

**Applied Cryptography and Computer  
Security  
CSE 664 Spring 2017**

**Lecture 10: Applications of Hash Functions**

**Department of Computer Science and Engineering  
University at Buffalo**

# Overview

- **Going back to our security goal of data integrity:**
  - **theoretical MAC constructions**
  - **CBC-MAC**
  - **HMAC**
- **Sample applications of hash functions**

# MAC Algorithms

- **Constructing a MAC algorithm from a hash function**
  - **one approach is to include the key  $k$  as part of the hash function**  
**input:  $\text{Mac}_k(x) = h(k||x)$**
  - **if the hash function is one-way, we won't be able to recover the key**
  - **but how about MAC forgery?**

# MAC Algorithms

- $\text{Mac}_k(x) = h(k||x)$ 
  - **assume we have a message**  $m = m_1m_2\dots m_t$
  - **consider an iterated hash function:**  $h_0 = IV, h_i = f(m_i, h_{i-1});$   
 $h(x) = h_t$
  - **then we can extend  $m$  by an arbitrary single block  $b$  and compute the MAC on  $m' = m_1m_2\dots m_tb$**
  - **compute  $\text{Mac}_k(m') = h(k||m||b)$  as  $f(\text{Mac}_k(m), b)$**
- **What if we construct a MAC from a hash function using the key  $k$  as the IV for the compression function?**

# MAC Algorithms

- **Hash-Based MAC – HMAC**
- **Goals:**
  - use available hash functions without modifications
  - preserve the original performance of the hash function
  - use and handle keys in a simple way
  - allow replacement of the underlying hash function
  - have a well-understood cryptographic analysis of its strength

# MAC Algorithms

- **HMAC**

- $\text{HMAC}_k(x) = h((K \oplus \text{opad}) || h((K \oplus \text{ipad}) || x))$
- $K$  is the key  $k$  padded to a full block
- $\text{ipad} = 0\mathbf{x}3636\dots36$  and  $\text{opad} = 0\mathbf{x}5C5C\dots5C$  are fixed padding constants

- **Properties of HMAC:**

- efficient
- security is related to that of the underlying hash function
  - we want  $k_1 = h(K \oplus \text{opad})$  and  $k_2 = h(K \oplus \text{ipad})$  to be rather independent and close to random
  - then HMAC is existentially unforgeable under an adaptive chosen-message attack for messages of any length

# MAC Algorithms

- **HMAC Security**
  - provides greater security than the security of the underlying hash function
  - no known practical attacks if a secure hash function is used and according to the specifications
- In general, HMAC can be attacked by:
  - brute force search on the key space
  - attacks on the hash function

## Other Uses of Hash Functions

- **Hash Chains**

- a method for authenticating multiple user logins or packet streams
- consists of successive application of a hash function to a string
- $n$  applications of the hash function on  $x$  is denoted by  $h^n(x)$
- this produces a hash chain of length  $n$

- **Example:**

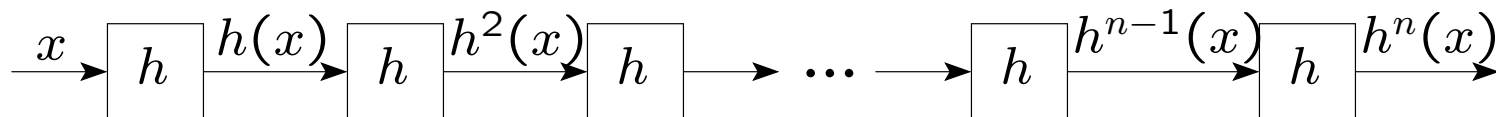
- $h^4(x) = h(h(h(h(x))))$  produces a hash chain of length 4



# Uses of Hash Functions

- **Authentication using hash chains**

- user generates a hash chain of length  $n$
- at time 1, the user sends  $auth_1 = h^n(x)$  (and possibly authenticates it through other means)
- the recipient stores  $auth = auth_1$
- at time 2, the user sends  $auth_2 = h^{n-1}(x)$
- the recipient checks whether  $h(auth_2) = auth_1$  and, if so, accepts
- the recipient updated  $auth = auth_2$
- etc.



# Uses of Hash Functions

- **Why is such authentication secure?**
- **Authentication in packet streams**
  - we can similarly authenticate each packet as belonging to the stream
  - need to take into account packet delivery delay
  - a packet authentication value is opened several packets later
  - see Perrig et al. “Efficient Authentication and Signing of Multicast Streams over Lossy Channels” (2000) for more information

# Uses of Hash Functions

- **Merkle Hash Tree**

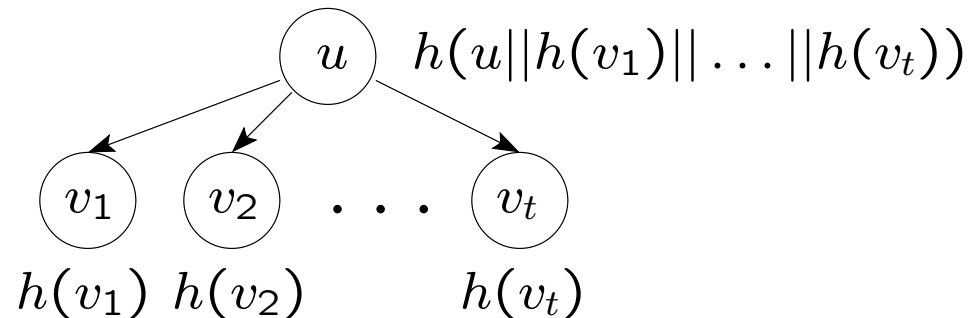
- integrity verification mechanism for hierarchically structured documents or databases
- the technique works on trees only
- the hash of the tree is computed in the bottom-up fashion

- **Generation of a Merkle hash tree**

- for a leaf node  $v$ , simply compute its hash  $h(v)$
- for a non-leaf node  $u$  with children  $v_1, \dots, v_t$ , compute its hash as  $h(u || h(v_1) || \dots || h(v_t))$

# Uses of Hash Functions

- **Merkle Hash Tree**



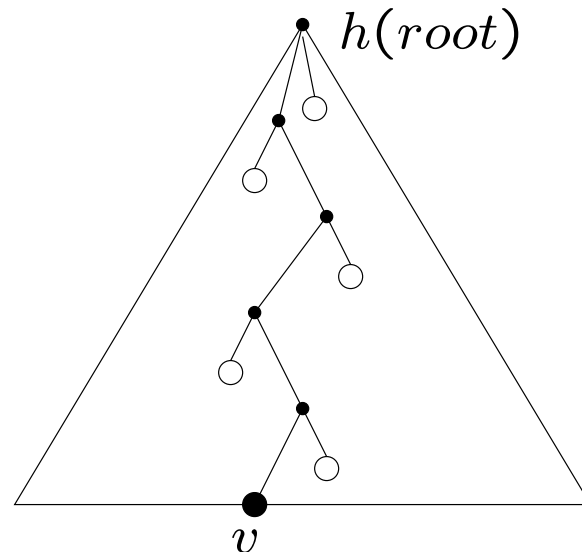
- **this computation continues until the hash of the root is computed**
- **the hash of the root corresponds to the hash of the entire tree**

- **Integrity verification**

- **node integrity verification is much faster than hashing the entire tree**
- **to check node  $v$ , obtain hashes of the nodes on the path from  $v$  to the root**

# Uses of Hash Functions

- **Integrity verification in Merkle Hash Tree**



- your node
- hash is given
- compute the hash

- **compute the hash of  $v$  and combine it with other hashes on the path to the root**
- **compare your hash of the root with what you are given**
- **the node you are authenticating doesn't have to be a leaf**

# Uses of Hash Functions

- **Merkle Hash Tree**
  - **why does this work?**
  - **what are the computation savings compared to just applying the hash function to the entire tree?**
  - **what needs to be done when a node's content changes?**

## Confidentiality + Integrity

- **How do we use a MAC in combination with encryption?**

- **message authentication**

- $A \xrightarrow{m, \text{Mac}_k(m)} B$

- **encrypt and authenticate**

- $A \xrightarrow{\text{Enc}_{k_1}(m), \text{Mac}_{k_2}(m)} B$

- **authenticate then encrypt**

- $A \xrightarrow{\text{Enc}_{k_1}(m, \text{Mac}_{k_2}(m))} B$

- **encrypt then authenticate**

- $A \xrightarrow{\text{Enc}_{k_1}(m), \text{Mac}_{k_2}(\text{Enc}_{k_1}(m))} B$

## Confidentiality + Integrity

- Which construction is good for achieving both objectives?
  - how do we define “good”?
- We want a combination that **always** achieves both confidentiality and integrity
  - given any CPA-secure encryption scheme and any secure MAC scheme, the construction must achieve both goals
  - if there are secure encryption and MAC schemes using which a construction doesn't achieve both goals, we say it is insufficient



## Confidentiality + Integrity

- **How do we combine two schemes into one?**
  - we are given encryption  $E = (\text{Gen}_E, \text{Enc}, \text{Dec})$  and MAC  $M = (\text{Gen}_M, \text{Mac}, \text{Vrfy})$
  - we build message transmission scheme  $T = (\text{Gen}, \text{EncMac}, \text{DecVrfy})$
- **Correctness** is defined as before
- **Security** is based on meeting the requirements of two experiments: authenticated communication and confidentiality experiments
  - there is a single authenticated communication experiment  $\text{AuthComm}_{\mathcal{A},T}(n)$

## Confidentiality + Integrity

- **Analysis of our constructions:**
  - **encrypt and authenticate**
    - transmitting  $\text{Mac}_{k_2}(m)$  may leak information about  $m$
  - **authenticate then encrypt**
    - has a chosen-ciphertext attack (limited version) against the general version
    - tampering with ciphertext might permit predictable changes to the encrypted content
  - **encrypt then authenticate**
    - satisfies the definition and is CCA-secure
- **The keys  $k_1$  and  $k_2$  must be different!**

# Summary

- **Hash functions** have many uses:
  - data integrity
  - data and user authentication
  - in various protocols as a one-way function
- **Combining confidentiality and integrity** requires care
- **Next time:**
  - public key cryptography!
  - number theory