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

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 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 :

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

- $opad, ipad$: 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.

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 $\Delta: \{m_i \in \{0,1\}^*\} \rightarrow \{0,1\}^n$
 - 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, VerPol will output 1.
 - Same for PoC.
- For PoI: security means that a PPT adversary cannot forge a valid PoI for a message that is not in 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

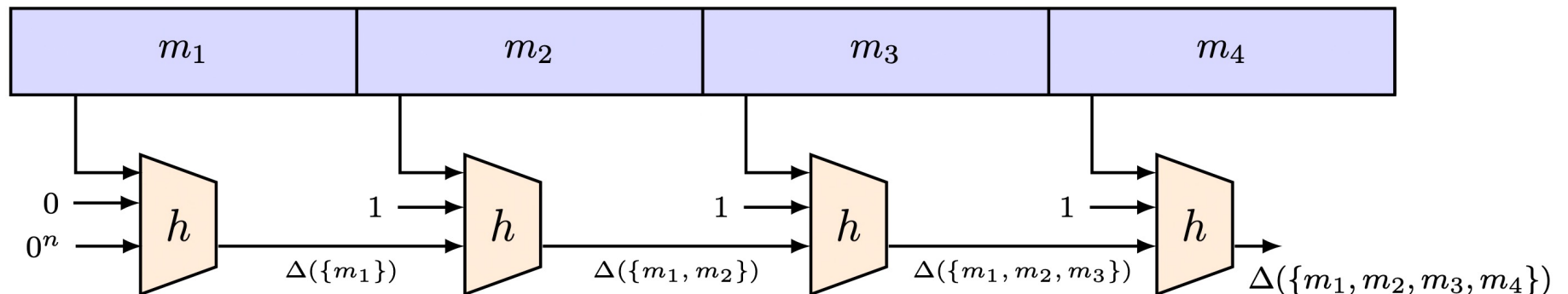
- Merkle-Damgard accumulators.
- Merkle trees.
- Blockchains.

The Merkle-Damgard Accumulator

- Idea: hash iteratively, message by message:

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

- Each message is a block of length n bits.
 - Put differently, a long message is divided into blocks each of length n bits.
- 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; avoids the requirement of having the message as an integer number of blocks.
- Let $pad(x) = 1 || 0^k || bin(|x|)$; $x' = x || pad(x)$
 - Where $bin(|x|)$ is the n -bit binary representation of $|x|$
- For $i=1, \dots, l$, where $l = |x'|/n$, and let x'_i be 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 is 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:

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

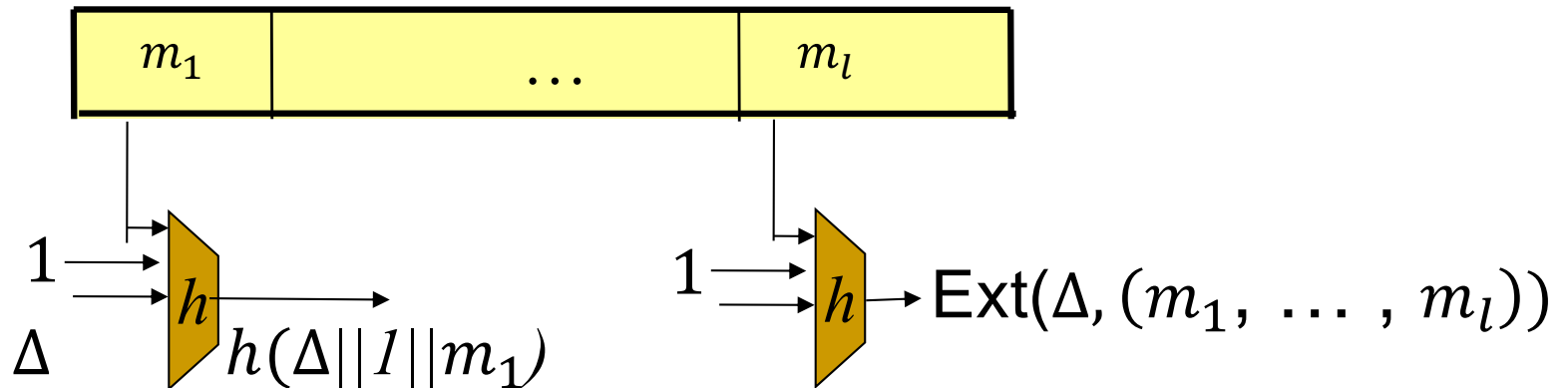
The Merkle-Damgard Extend Function

- We can define Extend for Merkle-Damgard:

- Idea: Just continue last digest!

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

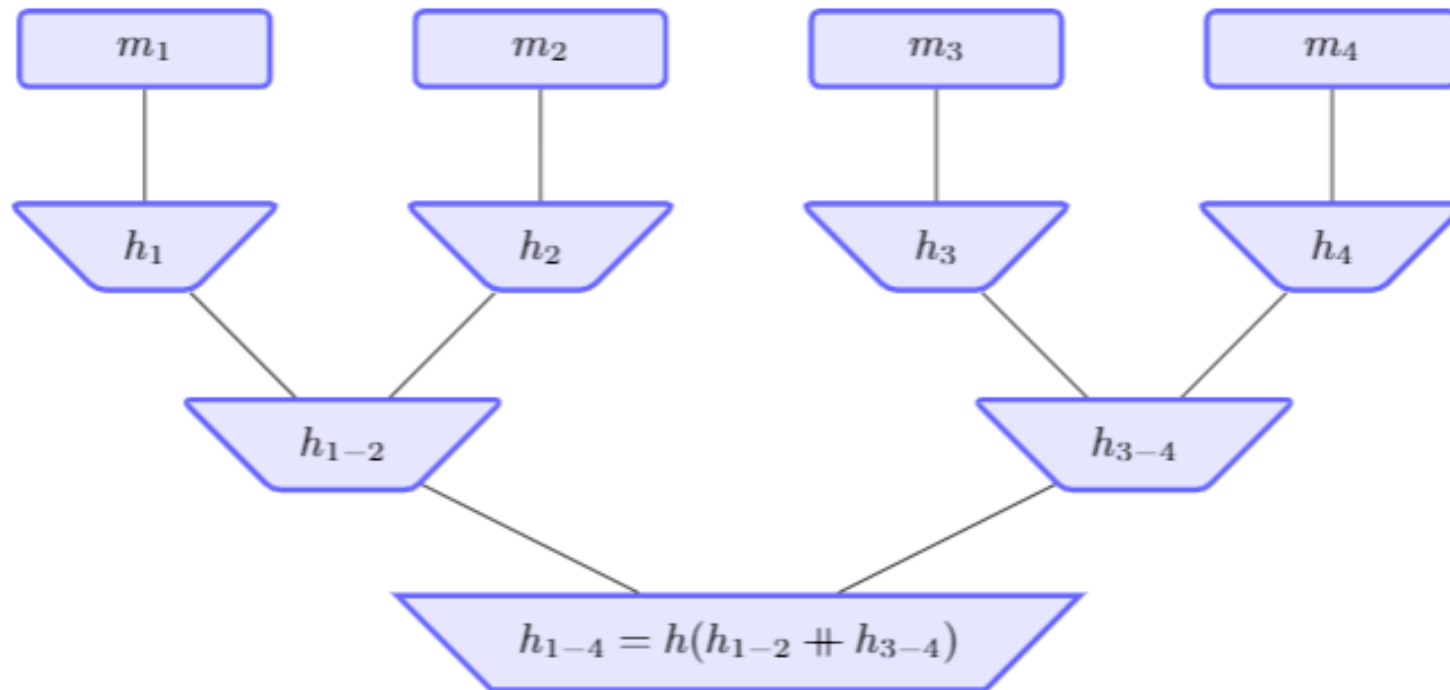
- Not secure to be used to construct a MAC! **Why?**



The Merkle Tree Accumulator

- Construct a tree of digests
- Short digest validates integrity of large objects.
 - Often, an object consists of multiple files or data pieces.
- Integrity relies on verifying the root of the tree
- It enables efficient proof of inclusion
 - Whether proof size or time needed to verify the proof.
 - It is logarithmic in the number of leaves (or files or data pieces).
- It also offers a privacy advantage
 - a prover only reveals their file but not those of others.
- Widely used in practice!
- Generally, it supports accumulate, Pol, and verify Pol functions.

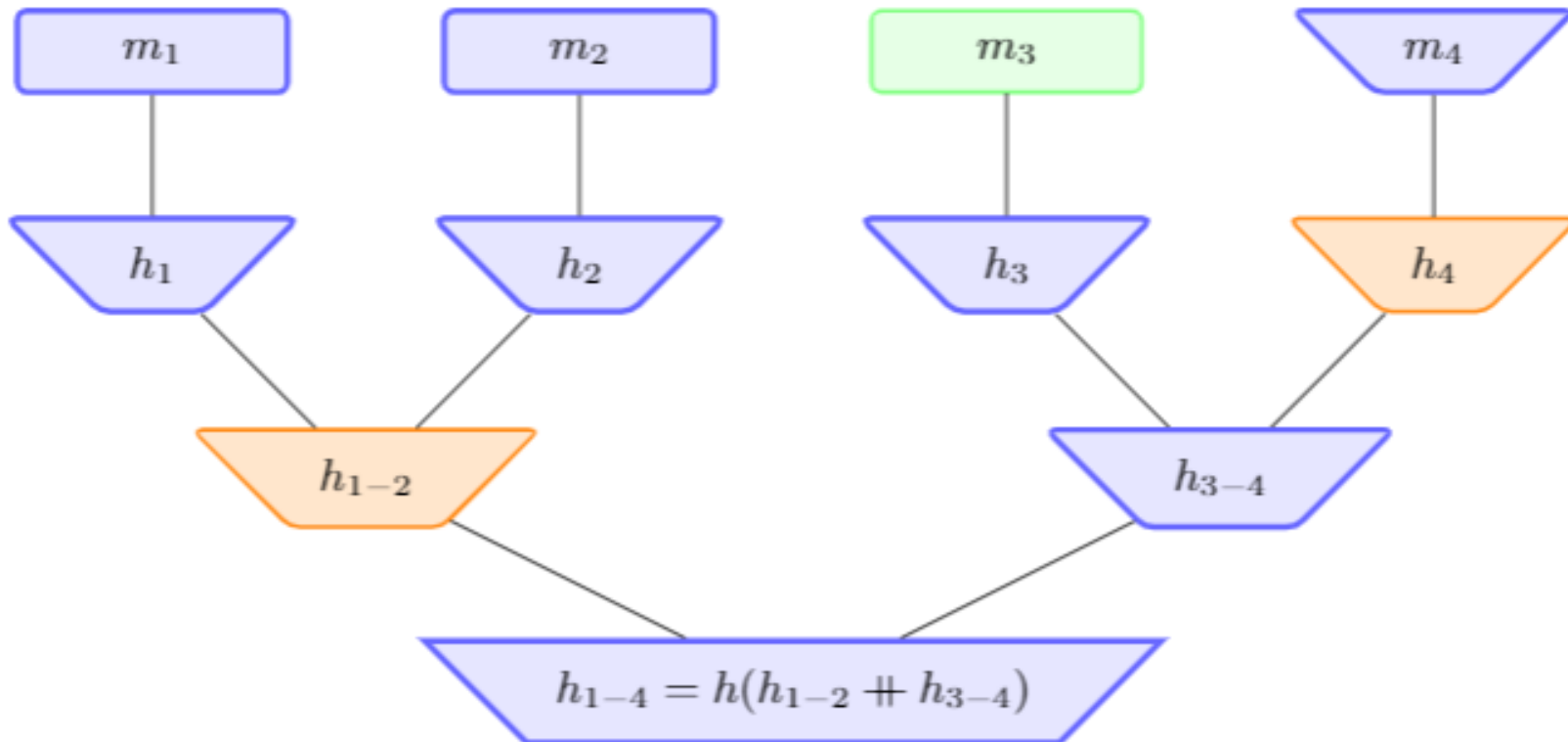
The Merkle Tree Accumulator



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

Merkle Tree: Proof of Inclusion (PoI)

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



The Merkle Tree Accumulator

- How many levels in a Merkle tree?
- What is the size of the Pol in terms of number of digests?
- Does the order of the messages (i.e., leaf nodes) matter? Why?

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!

