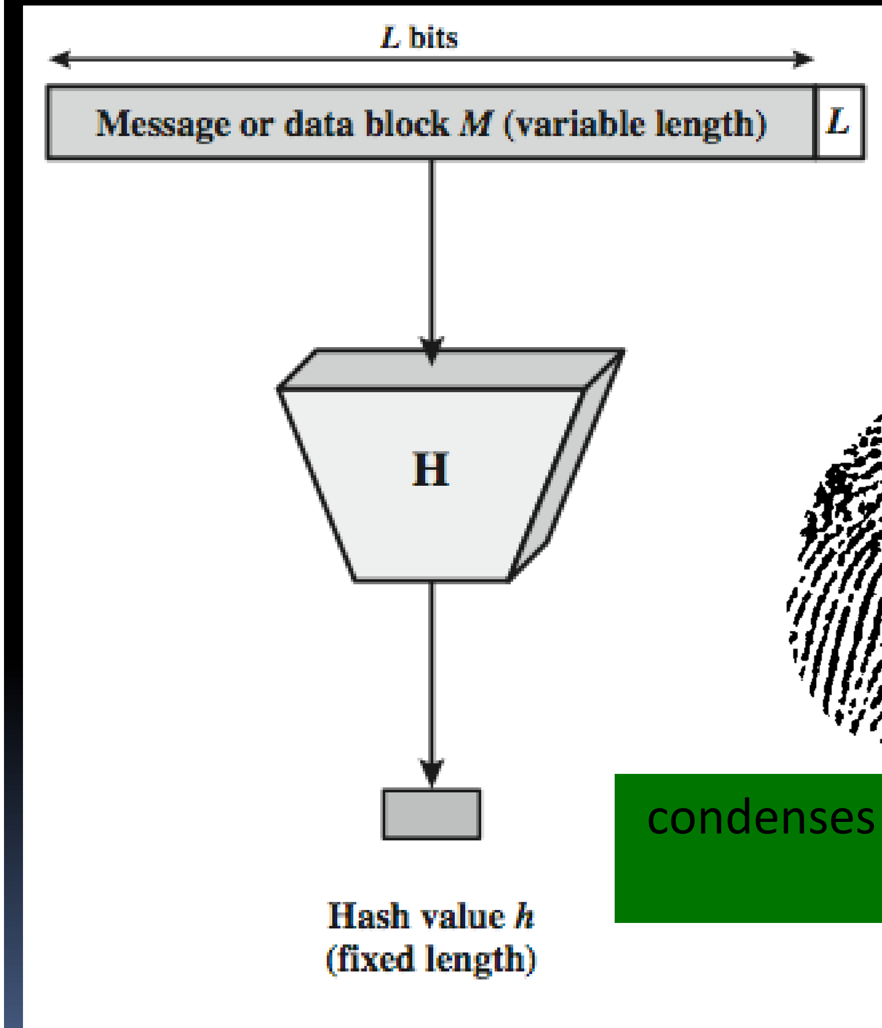
HASH AND MAC ALGORITHMS

Hash Function



- The hash value represents concisely the longer message
  - may called the *message digest*

- A message digest is as a ``digital fingerprint'' of the original document

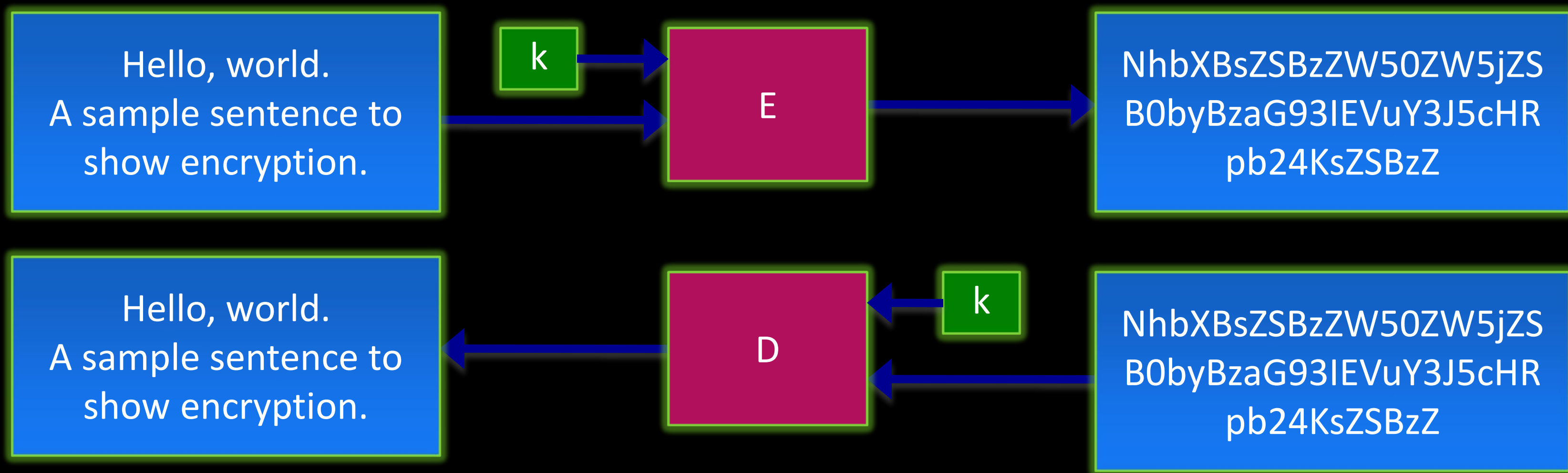
condenses arbitrary message to fixed size

$$h = H(M)$$

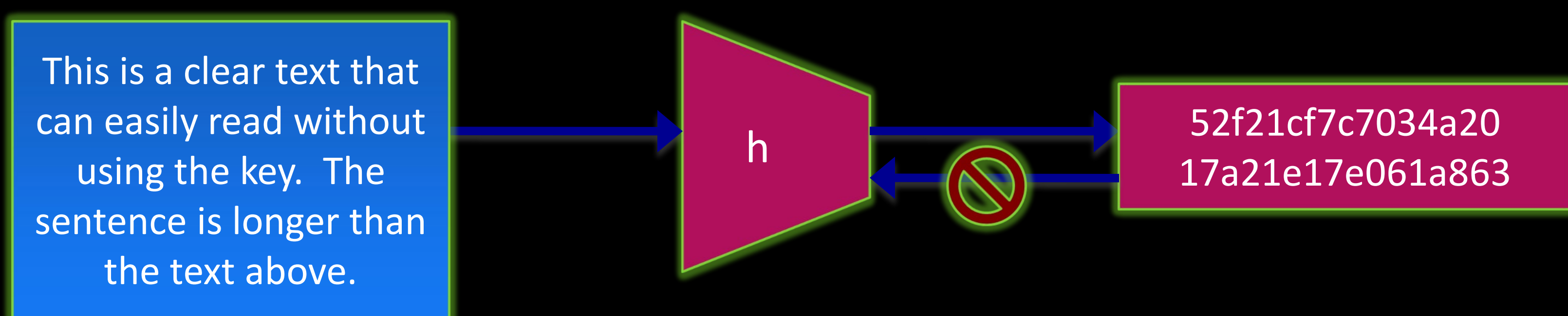
Chewing functions

- Hashing function as “chewing” or “digest” function

Hashing V.S. Encryption



- Encryption is two way, and requires a key to encrypt/decrypt



- Hashing is one-way. There is no 'de-hashing’

Motivation for Hash Algorithms

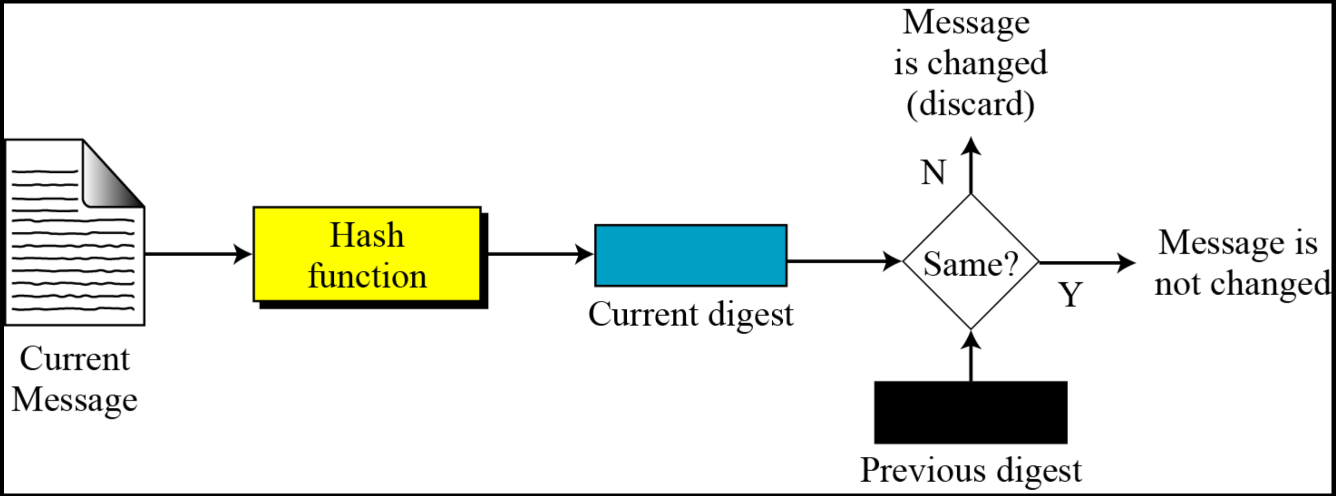
- Intuition
  - Limitation on non-cryptographic checksum
  - Very possible to construct a message that matches the checksum
- Goal
  - Design a code where the original message can not be inferred based on its checksum
  - such that an accidental or intentional change to the message will change the hash value

Hash Function Applications

- Used Alone
  - Fingerprint -- file integrity verification, public key fingerprint
  - Password storage (one-way encryption)

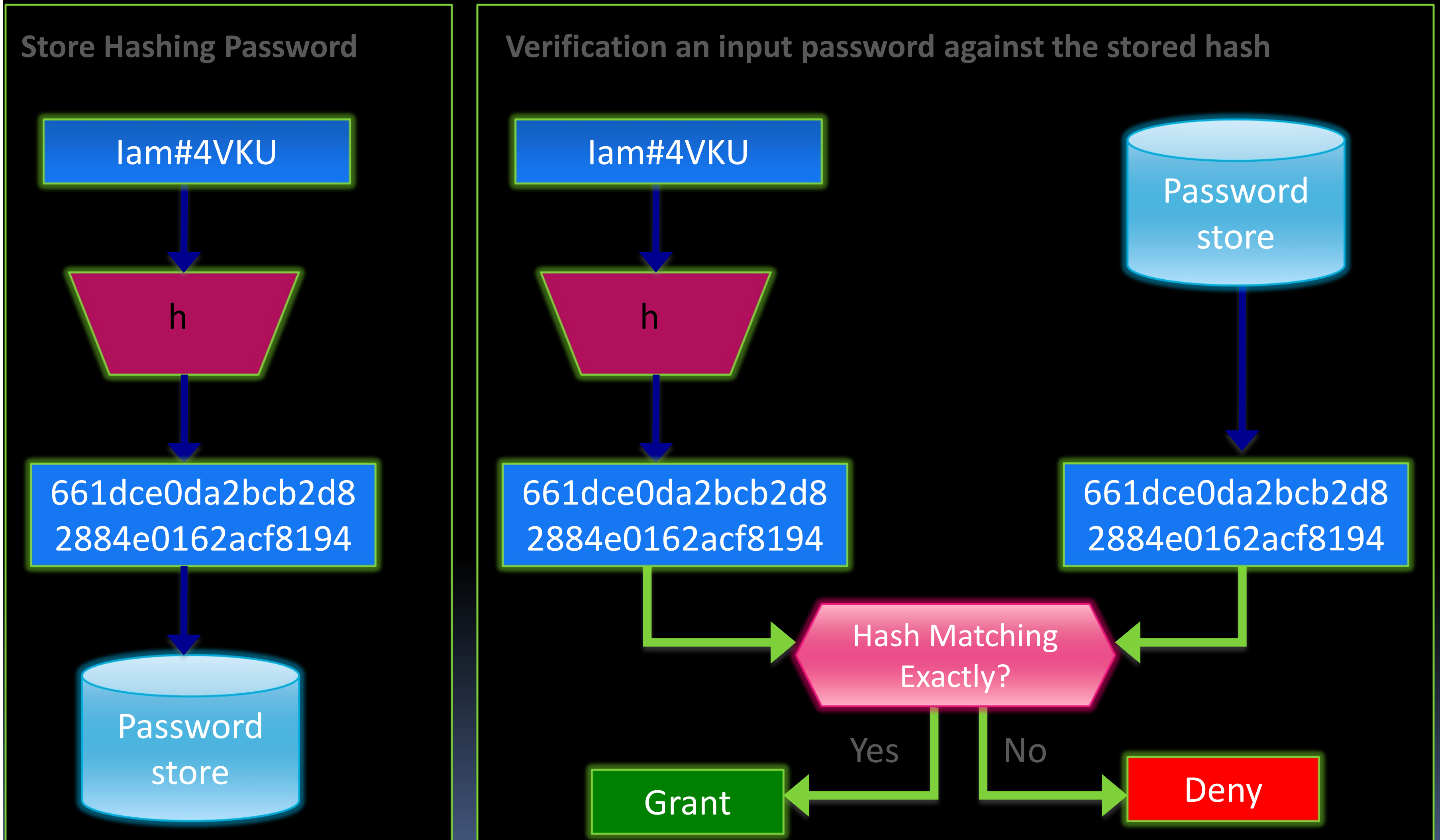
- Combined with encryption functions
  - Hash based Message Authentication Code (HMAC)
    - protects both a message's integrity and confideltaility
  - Digital signature
    - Ensuring Non-repudiation
    - Encrypt hash with private (signing) key and verify with public (verification) key

Integrity

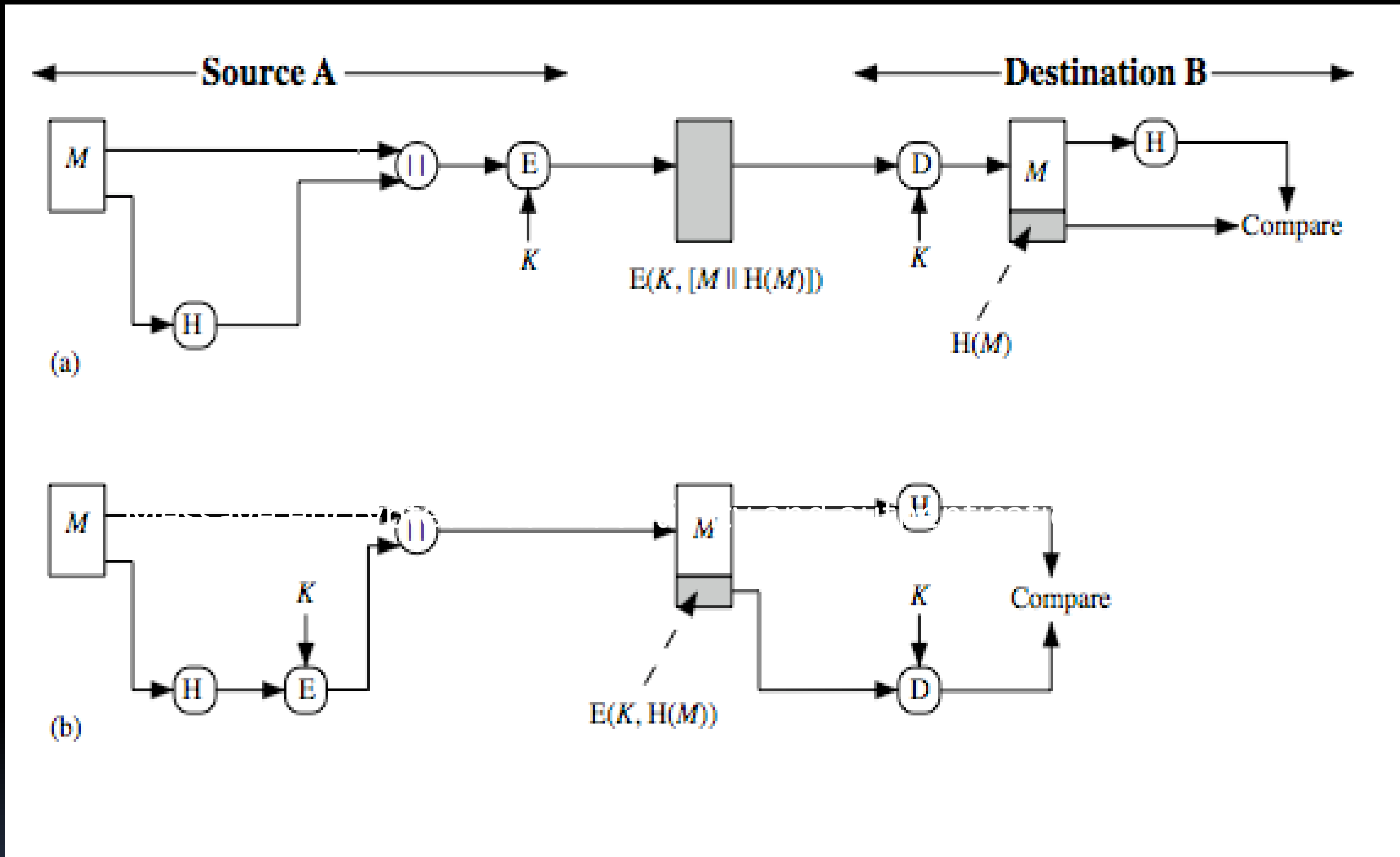


- to create a one-way password file
  - store hash of password not actual password
- for intrusion detection and virus detection
  - keep & check hash of files on system

Password Verification

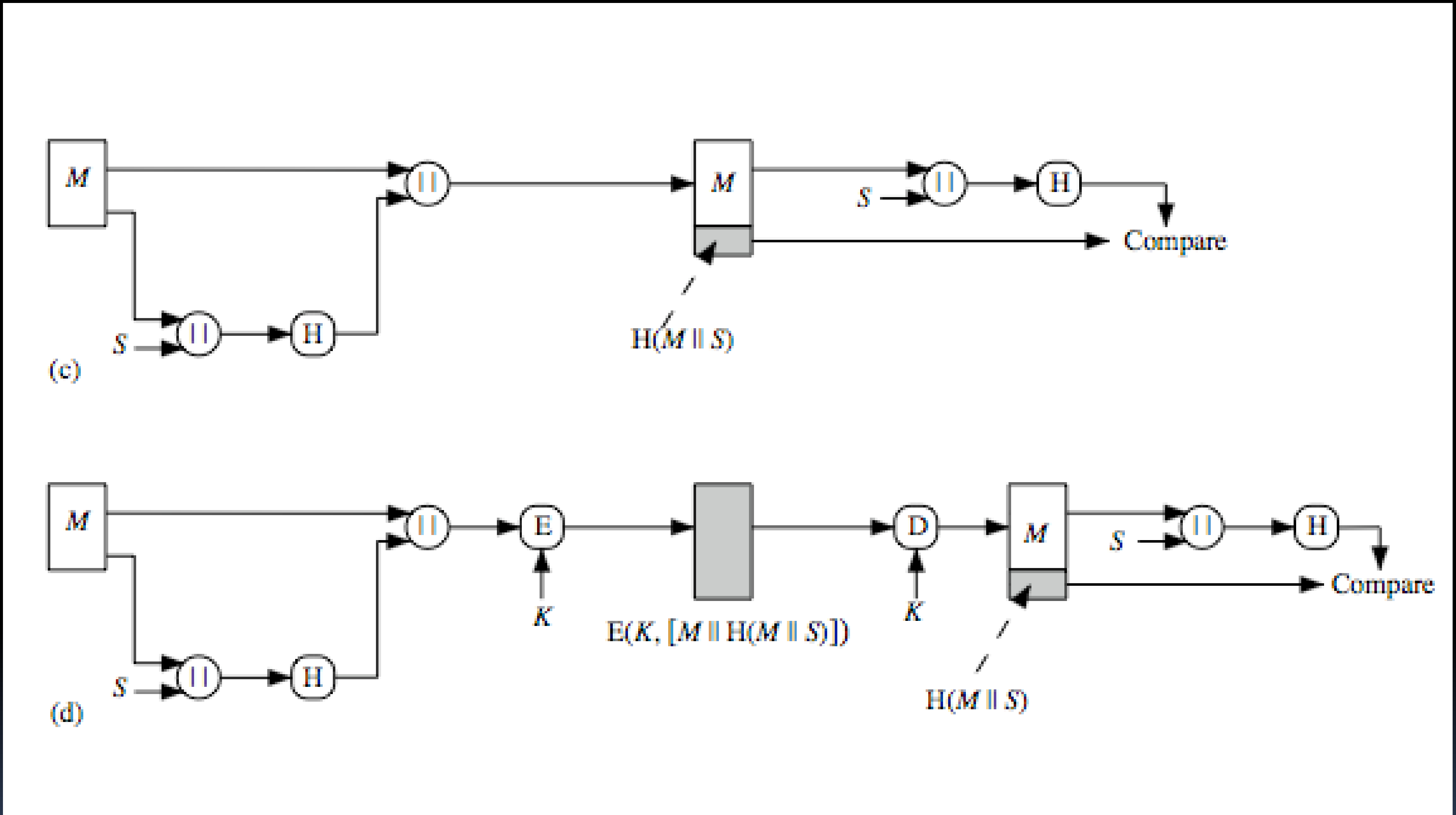


Hash Function Usages (I)

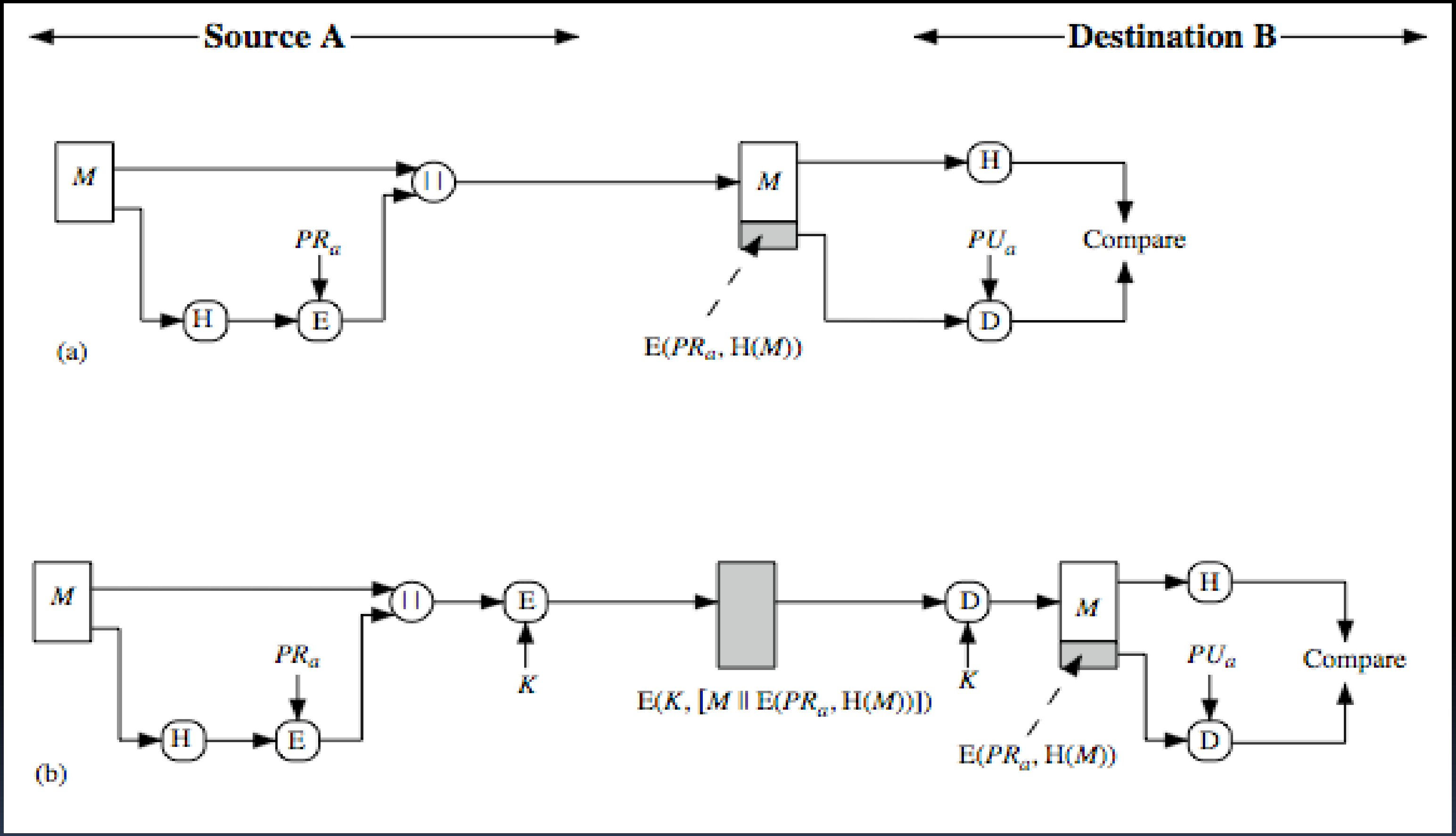


Message unencrypted: Authentication

Hash Function Usages (II)



Hash Function Usages (III)

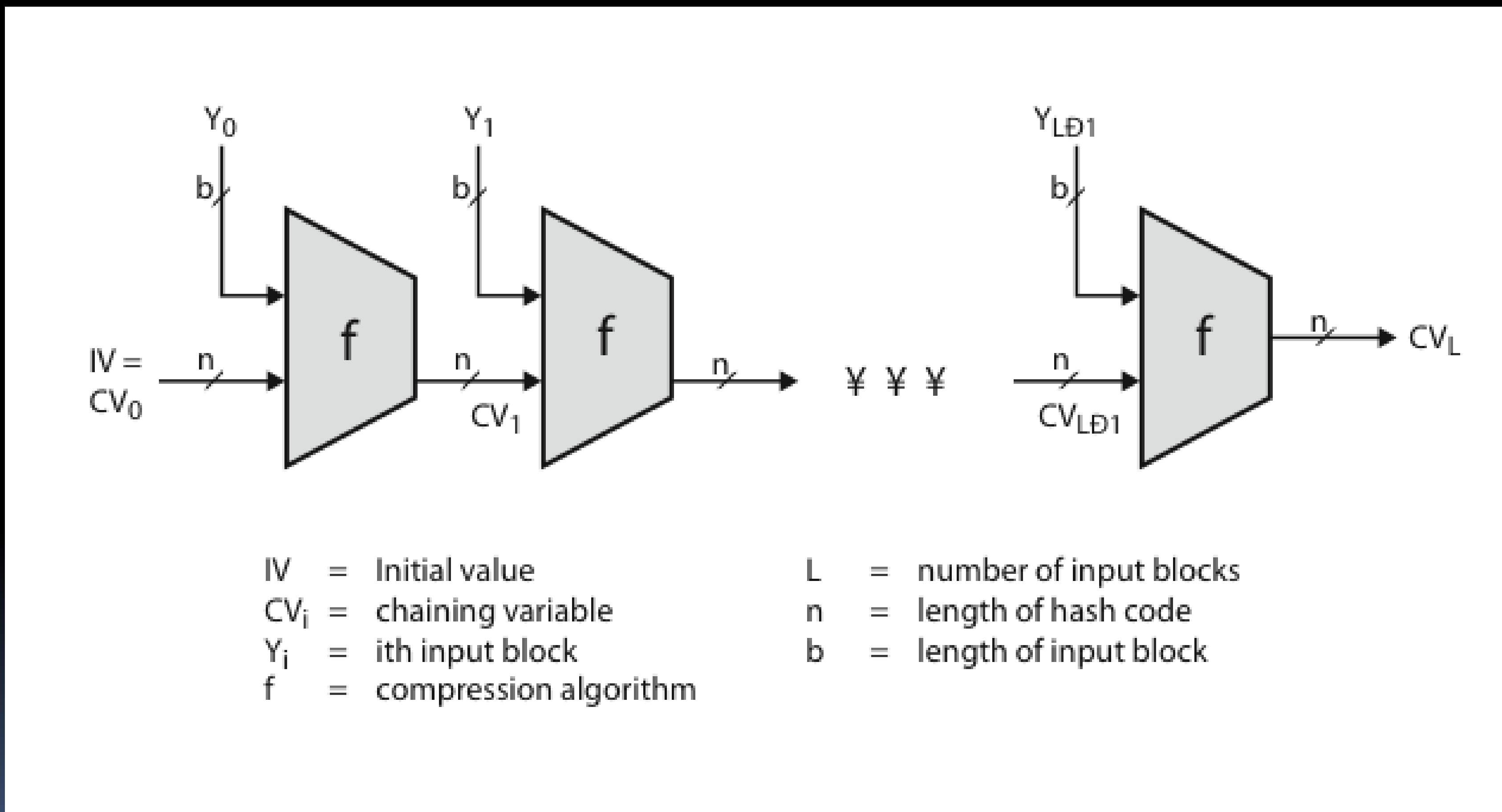


Authentication, digital signature, confidentiality

Hash and MAC Algorithms

- Hash Functions
  - condense arbitrary size message to fixed size
  - by processing message in blocks
  - through some compression function
  - either custom or block cipher based
- Message Authentication Code (MAC)
  - fixed sized authenticator for some message
  - to provide authentication for message
  - by using block cipher mode or hash function

Hash Algorithm Structure



Secure Hash Algorithm

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- based on design of MD4 with key differences
- produces 160-bit hash values
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

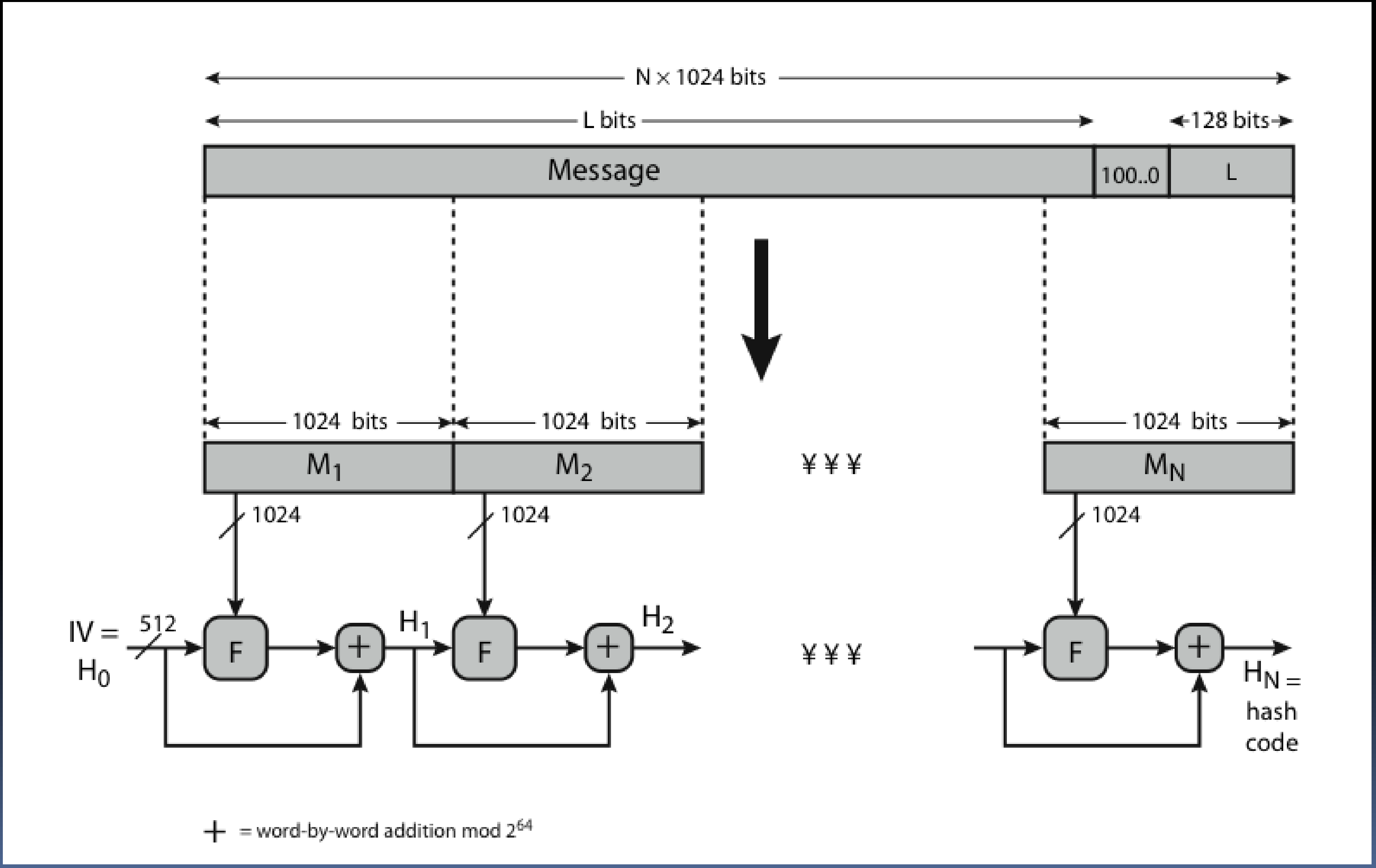
Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- 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-512

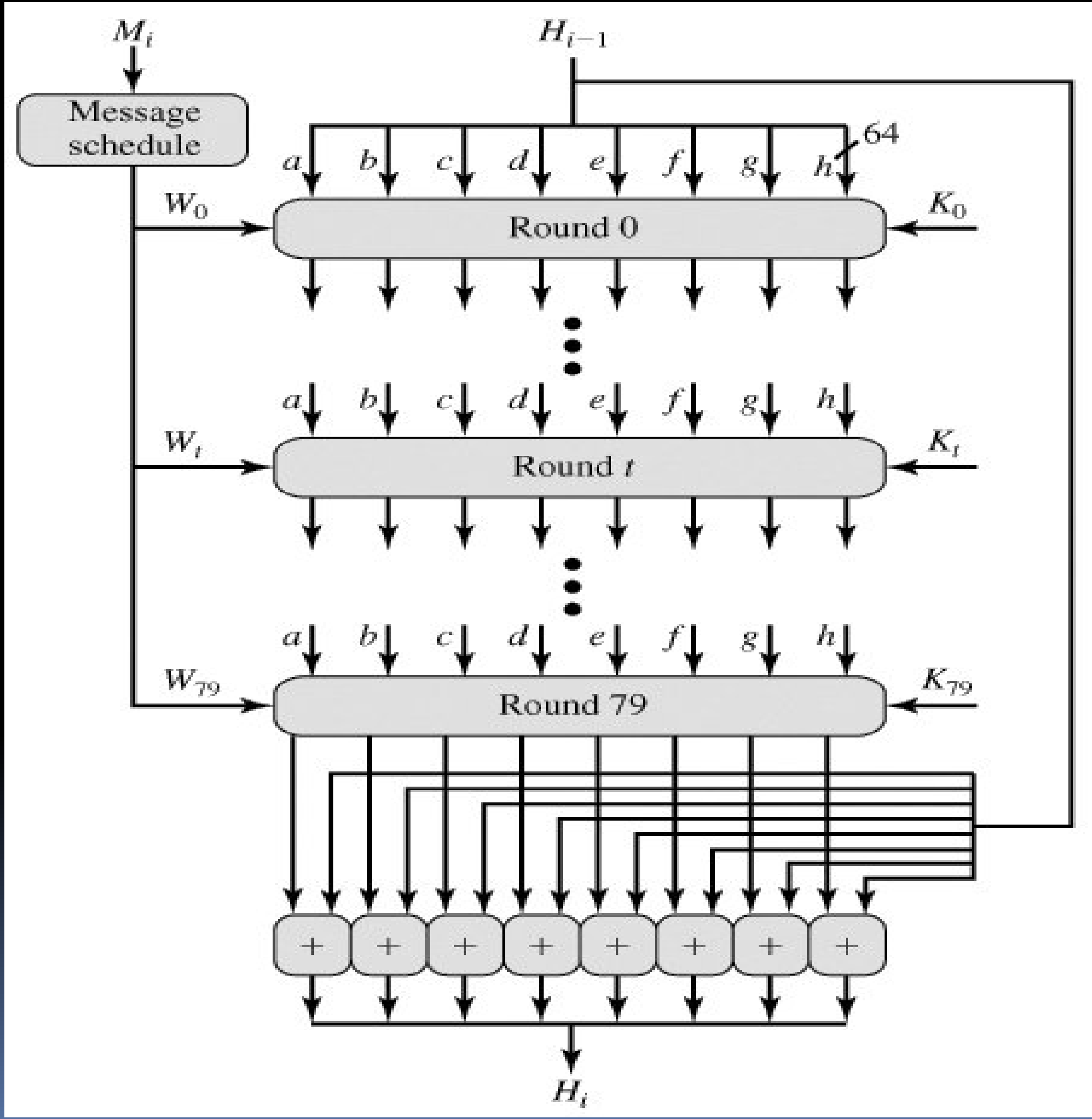
- Step 1: Append padding bits
- Step 2: Append length
- Step 3: Initialize hash buffer
- Step 4: Process the message in 1024-bit (128-word) blocks, which forms the heart of the algorithm
- Step 5: Output the final state value as the resulting hash

SHA-512 Overview



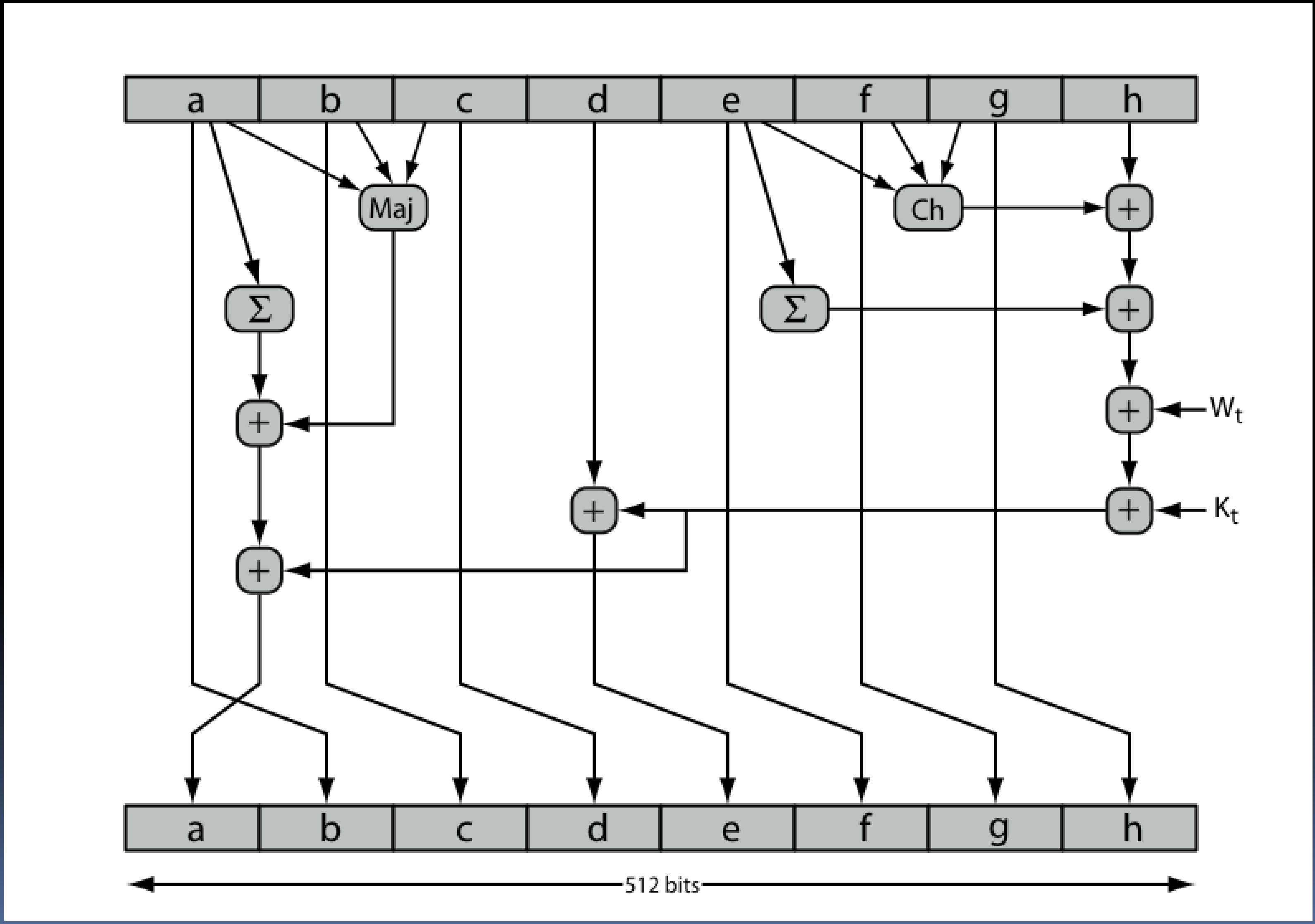
SHA-512 Compression Function

- heart of the algorithm
- processing message in 1024-bit blocks
- consists of 80 rounds
  - updating a 512-bit buffer
  - using a 64-bit value Wt derived from the current message block
  - and a round constant based on cube root of first 80 prime numbers



M_i
H_{i-1}
Message schedule
64
a
b
c
d
e
f
g
h
W_0
Round 0
K_0
W_t
Round t
K_t
W_{79}
Round 79
K_{79}
+
H_i

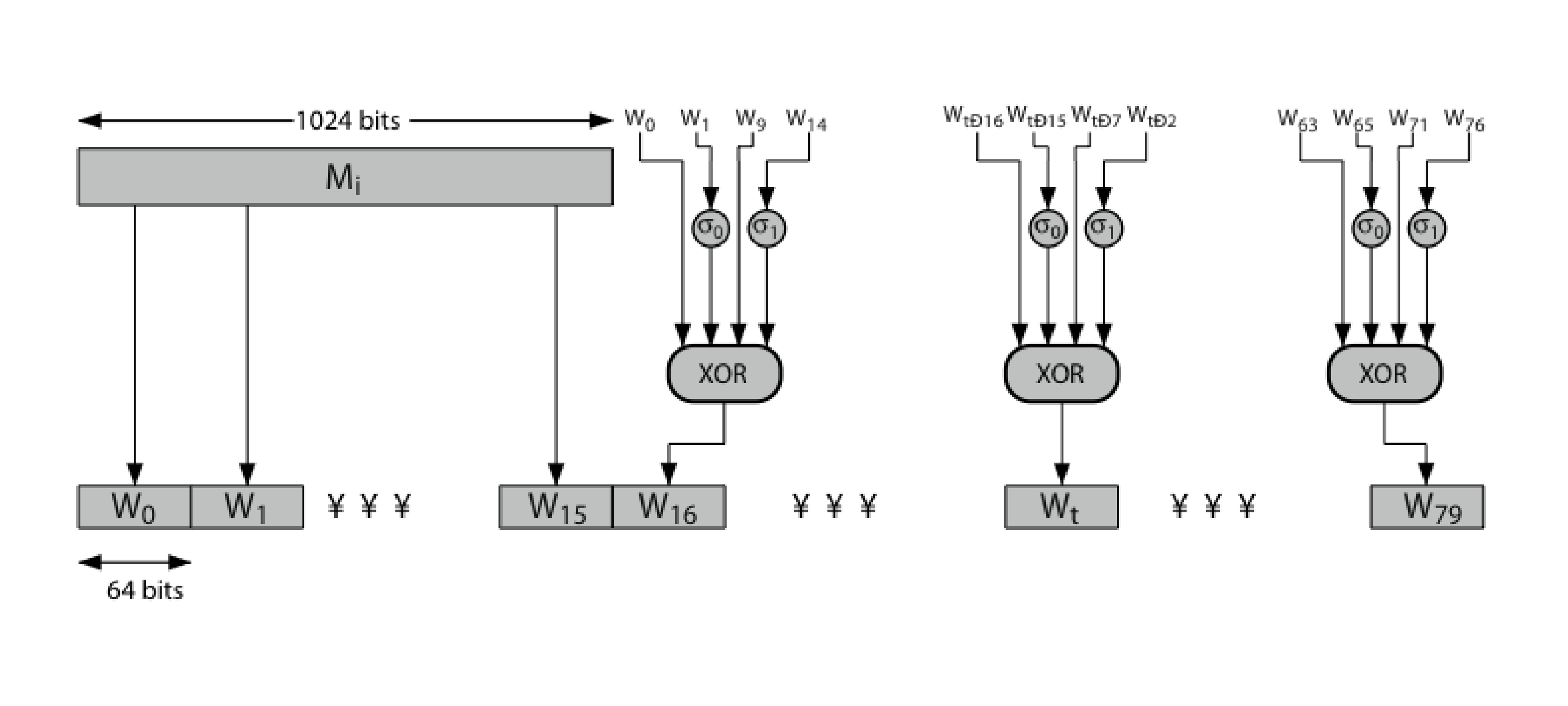
SHA-512 Round Function



Function Elements

- $Ch(e,f,g) = (e \text{ AND } f) \text{ XOR } (\text{NOT } e \text{ AND } g)$
- $Maj(a,b,c) = (a \text{ AND } b) \text{ XOR } (a \text{ AND } c) \text{ XOR } (b \text{ AND } c)$
- $\sum(a) = ROTR(a,28) \text{ XOR } ROTR(a,34) \text{ XOR } ROTR(a,39)$
- $\sum(e) = ROTR(e,14) \text{ XOR } ROTR(e,18) \text{ XOR } ROTR(e,41)$
- + = addition modulo 2^{64}
  - Kt = a 64-bit additive constant
  - Wt = a 64-bit word derived from the current 512-bit input block.

SHA-512 Round Function



Keyed Hash Functions as MACs

- want a MAC based on a hash function
  - because hash functions are generally faster
  - code for crypto hash functions widely available
- hash includes a key along with message
- original proposal:

```
KeyedHash = Hash(Key|Message)
```

  - some weaknesses were found with this
- eventually led to development of HMAC

HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:

  $\mathrm{HMAC}_K = \mathrm{Hash}[(K^+ \text{ XOR } opad) \,||$
  $\qquad\qquad \mathrm{Hash}[(K^+ \text{ XOR } ipad)||M)]]$
- where K^+ is the key padded out to size
- and opad, ipad are specified padding constants
- overhead is just 3 more hash calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC Overview