MD5 (Message-Digest Algorithm 5)

- designed by Ronald Rivest
  (MIT Lab for Computer Science and RSA Inc.)

- produces a 128-bit hash value

- the most widely used hash algorithm
  - in recent times have both brute-force & cryptanalytic concerns

- specified as Internet standard RFC1321

MD5 Overview

任意长度 → **MD5** → 128 bits

- ✓ pad message so its length is congruent to 448 mod 512

  与$(M)_{512}$=448同余)， 10…0 - MUST DO IT !!!!!!!

- ✓ append a 64-bit length value to message 填充前消息长度
- ✓ initialize 4 (128-bit) MD buffer (A,B,C,D)

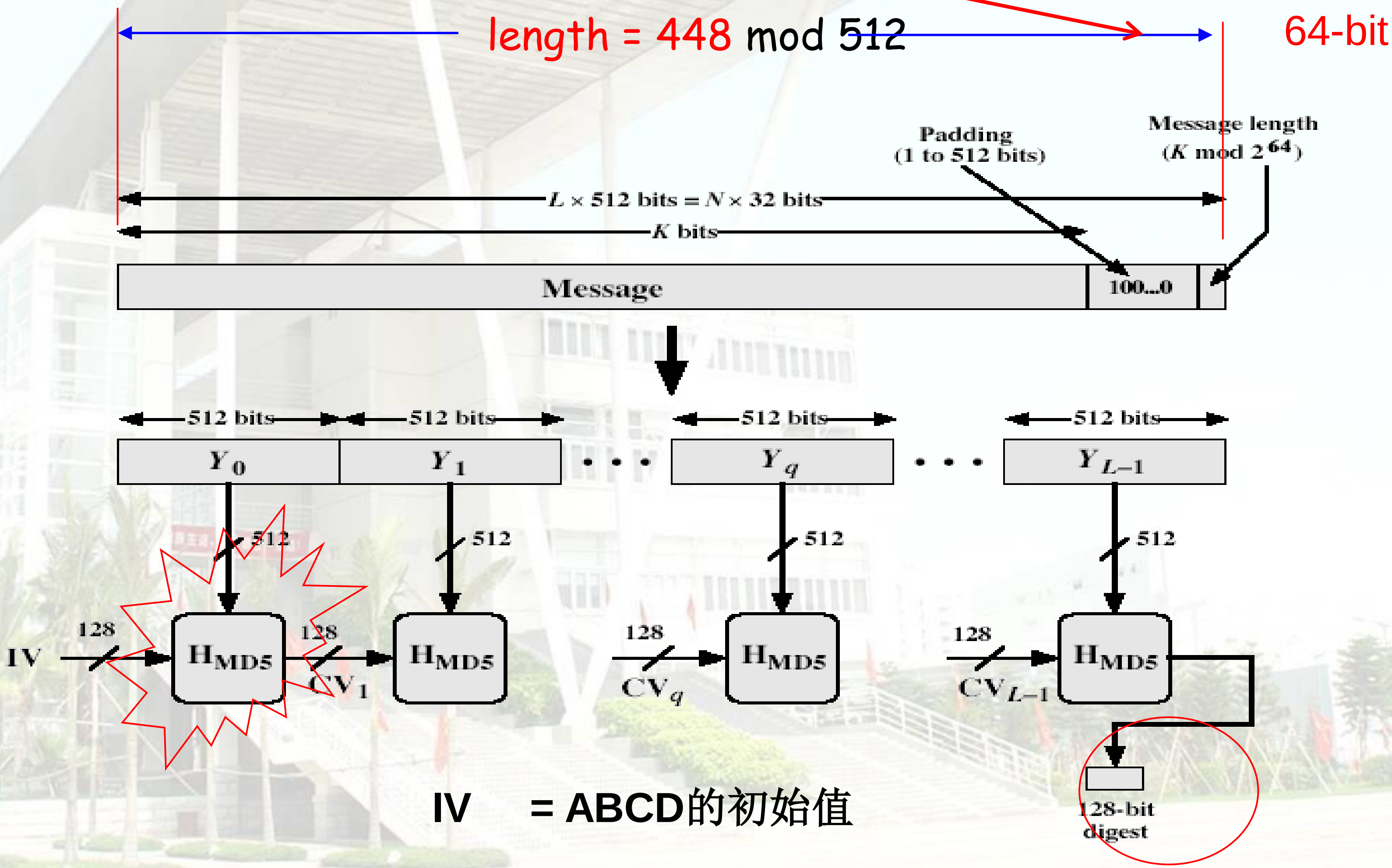
  A=0x01234567， B=0x89abcdef，

  C=0xfedcba98， D=0x76543210

- ✓ process message in 512-bit blocks:
  - ➢ use 4 rounds of operations on message block & buffer
  - ➢ add output to buffer input to form new buffer value
- ✓ output hash value is the final buffer value

MD5 Algorithm Structure

填充一个1和无数个0，然后，在后面附加一个以64位表示填充前信息长度。
信息字节长度=N*512+448+64=(N+1)*512，512的整数倍。



IV = ABCD的初始值

MD5 Compression Function

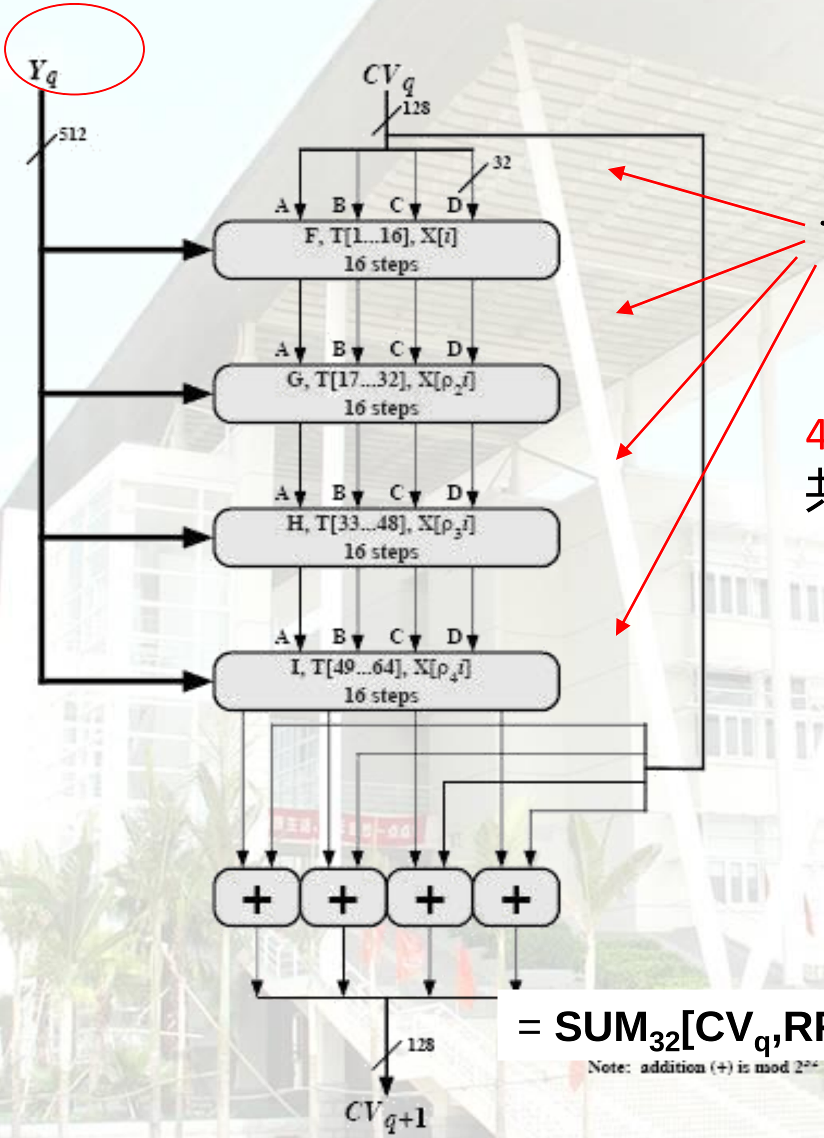


Figure 12.2 MD5 Processing of a Single 512-bit Block

T[i]: a constant value derived from sin

四轮循环运算

4 rounds of operations, 每轮16步
共 64步

$= SUM_{32}[CV_q, RF_I(Y_q, RF_H(Y_q, RF_G(Y_q, RF_G(Y_q, RF_F(Y_q, CV_q))))]$

MD5 Function and Truth Table

Table 12.1 Key Elements of MD5

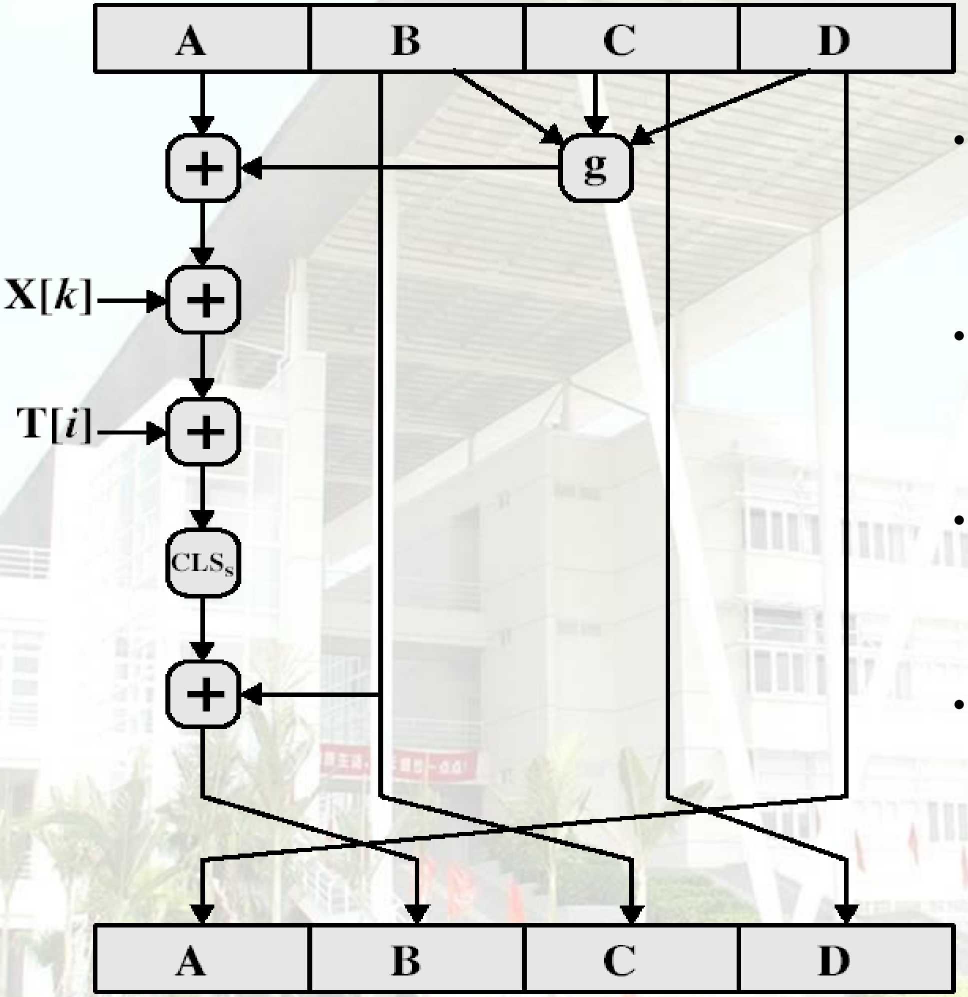
(a) Truth table of logical functions

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

$\wedge$: AND, $\vee$: OR,
$\overline{}$: NOT, $\oplus$:XOR

Function g	g(b,c,d)
1 F(b,c,d)	$(b \wedge c) \vee (\overline{b} \wedge d)$
2 G(b,c,d)	$(b \wedge d) \vee (c \wedge \overline{d})$
3 H(b,c,d)	$b \oplus c \oplus d$
4 I(b,c,d)	$c \oplus (b \vee \overline{d})$

MD5 Compression Function



- each round has 16 steps of the form:

  a = b+((a+g(b,c,d)+X[k]+T[i])<<<s)
- a,b,c,d refer to the 4 words of the buffer, but used in varying permutations
- where g(b,c,d) is a different nonlinear function in each round (F,G,H,I)
- T[i] is a constant value derived from sin

第一轮

- FF(a,b,c,d,M0,7,0xd76aa478)
  FF(d,a,b,c,M1,12,0xe8c7b756)
  FF(c,d,a,b,M2,17,0x242070db)
  FF(b,c,d,a,M3,22,0xc1bdceee)
  FF(a,b,c,d,M4,7,0xf57c0faf)
  FF(d,a,b,c,M5,12,0x4787c62a)
  FF(c,d,a,b,M6,17,0xa8304613)
  FF(b,c,d,a,M7,22,0xfd469501)
  FF(a,b,c,d,M8,7,0x698098d8)
  FF(d,a,b,c,M9,12,0x8b44f7af)
  FF(c,d,a,b,M10,17,0xffff5bb1)
  FF(b,c,d,a,M11,22,0x895cd7be)
  FF(a,b,c,d,M12,7,0x6b901122)
  FF(d,a,b,c,M13,12,0xfd987193)
  FF(c,d,a,b,M14,17,0xa679438e)
  FF(b,c,d,a,M15,22,0x49b40821)

第二轮

- GG(a,b,c,d,M1,5,0xf61e2562)
GG(d,a,b,c,M6,9,0xc040b340)
GG(c,d,a,b,M11,14,0x265e5a51)
GG(b,c,d,a,M0,20,0xe9b6c7aa)
GG(a,b,c,d,M5,5,0xd62f105d)
GG(d,a,b,c,M10,9,0x02441453)
GG(c,d,a,b,M15,14,0xd8a1e681)
GG(b,c,d,a,M4,20,0xe7d3fbc8)
GG(a,b,c,d,M9,5,0x21e1cde6)
GG(d,a,b,c,M14,9,0xc33707d6)
GG(c,d,a,b,M3,14,0xf4d50d87)
GG(b,c,d,a,M8,20,0x455a14ed)
GG(a,b,c,d,M13,5,0xa9e3e905)
GG(d,a,b,c,M2,9,0xfcefa3f8)
GG(c,d,a,b,M7,14,0x676f02d9)
GG(b,c,d,a,M12,20,0x8d2a4c8a)

第三轮

- HH(a,b,c,d,M5,4,0xfffa3942)
  HH(d,a,b,c,M8,11,0x8771f681)
  HH(c,d,a,b,M11,16,0x6d9d6122)
  HH(b,c,d,a,M14,23,0xfde5380c)
  HH(a,b,c,d,M1,4,0xa4beea44)
  HH(d,a,b,c,M4,11,0x4bdecfa9)
  HH(c,d,a,b,M7,16,0xf6bb4b60)
  HH(b,c,d,a,M10,23,0xbebfbc70)
  HH(a,b,c,d,M13,4,0x289b7ec6)
  HH(d,a,b,c,M0,11,0xeaa127fa)
  HH(c,d,a,b,M3,16,0xd4ef3085)
  HH(b,c,d,a,M6,23,0x04881d05)
  HH(a,b,c,d,M9,4,0xd9d4d039)
  HH(d,a,b,c,M12,11,0xe6db99e5)
  HH(c,d,a,b,M15,16,0x1fa27cf8)
  HH(b,c,d,a,M2,23,0xc4ac5665)

第四轮

- II(a,b,c,d,M0,6,0xf4292244)
  II(d,a,b,c,M7,10,0x432aff97)
  II(c,d,a,b,M14,15,0xab9423a7)
  II(b,c,d,a,M5,21,0xfc93a039)
  II(a,b,c,d,M12,6,0x655b59c3)
  II(d,a,b,c,M3,10,0x8f0ccc92)
  II(c,d,a,b,M10,15,0xffeff47d)
  II(b,c,d,a,M1,21,0x85845dd1)
  II(a,b,c,d,M8,6,0x6fa87e4f)
  II(d,a,b,c,M15,10,0xfe2ce6e0)
  II(c,d,a,b,M6,15,0xa3014314)
  II(b,c,d,a,M13,21,0x4e0811a1)
  II(a,b,c,d,M4,6,0xf7537e82)
  II(d,a,b,c,M11,10,0xbd3af235)
  II(c,d,a,b,M2,15,0x2ad7d2bb)
  II(b,c,d,a,M9,21,0xeb86d391)

MD5的实现

速度

用32 bits软件易于高速实现

简洁与紧致性

描述简单，短程序可实现

采用little-endian结构

Rivest选择little endian来表述消息(32 bit).

little endian 和 big endian

假设机器以每个内存单元以8位（一个字节）为单位.
little endian和big endian：表示计算机字节顺序的格式.

假设从地址0x00000000开始保存有数据0x1234abcd,
那么有两种不同的内存顺序:

1)little endian内存中的存放顺序：
0x00000000-0xcd,
0x00000001-0xab,
0x00000002-0x34,
0x00000003-0x12
2)big endian 内存中的存放顺序：
0x00000000-0x12,
0x00000001-0x34,
0x00000002-0xab,
0x00000003-0xcd
little endian把低字节存放在内存的低位; （如Intel）
而big endian将低字节存放在内存的高位. （如SUN）

Secure Hash Algorithm（SHA）

- SHA originally designed by NIST & NSA in 1993
- revised in 1995 as SHA-1

- US standard with DSA signature scheme
  - Standard:
    FIPS 180-1 （1995）， FIPS 180-2 （2002）
    FIPS 180-3 （2008），
    RFC3174 （2001）
- produces 160-bit hash values

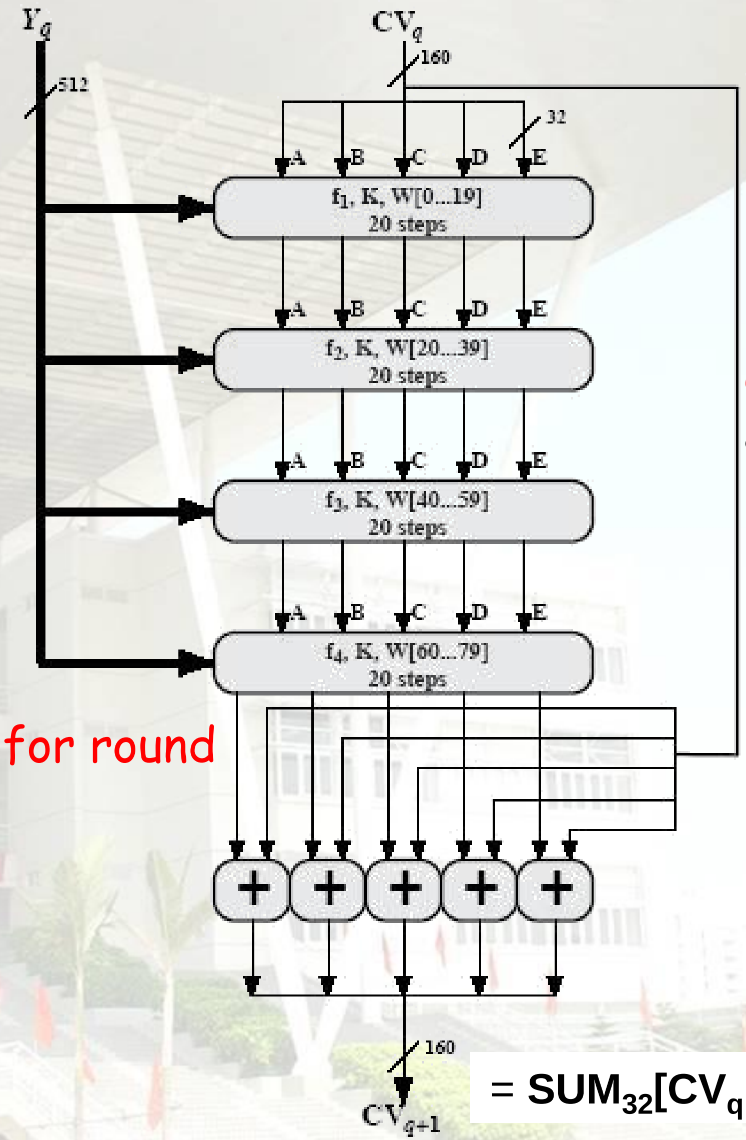
SHA Overview

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 5 buffer (A,B,C,D,E) to (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. process message in 512-bit chunks:
   - use 4 rounds of operations on message block & buffer
   - add output to input to form new buffer value
5. output hash value is the final buffer value

SHA-1

four rounds

f_i is nonlinear function for round



each round has 20 steps, 80步

$= SUM_{32}[CV_q, ABCD_q]$

Figure 12.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

SHA-1 Compression Function

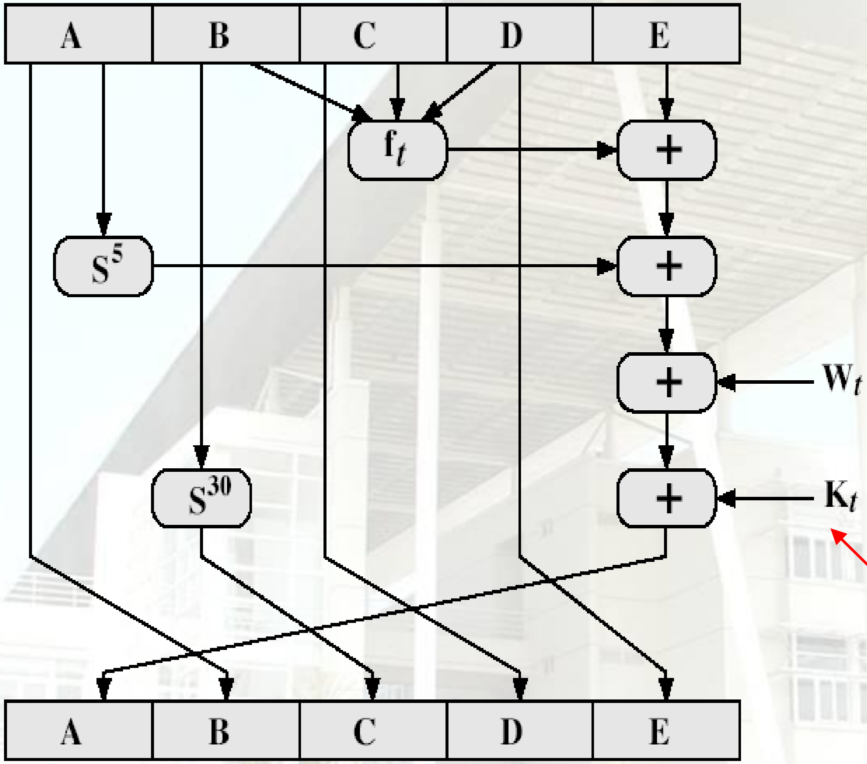


Table 12.2 Truth Table of Logical Functions for SHA-1

f(t,B,C,D) is nonlinear function for round

B	C	D	$f_{0..19}$	$f_{20..39}$	$f_{40..59}$	$f_{60..79}$
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

derived from the message block

- $0 \le t < 20$,
  K[t] = [230 x sqrt(2)]
- $20 \le t < 40$,
  K[t] =[230 x sqrt(3)]
- $30 \le t < 60$,
  K[t] = [230 x sqrt(5)]
- $60 \le t < 80$,
  K[t] =[230 x sqrt(10)]

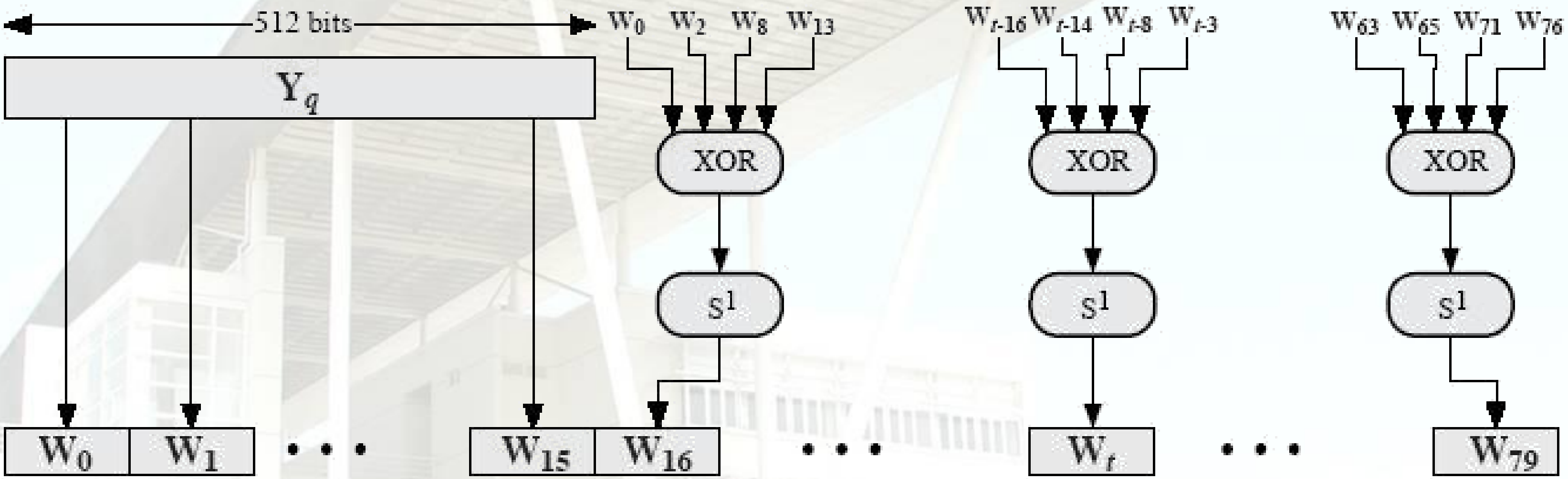
a constant value

(A,B,C,D,E) <-(E+f(t,B,C,D)+(A<<5)+Wt+Kt),A,(B<<30),C,D)

t is the step number

Step	Function Name	Function Value
$(0 \le t \le 19)$	$f_1 = f(t,B,C,D)$	$(B \wedge C) \vee (\overline{B} \wedge D)$
$(20 \le t \le 39)$	$f_2 = f(t,B,C,D)$	$B \oplus C \oplus D$
$(40 \le t \le 59)$	$f_3 = f(t,B,C,D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$(60 \le t \le 79)$	$f_4 = f(t,B,C,D)$	$B \oplus C \oplus D$

Creation of W



$W[t] = Y_i[t] \qquad 0 \le t < 16$

$W[t] = (W[t-16] \oplus W[t-14] \oplus W[t-8] \oplus W[t-3]) <<< 1$

$16 \le t < 80$

Figure 12.7 Creation of 80-word Input Sequence for SHA-1 Processing of Single Block

SHA-1 EXAMPLES

One-Block Message

Let the message, *M*, be the 24-bit ASCII string "**abc**",

01100001 01100010 01100011

1) pad message so its length is 448 mod 512

The message is padded by appending a "1" bit, followed by 423 "0" bits.

2) append a 64-bit length value to message (two 32-bit word)

ending with the hex value 00000000 00000018

3) initialise 5 buffer (A,B,C,D,E) to

67452301,efcdab89,98badcfe,10325476,c3d2e1f0

SHA-1 EXAMPLES

4) assign the words $W0,\ldots,W15$ of the message schedule:

$W0$ = 61626380	$W1$ = 00000000	$W2$ = 00000000	$W3$ = 00000000
$W4$ = 00000000	$W5$ = 00000000	$W6$ = 00000000	$W7$ = 00000000
$W8$ = 00000000	$W9$ = 00000000	$W10$ = 00000000	$W11$ = 00000000
$W12$ = 00000000	$W13$ = 00000000	$W14$ = 00000000	$W15$ = 00000018

5) The resulting 160-bit message digest is

a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d

SHA-1 EXAMPLES

Multi-Block Message

Let the message, *M*, be the 448-bit ASCII string

"**abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq**"

1) pad message so its length is 448 mod 512

The message is padded by appending a "1" bit, followed by 511 "0" bits,

2) append a 64-bit length value (448) to message (two 32-bit word)

ending with the hex value 00000000 000001c0

00000000 000001C0

448-bit 512-bit（ by 511 "0" ） 64-bit

$(M)_{512}=448$

3) initialise 5 buffer (A,B,C,D,E) to

67452301,efcdab89,98badcfe,10325476,c3d2e1f0

SHA-1 EXAMPLES (4)

4) The words of the first padded message block, *M*(1), are then assigned to the words *W*0,…,*W*15 of the message schedule:

*W*0 =	61626364	*W*1 =	62636465	*W*2 =	63646566	*W*3 =	64656667
*W*4 =	65666768	*W*5 =	66676869	*W*6 =	6768696a	*W*7 =	68696a6b
*W*8 =	696a6b6c	*W*9 =	6a6b6c6d	*W*10 =	6b6c6d6e	*W*11 =	6c6d6e6f
*W*12 =	6d6e6f70	*W*13 =	6e6f7071	*W*14 =	80000000	*W*15 =	00000000.

The words of the *second* padded message block, *M*(2), are then assigned to the words *W*0,…,*W*15 of the message schedule:

*W*0 =	00000000	*W*1 =	00000000	*W*2 =	00000000	*W*3 =	00000000
*W*4 =	00000000	*W*5 =	00000000	*W*6 =	00000000	*W*7 =	00000000
*W*8 =	00000000	*W*9 =	00000000	*W*10 =	00000000	*W*11 =	00000000
*W*12 =	00000000	*W*13 =	00000000	*W*14 =	00000000	*W*15 =	000001c0.

5) The resulting 160-bit message digest is:

84983e44 1c3bd26e baae4aa1 f95129e5 e54670f1

SHA-1 vs MD5

- brute force attack is harder (160 vs 128)
- slower than MD5 (80 vs 64 steps)
- both designed as simple and compact
- optimised for big endian CPU's

MD5 is optimised for little endian CPU’s)