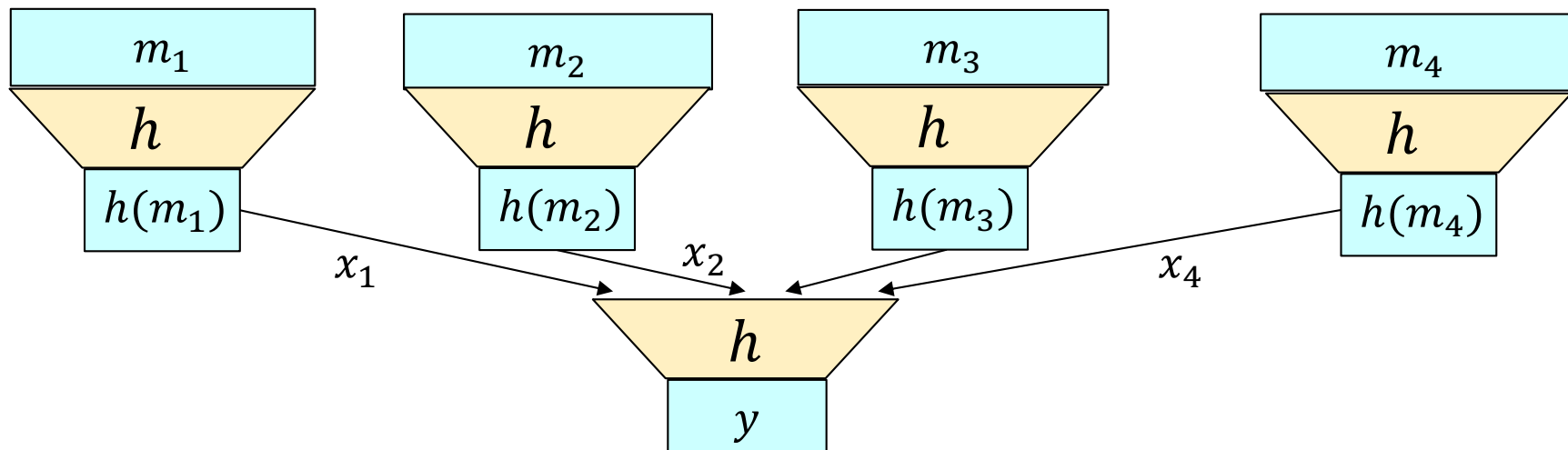   - Advantages: **efficiency** (computation, communication) and **privacy**
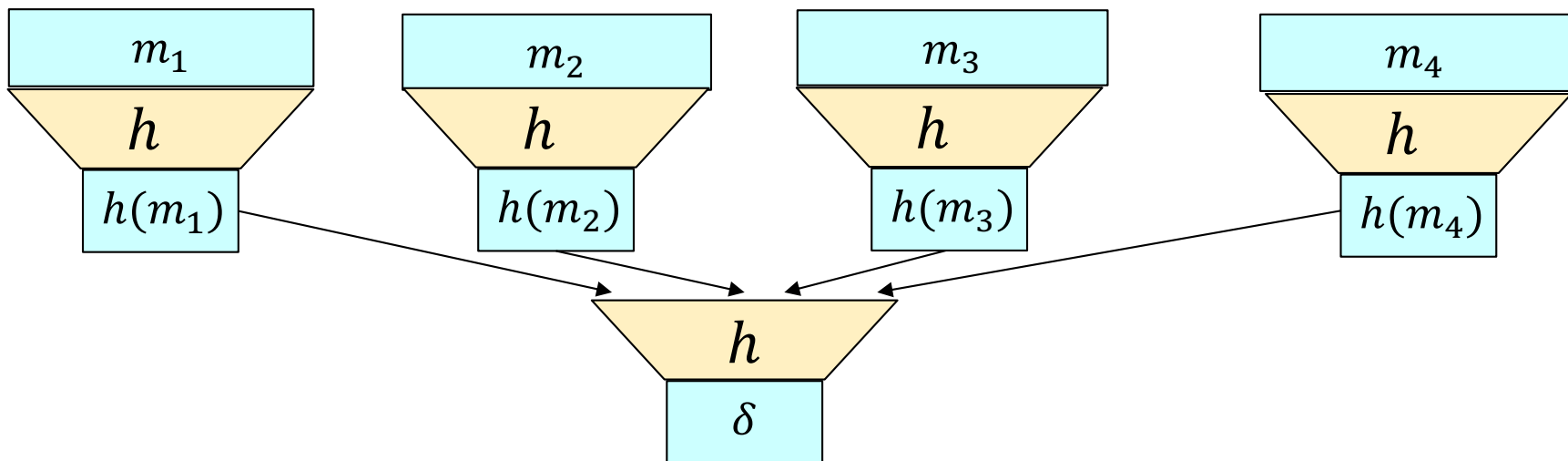
Two-layered Merkle tree

- Hash each item in block separately:
$$x_1 = h(m_1), x_2 = h(m_2), \quad \dots$$

- Digest is hash of hashes:
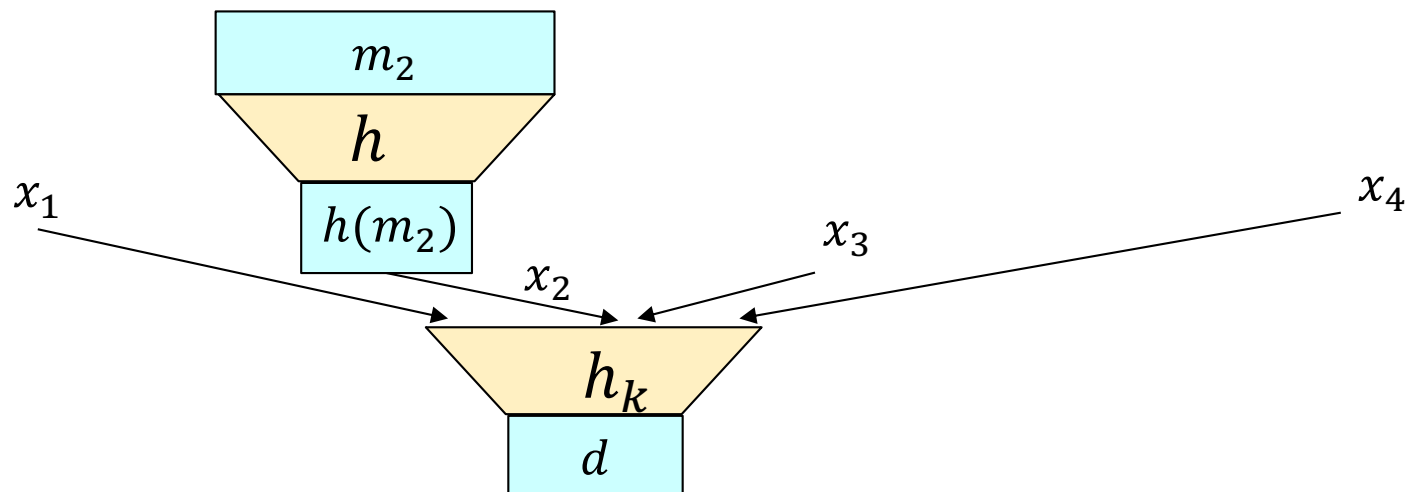$$y = \Delta(m_1, m_2, \dots) = h(x_1 || x_2 || \dots)$$

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

To verify inclusion of m_2 ...

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

$$2l\mathcal{MT}.PoI((m_1, \ldots, m_l), j) \equiv \{h(m_i)\}_{i=1}^{l}$$
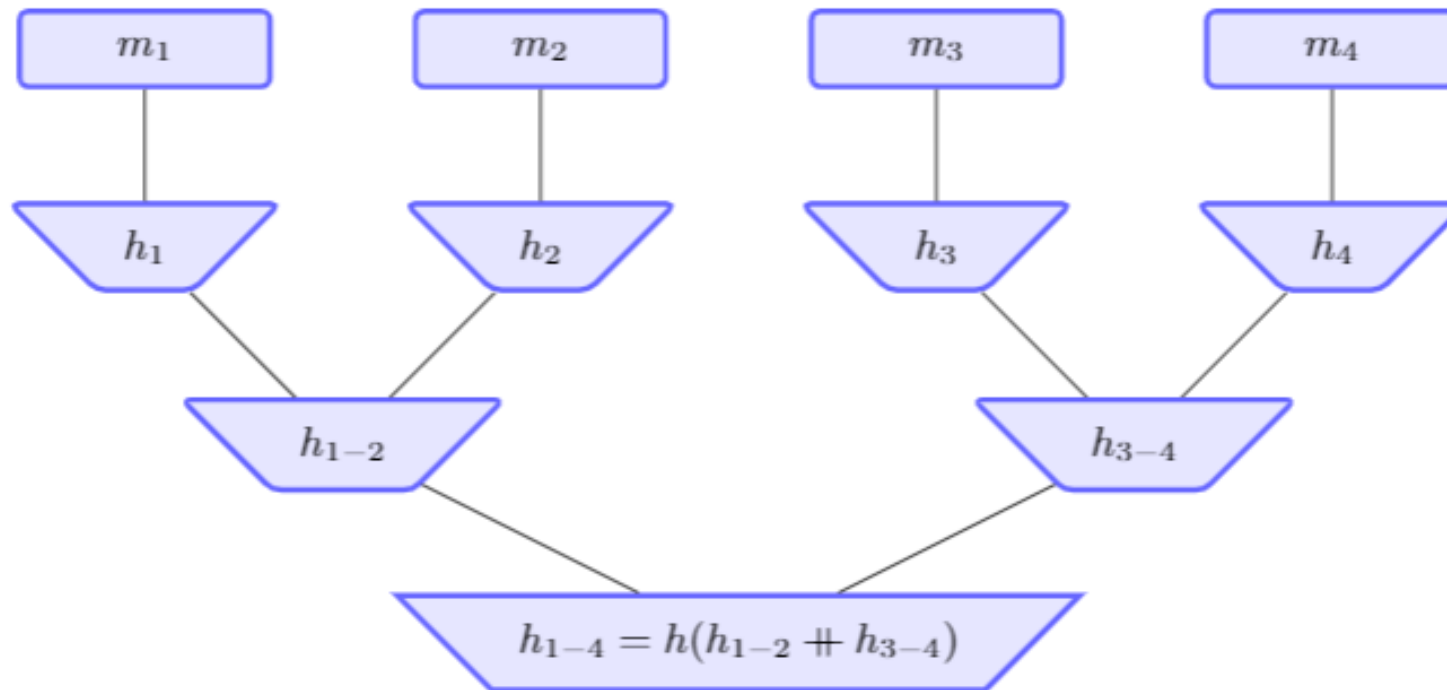
$$2l\mathcal{MT}.VerPoI(d, m, i, \{x_i\}_{i=1}^{l}) \equiv \left[\begin{array}{c} \text{TRUE } \textit{if } x_i = h(m), \textit{ and} \\ d = h(x_1 + \ldots + x_l) \end{array}\right]$$

Receive and validate only m_2. Other hashes still required, though.
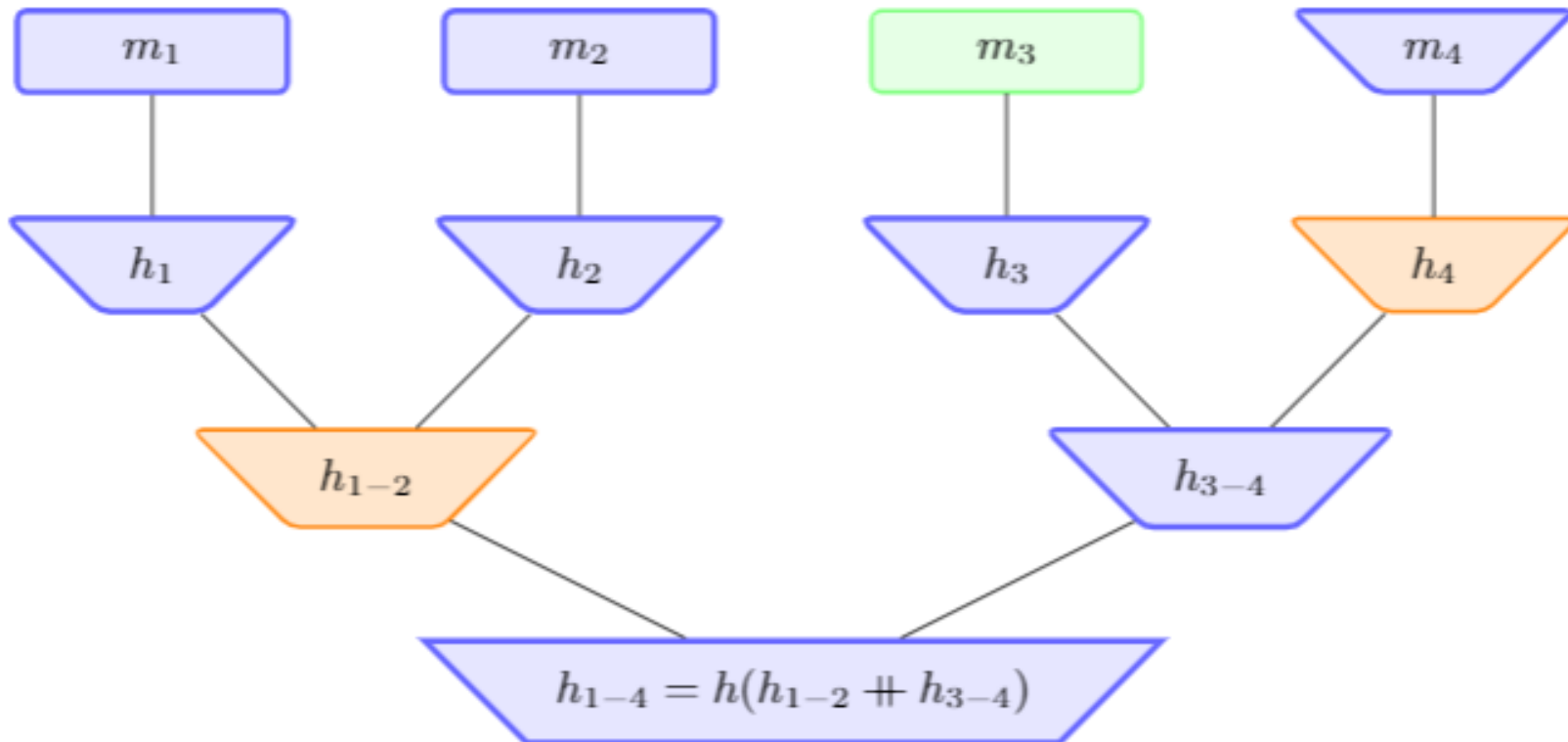
The Merkle Tree Construction

- 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

- ❑ Next slides set.

Covered Material From the Textbook

- Chapter 3: Sections 3.7, 3.8, and 3.9
- Chapter 4: Section 4.6.3

Thank You!