
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: $MAC_k^{PK}(m) = h(k \parallel m)$

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

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

- 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.), for max hamming distance btw $k \oplus opad$ and $k \oplus ipad$.
- [BCK]: secure MAC under ‘reasonable assumptions’ [beyond our scope]
- Widely deployed – for MAC, PRF and KDF
 - KDF – Key Derivation Function
- More results, more exposure → confidence!
- Hash are useful for MACs in another way:
 - Hash then MAC.

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-chain
 - Merkle Digest (and Merkle trees)
 - Blockchains (and Bitcoin)
- In other textbooks, this is referred to as Domain Extension.

Digest-Chain Schemes

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

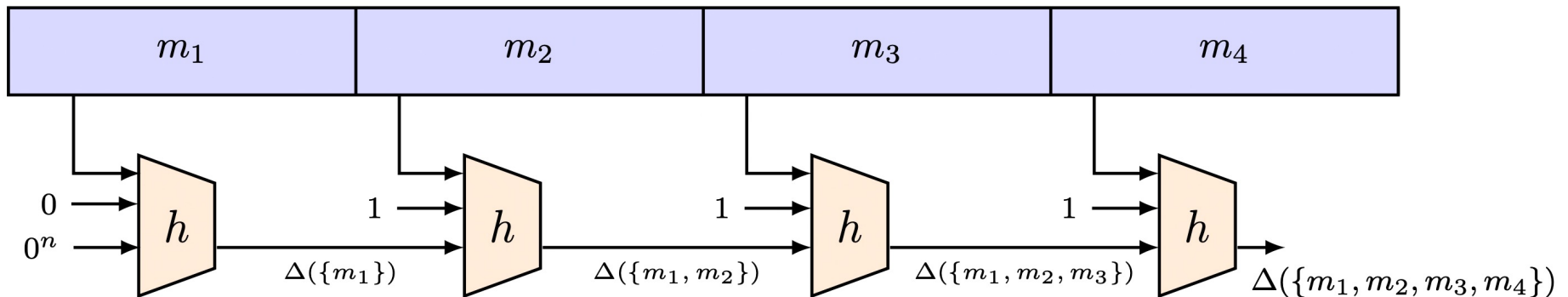
Definition 4.13. A digest function Δ is an efficiently computable function (in PPT) that maps blocks (finite sequences of binary strings) to n-bit binary strings, i.e., $\Delta : (\{0, 1\}^*)^* \rightarrow \{0, 1\}^*$, where n is the security parameter.

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:

$$\varepsilon_{\mathcal{A}, \Delta}^{DCR}(n) \equiv \Pr((B, B') \leftarrow \mathcal{A}(1^n) \text{ s.t. } B \neq B' \wedge \Delta(B) = \Delta(B')) \quad (4.21)$$

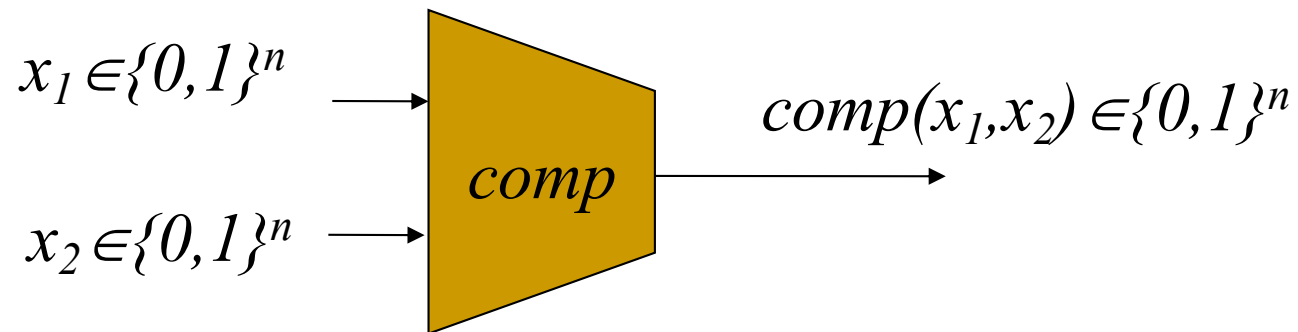
The Merkle-Damgard Digest Function

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

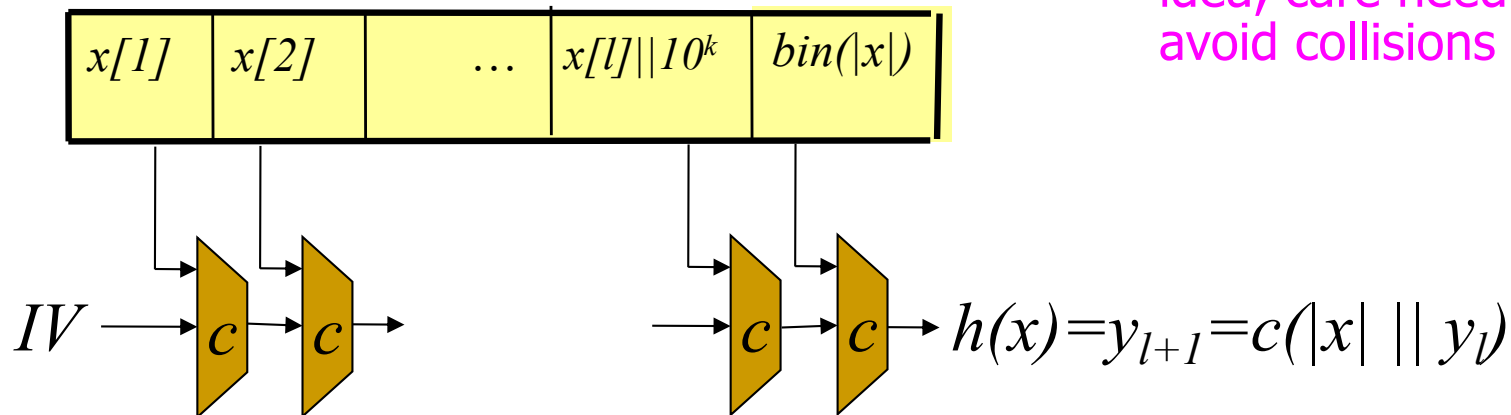


- The Merkle-Damgard constructs a CRHF from a compression function
- Requires 'MD-strengthening' extension [see text]

Merkle - Damgard Length-Padding

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

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:

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

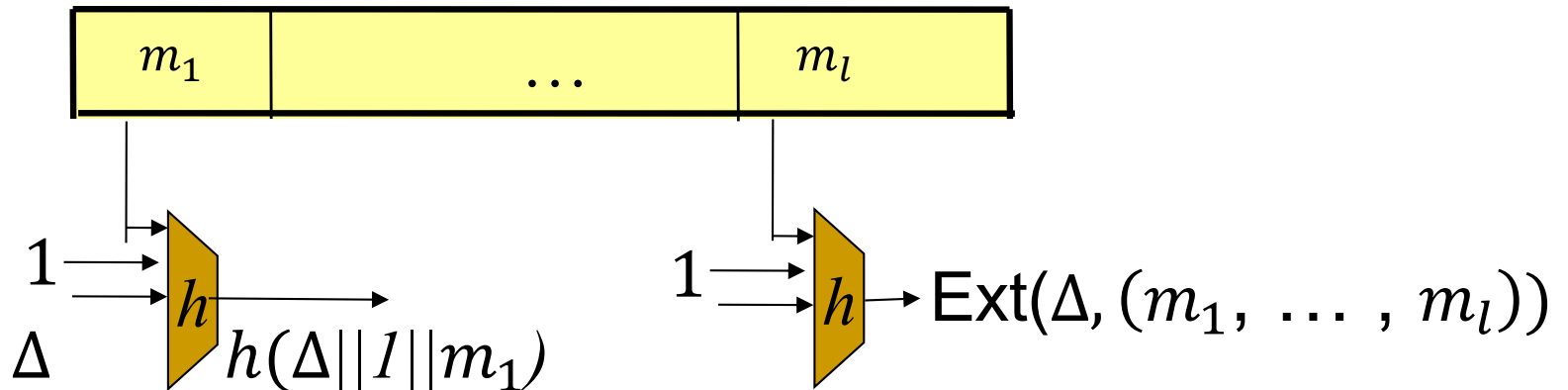
Use to (1) extend chain, (2) validate new digest (with new seq.), or (3) use digest to validate a message

The Merkle-Damgard Extend Function

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

$$\mathcal{MD}^h.\text{Extend}(\Delta, \{m_1, \dots, m_l\}) \equiv \begin{cases} \text{Let } \Delta_1 \leftarrow h(\Delta \# 1 \# m_1) \\ \text{For } l = 1: \Delta_1 \\ \text{For } l > 1: \\ \quad \mathcal{MD}^h.\text{Extend}(\Delta_1, \{m_2, \dots, m_l\}) \end{cases}$$

- Not secure to be used to construct a MAC!



Merkle Digest Schemes

- Digest function $\Delta: \{m_i \in \{0,1\}^*\} \rightarrow \{0,1\}^n$
 - 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 4.15 (Merkle digest scheme). A Merkle digest scheme \mathcal{M} is a tuple of three PPT functions $(\mathcal{M}.\Delta, \mathcal{M}.\text{PoI}, \mathcal{M}.\text{VerPoI})$, 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: $\mathcal{M}.\Delta : (\{0, 1\}^*)^* \rightarrow \{0, 1\}^n$.

$\mathcal{M}.\text{PoI}$ 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): $\mathcal{M}.\text{PoI} : (\{0, 1\}^*)^* \times \mathbb{N} \rightarrow \{0, 1\}^*$.

$\mathcal{M}.\text{VerPoI}$ 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'): $\mathcal{M}.\text{VerPoI} : \{0, 1\}^n \times \{0, 1\}^* \times \mathbb{N} \times \{0, 1\}^* \rightarrow \{0, 1\}$.

Merkle digest: correctness and security

A Merkle digest scheme \mathcal{M} is correct if for every sequence of messages $B = \{m_i \in \{0, 1\}^*\}_i$ and every index $i \in [1, |B|]$, the Proof-of-Inclusion verifies correctly, i.e.:

$$\mathcal{M}.VerPoI(\mathcal{M}.\Delta(B), m_i, i, \mathcal{M}.PoI(B, i)) = \text{TRUE} \quad (4.29)$$

A Merkle digest scheme \mathcal{M} is secure if for every efficient (PPT) algorithm \mathcal{A} , both the collision advantage $\varepsilon_{\mathcal{M}, \mathcal{A}}^{Coll}(n)$ and the PoI advantage $\varepsilon_{\mathcal{M}, \mathcal{A}}^{PoI}(n)$ are negligible in n , i.e., smaller than any positive polynomial for sufficiently large n (as $n \rightarrow \infty$), where:

$$\begin{aligned} \varepsilon_{\mathcal{M}, \mathcal{A}}^{Coll}(n) &\equiv \Pr \left[\begin{array}{l} (x, x') \leftarrow \mathcal{A}(1^n) \text{ s.t. } (x \neq x') \\ \wedge (\mathcal{M}.\Delta(x) = \mathcal{M}.\Delta(x')) \end{array} \right] \\ \varepsilon_{\mathcal{M}, \mathcal{A}}^{PoI}(n) &\equiv \Pr \left[\begin{array}{l} (\{m_1, \dots, m_l\}, d, m, i, p) \leftarrow \mathcal{A}(1^n) \text{ s.t. } m_i \neq m \wedge \\ d = \mathcal{M}.\Delta(\{m_1, \dots, m_l\}) \wedge \\ \mathcal{M}.VerPoI(d, m, i, p) = \text{TRUE} \end{array} \right] \end{aligned}$$

Where the probability is taken over the random coin tosses of \mathcal{A} .

Simply put, security means that a PPT adversary cannot find collisions 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 $\gamma_{CN} = m.PoC(B_C, B_N)$ is a binary string which we call the Proof-of-Consistency from $\Delta_C \equiv m.\Delta(B_C)$ to $\Delta_{CN} \equiv m.\Delta(B_{CN})$.

$m.VerPoC(\Delta_C, \Delta_{CN}, l_C, l_N, p) \in \{\mathbf{True}, \mathbf{False}\}$ 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 .

- **Correct PoC:**

$$m.VerPoC(m.\Delta(B_C), m.\Delta(B_C \# B_N), l_C, l_N, m.PoC(B_C, B_N)) = \mathbf{TRUE}$$

Secure Proof of Consistency

We say that \mathcal{M} has secure PoC, if for every efficient (PPT) algorithm \mathcal{A} , the PoC-advantage $\varepsilon_{\mathcal{M}, \mathcal{A}}^{\text{PoC}}(n)$ is negligible in n , where:

$$\varepsilon_{\mathcal{M}, \mathcal{A}}^{\text{PoC}}(n) \equiv \Pr \left[\begin{array}{l} (B_C, B_A, l_C, l_A, p) \leftarrow \mathcal{A}(1^n) \text{ s.t.} \\ \mathcal{M}.\text{VerPoC}(\mathcal{M}.\Delta(B_C), \mathcal{M}.\Delta(B_A), l_C, l_A, p) = \text{TRUE} \wedge \\ \wedge B_C \text{ is not a prefix of } B_A \end{array} \right]$$

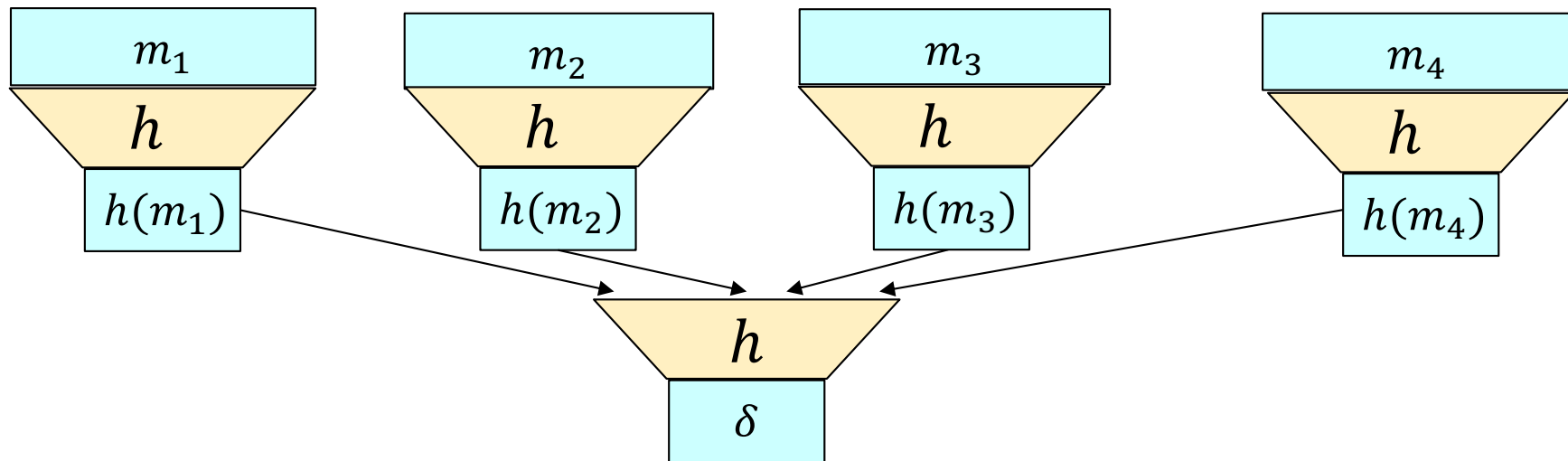
Where the probability is taken over the random coin tosses of \mathcal{A} .

To be consistent with previous slides, replace B_A with B_{CN}

Simply put, the above says that 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**

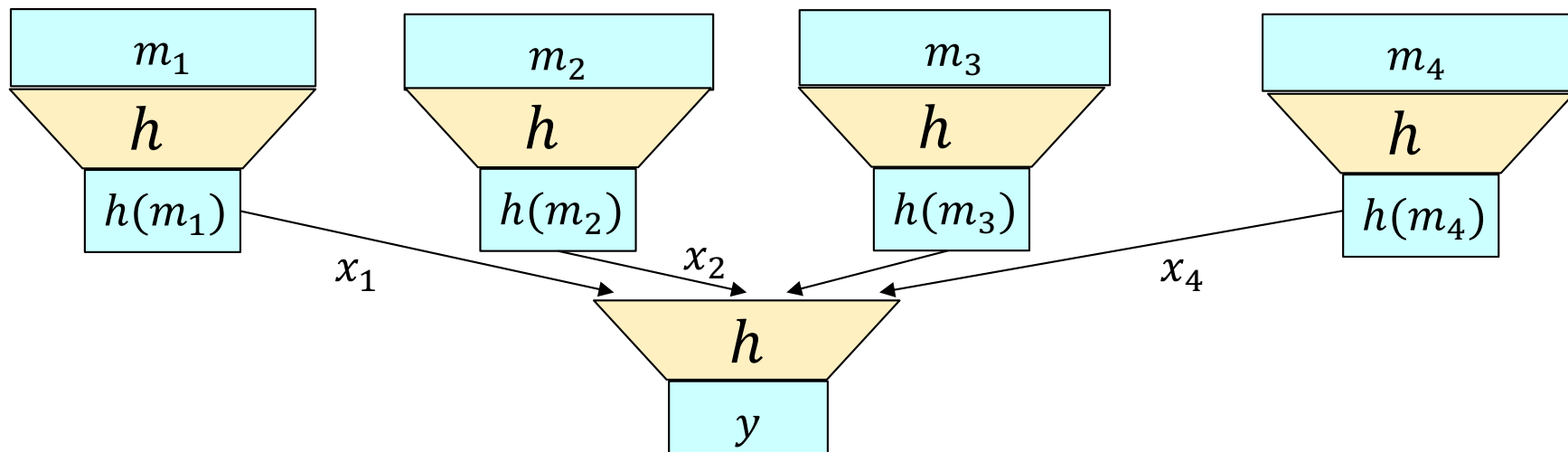


Two-layered Merkle tree

$$2lMT.\Delta(m_1, \dots, m_l) \equiv h[h(m_1) \# \dots \# h(m_l)]$$

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

$$2lMT.VerPoI(d, m, i, \{x_i\}_{i=1}^l) \equiv \left[\begin{array}{l} \text{TRUE if } x_i = h(m), \text{ and} \\ d = h(x_1 \# \dots \# x_l) \end{array} \right]$$



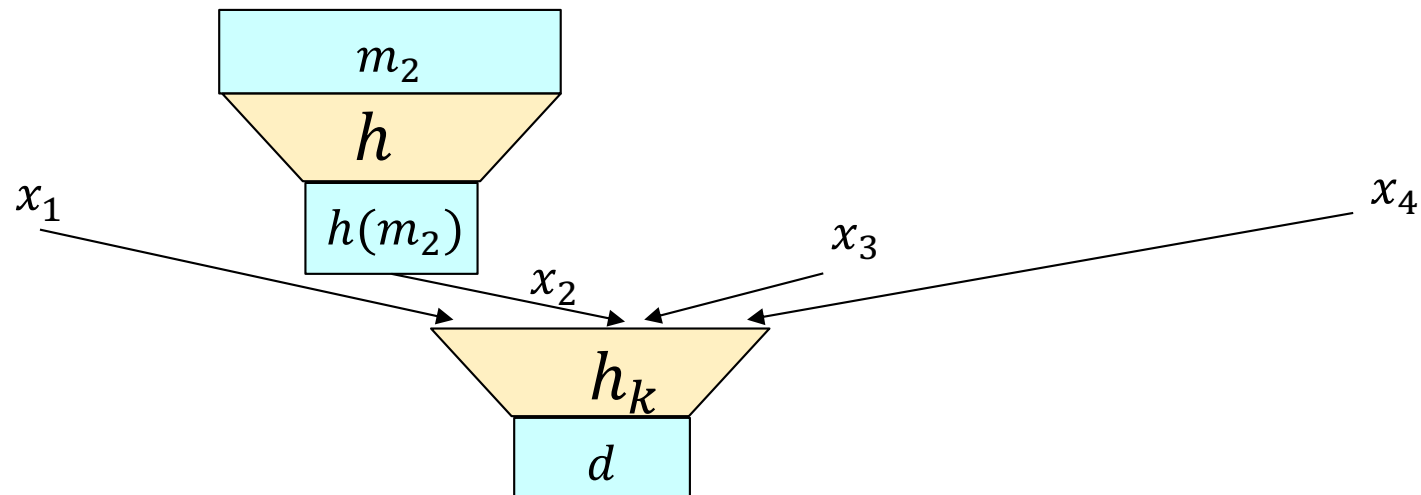
Allows each user to receive, validate only required items. How?

To verify inclusion of m_2 ...

$$2lMT.\Delta(m_1, \dots, m_l) \equiv h[h(m_1) \# \dots \# h(m_l)]$$

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

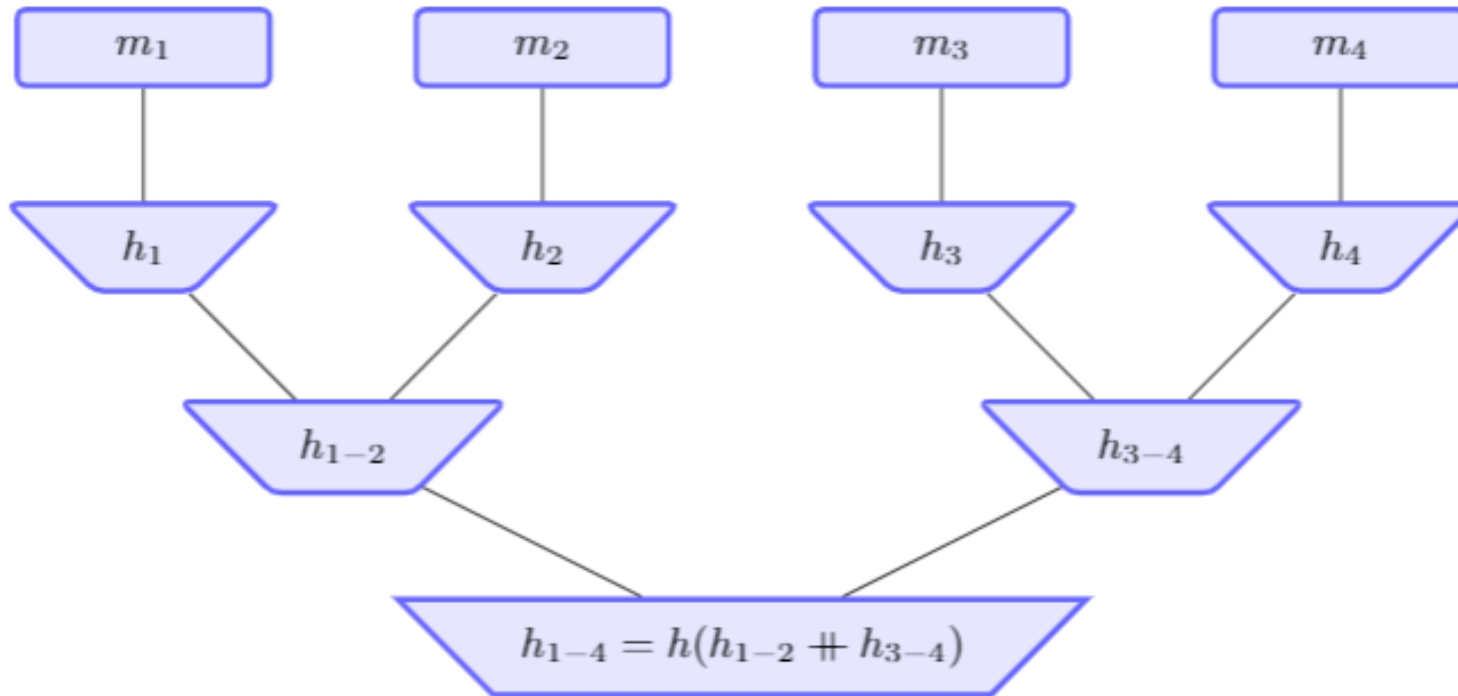
$$2lMT.VerPoI(d, m, i, \{x_i\}_{i=1}^l) \equiv \left[\begin{array}{l} \text{TRUE if } x_i = h(m), \text{ and} \\ d = h(x_1 \# \dots \# x_l) \end{array} \right]$$



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

The Merkle Tree Construction

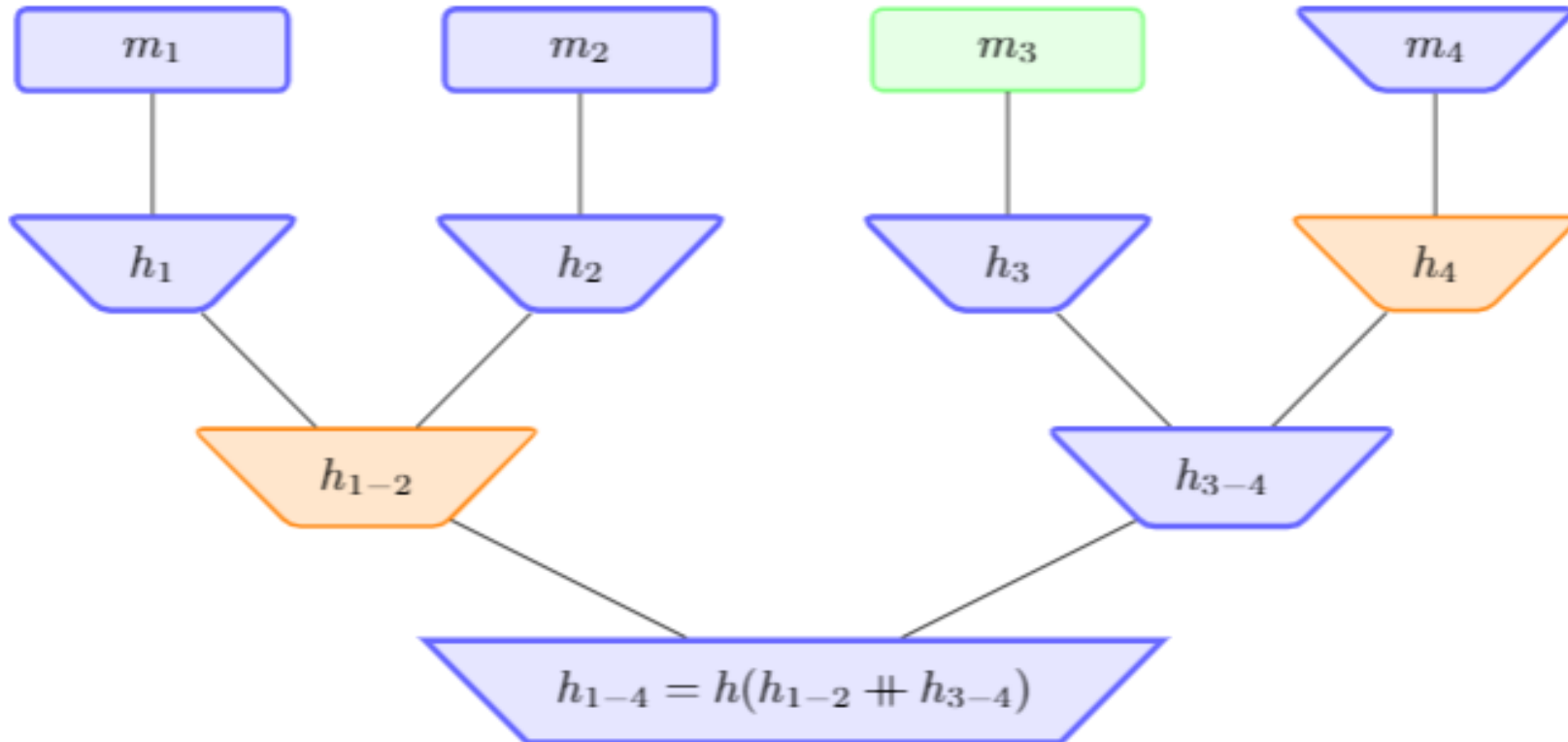
- Reduce length of 'proofs' – send less hashes of 'other msgs'



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

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 4
 - Sections 4.6, 4.7, and 4.8

Thank You!

