Cryptography and Security

Cunsheng DING

HKUST, Hong Kong

Version 3

Lecture 12: The Secure Hash Algorithm V1 (SHA-1)

Main Topics of This Lecture

1. Some information of SHA-1.
2. The internal structure of SHA-1.

Brief Information on SHA-1

- SHA was designed by the NIST, and published as a federal information processing standard in 1993.
- SHA-1 is a revised version of SHA by the NSA in 1995, and has a 160-bit hash value.
- The maximum input size is $2^{64} - 1$ bits. This upper bound is not a problem for real-world applications, as no application data has more than $2^{64} - 1$ bits.
- It is used in the Digital Signature Standard (DSS), and Pretty Good Privacy (PGP), and other security systems.

Design Considerations of SHA-1: Security & Performance

- It should be computationally infeasible
  - to find a preimage given a hash value (i.e., the one-way property), and
  - to find a collision. (i.e., the collision-resistance property).

  To this end, SHA-1 should have good confusion and diffusion. Hence, both linear and nonlinear functions should be used in SHA-1.
- SHA-1 was designed to be software-oriented and was implemented in general-purpose computers. In the 1990's, the processor size was 32 bits. To make full use of the processor size, the operations of SHA-1 work on words of 32 bits.

SHA-1 Description

Step 1: Append padding bits

- The message is padded so that its length in bits is $\equiv 448 \bmod 512$. Padding is always added, even if the message is already of the desired length. Thus the number of padding bits is in the range of 1 to 512.
- The padding rule: 1 followed by the number of necessary 0's.

Remark: 448 is chosen here because another 64 bits will be appended in Step 2. Note that $448 + 64 = 512$.

SHA-1 Description

Step 2: Append the binary representation of the length of the original message

- Since the bit length of the original message $< 2^{64}$, append the 64 bits of the binary representation of the length of the original message after Step 1.
- This block of 64 bits is treated as an unsigned 64-bit integer (most significant byte first).

Remark: After the padding of Step 2, the length of the new message is a multiple of 512 bits.



SHA-1 Description

Step 3: Initialize the MD (message digest) buffer

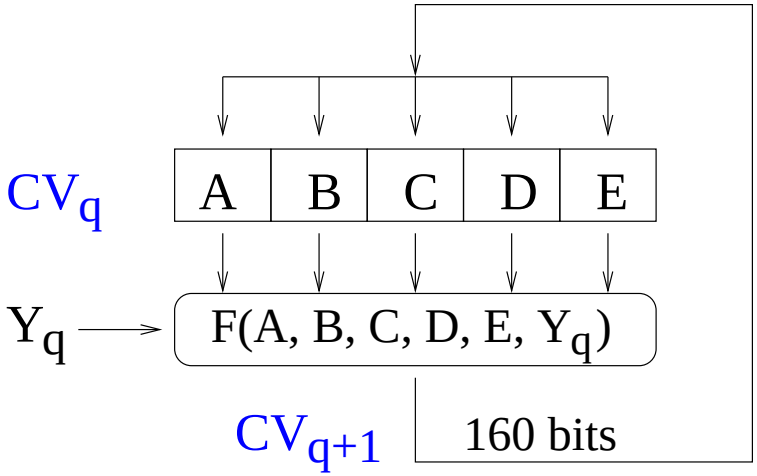
- A 160-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as five 32-bit registers A, B, C, D, E.
- These registers are initialized to the following 32-bit integers (hexadecimal values):

$$A = 67452301,\ B = efcdab89,\ C = 98badcfe,$$
$$D = 10325476,\ E = c3d2e1f0$$

SHA-1 Description

Step 4: Process padded message in 512-bit blocks

Let $Y_{L-1} \cdots Y_1 Y_0$ be the message after the padding, where Y_i is a 512-bit block. Then SHA-1 computes the hash value as follows.

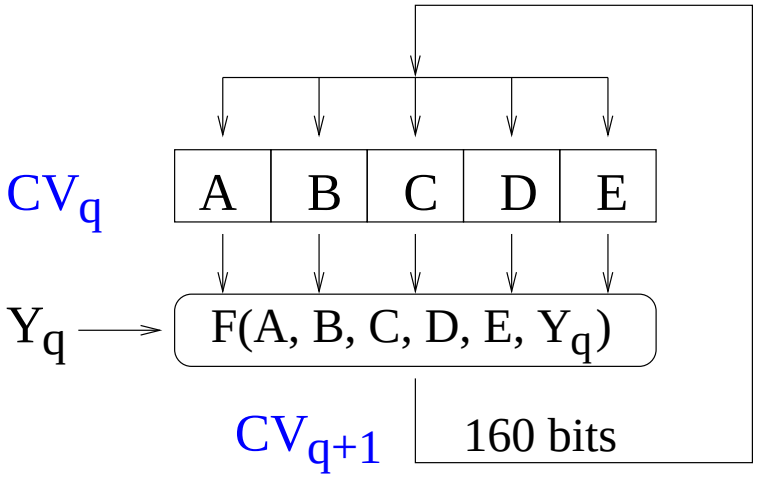


Where $F(A, B, C, D, E, Y_q)$ is a function to be specified later, CV_q is the content of the buffer at time unit q, and CV_{q+1} is the new value for replacing CV_q. The final content of the buffer is the hash code.

SHA-1 Description

Step 5: Output

After processing the last block of message, the content of the buffer is the hash code. That is, $MD = CV_L$.



Remark: The core part of SHA-1 is the function

$$F(A, B, C, D, E, Y_q).$$

SHA-1 Description: Function $F(A, B, C, D, E, Y_q)$

The constants K_i:

step number	hexadecimal	where from
$0 \le t \le 19$	$K_t = 5a827999$	$[2^{30}\sqrt{2}]$
$20 \le t \le 39$	$K_t = 6ed9eba1$	$[2^{30}\sqrt{3}]$
$40 \le t \le 59$	$K_t = 8f1bbcdc$	$[2^{30}\sqrt{5}]$
$60 \le t \le 79$	$K_t = ca62c1d6$	$[2^{30}\sqrt{10}]$

Remark: These constants are needed inside the function $F(A, B, C, D, E, Y_q)$, and are called **magic numbers**.

SHA-1 Description: Function $F(A, B, C, D, E, Y_q)$

The $W[0..79]$ computed from each block Y_q:

For each message block Y_q, 80 words W_i of 32 bits are computed:

- Define $W_0||W_1||\cdots||W_{15} = Y_q$, i.e., divide Y_q into 16 blocks of 32 bits.
- For $16 \leq t \leq 79$, W_t is computed by

$$W_t = S^1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3})$$

recursively, where $S^i(x)$ is the circular left shift of the 32-bit word x for i positions. For example,

$$S^1(x_1x_2 \ldots x_{32}) = x_2x_3 \ldots x_{32}x_1$$

and

$$S^2(x_1x_2 \ldots x_{32}) = x_3x_4 \ldots x_{32}x_1x_2.$$

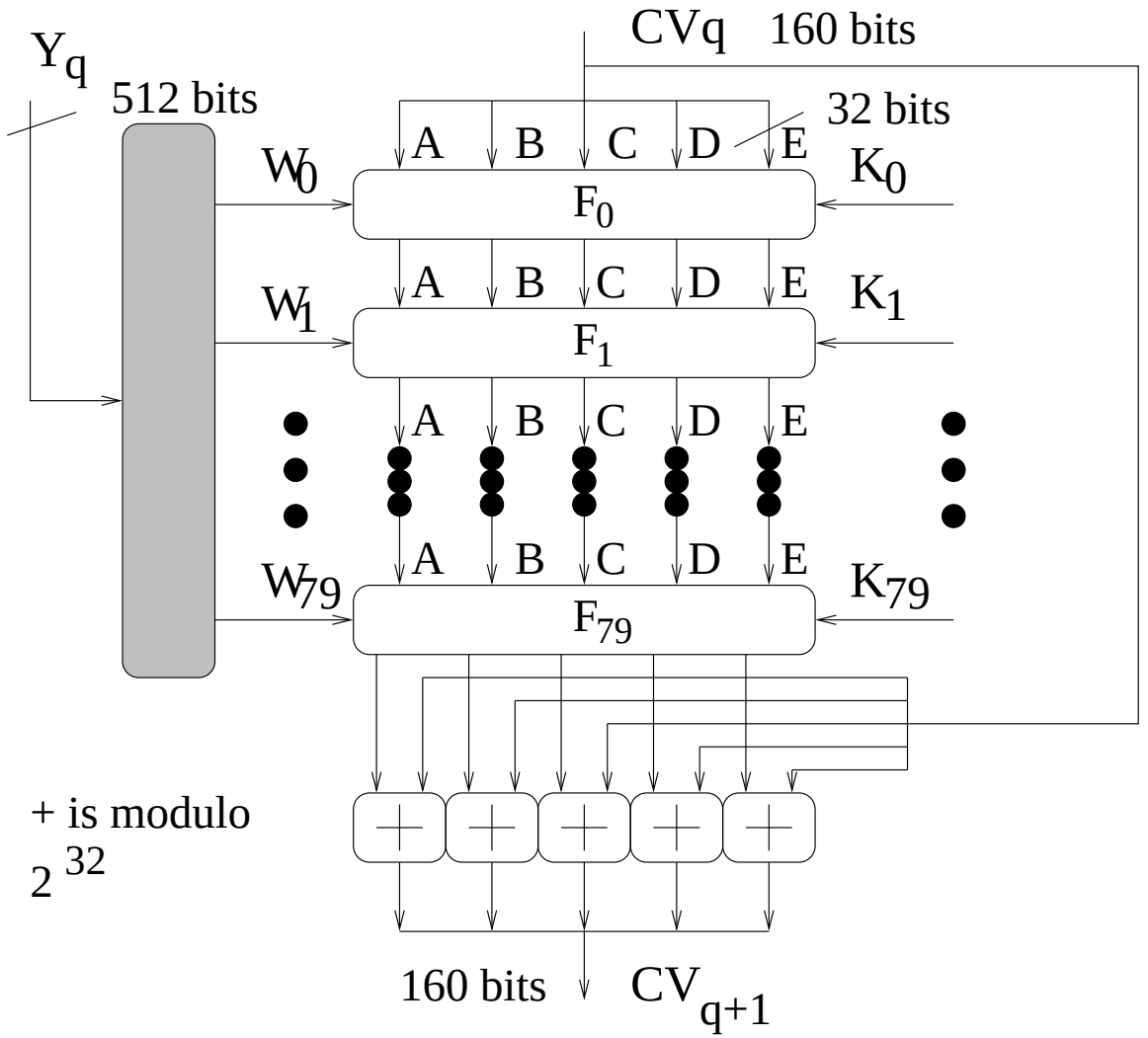
SHA-1 Description: Function $F(A, B, C, D, E, Y_q)$

The $W[0..79]$ computed from each block Y_q:

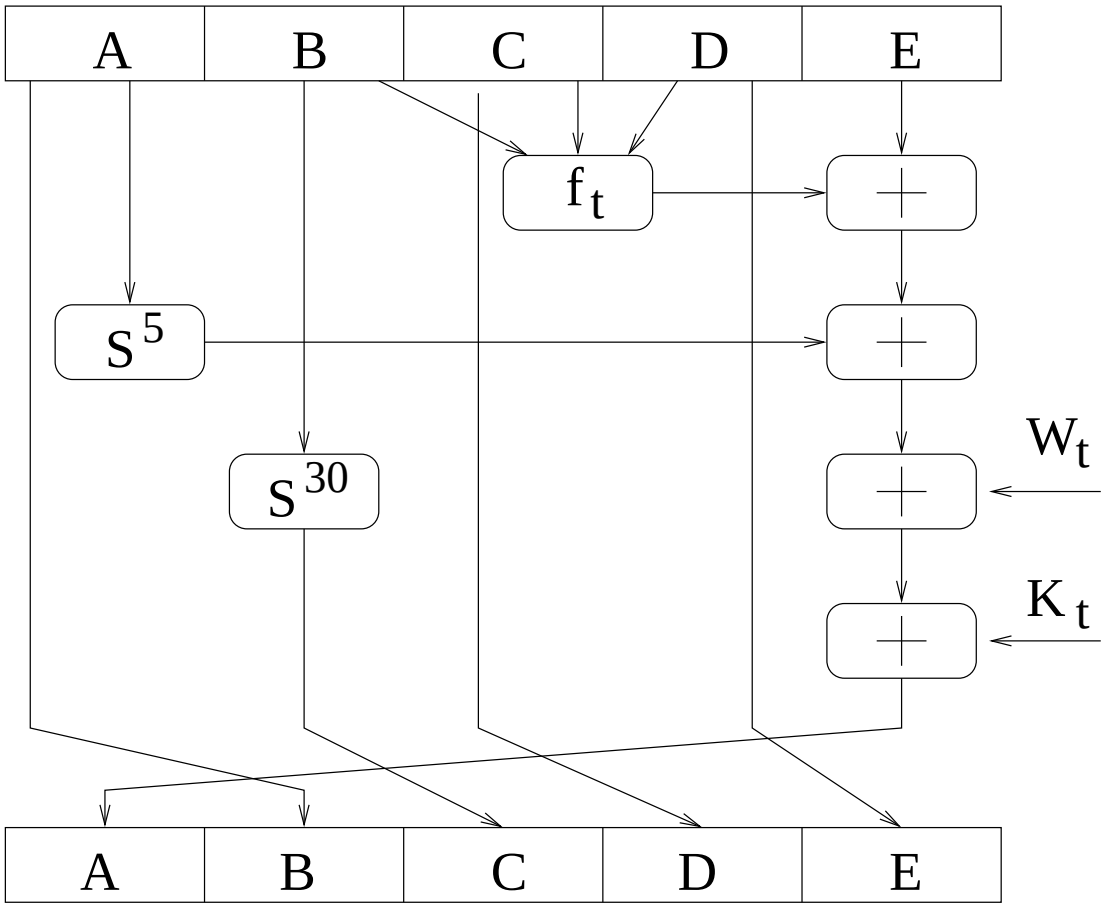
Comments and design ideas: It would be desirable to have all the 80 words $W_0, \cdots, W_{79}$ take as many possible 32-bit strings as possible. Hence linear recurrence approach is a good choice.

The linear operation $S^1(x)$ is used as it will change the exact linear recurrence formula.

SHA-1 Description: Function $F(A, B, C, D, E, Y_q)$



SHA-1 Description: Function F_t



The function F_t, where f_t will be described next.

SHA-1 Description: Function $F(A, B, C, D, E, Y_q)$

The functions f_t:

step number	function	function value
$0 \le t \le 19$	$f_t(B, C, D)$	$(B \wedge C) \vee (\overline{B} \wedge D)$
$20 \le t \le 39$	$f_t(B, C, D)$	$B \oplus C \oplus D$
$40 \le t \le 59$	$f_t(B, C, D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$60 \le t \le 79$	$f_t(B, C, D)$	$B \oplus C \oplus D$

Remark: $\vee =$ OR, $\wedge =$ AND, $^- =$ NOT, $\oplus =$ EXOR.

Remark: $B \wedge C$ is the bitwise logical AND.

Understanding the Design Ideas

1. The 80 functions f_t use only three distinct functions. What is the idea of using only three distinct functions?

   – This is for having a small program size in software implementation.

2. In the single step operation, why four different kinds of operations $\wedge$, $\vee$, $\oplus$, and $+_{2^{32}}$ are used?

   – This is for confusion.

3. Why shift operations are used in the single step operation?

   – This is for diffusion.

4. What is the purpose of having 80 passes?

   – This is for both confusion and diffusion.

Security and other Issues

1. It is simple to describe and implement SHA-1 in software. Its software implementation does not require large programs or substitution tables.

2. Strong collisions of SHA-1 and MD5 were found in about 2004. But for any message x, it is not known how to find a y such that $H(x) = H(y)$. So it is not really a threat for some applications. But it was time to design new hash functions in early 2000.