CS396: Security, Privacy & Society

Fall 2022

Lecture 10: MACs & Hashing

Instructor: Abrar Alrumayh

October 3, 2022

Outline

- Hash functions
  - Collision resistance (CR)
  - Design framework
  - Generic attacks
  - Applications of hashing to cryptography

Hash functions

Recall from algorithms/data-structures

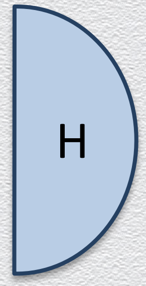
- store a small number of elements coming from a large set.
- example: store $m = n^2$ values where each value is a string of length n.
- total strings to be stored are few in comparison to the full set of 2^n elements
- **deterministic method** to quickly store and "look-up" elements
- Want: low collisions (otherwise, useless)

Cryptographic hash functions

Basic cryptographic primitive

- maps **“objects”** to a **fixed-length** binary strings
- core security property: mapping **avoids collisions**
    - **collision**: distinct objects ($x \neq y$) are mapped to the same hash value ($H(x) = H(y)$)
    - although collisions **necessarily exist**, they are **infeasible to find**

input
arbitrarily long string



output
short digest, fingerprint, “secure” description

Important role in modern cryptography

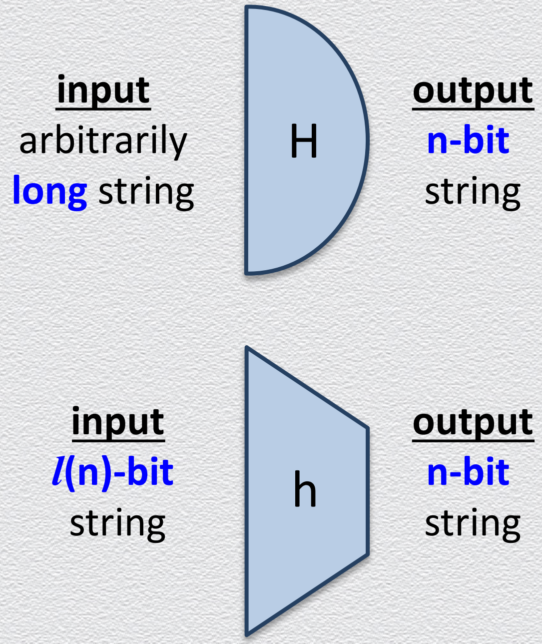
- lie between symmetric- and asymmetric-key cryptography
- capture different security properties of “idealized random functions”
- qualitative stronger assumption than PRF

Hash & compression functions

Map messages to short digests

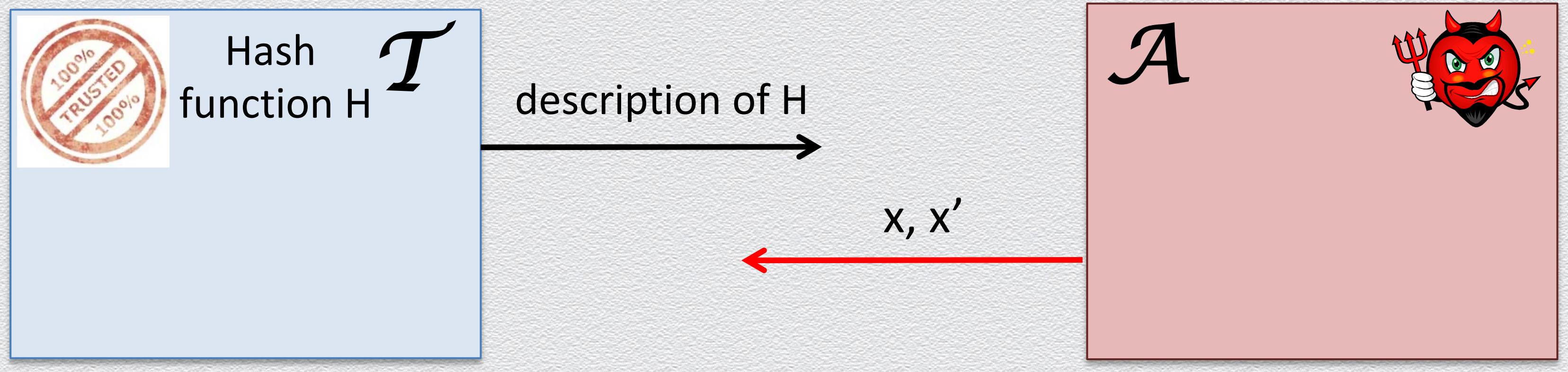
- a **general** hash function H() maps
  - a message of an **arbitrary length** to a **n-bit** string

- a **compression** (hash) function h() maps
  - a **long** binary string to a **shorter** binary string
  - an ***l*(n)-bit string** to a **n-bit string**, with ***l*(n) > n**



Collision resistance (CR)

Attacker wins the game if $x \neq x'$ & $H(x) = H(x')$



H is collision-resistant if any PPT $\mathcal{A}$ wins the game only negligibly often.

Weaker security notions

Given a hash function H: $X \rightarrow Y$, then we say that H is

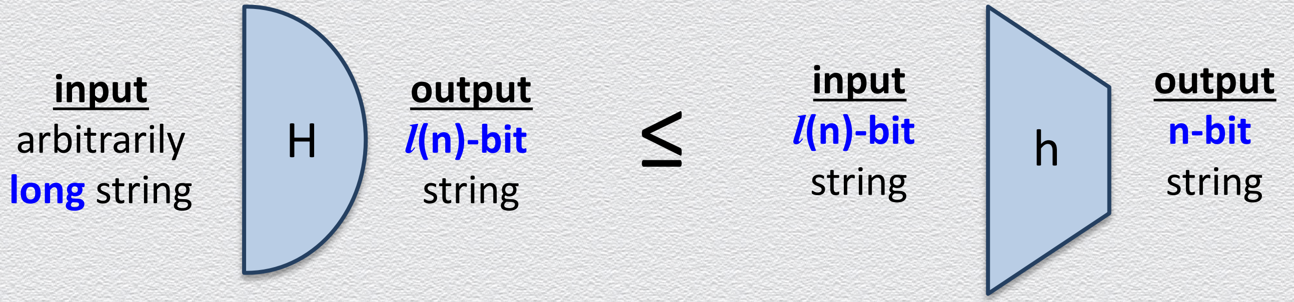
- **preimage resistant** (or **one-way**)
  - if given $y \in Y$, finding a value $x \in X$ s.t. $H(x) = y$ happens negligibly often
- **2-nd preimage resistant** (or **weak collision resistant**)
  - if given a **uniform** $x \in X$, finding a value $x' \in X$, s.t. $x' \neq x$ and $H(x') = H(x)$ happens negligibly often
- **collision resistant** (or **strong collision resistant**)
  - if finding two distinct values $x', x \in X$, s.t. $H(x') = H(x)$ happens negligibly often

Design framework

Domain extension via the Merkle-Damgård transform

General design pattern for cryptographic hash functions

- reduces CR of general hash functions to CR of compression functions



- thus, in practice, it suffices to realize a collision-resistant compression function h
- compressing by 1 single bit is a least as hard as compressing by any number of bits!

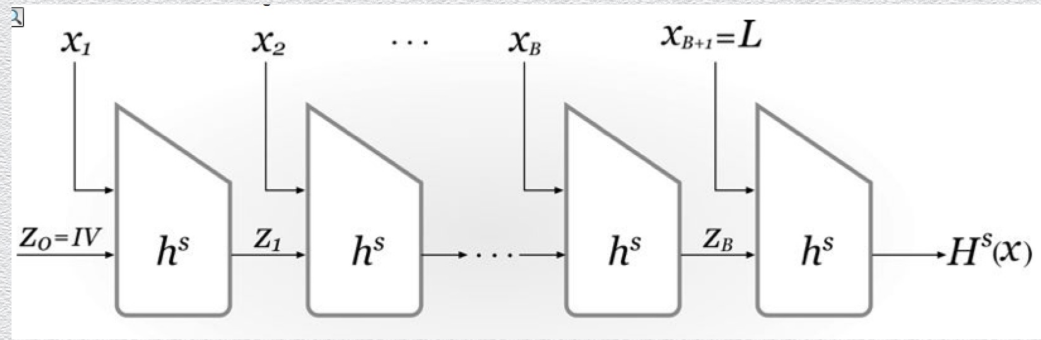
Merkle-Damgård transform: Design

Suppose that h: $\{0,1\}^{2n} \rightarrow \{0,1\}^n$ is a collision-resistant compression function

Consider the general hash function H: $\mathcal{M} = \{x : |x| < 2^n\} \rightarrow \{0,1\}^n$, defined as

Merkle-Damgård design

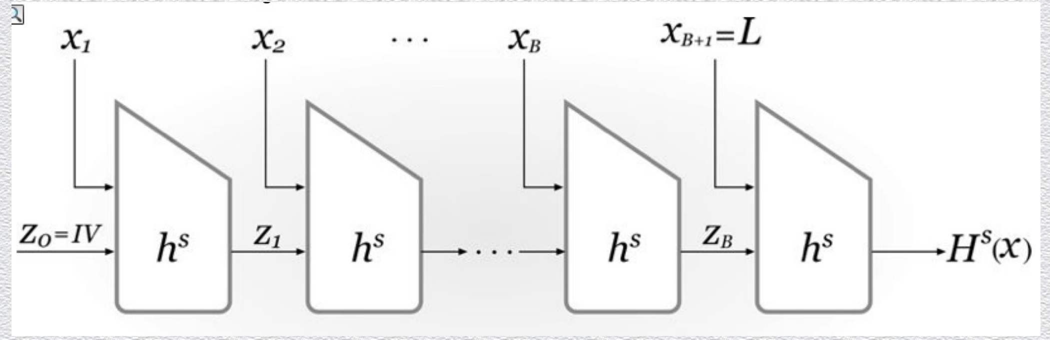
- H(x) is computed by applying h() in a **"chained" manner** over n-bit message blocks



  - pad x to define a number, say B, **message blocks $x_1, \ldots, x_B$**, with $|x_i| = n$
  - set extra, final, message block **x_{B+1} as an n-bit encoding L of $|x|$**
  - starting by initial digest **$z_0 = IV = 0^n$**, output **$H(x) = z_{B+1}$**, where **$z_i = h^s(z_{i-1}||x_i)$**

Merkle-Damgård transform: Security

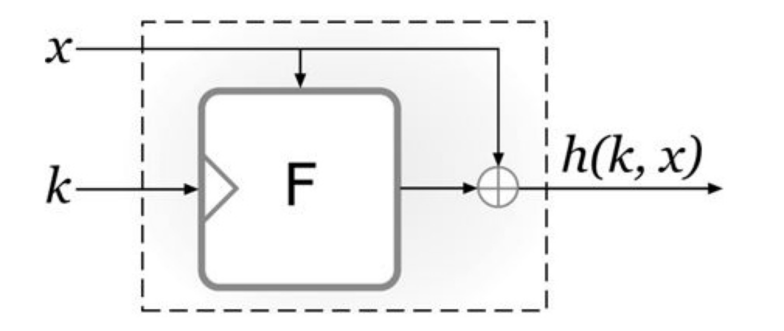
If the compression function h is CR,
then the derived hash function H is also CR!



Compression function design: The Davies-Meyer scheme

Employs PRF w/ key length m & block length n

- define h: $\{0,1\}^{n+m} \rightarrow \{0,1\}^n$ as **$h(x||k) = F_k(x)$ XOR x**



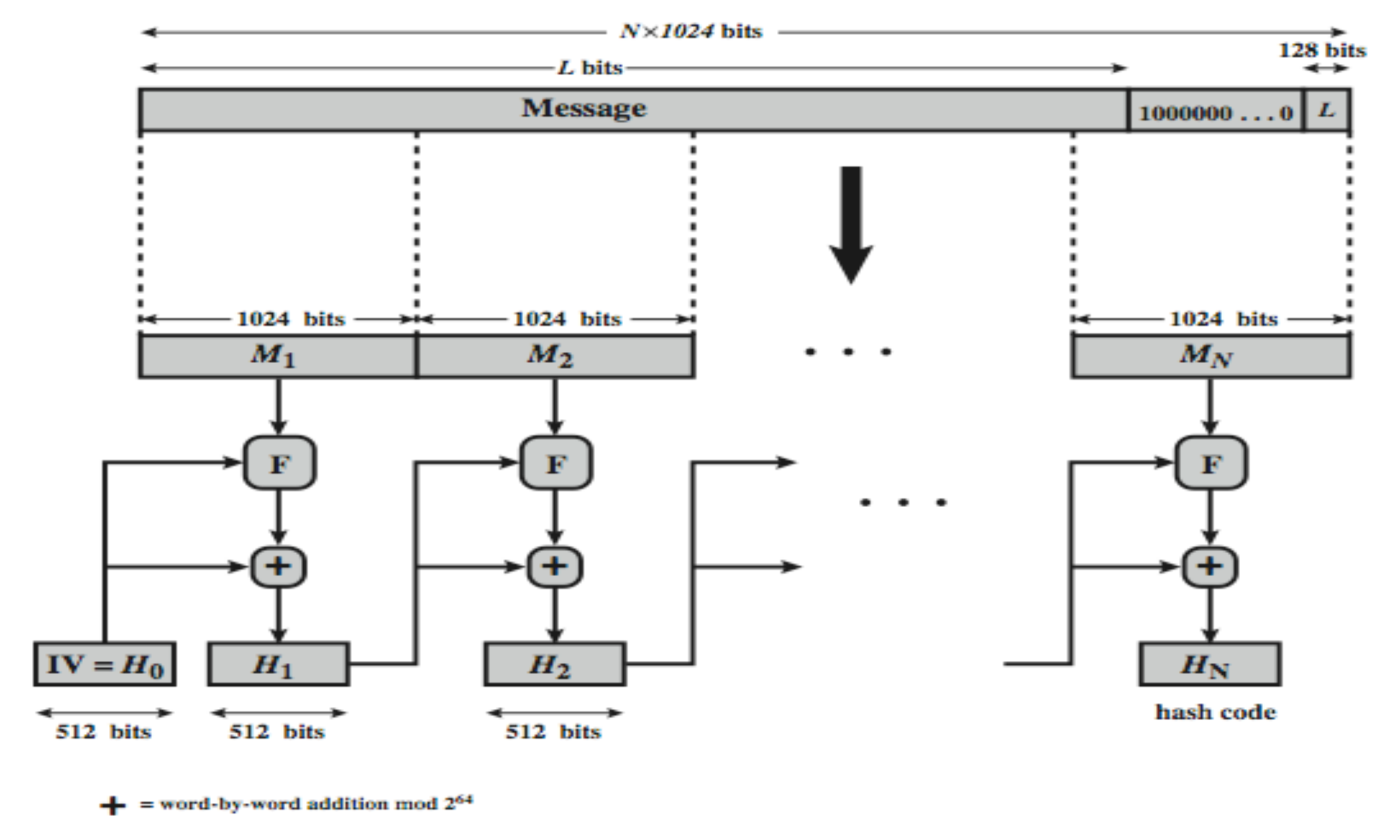
Security

- h is CR, if F is an **ideal cipher**

Well known hash functions

- MD5 (designed in 1991)
  - output 128 bits, collision resistance **completely broken** by researchers in 2004
  - today (controlled) collisions can be found in less than a minute on a desktop PC
- SHA1 – the Secure Hash Algorithm (series of algorithms standardized by NIST)
  - output 160 bits, considered **insecure** for collision resistance
  - **broken** in 2017 by researchers at CWI
- SHA2 (SHA-224, SHA-256, SHA-384, SHA-512)
  - outputs 224, 256, 384, and 512 bits, respectively, **no real security concerns yet**
  - based on Merkle-Damgård + Davies-Meyer generic transforms
- SHA3 (Kessac)
  - **completely new philosophy** (sponge construction + unkeyed permutations)

SHA-2-512 overview



Current hash standards

Algorithm	Maximum Message Size (bits)	Block Size (bits)	Rounds	Message Digest Size (bits)
MD5	2^{64}	512	64	128
SHA-1	2^{64}	512	80	160
SHA-2-224	2^{64}	512	64	224
SHA-2-256	2^{64}	512	64	256
SHA-2-384	2^{128}	1024	80	384
SHA-2-512	2^{128}	1024	80	512
SHA-3-256	unlimited	1088	24	256
SHA-3-512	unlimited	576	24	512

Generic attacks

Generic attacks against cryptographic hashing

Assume a CR compression function $h : \{0,1\}^{l'(n)} \rightarrow \{0,1\}^{l(n)}$

- **brute-force** attack
  - for each string x in the domain
    - compute and record hash value h(x)
    - if h(x) equals a previously recorded hash h(y) (i.e., $x \neq y$ but $h(x)=h(y)$), halt and output collision on $x \neq y$
- **birthday** attack
  - surprisingly, a more efficient generic attack exists!

Birthday paradox

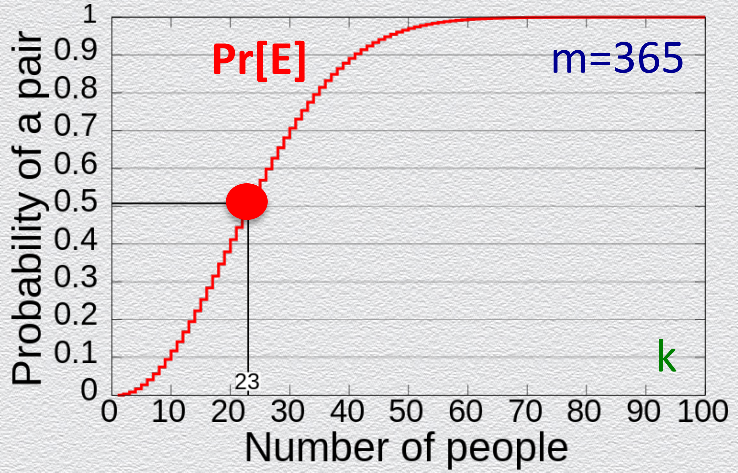
"In any group of 23 people (or more), it is **more likely** (than not) that **at least two** individuals have their birthday on the **same** day"

- based on probabilistic analysis of a random "balls-into-bins" experiment:

  "k balls are each, independently and randomly, thrown into one out of m bins"

- captures likelihood that event E = "**two balls land into the same bin**" occurs

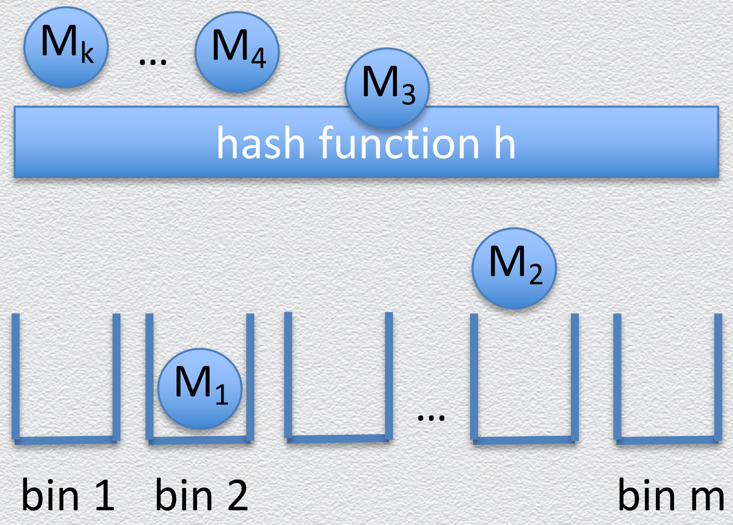
- analysis shows: $\Pr[E] \approx 1 - e^{-k(k-1)/2m}$ (1)
  - if **Pr[E] = 1/2**, Eq. (1) gives $\mathbf{k \approx 1.17\ m^{1/2}}$
  - thus, for m = 365, k is around 23 (!)
    - assuming a uniform birth distribution



Birthday attack

Applies “birthday paradox” against cryptographic hashing

- exploits the likelihood of finding collisions for hash function h using a **randomized** search, rather than an **exhausting** search
- analogy
  - k balls: distinct messages chosen to hash
  - m bins: number of possible hash values
  - independent & random throwing
    - how is this achieved?
    - message selection, hash mapping



Probabilistic analysis

Experiment

- k balls are each, independently and randomly, thrown into one out of m bins

Analysis

- the probability that the i-th ball lands in an empty bin is: $1 - (i - 1)/m$
- the probability F_k that after k throws, no balls land in the same bin is:

$$F_k = (1 - 1/m)\,(1 - 2/m)\,(1 - 3/m)\,\ldots\,(1 - (k - 1)/m)$$

- by the standard approximation $1 - x \approx e^{-x}$: $F_k \approx e^{-(1/m + 2/m + 3/m + \ldots + (k-1)/m)} = e^{-k(k-1)/2m}$
- thus, two balls land in same bin with probability $\Pr[E] = 1 - F_k = 1 - e^{-k(k-1)/2m}$
- **lower bound** – Pr[E] increases if the bin-selection distribution is not uniform

What birthday attacks mean in practice…

- # hash evaluations for finding collisions on n-bit digests with probability p

Bits n	Possible outputs (2 s.f.) (H) m	Desired probability of random collision (2 s.f.) (p)									
		10^{-18}	10^{-15}	10^{-12}	10^{-9}	10^{-6}	0.1%	1%	25%	50%	75%
16	65,536	<2	<2	<2	<2	<2	11	36	190	300	430
32	4.3×10^{9}	<2	<2	<2	3	93	2900	9300	50,000	77,000	110,000
64	1.8×10^{19}	6	190	6100	190,000	6,100,000	1.9×10^{8}	6.1×10^{8}	3.3×10^{9}	5.1×10^{9}	7.2×10^{9}
128	3.4×10^{38}	2.6×10^{10}	8.2×10^{11}	2.6×10^{13}	8.2×10^{14}	2.6×10^{16}	8.3×10^{17}	2.6×10^{18}	1.4×10^{19}	2.2×10^{19}	3.1×10^{19}
256	1.2×10^{77}	4.8×10^{29}	1.5×10^{31}	4.8×10^{32}	1.5×10^{34}	4.8×10^{35}	1.5×10^{37}	4.8×10^{37}	2.6×10^{38}	4.0×10^{38}	5.7×10^{38}
384	3.9×10^{115}	8.9×10^{48}	2.8×10^{50}	8.9×10^{51}	2.8×10^{53}	8.9×10^{54}	2.8×10^{56}	8.9×10^{56}	4.8×10^{57}	7.4×10^{57}	1.0×10^{58}
512	1.3×10^{154}	1.6×10^{68}	5.2×10^{69}	1.6×10^{71}	5.2×10^{72}	1.6×10^{74}	5.2×10^{75}	1.6×10^{76}	8.8×10^{76}	1.4×10^{77}	1.9×10^{77}

- for large $m = 2^n$, average # hash evaluations before finding the first collision is

$1.25(m)^{1/2}$

Overall

Assume a CR function h producing hash values of size n

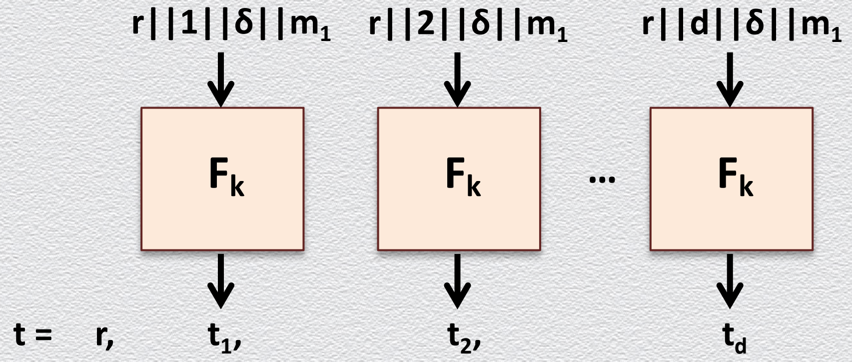
- **brute-force** attack
  - evaluate h on **$2^n + 1$** distinct inputs
  - by the "pigeon hole" **principle**, at least 1 collision **will be** found
- **birthday** attack
  - evaluate h on (much) **fewer** distinct inputs that hash to **random** values
  - by "balls-into-bins" **probabilistic analysis**, at least 1 collision will **likely** be found
  - when hashing **only half** distinct inputs, it's **more likely** to find a collision!
  - thus, in order to get **n-bit security**, we (**at least**) need **hash values of length 2n**

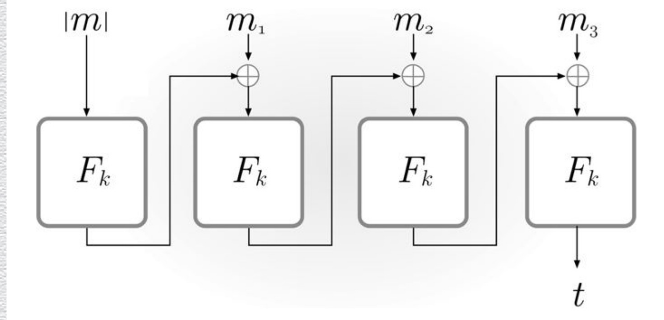
Applications of hashing to cryptography

Hash functions enable efficient MAC design!

Back to problem of designing **secure MAC for messages of arbitrary lengths**

- so far, we have seen two solutions
  - block-based “tagging”
    - based on PRFs
    - inefficient
  - CBC-MAC
    - also based on PRFs
    - more efficient

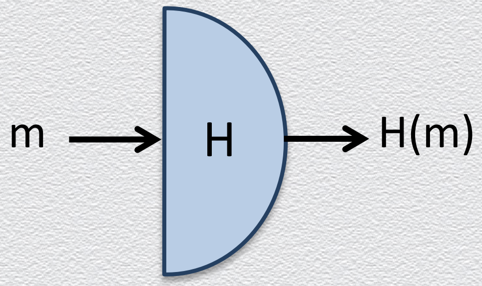


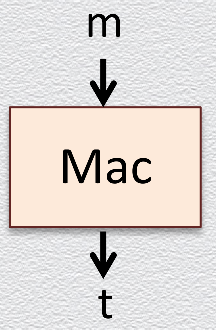


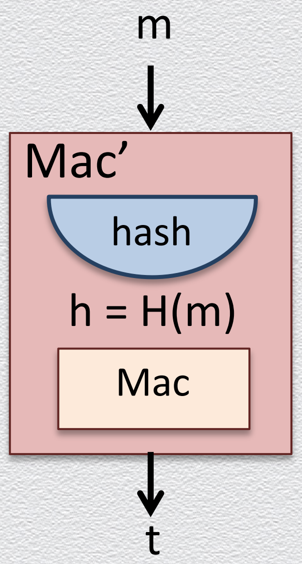
[1] Hash-and-MAC: Design

Generic method for designing **<u>secure MAC for messages of arbitrary lengths</u>**

- based on **CR hashing** and **<u>any</u>** **fix-length secure MAC**







- new MAC (Gen', Mac', Vrf') **<u>as the name suggests</u>**
  - Gen': **<u>instantiate</u>** H and Mac_k with key k
  - Mac': **<u>hash</u>** message m into h = H(m), output **<u>Mac_k</u>**-tag t on h
  - Vrf': **<u>canonical</u>** verification

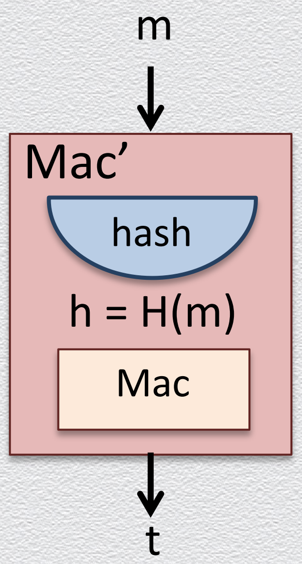
[1] Hash-and-MAC: Security

The Hash-and-MAC construction is a secure as long as

- H is **collision resistant**; and
- the underlying MAC is **secure**

Intuition

- since **H is CR**:
  authenticating **digest H(m)** is **a good as** authenticating **m itself**!



[2] Hash-based MAC

- so far, MACs are based on block ciphers
- can we construct a MAC based on CR hashing?

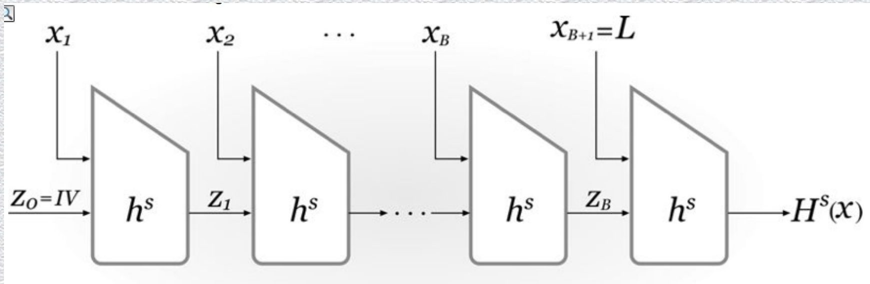
[2] A naïve, insecure, approach

Set tag t as:

$$\mathrm{Mac}_k(m) = \mathbf{H}(k||\mathbf{m})$$

- intuition: given H(k||m) it should be infeasible to compute H(k||m'), m' ≠ m

Insecure construction



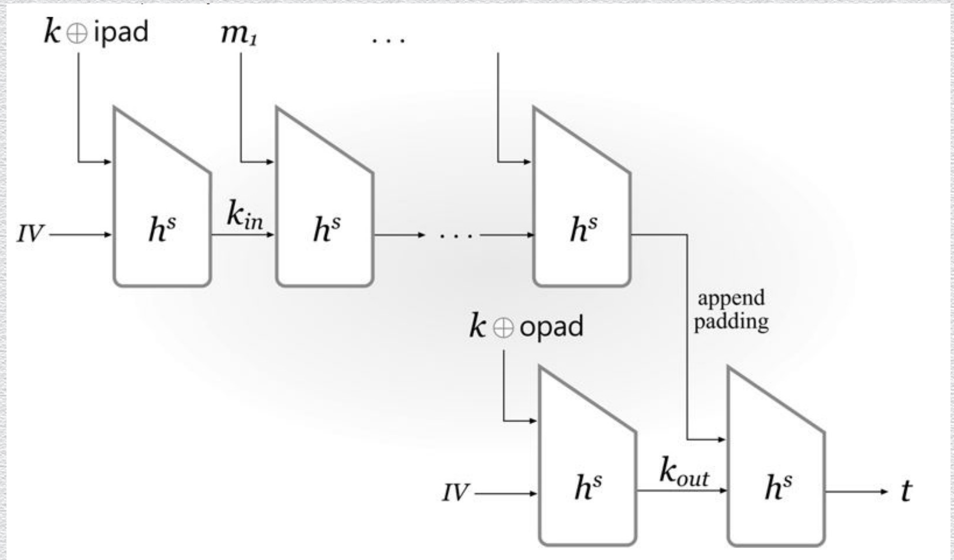
- practical CR hash functions employ the Merkle-Damgård design
- **length-extension attack**
  - knowledge of $H(m_1)$ makes it feasible to compute $H(m_1||m_2)$
  - by knowing the length of m_1, one can learn internal state z_B even without knowing m_1!

[2] HMAC: Secure design

Set tag t as:

$$\mathrm{HMAC}_k[m] \;=\; H\,[\;(k \oplus \mathrm{opad})\;||\;H[\;(k \oplus \mathrm{ipad})\;||\;m\;]\;]$$

- intuition: **instantiation of hash & sign paradigm**
- two layers of hashing H
  - **upper layer**
    - y = H($(k \oplus \mathrm{ipad})$ || m)
    - y = H'(m), i.e., "hash"
  - **lower layer**
    - t = H ($(k \oplus \mathrm{opad})$ || y')
    - t = Mac'(k_{out}, y'), i.e., "sign"



[2] HMAC: Security

If used with a secure hash function and according to specs, HMAC is secure

- no practical attacks are known against HMAC