Applied Cryptography and Computer Security
CSE 664 Spring 2020

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: $\mathsf{Mac}_k(x) = h(k||x)$
  - if the hash function is one-way, we won't be able to recover the key
  - but how about MAC forgery?

MAC Algorithms

- $\mathsf{Mac}_k(x) = h(k||x)$
  - assume we have a message $m = m_1 m_2 ... m_t$
  - 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 $m' = m_1 m_2 ... m_t b$
  - compute $\mathsf{Mac}_k(m') = h(k||m||b)$ as $f(\mathsf{Mac}_k(m), b)$
- 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
  - $\mathsf{HMAC}_k(x) = h((K \oplus opad)||h((K \oplus ipad)||x))$
  - K is the key k padded to a full block
  - $ipad = \texttt{0x3636...36}$ and $opad = \texttt{0x5C5C...5C}$ are fixed padding constants

- Properties of HMAC:
  - efficient
  - security is related to that of the underlying hash function
    - we want $k_1 = h(K \oplus opad)$ and $k_2 = h(K \oplus ipad)$ 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

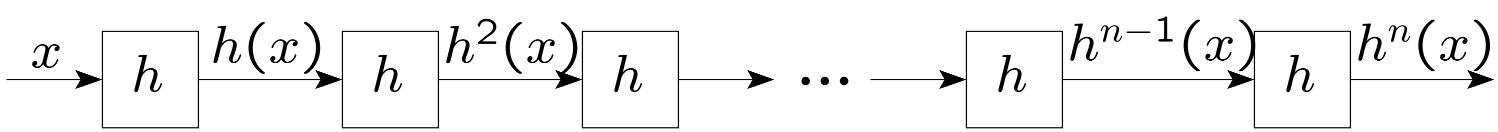
- HMAC Security
  - provides greater security than the security of the underlying hash function
  - no known practical attacks if a secure hash function is used and according to the specifications
- In general, HMAC can be attacked by:
  - brute force search on the key space
  - attacks on the hash function

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

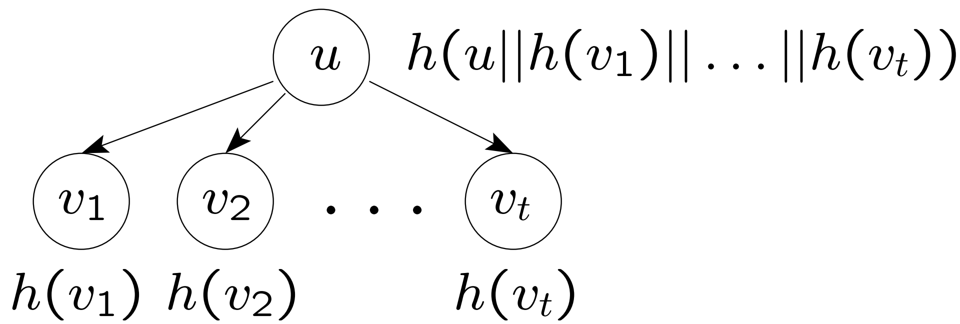
- Why is such authentication secure?
- Authentication in packet streams
  - we can similarly authenticate each packet as belonging to the stream
  - need to take into account packet delivery delay
  - a packet authentication value is opened several packets later
  - see Perrig et al. "Efficient Authentication 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 $v_1, \ldots, v_t$, compute its hash as $h(u||h(v_1)||\ldots||h(v_t))$

Uses of Hash Functions

- Merkle Hash Tree

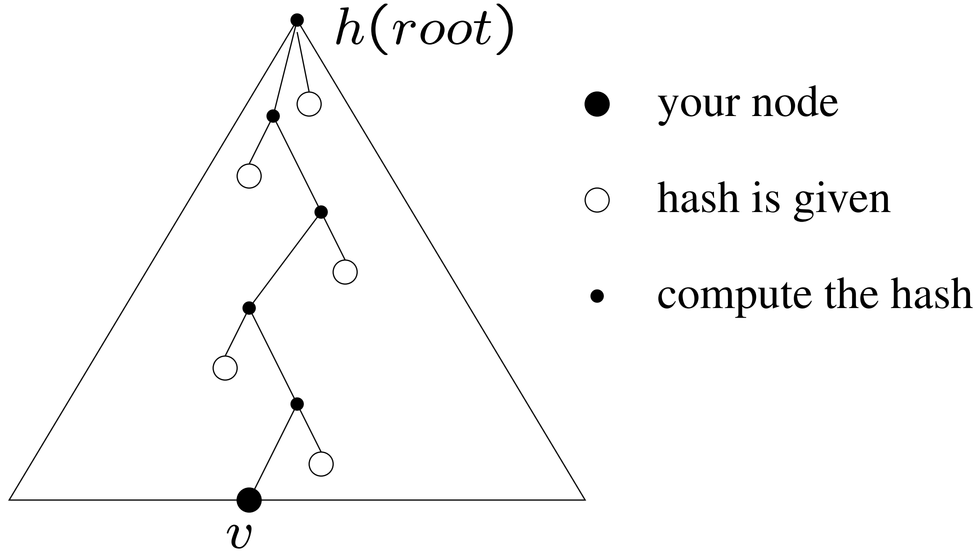


  - this computation continues until the hash of the root is computed
  - the hash of the root corresponds to the hash of the entire tree

- 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



  - 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?

Uses of Hash Functions

- Commitment schemes
  - a commitment scheme allows one to "commit" to a message m by computing a committed value com
  - it can later be opened to reveal m
  - the following properties are required to hold:
    - hiding property: commitment com reveals nothing about message m
    - binding property: it is infeasible to find another message $m' \neq m$ such that com can be opened to m'

Uses of Hash Functions

- A commitment scheme is defined by three algorithms
  - Gen: randomized algorithm that takes a security parameter 1^n and outputs public parameters params
  - Com: randomized algorithm that takes params and a message $m \in \{0,1\}^n$ and outputs commitment com
    - we make the randomness that Com uses explicit, denote it by r, and use $\mathsf{com} = \mathsf{Com}(\mathsf{param}, m, r)$
  - Open: a deterministic algorithm that decommits to m by typically disclosing m and r
    - the verifier that check whether com is in fact equal to $\mathsf{Com}(\mathsf{params}, m, r)$

Uses of Hash Functions

- The security properties of a commitment scheme can be fromally defined as two experiments
  - to achieve hiding, $\mathcal{A}$ chooses m_0, m_1, receives a commitment to one of them and is asked to determine which message was used
  - to achieve binding, $\mathcal{A}$ is challenged to create a commitment com and two different opennings for it (m, r) and (m', r')
- We require that a commitment scheme is secure if
  - the probability of succeeding in the hiding experiment is at most negligibly larger than 1/2 and
  - the success in the binding experiement is at most negligigle

Uses of Hash Functions

- We can use hash functions to create a commitment scheme (in the random oracle model):
  - Gen takes a security prameter 1^n and chooses an appropriate hash funcion h
  - to commit to m, choose uniform $r \in \{0,1\}^n$ and output $\mathsf{com} := h(m||r)$
  - hiding follows because adversary can query $h(*||r)$ with only a negligible probability
  - binding follows from the collision resistant property of h

Confidentiality + Integrity

- How do we use a MAC in combination with encryption?
  - message authentication
    - $A \xrightarrow{m,\mathsf{Mac}_k(m)} B$
  - encrypt and authenticate
    - $A \xrightarrow{\mathsf{Enc}_{k_1}(m),\mathsf{Mac}_{k_2}(m)} B$
  - authenticate then encrypt
    - $A \xrightarrow{\mathsf{Enc}_{k_1}(m,\mathsf{Mac}_{k_2}(m))} B$
  - encrypt then authenticate
    - $A \xrightarrow{\mathsf{Enc}_{k_1}(m),\mathsf{Mac}_{k_2}(\mathsf{Enc}_{k_1}(m))} B$

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 $E = (\mathsf{Gen}_E, \mathsf{Enc}, \mathsf{Dec})$ and MAC $M = (\mathsf{Gen}_M, \mathsf{Mac}, \mathsf{Vrfy})$
  - we build message transmission scheme $T = (\mathsf{Gen}, \mathsf{EncMac}, \mathsf{DecVrfy})$
- 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 $\mathsf{AuthComm}_{\mathcal{A},T}(n)$

Confidentiality + Integrity

- Analysis of our constructions:
  - encrypt and authenticate
    - transmitting $\mathsf{Mac}_{k_2}(m)$ may leak information about m
  - authenticate then encrypt
    - has a chosen-ciphertext attack against the general version, which has been successfully applied in practice
    - 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!

Authenticated Encryption

- Do I have to use encryption and MAC separately or are there authenticated encryption modes?
  - recently, authenticated encryption modes have been proposed
- Some good reads:
  - https://blog.cryptographyengineering.com/2012/05/19/how-to-choose-authenticated-encryption/
  - https://stackoverflow.com/questions/1220751/how-to-choose-an-aes-encryption-mode-cbc-ecb-ctr-ocb-cfb

Authenticated Encryption

- Good options to consider:
  - Offset Codebook (OCB) mode
    - state of the art in authenticated encryption
    - proposed internet standard
    - has licensing restrictions
    - see http://web.cs.ucdavis.edu/~rogaway/ocb/ocb-faq.htm for more information
  - Galois/Counter Mode (GCM)
    - does not have licensing restrictions
    - can be used as an alternative for commercial software

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