The Secure Hash Function (SHA)

Information Security

The Secure Hash Function (SHA)

Objectives of the Topic

- After completing this topic, a student will be able to
  - explain working of the secure hash algorithm (SHA).

The Secure Hash Function (SHA)

Figures and material in this topic have been adapted from

- *"Network Security Essentials : Applications and Standards"*, by William Stallings.

The Secure Hash Function (SHA)

Secure Hash Algorithm (SHA)

- Is the most widely used hash function in recent years.
- Developed by the National Institute of Standards and Technology (NIST)
- FIPS 180 in 1993.

The Secure Hash Function (SHA)

- The actual standards document is entitled “Secure Hash Standard.”
- SHA is based on the hash function (Message-Digest) MD4, and its design closely models MD4.

The Secure Hash Function (SHA)

- When weaknesses were discovered in SHA (now known as SHA-0), a revised version was issued as FIPS 180-1 in 1995 and is referred to as SHA-1.

The Secure Hash Function (SHA)

- In 2002, NIST produced FIPS 180-2.
- Three new versions of SHA with hash value lengths of 256, 384, and 512 bits known as SHA-256, SHA-384, and SHA-512 were defined.
- Collectively, these are known as SHA-2.

The Secure Hash Function (SHA)

Comparison of SHA Parameters

	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

Note: All sizes are measured in bits.

The Secure Hash Function (SHA)

- In 2005 NIST announced the intention to phase out approval of SHA-1 and move to a reliance on SHA-2 by 2010.
- We focus on SHA-512.

The Secure Hash Function (SHA)

SHA-512 Logic:

- The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest.
- The input is processed in 1024-bit blocks.

The Secure Hash Function (SHA)

Step 1: Append padding bits

- Padding is added, even if the message is already of the desired length. No. of Padding bits = [1 1024]
- Padding consists of a single 1 bit followed by the necessary number of 0 bits.

The Secure Hash Function (SHA)

Step 2: Append length

- A block of 128 bits is appended to the message.
- This block is treated as an unsigned 128-bit integer and contains the length of the original message (before the padding).

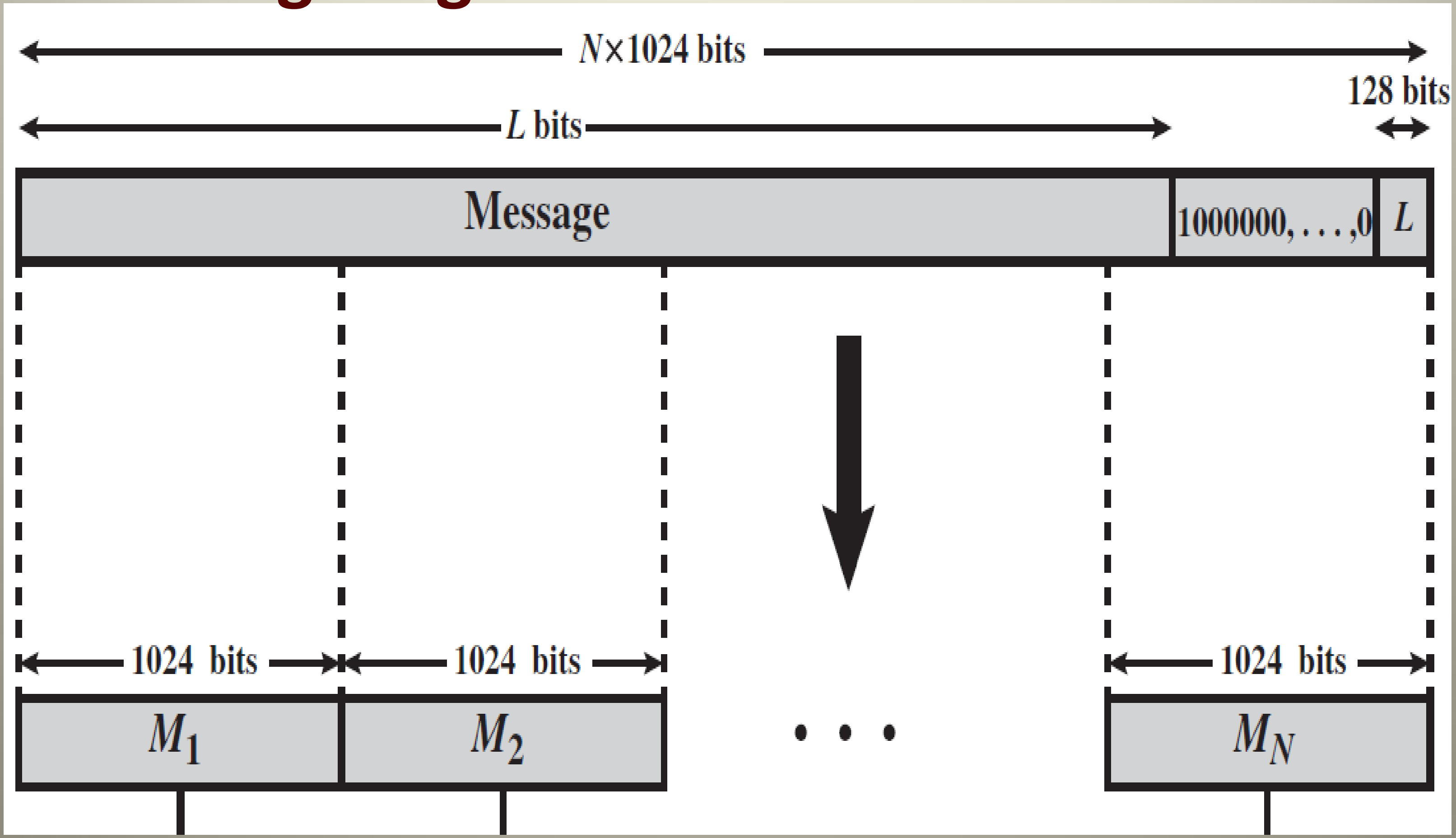
The Secure Hash Function (SHA)

- Outcome of first two steps yields a message an integer multiple of 1024 bits in length.
- Total length of the expanded message is

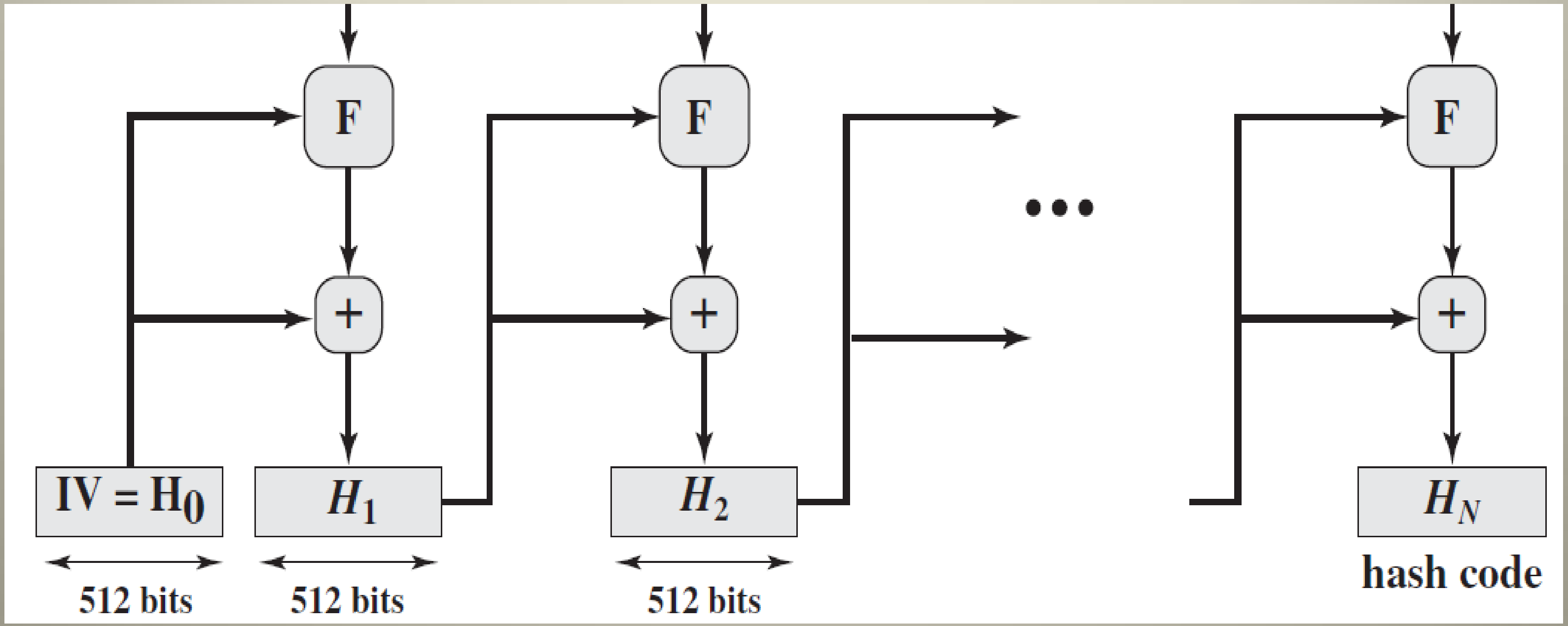
$N \times 1024$ bits as the expanded message is a sequence of 1024-bit blocks **$M_1, M_2, \ldots, M_N$.**

The Secure Hash Function (SHA)

Message Digest Generation of SHA-512



The Secure Hash Function (SHA)



The Secure Hash Function (SHA)

Step 3: Initialize hash buffer

- A 512-bit buffer is used to hold intermediate and final results of the hash function.
- The buffer can be represented as **eight 64-bit registers (*a, b, c, d, e, f, g, h*)**

The Secure Hash Function (SHA)

- Initialize these registers by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

The Secure Hash Function (SHA)

Initialization of the registers

$$a = \mathtt{6A09E667F3BCC908} \qquad e = \mathtt{510E527FADE682D1}$$

$$b = \mathtt{BB67AE8584CAA73B} \qquad f = \mathtt{9B05688C2B3E6C1F}$$

$$c = \mathtt{3C6EF372FE94F82B} \qquad g = \mathtt{1F83D9ABFB41BD6B}$$

$$d = \mathtt{A54FF53A5F1D36F1} \qquad h = \mathtt{5BE0CD19137E2179}$$

The Secure Hash Function (SHA)

Step 4: Process message in 1024-bit (128-word) blocks

- The module labeled F consists of 80 rounds.
- Each round takes as input the 512-bit buffer value ***abcdefgh*** and updates the contents of the buffer.

The Secure Hash Function (SHA)

- At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}.
- Each round **t** makes use of a 64-bit value W_t derived from the current 1024-bit block being processed (M_i).

The Secure Hash Function (SHA)

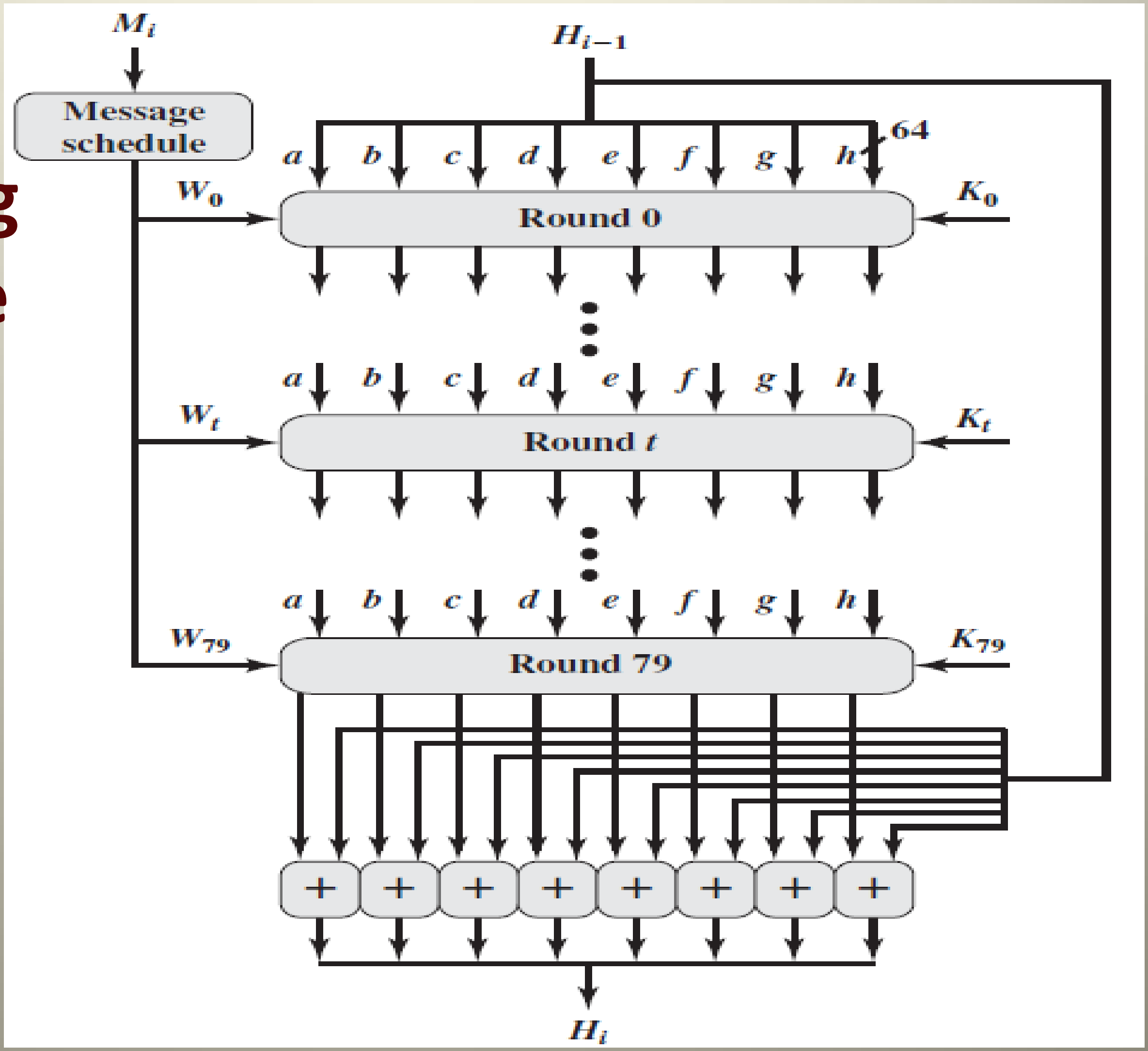
- Each round also makes use of an additive constant K_t, where t = 0 … … 79.
- The constants eliminate any regularities in the input data.

The Secure Hash Function (SHA)

- The output of the 80th round is added to the input to the first round (H_{i-1}) to produce H_i .

The Secure Hash Function (SHA)

Processing of a Single 1024-Bit Block



The Secure Hash Function (SHA)

Step 5 Output:

- After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.
- In 2012, NIST formally published SHA-3.

SHA-512 Example

Input: "abc" (3 characters = 24 bits)

- **Step 1: Convert to binary**

  'a' = 01100001

  'b' = 01100010

  'c' = 01100011

  Message: 01100001 01100010 01100011

- **Step 2: Append '1' bit**

  01100001 01100010 01100011 1

SHA-512 Example

- **Step 3: Append '0' bits until length ≡ 896 mod 1024**
  Current length: 25 bits
  Need: 25 + k ≡ 896 mod 1024
  Solution: k = 871 zeros
  01100001 01100010 01100011 1 [871 zeros...]
- **Step 4: Append original length as 128-bit number**
  Original length = 24 bits = 0x18 in hexadecimal
  Append: 000...00011000 (128 bits)
  Final padded message: 1024 bits

SHA-512 Example

- **2. Initialize Hash Values**

  SHA-512 starts with 8 initial constants (first 64 bits of fractional parts of square roots of first 8 primes):

  h0 = 6a09e667f3bcc908

  h1 = bb67ae8584caa73b

  h2 = 3c6ef372fe94f82b

  h3 = a54ff53a5f1d36f1

  h4 = 510e527fade682d1

  h5 = 9b05688c2b3e6c1f

  h6 = 1f83d9abfb41bd6b

  h7 = 5be0cd19137e2179

- **3. Process Each 1024-bit Block**

For our "abc" example, we have exactly one 1024-bit block after padding.

SHA-512 Example

- **Round 0 Calculations**
- **Working variables initialized:**

  a = h0 = 6a09e667f3bcc908

  b = h1 = bb67ae8584caa73b

  c = h2 = 3c6ef372fe94f82b

  d = h3 = a54ff53a5f1d36f1

  e = h4 = 510e527fade682d1

  f = h5 = 9b05688c2b3e6c1f

  g = h6 = 1f83d9abfb41bd6b

  h = h7 = 5be0cd19137e2179
- **Constants (K):** First 80 primes' cube roots fractional parts
- **Message Schedule (W):** Derived from our 1024-bit block

SHA-512 Example

- **Round 0 Step-by-Step**

  1. Calculate S1:

  S1 = ROTR(e, 14) ⊕ ROTR(e, 18) ⊕ ROTR(e, 41)
  e = 510e527fade682d1

  ROTR(e, 14) = rotate right by 14 bits
  ROTR(e, 18) = rotate right by 18 bits
  ROTR(e, 41) = rotate right by 41 bits

  S1 = [calculation result]

  2. Calculate Ch (Choose) function:

  Ch(e, f, g) = (e ∧ f) ⊕ (¬e ∧ g)
  e = 510e527fade682d1
  f = 9b05688c2b3e6c1f
  g = 1f83d9abfb41bd6b

  Ch = (e AND f) XOR (NOT e AND g)

SHA-512 Example

3. Calculate temp1:

temp1 = h + S1 + Ch + K[0] + W[0]

h = 5be0cd19137e2179

S1 = [from step 1]

Ch = [from step 2]

K[0] = 428a2f98d728ae22

W[0] = first 64 bits of our message block

temp1 = h + S1 + Ch + K[0] + W[0]

4. Calculate S0:

S0 = ROTR(a, 28) ⊕ ROTR(a, 34) ⊕ ROTR(a, 39)

a = 6a09e667f3bcc908

S0 = ROTR(a,28) ⊕ ROTR(a,34) ⊕ ROTR(a,39)

SHA-512 Example

5. Calculate Maj (Majority) function:

$\text{Maj}(a, b, c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$

a = 6a09e667f3bcc908

b = bb67ae8584caa73b

c = 3c6ef372fe94f82b

Maj = (a AND b) XOR (a AND c) XOR (b AND c)

6. Calculate temp2:

temp2 = S0 + Maj

SHA-512 Example

7. Update working variables:

h = g

g = f

f = e

e = d + temp1

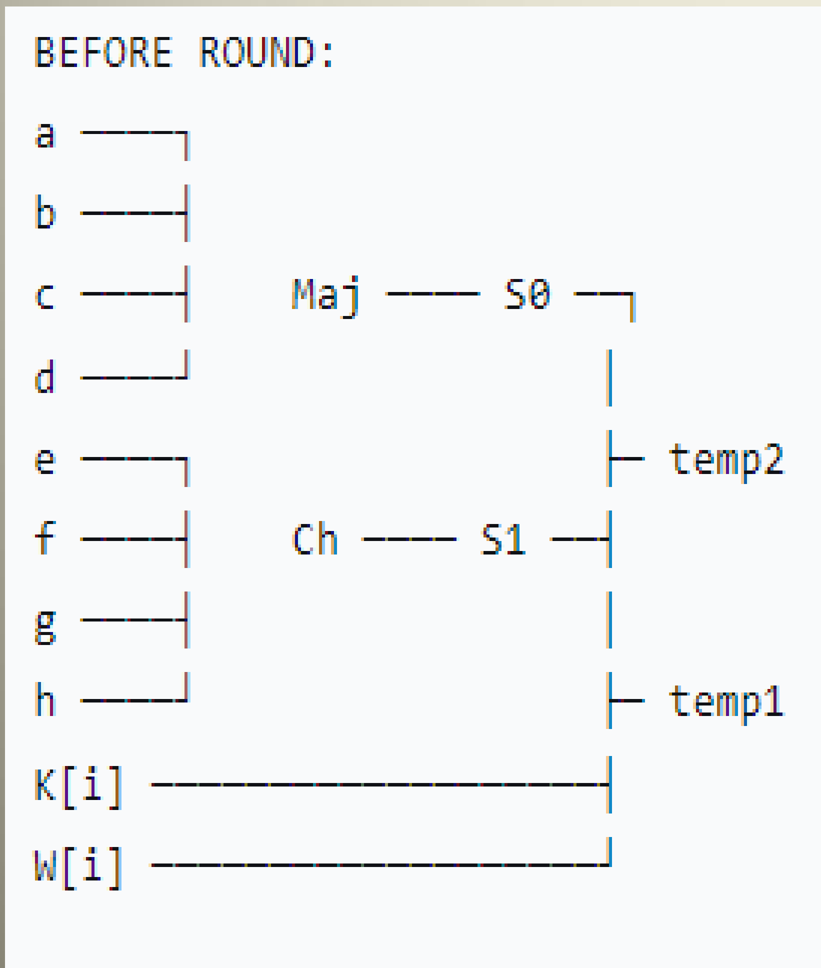
d = c

c = b

b = a

a = temp1 + temp2

This process repeats for 80 rounds!



AFTER ROUND

a = temp1 + temp2

b = old a

c = old b

d = old c

e = old d + temp1

f = old e

g = old f

h = old g

- **Expected Output:**
- Message: 'abc'
- SHA-512: ddaf35a193617abacc417349ae204131 12e6fa4e89a97ea20a9eeee64b55d39a...
- Length: 512 bits (128 hex characters)