CS 4920/5920 Applied Cryptography

Chapter 11 Cryptographic Hash Functions

Hash Function

Functions transforming large input (variable size) to small fixed output

h = H(M)

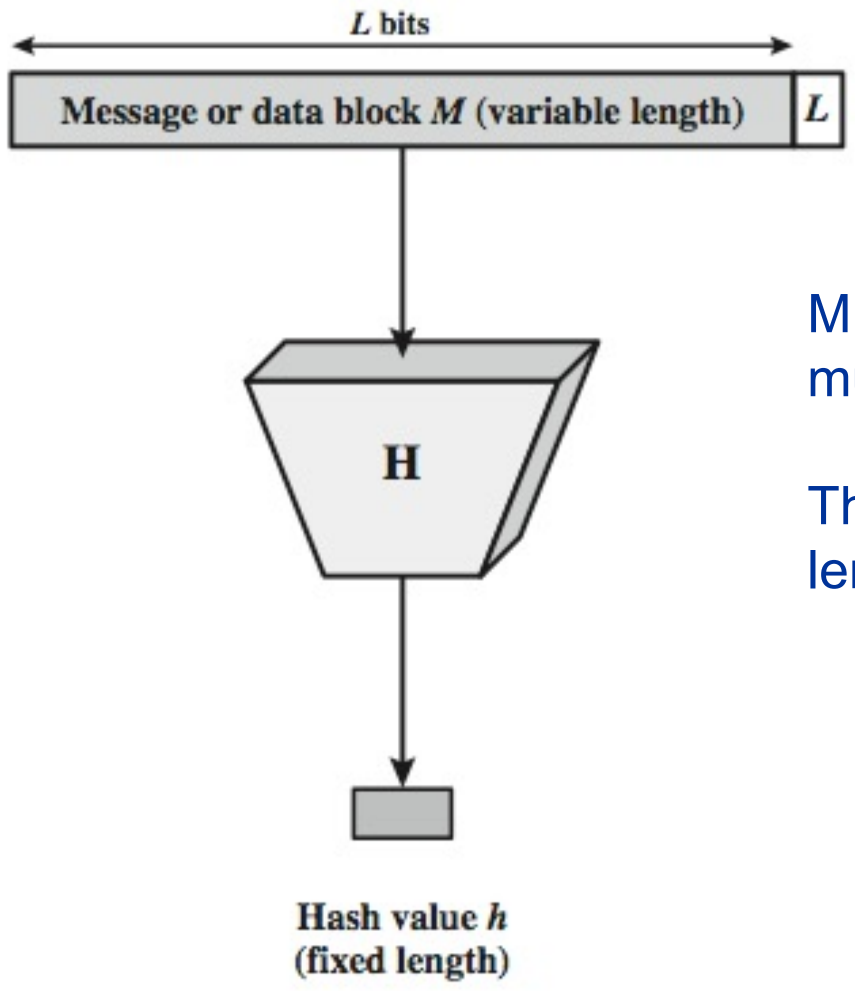
Deterministic and efficient computation given the input

Output is random (uniformly distributed)

Input is also called *message* and output can be called *digest*, *fingerprint*, *hash*

Used to detect changes to the message (data integrity)

Cryptographic Hash Function



M is padded out to an integer multiple of some fixed length.

The padding often includes the length of the original message.

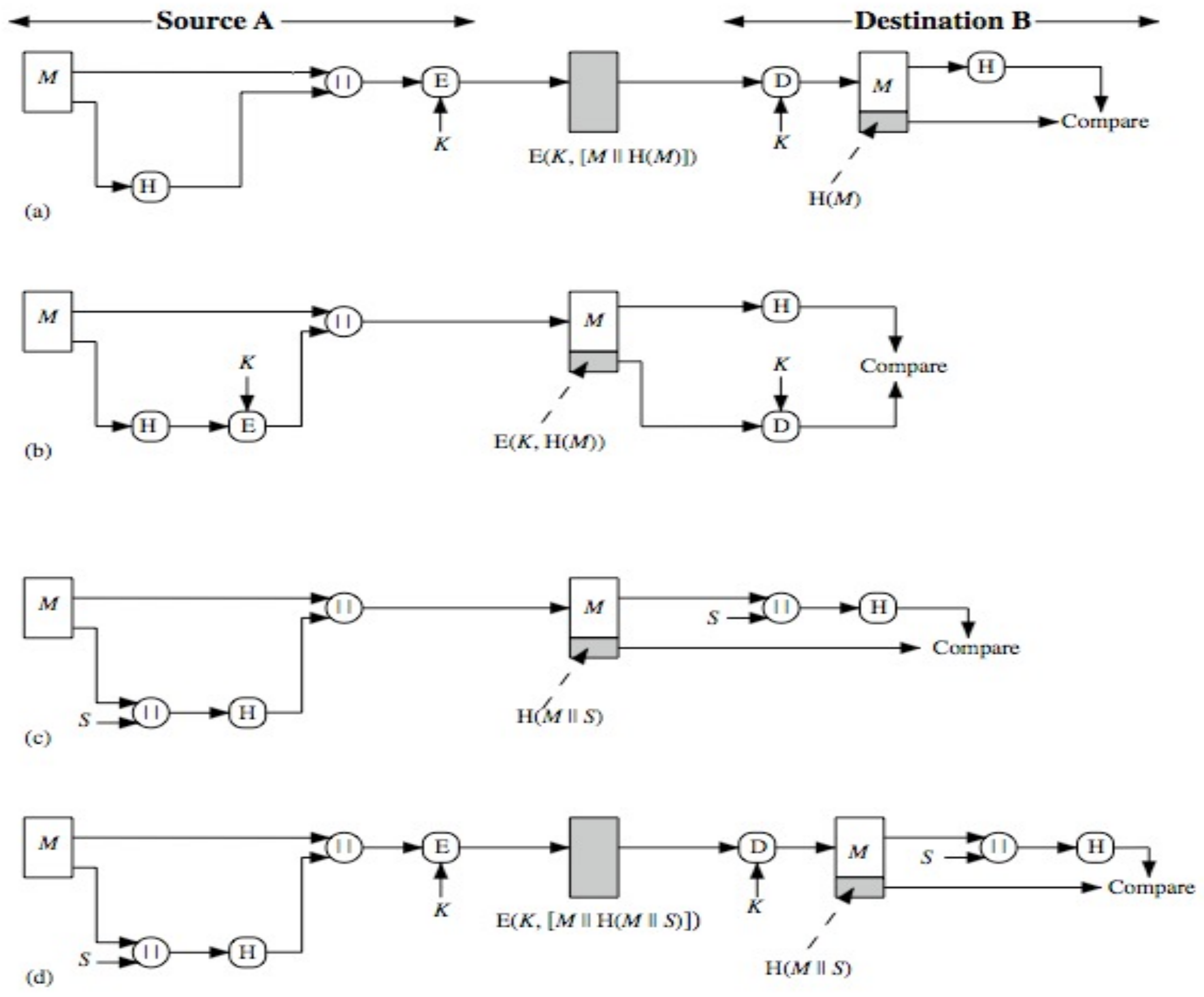
Cryptographic Hash (h) Requirements

The output of h is pseudo-random and exhibits avalanche effect

One-way property: Difficult to find a input that maps to a given hash output

Collision resistance: Difficult to find two inputs mapping to same hash output

Hash Functions used to provide Message Authentication



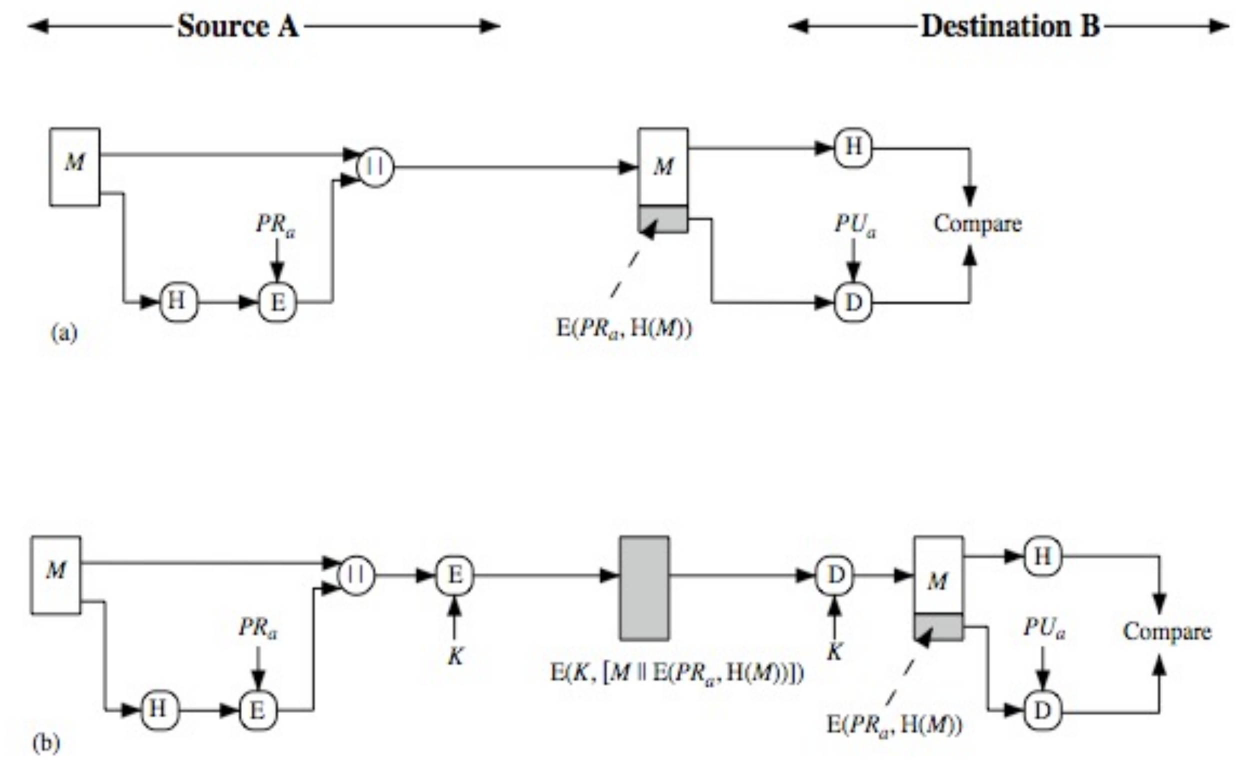
scheme (a) and (d) also provide confidentiality

scheme (a)(b)(d) need to use encryption

specific MAC algorithms are normally used (chap12)

Comparison between hash computations performed by different entities

Hash Functions used to provide Digital Signatures



scheme (a) is similar to (b) in the last slide, but is a digital signature

scheme (b) provides confidentiality

specific digital signature algorithms are used (ch.13)

Other Hash Function Uses

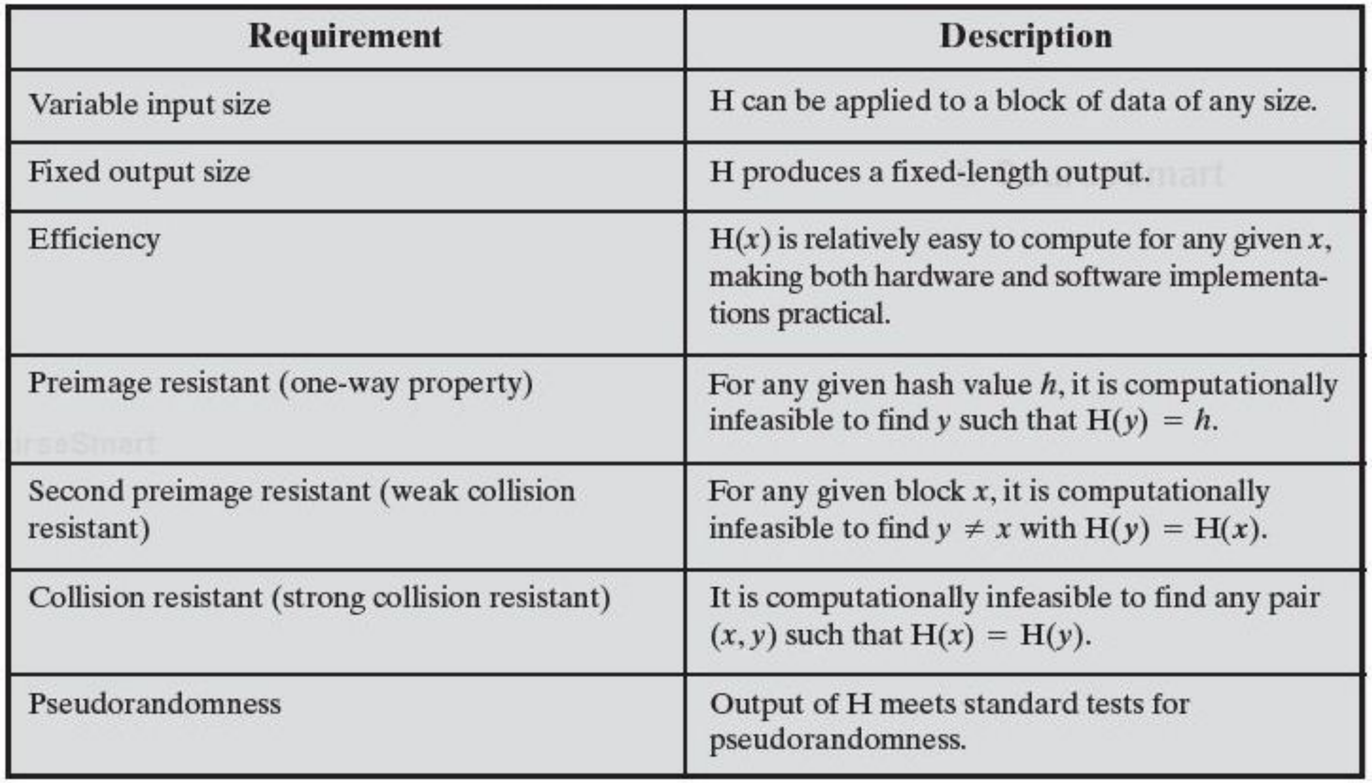
- to create a one-way password file
  - store hash of password not actual password
- for intrusion detection and virus detection
  - keep & check hash of files on system
  - keep H(F) secure, compute and compare later
- blockchain’s hash chain and Merkle root
- pseudorandom function (PRF) or pseudorandom number generator (PRNG)
  - chap 12

Two Simple Insecure Hash Functions

- bit-by-bit exclusive-OR (XOR) of every block
  - $C_i = b_{i1} \oplus b_{i2} \oplus . . . \oplus b_{im}$ *//C_i: the ith bit of the hash code*
  - produces a simple parity for each bit position
  - reasonably effective as data integrity check
  - Regularities in the input will reduce its effectiveness
- one-bit circular shift on hash value (an improvement)
  - rotate current hash value by 1 bit then XOR each new block
- good for data integrity but useless for security
  - given a message (e.g., scheme-b on slide 5, scheme-a on slide 6), an alternate message + additional blocks $\rightarrow$ desired hash code
  - be careful with encryption (e.g., scheme-a on slide 5, CBC mode)

Cryptographic Hash Function Requirements

- h = H(x); x is referred to as the preimage of h; H is a many-to-one mapping; a collision occurs if x≠y, but H(x)=H(y) → undesirable
- length of x: b bits, length of h: n bits → each h, 2^{b-n} preimages
- length of x can be arbitrary, more preimages



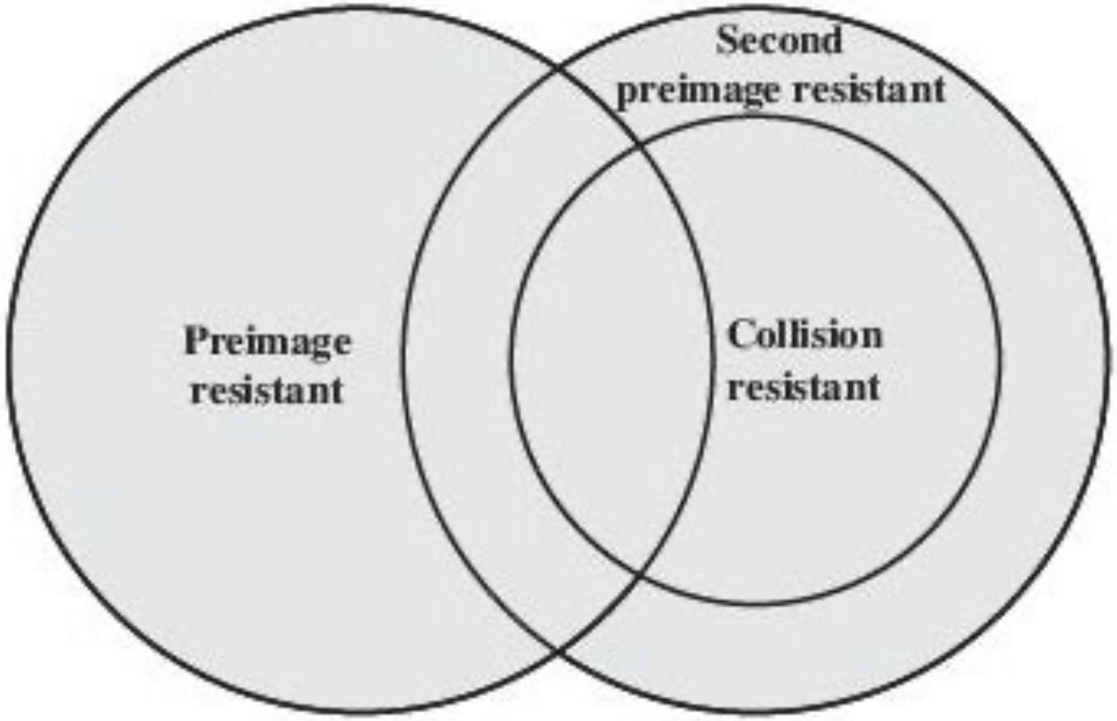
Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that H(y) = h.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with H(y) = H(x).
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

→ Slide 5(c): protects S

→ Slides 5(b) and 6(a): prevents forgery

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

Relationship among Hash Function Properties

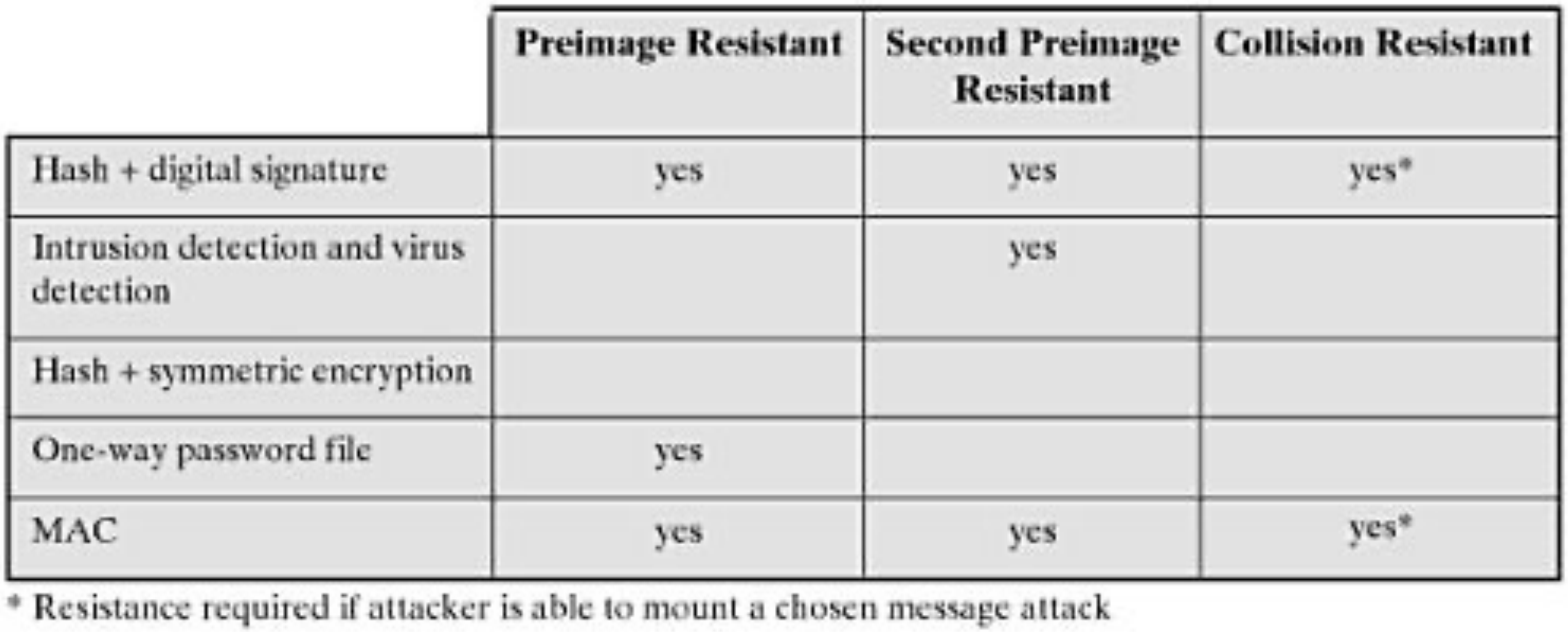


Preimage resistance (one-wayness)
For any output h', it is computationally infeasible to find y such that h(y)=h'

Second preimage resistance (a.k.a. weak collision resistance)
For any x, it is computationally infeasible to find y≠x with h(y)=h(x)

Collision resistance
It is computationally infeasible to find any pair (x,y) such that h(x)=h(y)

Hash Function Resistance Properties Required for Various Data Integrity Applications



	Preimage Resistant	Second Preimage Resistant	Collision Resistant
Hash + digital signature	yes	yes	yes*
Intrusion detection and virus detection		yes	
Hash + symmetric encryption			
One-way password file	yes		
MAC	yes	yes	yes*

* Resistance required if attacker is able to mount a chosen message attack

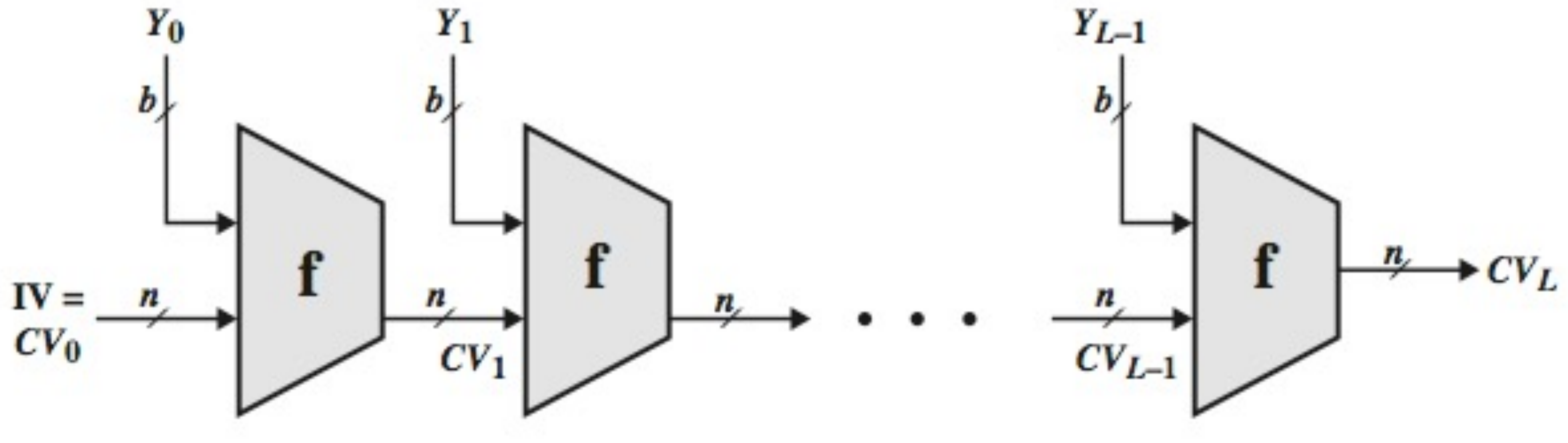
Brute-force Attacks on Hash Functions

- depend on the bit length (m) of the hash value
  - attack on preimage or second preimage resistance
    - on average, 2^{m-1} attempts, (proof in appendix 11A)
  - attack on (strong) collision resistance
    - $2^{m/2}$ attempts
    - birthday paradox, e.g., P(365, 23) = 0.5073

      "In a room of 23 people, there are 0.5 chance of two people having the same birthday"
  - 128-bits inadequate, 160-bits suspect

Cryptanalytic Attacks on Hash Functions

- exploit properties of algorithms
- good hash algorithms should make the effort of best cryptanalytic attacks greater than or equal to brute force
- hash functions use iterative structure processing blocks
- if f is collision resistant, then so is the hash function
- attacks focus on collisions in the **compression function** f



Block Ciphers as Hash Functions

- can use block ciphers as hash functions
  - divide a message into N blocks M_1, M_2, …, M_N
  - using H_0=0 and zero-pad of final block
  - compute: $H_i = E(M_i, H_{i-1})$
  - $h = H_N$
  - similar to CBC but without a key
- vulnerable to attacks: small hash code if DES (64-bit) is used, birthday attack, “meet-in-the-middle” attack
- other variants also susceptible to attack

Secure Hash Algorithm (SHA)

- the most widely used hash function
- SHA (SHA-0) originally designed by NIST in 1993
- was revised in 1995 as SHA-1
  - based on design of MD4
  - produces 160-bit hash values
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications
  - estimated collision attack effort: 2^{69}, 2^{63} [Wang05]
- In Feb 2017, SHA-1 collision found: shattered.io

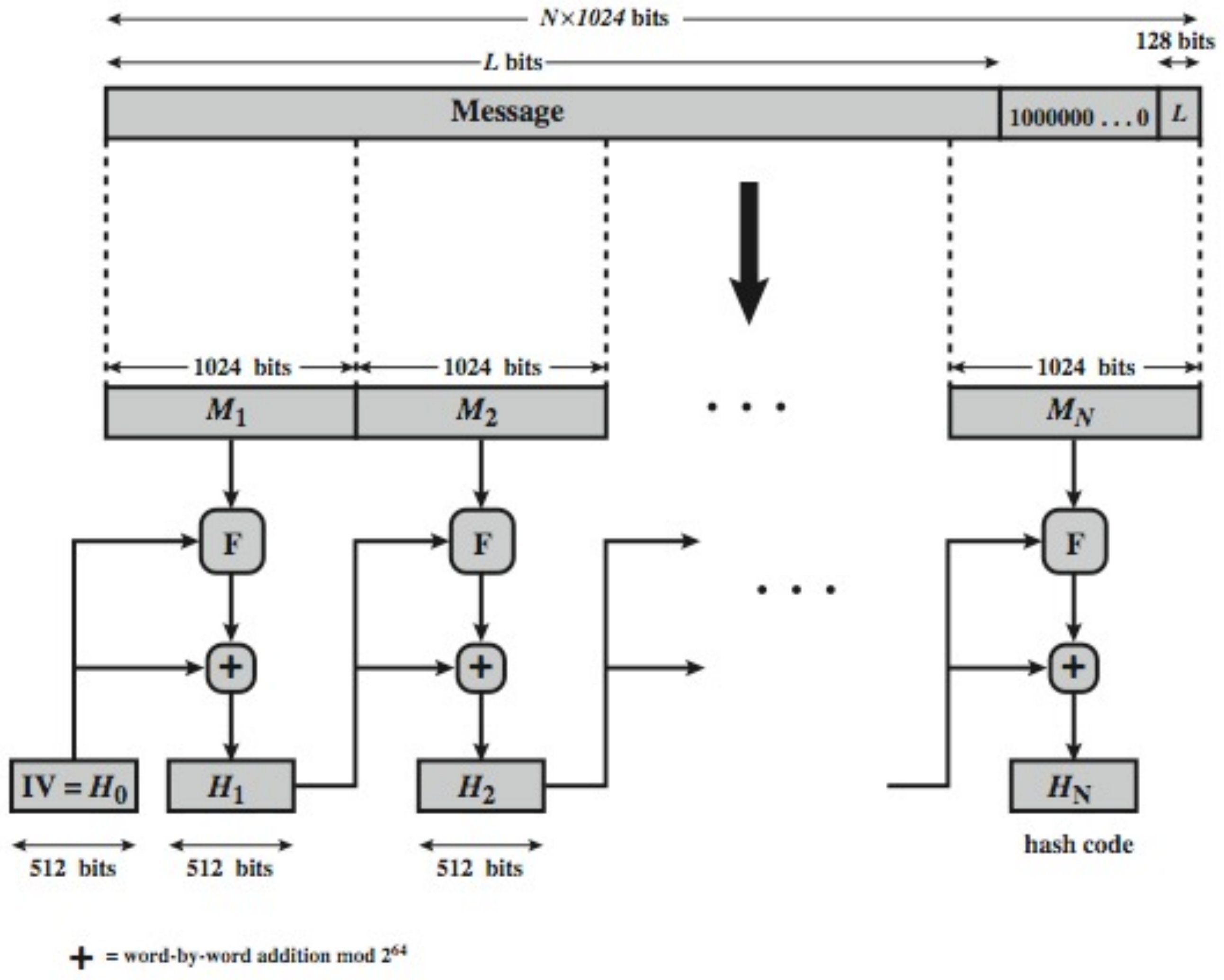
Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA (SHA-2)
  - SHA-256, SHA-384, SHA-512 with different hash value lengths in 2002; SHA-224 in 2008
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

SHA Versions

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message digest size	160	224	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	512	1024	1024
Word size	32	32	32	64	64
Number of steps	80	64	64	80	80

SHA-512 Overview



1. Append padding bits to message so that its length≡896(mod 1024)

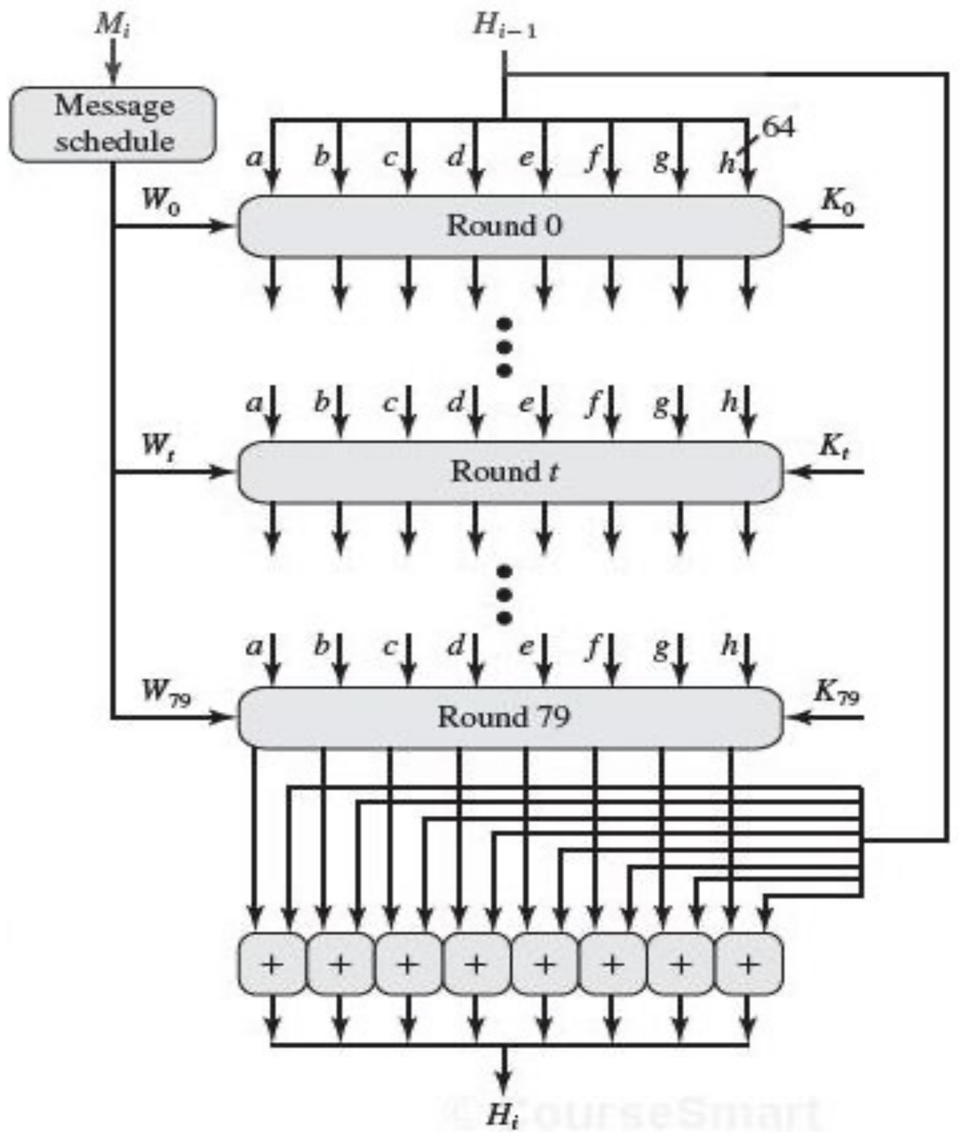
2. Append length L

3. Initialize a 512-bit hash buffer

4. Process message in 1024-bit (the heart is a compression function F)

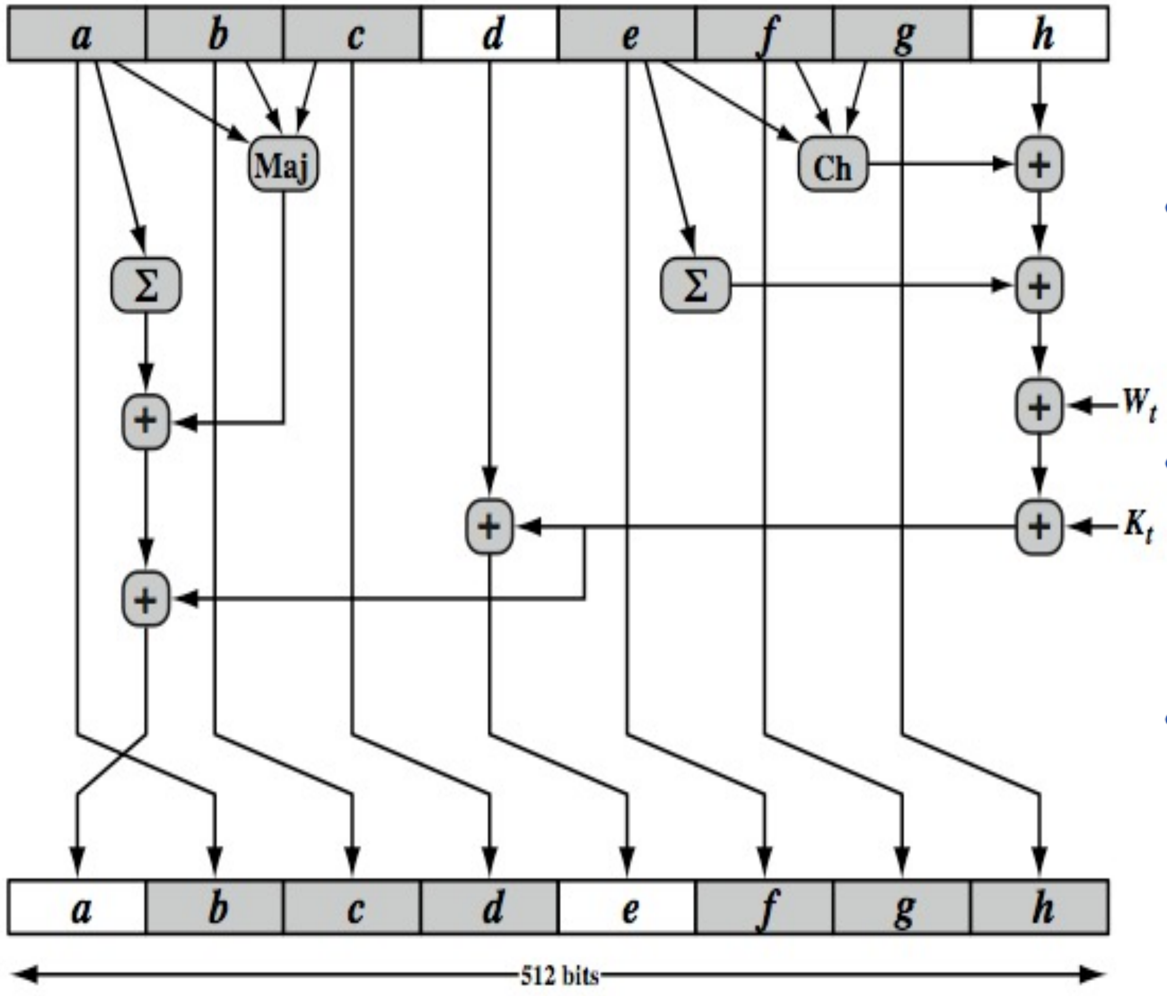
5. Output the Nth stage result h = H_N

SHA-512 Compression Function F



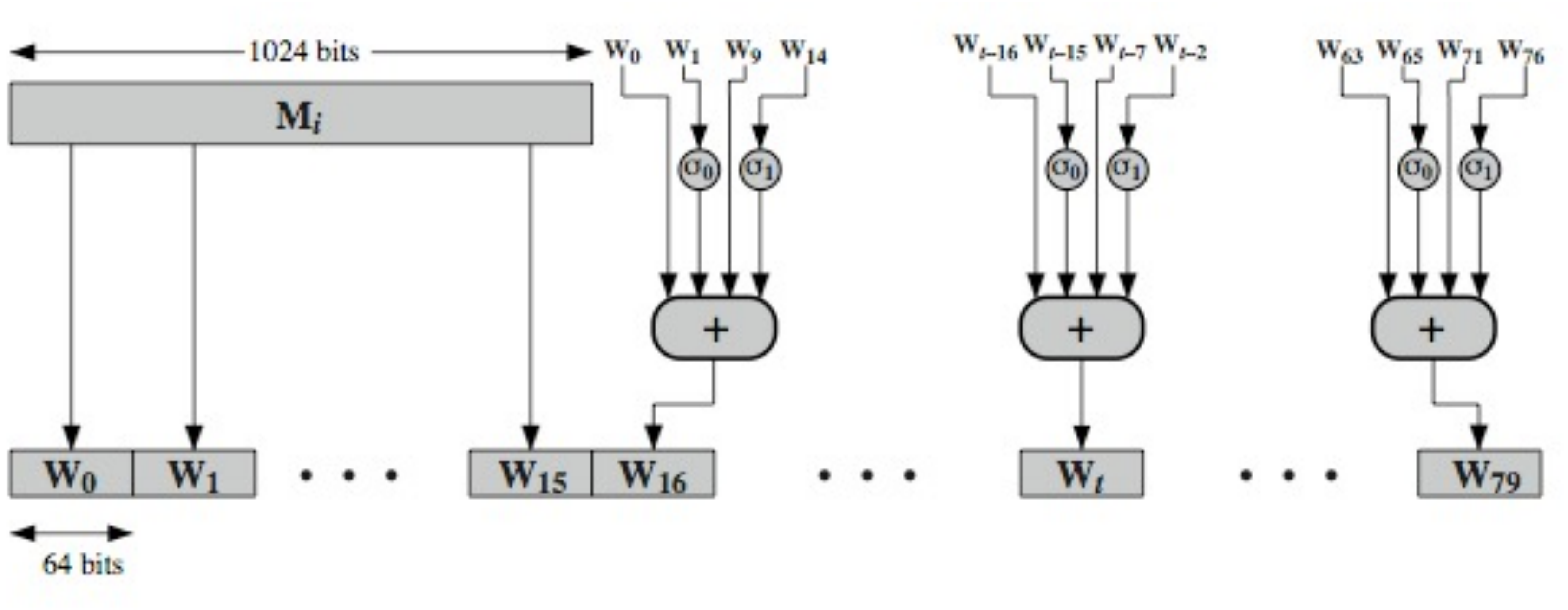
- 1024-bit message block M_i
- 512-bit hash value H_{i-1}
- 80 rounds, in each round t
  - update a 512-bit buffer value
  - use a 64-bit value W_t derived from current message block
  - use a round constant K_t based on fractional parts of cube roots of first 80 primes
  - last round result added (mod 2^{64}) with H_{i-1} to get H_i

SHA-512 Round Function



- Six of the eight output words involve simply permutation
- Two of the output words are generated by substitution
- Ch, Maj, $\sum$: *bit operations* like XOR, rotation, etc.

SHA-512 Creation of W_t



1024 bits
M_i
W_0 W_1 W_9 W_{14}
W_{t-16} W_{t-15} W_{t-7} W_{t-2}
W_{63} W_{65} W_{71} W_{76}
σ_0
σ_1
+
W_0 W_1 W_{15} W_{16} W_t W_{79}
64 bits

- The first 16 steps of processing, the value of W_t is equal to the corresponding word in the message block.
- For the remaining 64 steps, the value of W_t is calculated from four of the preceding values of W_t

SHA-3

- SHA-1 broken in Feb 2017
  - similar in structure and mathematical operations with MD5 & SHA-0, so considered insecure
- SHA-2 (esp. SHA-512) seems secure
  - shares same structure and mathematical operations as predecessors, so have concern
- NIST announced in 2007 a competition for the SHA-3 next generation NIST hash function
  - NIST released SHA-3 standard in 2015

SHA-3 Requirements

- can replace SHA-2 with SHA-3 in any application
  - by a simple drop-in substitution, support same hash value lengths
- preserve the online nature of SHA-2
  - process small blocks (512/1024 bits) at a time without buffering
- evaluation criteria
  - security strength close to theoretical maximum for different hash value lengths and for preimage resistance and collision resistance
  - Efficient in both time & memory
  - algorithm/implementation characteristics: flexibility & simplicity
  - NIST released SHA-3 in 2015: https://en.wikipedia.org/wiki/SHA-3

Summary

- Hash functions
  - Uses, requirements, security
- Hash applications
- Hash functions based on block ciphers
- SHA-1, SHA-2, SHA-3