MH4311 Cryptography

Lecture 11

Hash Function

Wu Hongjun

Lecture Outline

- Classical ciphers
- Symmetric key encryption
- Hash function and Message Authentication Code
  - Birthday attack
  - **Hash function**
  - Message Authentication Code
- Public key encryption
- Digital signature
- Key establishment and management
- Introduction to other cryptographic topics

Recommended Reading

- CTP Section 4.1, 4.2, 4.3
- HAC Section 9.1, 9.2, 9.3, 9.4
- Wikipedia
  - Cryptographic hash function
    http://en.wikipedia.org/wiki/Cryptographic_hash_function
  - Merkle-Damgard construction
    http://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd_construction
  - SHA-1
    http://en.wikipedia.org/wiki/SHA-1
  - SHA-2
    http://en.wikipedia.org/wiki/SHA-2
  - SHA-3 competition
    http://en.wikipedia.org/wiki/SHA-3
- Full SHA-1, SHA-2 specifications
  http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

Hash Function

- Hash Function
  - Compress a message with arbitrary length into a fixed-length output
  - Used for sorting and searching in computer science
- Cryptographic hash function
  - It is a type of hash function. Whenever we talk about hash function in this course, we mean cryptographic hash function.
  - We try to ensure that every output of the hash function (message digest) represents a message uniquely, i.e., each message digest represents only one message

Hash Function

- Importance of cryptographic hash function
  - Important for data integrity
    - Example: Checksum for downloading software (the Checksum of a file is typically computed using some hash function. After downloading a file, you compute the Checksum and compare it with the Checksum provided at the website)
  - Important for digital signature (for authentication)
    - The research on cryptographic hash function is mainly due to the invention of digital signature
  - Key generation, security token …………

Hash Function

- How to ensure that each message digest represents a message uniquely ?
  - The message space size is much larger than the size of the message digest space

    => Theoretically, it is impossible for a message digest to represent only one message
  - Solution: we try to ensure that it is computationally impossible to find two messages with the same message digest

    => then it becomes computationally possible for a message digest to represent only one message

Hash Function

- A strong cryptographic hash function *h* with *n*-bit message digest size has three properties
  - Property 1: Preimage Resistance
    - For any given *y*, it is difficult to find an imput *m* satisfying $h(m) = y$
    - For a strong cryptographic hash function, it requires about 2^n computations to find a preimage

Definition:
Image is a subset of a function's outputs.
Preimage (also called inverse image) of a function's image is the subset of the inputs that maps to that image.

Hash Function

- A strong cryptographic hash function h with n-bit message digest size has three properties
  - Property 2: Second-Preimage Resistance
    - For any given input m, it is difficult to find a different input m' so that $h(m) = h(m')$
    - For a strong cryptographic hash function, it requires about 2^n computations to find a second-preimage

Difference between preimage resistance and second-preimage resistance:
In preimage resistance, only the output is given;
In second-preimage resistance, one of the inputs is given.

Hash Function

- A strong cryptographic hash function *h* with *n*-bit message digest size has three properties
  - Property 3: Collision Resistance
    - It is difficult to find two different inputs *m* and *m'* so that $h(m) = h(m')$
    - For a strong cryptographic hash function, it requires about $2^{n/2}$ computations to find a collision (birthday attack)

Difference between second-preimage resistance and collision resistance:
In second-preimage resistance, one of the inputs is given.

In collision resistance, both inputs can be chosen freely.

Hash Function Overall Structure

- In order to compress a long message, hash function is normally based on the iterative structure:
  - Divide a message into many message blocks

    $$m = m_1 \,||\, m_2 \,||\, m_3 \;\ldots.$$

  - Hash each message block iteratively:

    H_0 = IV (here **IV is a fixed constant**)

    $H_i = f\,(H_{i-1}, m_i)$ (f is called compression function)

    (the size of H_i must be at least as large as the size of the message digest)

But, the above construction is insecure!

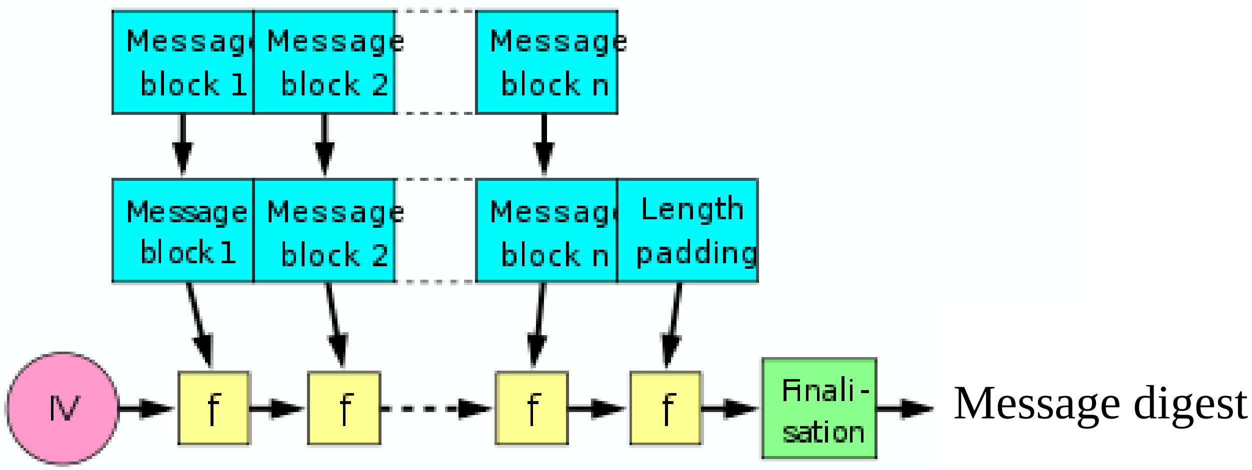
- For example, the message being represented by the last message block is ambiguous! (how many zeros at the end of the message?)

Hash Function Overall Structure

- Merkle-Damgard structure
  - Strengthen the iterative structure with **padding**
    - pad bit ‘1’ to the end of the message
    - pad some zeros
    - pad the message length (in bits)
    - After padding, the overall length should be a multiple of the block size
  - **Finalization** stage: process the output from the last message block, then to generate the message digest (There is no finalization in some hash functions)
  - The most widely used hash function overall structure

Hash Function Overall Structure

- Merkle-Damgard structure



f is a compression function

Hash Function Overall Structure

- Exercises: padding in the Merkle-Damgard structure, assume 512-bit message block size, 64 bits are used to store the message length
  - Message with only one bit, what is the padding?
  - Message with 447 bits, what is the padding?
  - Message with 448 bits, what is the padding?

Hash Function Overall Structure

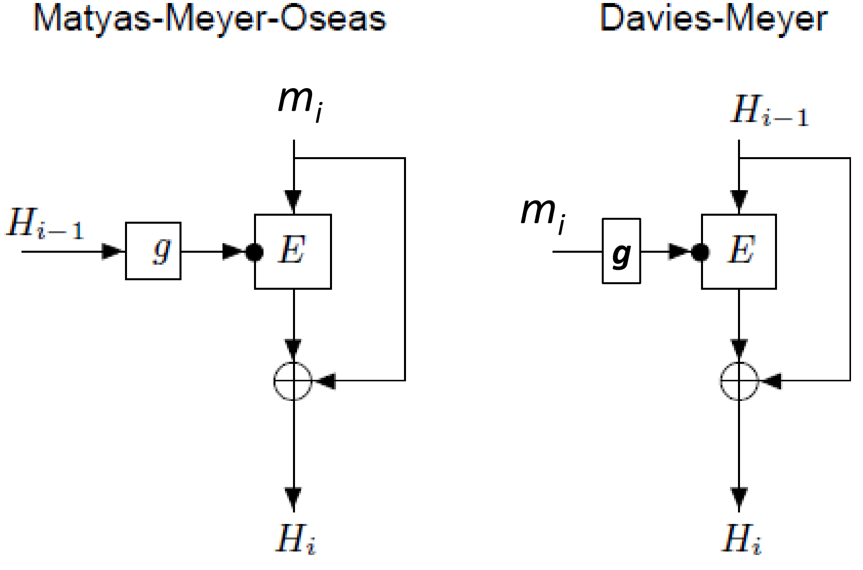
- Exercises: padding in the Merkle-Damgard structure, assume 512-bit message block size, 64 bits are used to store the message length
  - Message with 510 bits, what is the padding?
  - Message with 512 bits, what is the padding?
  - Message with 960 bits, what is the padding?

Compression Function Structure

- Compression function is applied to process each message block:

$$H_i = f\,(H_{i-1}, m_i)$$

- Many different compression function structures
- We learn the compression function based on single block cipher:



  – Davies-Meyer structure is so far the most widely used:
    MD4, MD5, SHA-1, SHA-2

Hash Functions

Hash Functions

- MD4 (1990)
  - 128-bit message digest
- MD5 (1991)
  - 128-bit message digest

Designed by Ron Rivest,
Extremely weak,
MD5 broken by Wang Xiaoyun
etc in 2005

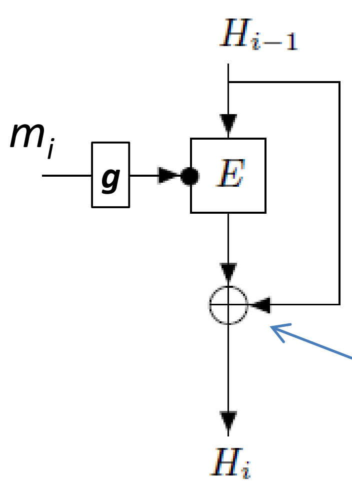
Hash Functions

- Hash function standards of NIST
  - SHA-0, published in 1993 (designed by NSA)
    - 160-bit message digest size
    - Insecure – withdrawn shortly, replaced by SHA-1
    - SHA indicates "Secure Hash Algorithm"
  - SHA-1, published in 1995 (designed by NSA)
    - 160-bit message digest size
    - Insecure (2^{69}, Wang Xiaoyun, etc, 2005)
      - but so far not broken on computer
  - SHA-2, published in 2001 (designed by NSA)
    - SHA-256, SHA-224
      - SHA-224 is based on SHA-256: different IV, truncating 32 bits
    - SHA-512, SHA-384
      - SHA-384 is based on SHA-512: different IV, truncating 64 bits
  - SHA-3, published in 2015 (designed by Belgium cryptographers)

SHA-1

- 160-bit message digest
- 512-bit message block size
- Merkle-Damgard construction
- Davies-Meyer compression function structure

Davies-Meyer



Addition modulo 2^{32} for SHA-1

SHA-1

- Message expansion
  - Expand a 512-bit message block
  - Message block: m_0, m_1, …. m_{15} (each m_i is 32-bit)
  - Expanded message: w_0, w_1, …. w_{79}

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \le t \le 79 \end{cases}$$

SHA-1

- 80 Steps to compress the expanded message
  - A 32-bit constant k_i for each step
- $(a_0, b_0, c_0, d_0, e_0) = H_{i\text{-}1}$ (H_0 is a fixed constant)

$$T = ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t$$

$$e = d$$

$$d = c$$

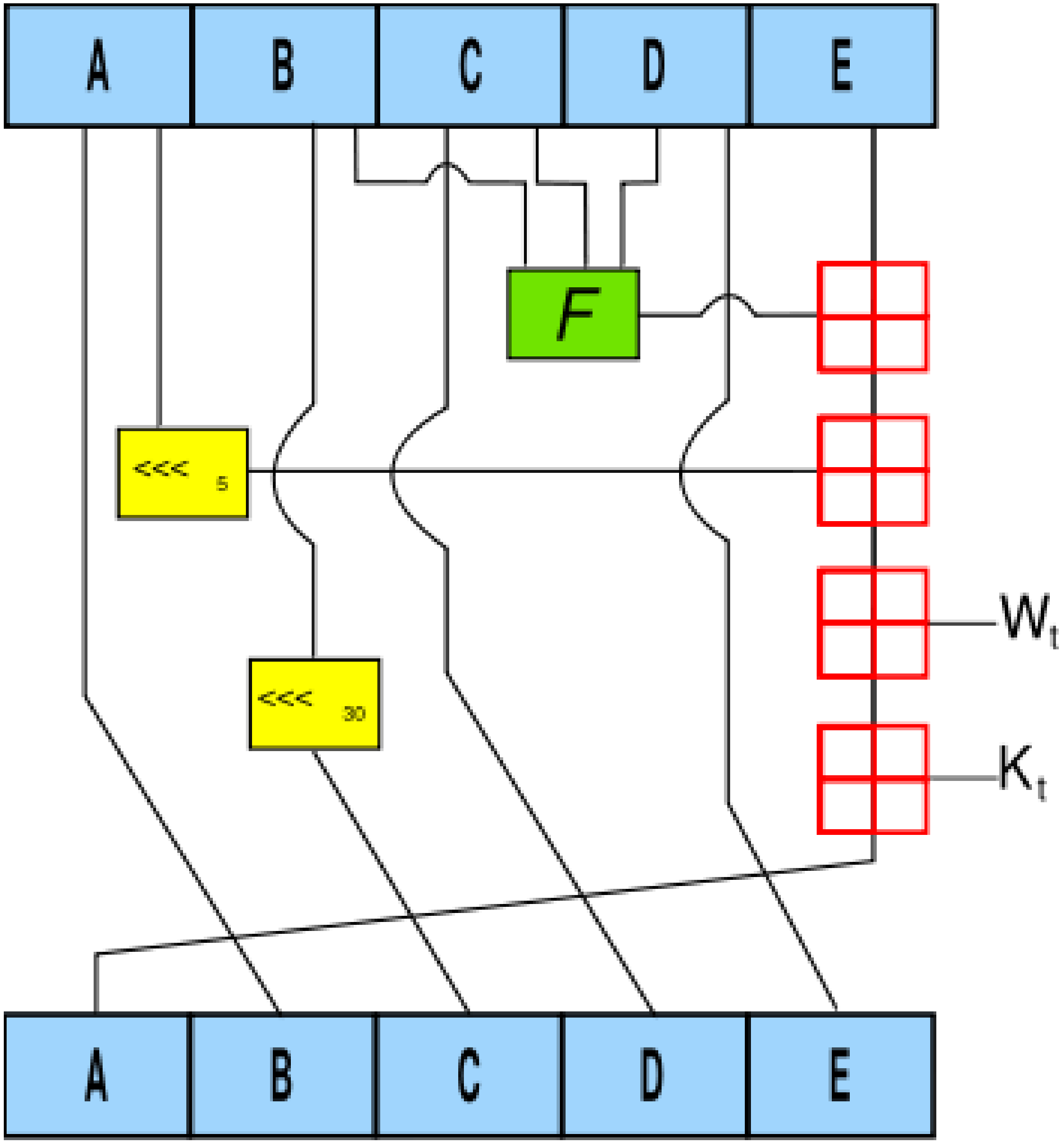
$$c = ROTL^{30}(b)$$

$$b = a$$

$$a = T$$

Each word is 32-bit

$$f_t(x,y,z) = \begin{cases} Ch(x,y,z)=(x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x,y,z)=x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x,y,z)=(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x,y,z)=x \oplus y \oplus z & 60 \le t \le 79. \end{cases}$$



A
B
C
D
E
F
<<< 5
<<< 30
Wt
Kt
A
B
C
D
E

SHA-256

- 256-bit message digest
- 512-bit message block size
- Merkle-Damgard construction
- Davies-Meyer compression function structure

SHA-256

- Message expansion
  - Expand a 512-bit message block
  - Message block: m_0, m_1, …. m_{15} (each m_i is 32-bit)
  - Expanded message: w_0, w_1, …. w_{63}

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$
$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

SHA-256

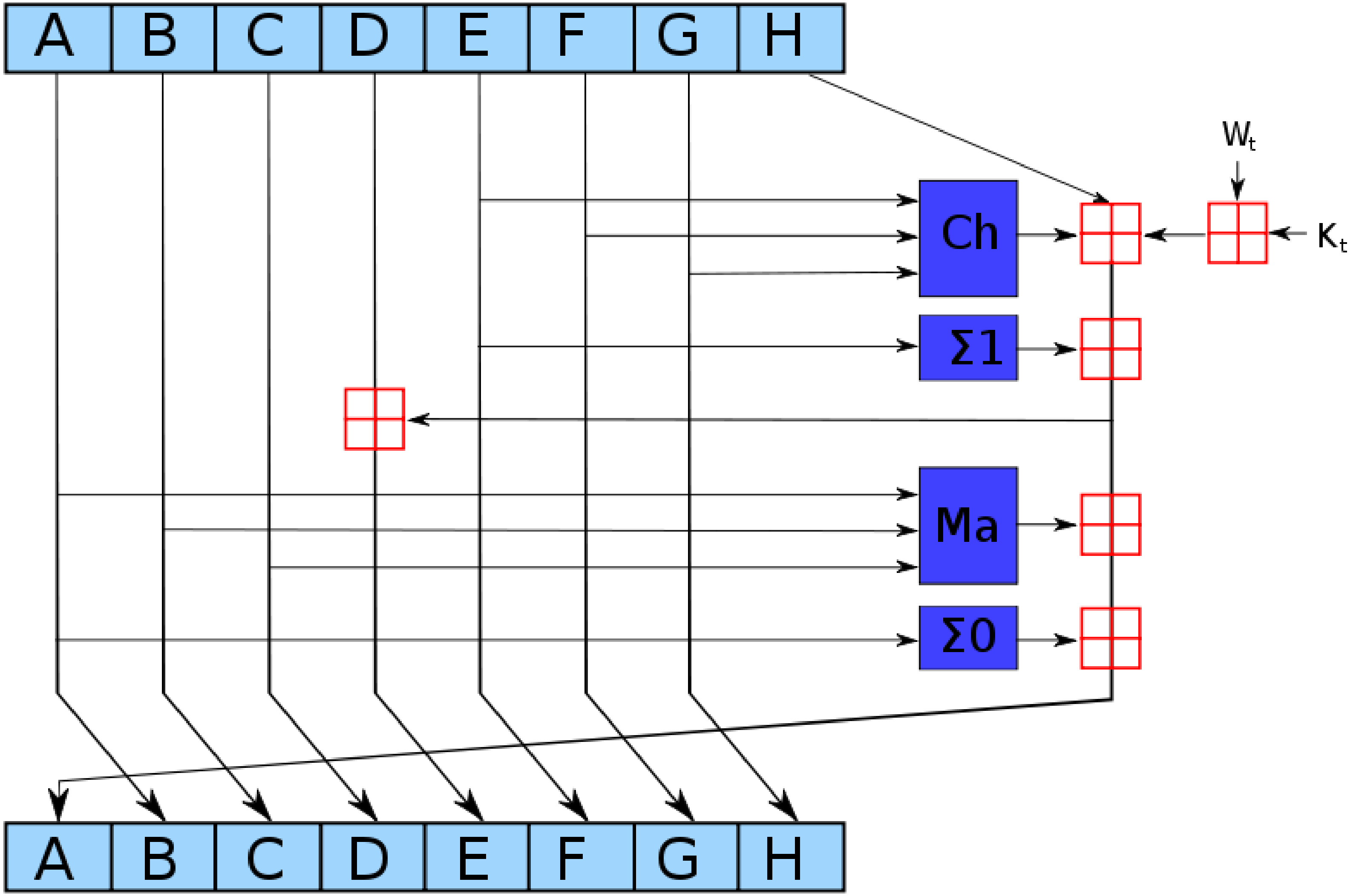
- 64 steps to compress the expanded message
  - A random constant k_i for each step
- $(a_0,b_0,c_0,d_0,e_0,f_0,g_0,h_0) = H_{i\text{-}1}$ (H_0 is a fixed constant)

$$T_1 = h + \sum\nolimits_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$
$$T_2 = \sum\nolimits_0^{\{256\}}(a) + Maj(a,b,c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

Each word is 32-bit

$$\begin{aligned} Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \end{aligned}$$

$$\begin{aligned} \sum\nolimits_0^{\{256\}}(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \\ \sum\nolimits_1^{\{256\}}(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \end{aligned}$$


A
B
C
D
E
F
G
H
Wt
Kt
Ch
Σ1
Ma
Σ0
A
B
C
D
E
F
G
H

SHA-512

- 512-bit message digest
- 1024-bit message block size
- Merkle-Damgard construction
- Davies-Meyer compression function structure

SHA-512

- Message expansion
  - Expand a 1024-bit message block
  - Message block: m_0, m_1, …. m_{15} (each m_i is 64-bit)
  - Expanded message: w_0, w_1, …. w_{79}

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \\ \sigma_1^{\{512\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{512\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 79 \end{cases}$$

$$\sigma_0^{\{512\}}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x)$$
$$\sigma_1^{\{512\}}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x)$$

SHA-512

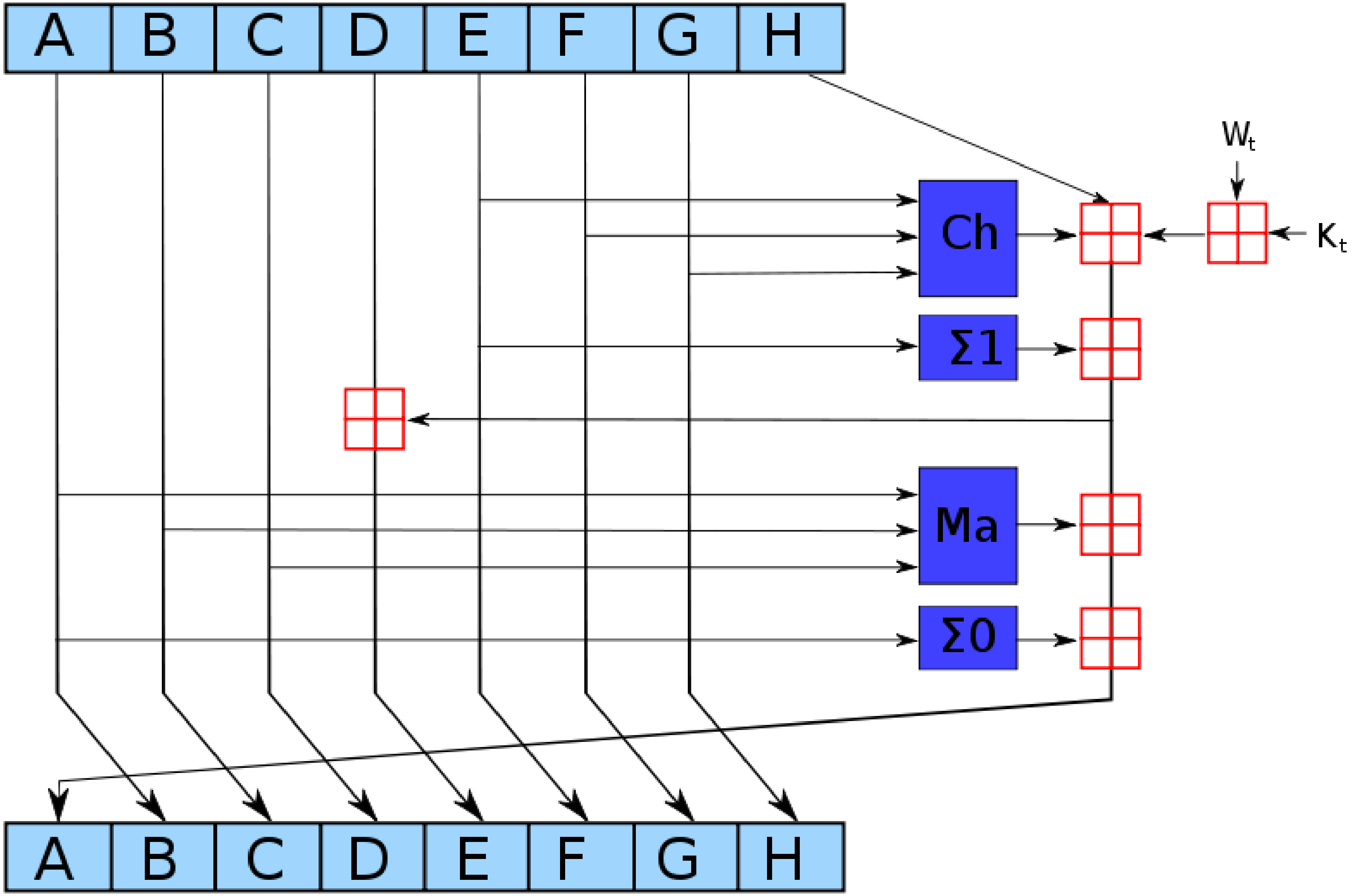
- 80 steps to compress the expanded message
  - A random 64-bit constant k_i for each step
- $(a_0,b_0,c_0,d_0,e_0,f_0,g_0,h_0) = H_{i\text{-}1}$ (H_0 is a fixed constant)

$$T_1 = h + \sum\nolimits_1^{\{512\}}(e) + Ch(e,f,g) + K_t^{\{512\}} + W_t$$
$$T_2 = \sum\nolimits_0^{\{512\}}(a) + Maj(a,b,c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

Each word is 64-bit

$$\begin{aligned} Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \end{aligned}$$

$$\begin{aligned} \sum\nolimits_0^{\{512\}}(x) &= ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \\ \sum\nolimits_1^{\{512\}}(x) &= ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \end{aligned}$$



A
B
C
D
E
F
G
H
Wt
Kt
Ch
Σ1
Ma
Σ0
A
B
C
D
E
F
G
H

SHA-3 Competition (2008—2012)

- NIST hash function competition
  - In order to select a strong and efficient hash functions
  - Received 64 submissions in 2008
  - In 2012, KECCAK was selected as the winner
  - SHA-3 (KECCAK) was published by NIST in 2015
    - Joan Daemen is a co-designer of Keccak, he is also the co-designer of AES

Summary

- Cryptographic hash function
  - Aim: Each message digest represents only one message (computationally)
  - Three security requirements
    - Preimage resistance
    - Second-preimage resistance
    - Collision resistance
- SHA-1
  - Insecure
- SHA-2
  - SHA-224,SHA-256, SHA-384, SHA-512
- SHA-3
  - KECCAK