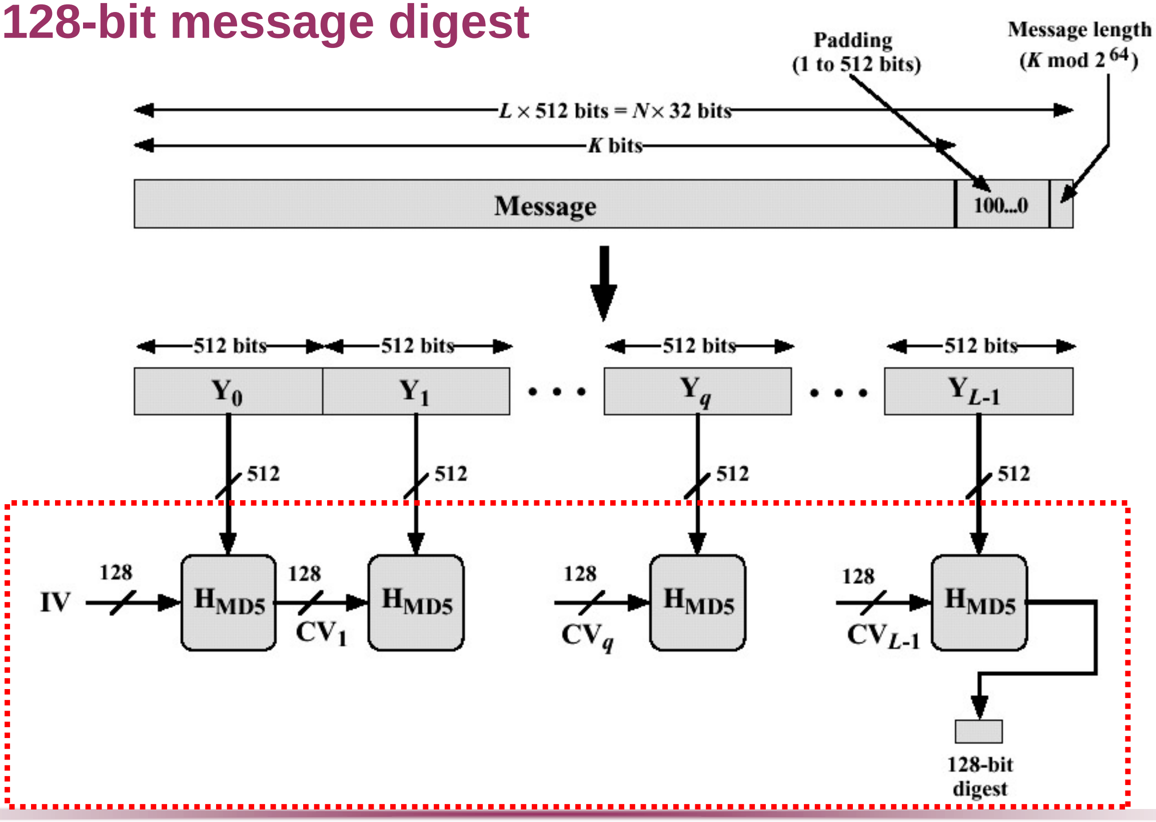
CHAPTER 9 HASH ALGORITHMS

- MD5 Message Digest Algorithm
- Secure Hash Algorithm
- RIPEMD-160
- HMAC

MD5 Message Digest Algorithm

MD5 Logic –

- Input – an arbitrary-length message which is processed in 512-bit block
- Output – 128-bit message digest



MD5 Message Digest Algorithm (cont.)

MD5 Logic (cont.)

- Step 1: append padding bits
  - The message is padded so that its length in bits is congruent to 448 mod 512 (length $\equiv$ 448 mod 512)
  - Padding is always added (1 to 512 bits)
  - The padding pattern is 100…0
- Step 2: append length
  - A 64-bit length in bits of the original message is appended
  - If the original length is greater than 2^{64}, the length is modulo 2^{64}
  - The expanded message is $L \times 512$ bits: $Y_0, Y_1, \ldots, Y_{L-1}$
  - A total of $L \times 16$ 32-bit words: M[0...N-1], $N = L \times 16$

MD5 Message Digest Algorithm (cont.)

MD5 Logic (cont.)

- Step 3: initialize MD buffer
  - A 128-bit buffer is used to hold intermediate and final results of the hash function
  - The buffer is represented as 4 32-bit registers (A, B, C, D) initialized to the following integers (hexadecimal values):

    A = 67452301
    B = EFCDAB89
    C = 98BADCFE
    D = 10325476

  - The values are stored in little-ending order, i.e., the least significant byte of a word in the low-address byte position:

    word A = 01 23 45 67
    word B = 89 AB CD EF
    word C = FE DC BA 98
    word D = 76 54 32 10

MD5 Message Digest Algorithm (cont.)

MD5 Logic (cont.)

- Step 4: process message in 512-bit (16-word) blocks
  - A compression function H_{MD5} consists of 4 "rounds" of processing
    - Input – 512-bit block Y_q and 128-bit buffer value CV_q represented by ABCD
    - Output – 128-bit chaining variable CV_{q+1}
  - For each round operation
    - Input –Y_q and ABCD
    - Output – updated ABCD
    - Each round makes use of 1/4 of a 64-element table T[1…64] – T[i] is the integer part of $2^{32} \times \text{abs}[\sin(i)]$, where i is in radian
    - Provides a "randomized" set of 32-bit patterns and thus eliminate the regularities in the input data
  - The output of the last round is added to the input of the first round (CV_q) to produce CV_{q+1}

MD5 Message Digest Algorithm (cont.)

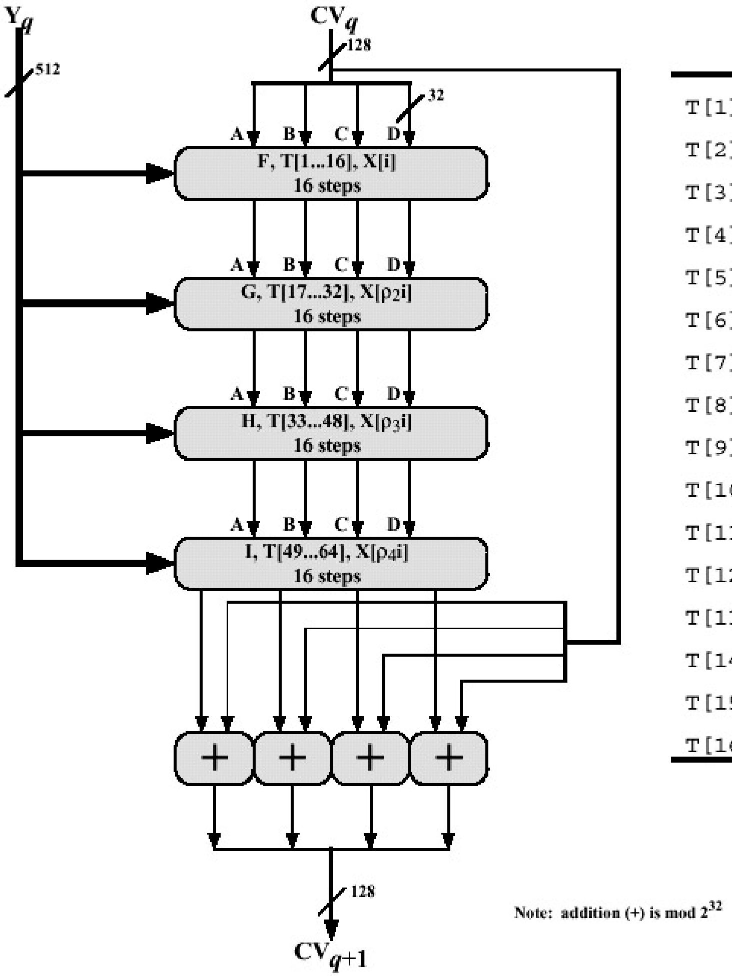


Table T, constructed from the sine function

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E5A51	T[35] = 699D6122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57C0FAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBC8	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

MD5 Message Digest Algorithm (cont.)

MD5 Logic (cont.)

- Step 5: output
  - The output from the L-th stage is the 128-bit message digest
- Summary of the MD5:

$$CV_0 = IV$$
$$CV_{q+1} = SUM_{32}(CV_q, RF_I[Y_q, [RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]])$$
$$MV = CV_L$$

where

IV = initial value of the ABCD buffer
Y_q = the q-th 512-bit block of the message
L = the number of blocks in the message
CV_q = chaining variable processed with the q-th block of the message
RF_x = round function using primitive logical function x
MD = final message digest value
SUM_{32} = addition modulo 2^{32} performed on each word of the pair of inputs

MD5 Message Digest Algorithm (cont.)

MD5 Compression Function

- Each round consists of 16 steps operating on the buffer ABCD with each step of the form:

  $$a \leftarrow b + ((a + g(b, c, d) + X[k] + T[I]) <<< s)$$

  where

  a,b,c,d = the four words of the buffer

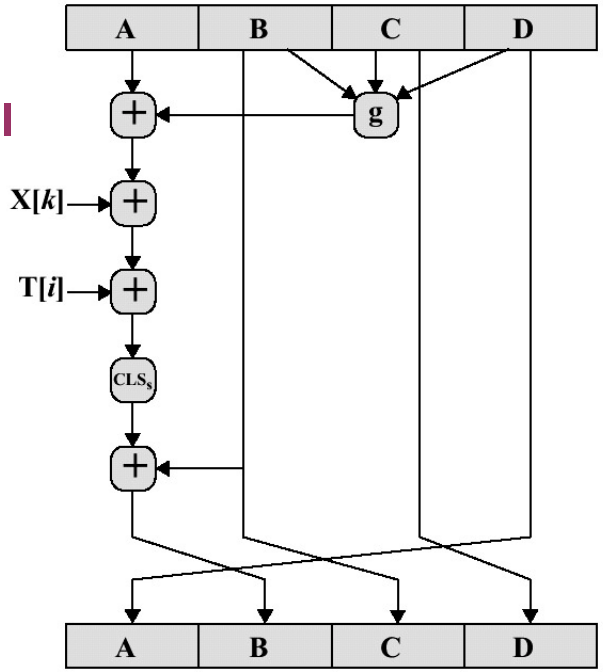
  g = one of the primitive functions F, G, H, I

  <<< s = circular left shift (rotation) of the 32-bit argument by s bits

  X[k] = $M[q \times 16 + k]$ = the k-th 32-bit word in the q-th 512-bit block of the message

  T[i] = the i-th 32-bit word in matrix T

  + = addition modulo 2^{32}



MD5 Message Digest Algorithm (cont.)

MD5 Compression Function (cont.)

- Primitive functions F,G,H,I:
  - Input – 3 32-bit words
  - Output – a 32-bit word
  - Each function performs a set of bitwise logical operations

Round	Primitive function g	g(b, c, d)
1	F(b, c, d)	$(b \wedge c) \vee (\bar{b} \wedge d)$
2	G(b, c, d)	$(b \wedge d) \vee (c \wedge \bar{d})$
3	H(b, c, d)	$b \oplus c \oplus d$
4	I(b, c, d)	$c \oplus (b \vee \bar{d})$

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

MD5 Message Digest Algorithm (cont.)

MD5 Compression Function (cont.)

- Permutations on the 32-bit words X[0..15] vary from round to round:
  - round 1: i
  - round 2: $\rho_2(i) = (1 + 5i) \bmod 16$
  - round 3: $\rho_3(i) = (5 + 3i) \bmod 16$
  - round 4: $\rho_4(i) = 7i \bmod 16$
- Note that for each step, only one of the four bytes of the ABCD buffer is updated
- Four different circular left shift amounts are used each round and different from round to round:
  - round 1: 7, 12, 17, 22
  - round 2: 5, 9, 14, 20
  - round 3: 4, 11, 16, 23
  - round 4: 6, 10, 15, 21
- Difficult to generate collisions (two blocks produce the same output)

MD5 Message Digest Algorithm (cont.)

Goals of MD4

- **Security** – computationally infeasible to find two messages that have the same message digest
- **Speed** – intended to be fast on 32-bit architectures, based on primitive operations on 32-bit words
- **Simplicity and Compactness** – simple to describe and simple to program
- **Favor Little-Endian Architecture** – Intel (little-endian) vs. SUN (big-endian)

MD5 Message Digest Algorithm (cont.)

Differences between MD4 and MD5

MD5	MD4
4 rounds of 16 steps each	3 rounds of 16 steps each
A different additive constant T[*i*] is used for each of the 64 steps	No additive constant is used for the 1st round The same additive constant is used for each of the steps of the other round
4 primitive logical functions, one for each round	3 primitive logical functions
Each step adds in the result of the preceding step	No previous step's result is included

MD5 Message Digest Algorithm (cont.)

Strength of MD5

- Property – every bit of the hash code is a function of every bit in the input
- Rivest conjectures that MD5 is as strong as possible for a 128-bit hash code
  - The difficulty of finding two messages having the same message digest is on the order of 2^{64} operations
  - The difficulty of finding a messages with a given digest is on the order of 2^{128} operations
- Many attacks on MD5 have been shown in the literatures

Secure Hash Algorithm

Development

- Secure Hash Algorithm (SHA) - developed by the National Institute of Standards and Technology (NIST)
- A federal information processing standard (FIPS PUB 180) in 1993
- SHA-1 – a revised version issued as FIPS PUB 180-1 in 1995
- Based on MD4 algorithm

SHA-1 Logic

- Input – a message with a maximum length of less than 2^{64} bits and is processed in 512-bit block
- Output – 160-bit message digest

Secure Hash Algorithm (cont.)

SHA-1 Logic (cont.)

- Step 1: append padding bits
  - The message is padded so that its length is congruent to 448 mod 512 (length $\equiv$ 448 mod 512)
  - Padding is always added (1 to 512 bits)
  - The padding pattern is 100…0
- Step 2: append length
  - A 64-bit length in bits of the original message is appended
- Step 3: initialize MD buffer
  - A 160-bit buffer is used to hold intermediate and final results of the hash function

Secure Hash Algorithm (cont.)

SHA-1 Logic (cont.)

- Step 3: initialize MD buffer (cont.)
  - The buffer is represented as 5 32-bit registers (A, B, C, D, E) initialized to the following integers (hexadecimal values):

    A = 67452301
    B = EFCDAB89
    C = 98BADCFE
    D = 10325476
    E = C3D2E1F0

  - The values are stored in big-ending order, i.e., the most significant byte of a word in the low-address byte position:

    word A = 67 45 23 01
    word B = EF CD AB 89
    word C = 98 BA DC FE
    word D = 10 32 54 76
    word E = C3 D2 E1 F0

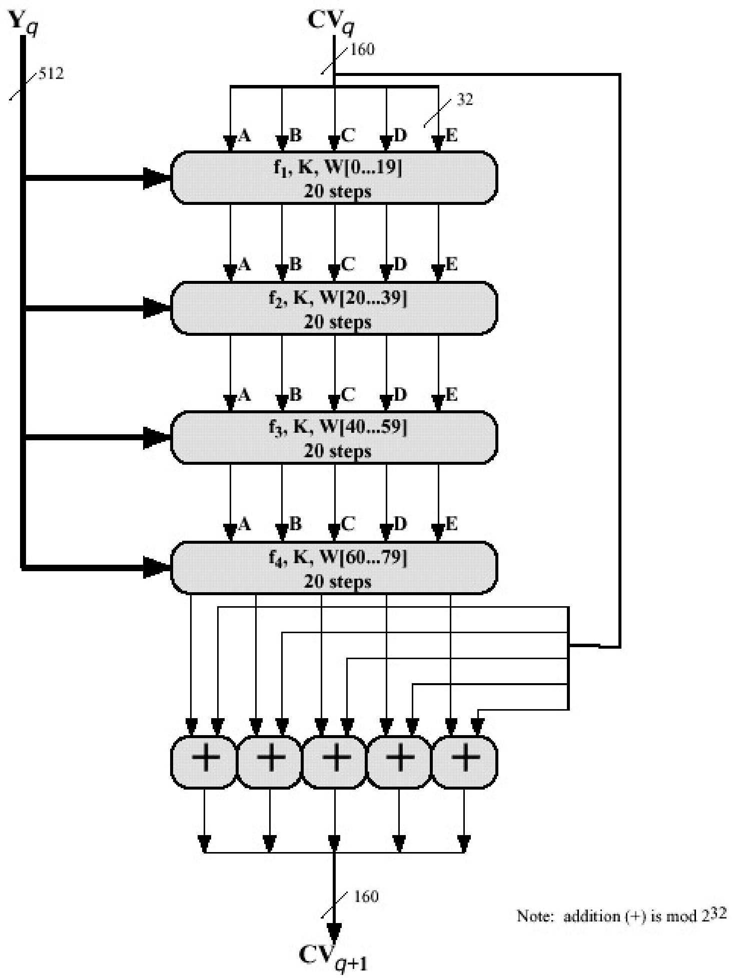
Secure Hash Algorithm (cont.)

SHA-1 Logic (cont.)

- Step 4: process message in 512-bit (16-word) blocks
  - A compression function with 4 rounds of processing of 20 steps each
  - For each round operation
    - Input – 512-bit block Y_q, 160-bit buffer value CV_q represented by ABCDE
    - Output – 160-bit chaining variable CV_{q+1} (updated ABCDE)
    - Makes use of additive constant K_t where $0 \le t \le 79$
  - The output of the last round is added to the input of the first round (CV_q) to produce CV_{q+1}

Step Number	Hexadecimal	Take Integer Part of:
$0 \le t \le 19$	K_t = 5A827999	$[2^{30} \times \sqrt{2}]$
$20 \le t \le 39$	K_t = 6ED9EBA1	$[2^{30} \times \sqrt{3}]$
$40 \le t \le 59$	K_t = 8F1BBCDC	$[2^{30} \times \sqrt{5}]$
$60 \le t \le 79$	K_t = CA62C1D6	$[2^{30} \times \sqrt{10}]$

Secure Hash Algorithm (cont.)



Secure Hash Algorithm (cont.)

SHA-1 Logic (cont.)

- Step 5: output
  - The output from the L-th stage is the 160-bit message digest
- Summary of the SHA-1:

$$CV_0 = IV$$
$$CV_{q+1} = SUM_{32}(CV_q, ABCDE_q)$$
$$MV = CV_L$$

where

IV = initial value of the ABCDE buffer

$ABCDE_q$ = the output of the last round processing of q-th message block

L = the number of blocks in the message

CV_q = chaining variable processed with the q-th block of the message

SUM_{32} = addition modulo 2^{32} performed on each word of the pair of inputs

MD = final message digest value

Secure Hash Algorithm (cont.)

SHA-1 Compression Function

- Each round consists of 16 steps operating on the buffer ABCDE with each step of the form:

  $$A, B, C, D, E \leftarrow (E + f(t,B,C,D) + S^5(A) + W_t + K_t), A, S^{30}(B), C, D$$

  where

  A,B,C,D,E = the five words of the buffer

  t = step number, $0 \leq t \leq 79$

  $f(t,B,C,D)$ = primitive logical function for step t

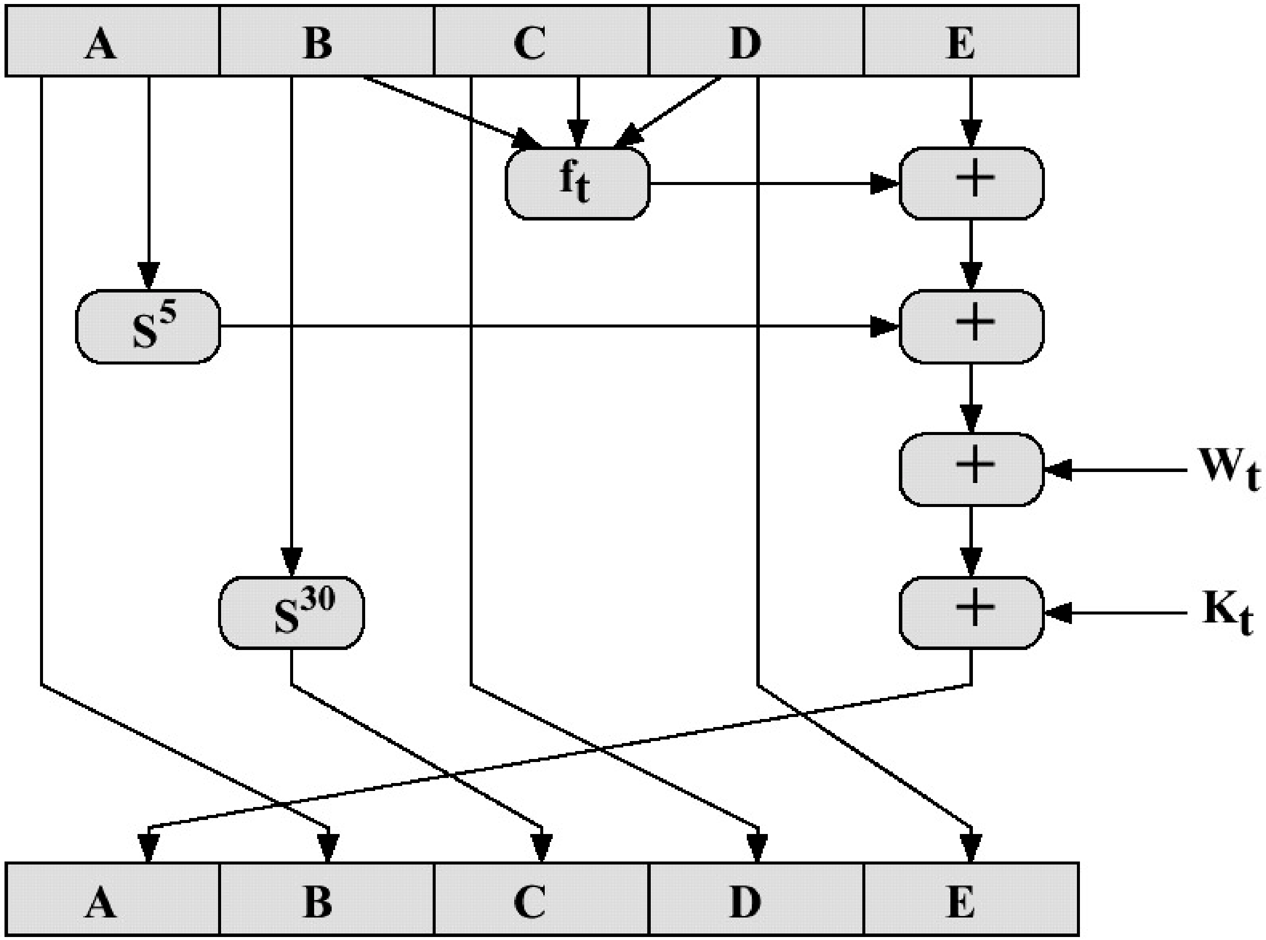
  S^k = circular left shift (rotation) of the 32-bit argument by k bits

  W_t = a 32-bit word derived from the 512-bit input block

  K_t = an additive constant, four distinct values are used

  $+$ = addition modulo 2^{32}

Secure Hash Algorithm (cont.)



Secure Hash Algorithm (cont.)

SHA-1 Compression Function (cont.)

- Primitive functions f(t,B,C,D):
  - Input – 3 32-bit words
  - Output – a 32-bit word
  - Each function performs a set of bitwise logical operations

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

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

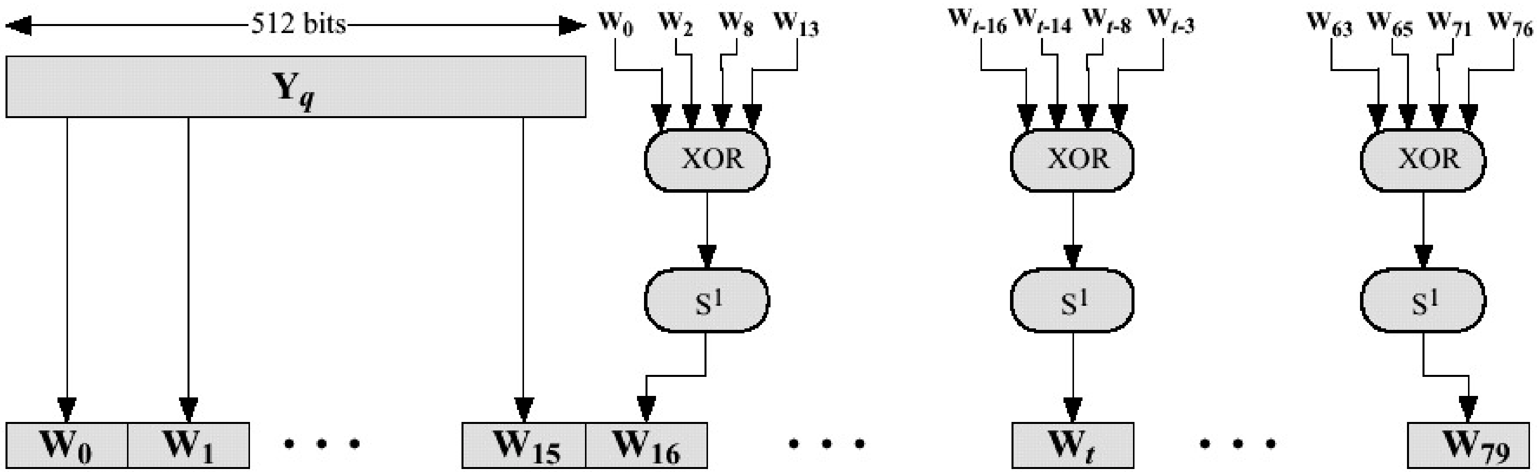
Secure Hash Algorithm (cont.)

SHA-1 Compression Function (cont.)

- Derivation of the 32-bit word W_t from the 512-bit input block
  - The first 16 values of W_t are taken directly from the 16 words of the current block
  - The remaining values are defined as follows:

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

- Difficult to generate collisions (two blocks produce the same output)



Secure Hash Algorithm (cont.)

Comparison of SHA-1 and MD5

	SHA-1	MD5
Security against brute-force attacks	160-bit message digest Stronger to brute-force attack	128-bit message digest
Security against cryptanalysis	not to be vulnerable	vulnerable to attacks
Speed	80 steps, 160-bit buffer slower	64 steps, 128-bit buffer faster
Simplicity and Compactness	Simple to describe and implement	Simple to describe and implement
Little-endian vs. big-endian architecture	big-endian	little-endian

RIPEMD-160

Development

- RIPEMD-160 was developed under the European RACE Integrity Primitives Evaluation (RIPE) project
- Originally developed a 128-bit version of RIPEM
- H. Dobbertin found attacks on two rounds of RIPEMD and later on MD4 and MD5
- Upgraded RIPEMD: RIPEMD-160

RIPEMD-160 Logic

- Input – a message of arbitrary length, processed in 512-bit block
- Output – 160-bit message digest

RIPEMD-160 (cont.)

RIPEMD-160 Logic (cont.)

- Step 1: append padding bits
  - The message is padded so that its length is congruent to 448 mod 512 (length $\equiv$ 448 mod 512)
  - Padding is always added (1 to 512 bits)
  - The padding pattern is 100…0
- Step 2: append length
  - A 64-bit length in bits of the original message is appended
  - If the original length is greater than 2^{64}, the length is modulo 2^{64}
- Step 3: initialize MD buffer
  - A 160-bit buffer is used to hold intermediate and final results of the hash function

RIPEMD-160 (cont.)

RIPEMD-160 Logic (cont.)

- Step 3: initialize MD buffer (cont.)
  - The buffer is represented as 5 32-bit registers (A, B, C, D, E) initialized to the following integers (hexadecimal values):

    A = 67452301
    B = EFCDAB89
    C = 98BADCFE
    D = 10325476
    E = C3D2E1F0

  - The values are stored in little-ending order, i.e., the least significant byte of a word in the low-address byte position:

    word A = 01 23 45 67
    word B = 89 AB CD EF
    word C = FE DC BA 98
    word D = 76 54 32 10
    word E = F0 E1 D2 C3

RIPEMD-160 (cont.)

RIPEMD-160 Logic (cont.)

- Step 4: process message in 512-bit (16-word) blocks
  - A module with 10 rounds of processing of 16 steps each
  - The 10 rounds are arranged in 2 parallel lines of 5 rounds each
    - Input – 512-bit block Y_q, 160-bit buffer value CV_q (ABCDE or A'B'C'D'E')
    - Output – 160-bit chaining variable CV_{q+1} (updated ABCDE)
    - Makes use of additive constant K_j

	Left Half		Right Half	
Step Number	**Hexadecimal**	**Integer part of:**	**Hexadecimal**	**Integer part of:**
$0 \le j \le 15$	$K_1 = K(j)$ = 00000000	0	$K'_1 = K'(j)$ = 50A28BE6	$2^{30} \times \sqrt[3]{2}$
$16 \le j \le 31$	$K_2 = K(j)$ = 5A827999	$2^{30} \times \sqrt{2}$	$K'_2 = K'(j)$ = 5C4DD124	$2^{30} \times \sqrt[3]{3}$
$32 \le j \le 47$	$K_3 = K(j)$ = 6ED9EBA1	$2^{30} \times \sqrt{3}$	$K'_3 = K'(j)$ = 6D703EF3	$2^{30} \times \sqrt[3]{5}$
$48 \le j \le 63$	$K_4 = K(j)$ = 8F1BBCDC	$2^{30} \times \sqrt{5}$	$K'_4 = K'(j)$ = 7A6D76E9	$2^{30} \times \sqrt[3]{7}$
$64 \le j \le 79$	$K_5 = K(j)$ = A953FD4E	$2^{30} \times \sqrt{7}$	$K'_5 = K'(j)$ = 00000000	0

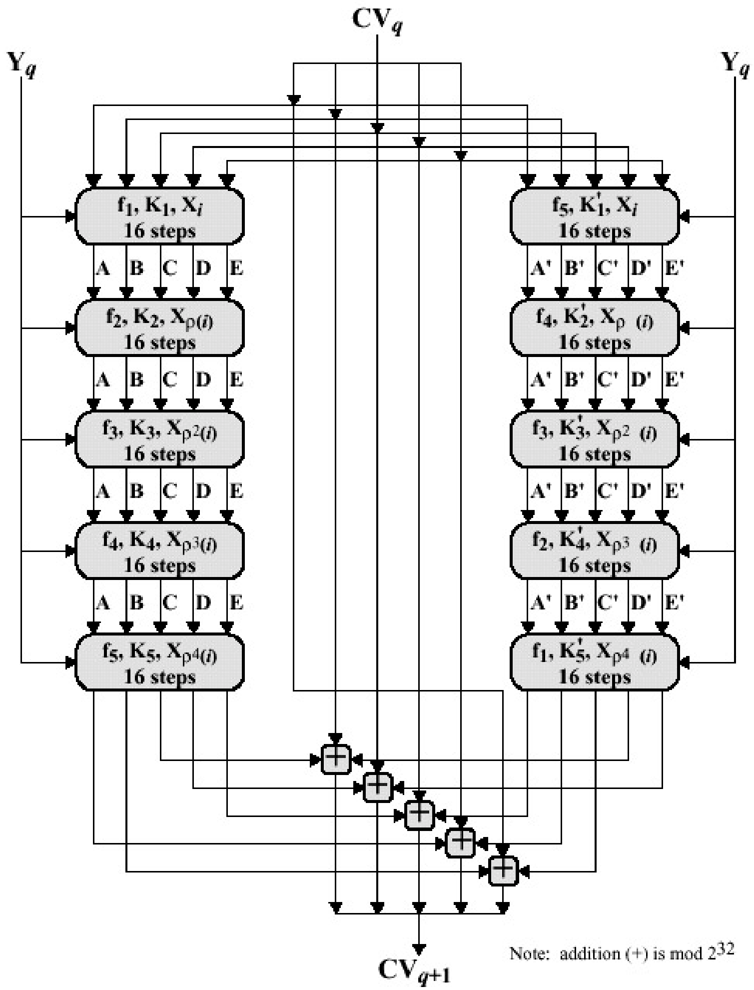
RIPEMD-160 (cont.)

RIPEMD-160 Logic (cont.)

- Step 4: (cont.)
  - The output of the last round is added to the input of the first round (CV_q) to produce CV_{q+1} in the following fashion:

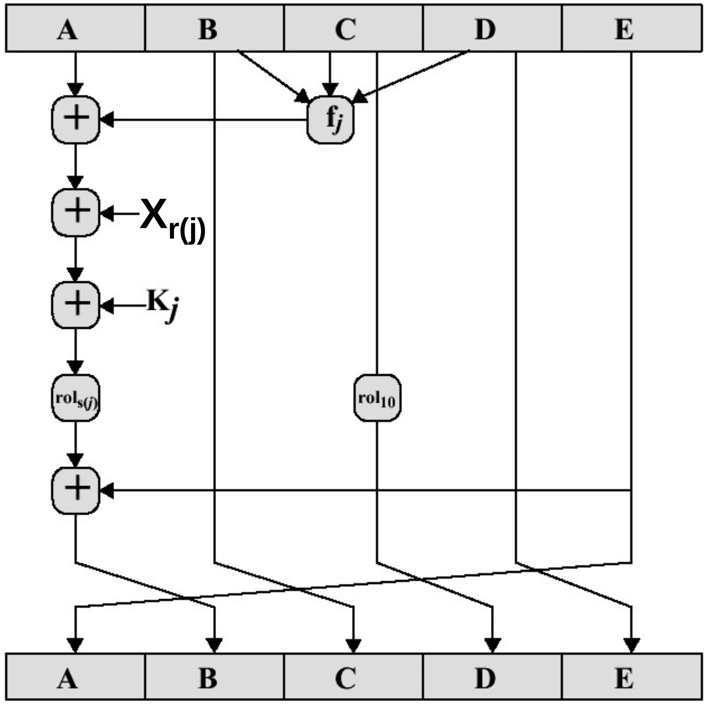
    $CV_{q+1}(0) = CV_q(1) + C + D'$
    $CV_{q+1}(1) = CV_q(2) + D + E'$
    $CV_{q+1}(2) = CV_q(3) + E + A'$
    $CV_{q+1}(3) = CV_q(4) + A + B'$
    $CV_{q+1}(4) = CV_q(0) + B + C'$
- Step 5: output
  - The output from the L-th stage is the 160-bit message digest

RIPEMD-160 (cont.)



RIPEMD-160 (cont.)

RIPEMD-160 Compression Function



- Each round consists of 16 steps operating with each round:

  $A := CV_q(0)$; $B := CV_q(1)$; $C := CV_q(2)$;
  $D := CV_q(3)$; $E := CV_q(4)$;
  $A' := CV_q(0)$; $B' := CV_q(1)$; $C' := CV_q(2)$;
  $D' := CV_q(3)$; $E' := CV_q(4)$;
  for j := 0 to 79 do
  $\quad T := rol_{s(j)}(A + f(j, B, C, D) + X_{r(j)} + K(j)) + E$;
  $\quad A := E$; $E := D$; $D := rol_{10}(C)$; $C := B$; $B := T$;
  $\quad T := rol_{s'(j)}(A + f(79-j, B', C', D') + X_{r;(j)} + K'(j)) + E'$;
  $\quad A' := E'$; $E' := D'$; $D' := rol_{10}(C')$; $C' := B'$; $B' := T'$;
  enddo
  $CV_{q+1}(0) := CV_q(1)+C+D'$; $CV_{q+1}(1) = CV_q(2)+D+E'$; $CV_{q+1}(2) = CV_q(3)+E+A'$;
  $CV_{q+1}(3) = CV_q(4)+A+B'$; $CV_{q+1}(4) = CV_q(0)+B+C'$;

RIPEMD-160 (cont.)

RIPEMD-160 Compression Function (cont.)

where

A,B,C,D,E = the five words of the buffer for the left line

A',B',C',D',E' = the five words of the buffer for the right line

j = step number, $0 \leq j \leq 79$

f(j,B,C,D) = primitive logical function for step j of left line and step 79-j of right line

$rol_{s(j)}$ = circular left shift (rotation) of the 32-bit argument by s(j) bits

$X_{r(j)}$ = a 32-bit word derived from the 512-bit input block determined by r(j)

K(j) = an additive constant used in step j

\+ = addition modulo 2^{32}

RIPEMD-160 (cont.)

RIPEMD-160 Compression Function (cont.)

- Primitive functions f(j,B,C,D):
  - Input – 3 32-bit words
  - Output – a 32-bit word
  - Each function performs a set of bitwise logical operations

Step	Function Name	Function Value
$0 \le j \le 15$	$f_1 = f(j, B, C, D)$	$B \oplus C \oplus D$
$16 \le j \le 31$	$f_2 = f(j, B, C, D)$	$(B \wedge C) \vee (\bar{B} \wedge D)$
$32 \le j \le 47$	$f_3 = f(j, B, C, D)$	$(B \vee \bar{C}) \oplus D$
$48 \le j \le 63$	$f_4 = f(j, B, C, D)$	$(B \wedge D) \vee (C \wedge \bar{D})$
$64 \le j \le 79$	$f_5 = f(j, B, C, D)$	$B \oplus (C \vee \bar{D})$

B	C	D	f1	f2	f3	f4	f5
0	0	0	0	0	1	0	1
0	0	1	1	1	0	0	0
0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	1
1	0	0	1	0	1	0	0
1	0	1	0	0	0	1	1
1	1	0	0	1	1	1	0
1	1	1	1	1	0	1	0

RIPEMD-160 (cont.)

RIPEMD-160 Compression Function (cont.)

- Circular left shift s(j):

Round	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}
1	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
2	12	13	11	15	6	9	9	7	12	15	11	13	7	8	7	7
3	13	15	14	11	7	7	6	8	13	14	13	12	5	5	6	9
4	14	11	12	14	8	6	5	5	14	12	15	14	9	9	8	6
5	15	12	13	13	9	5	8	6	15	11	12	11	8	6	5	5

- Derivation of the 32-bit word $X_{r(j)}$ from the 512-bit input block
  - The first 16 values of $X_{r(j)}$ holds the value of the current 512-bit block
  - The remaining values are permuted using $\rho(i)$ and $\pi(i)$:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho(i)$	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
$\pi(i)$	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12

Line	Round 1	Round 2	Round 3	Round 4	Round 5
left	identity	ρ	ρ^2	ρ^3	ρ^4
right	π	$\rho\pi$	$\rho^2\pi$	$\rho^3\pi$	$\rho^4\pi$

RIPEMD-160 (cont.)

Comparison with MD5 and SHA-1

	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	∞	$2^{64} - 1$ bits	∞
Primitive logical functions	4	4	5
Additive constants used	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

Relative performance on a 266-MHz Pentium:

Algorithm	Mbps
MD5	32.4
SHA-1	14.4
RIPEMD-160	13.6

HMAC

- An increased interest in developing a MAC derived from a cryptographic hash code since
  - Cryptographic hash functions (e.g., MD5 and SHA-1) generally execute faster than symmetric block ciphers (e.g., DES)
  - Library code for cryptographic hash functions is widely available
  - There are no export restrictions for cryptographic hash functions
- A hash function such as MD5 cannot be used directly for a MAC because <u>it does not rely on a secret key</u>
- HMAC
  - Issued as RFC 2104
  - Chosen as the mandatory-to-implement MAC for IP security
  - Used in other Internet protocols, such as SSL

HMAC (cont.)

HMAC Design Objectives

- To use, without modifications, available hash functions
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required
- To use and handle keys in a simple way
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the embedded hash function

HMAC (cont.)

- **HMAC Algorithm:**

  $$HMAC_K = H[(K^+ \oplus opad) \| H[(K^+ \oplus ipad) \| M]]$$

  where

  H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

  M = message input to HMAC

  Y_i = i-th block of M, $0 \le i \le L - 1$

  L = number of blocks in M

  b = number of bits in a block

  n = length of hash code produced by embedded hash function

  K = secret key; if key length is greater than b, the key is input to the hash function to produce an n-bit key; recommended length is $\ge$ n

  K^+ = K padded with 0's on the left so that the result is b bits in length

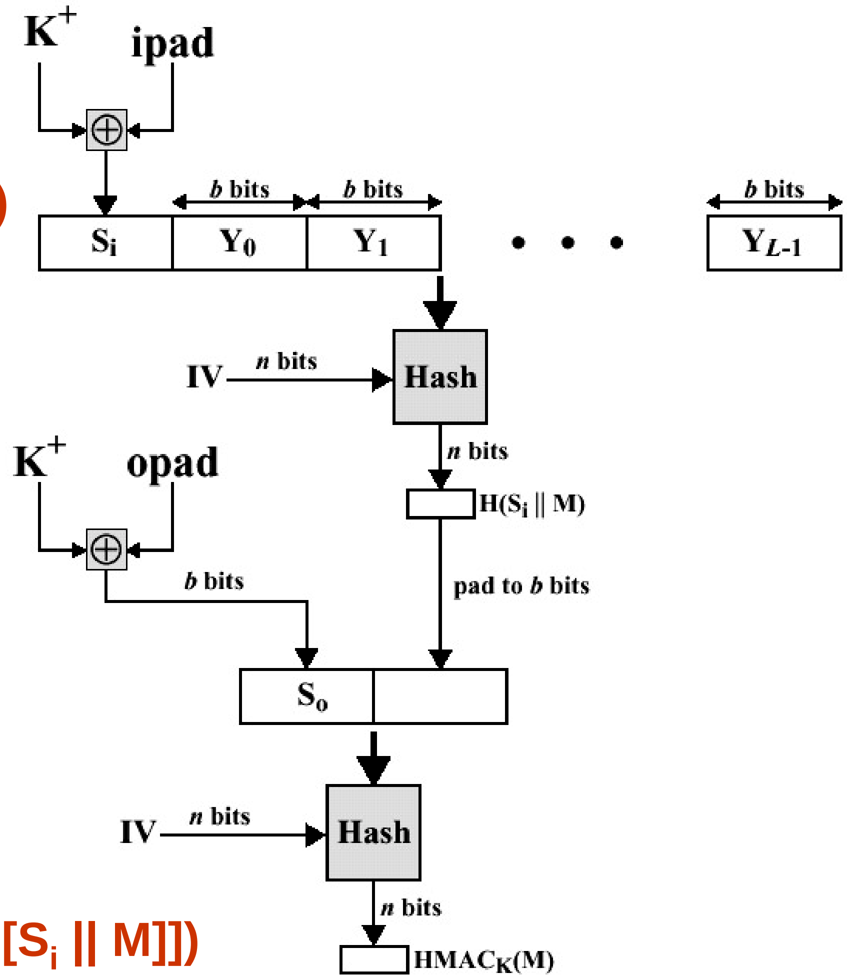
  ipad = 00110110 repeated b/8 times

  opad = 01011100 repeated b/8 times

HMAC (cont.)

HMAC Algorithm (cont.)

1. Append 0's to the left end of K to create a b-bit string K^+ (0…0 || K ⇒K^+)
2. XOR K^+ with ipad to produce the b-bit block S_i ($K^+ \oplus$ipad)
3. Append M to S_i (S_i || M)
4. Apply H to the stream generated in step 3 (H[S_i || M])
5. XOR K^+ with opad to produce the b-bit block S_o ($K^+ \oplus$opad)
6. Append the hash result to from step 4 to S_o (S_o || H[S_i || M])
7. Apply H to the stream generated in step 6 and output the result (H[S_o || H[S_i || M]])



HMAC (cont.)

Efficient Implementation of HMAC

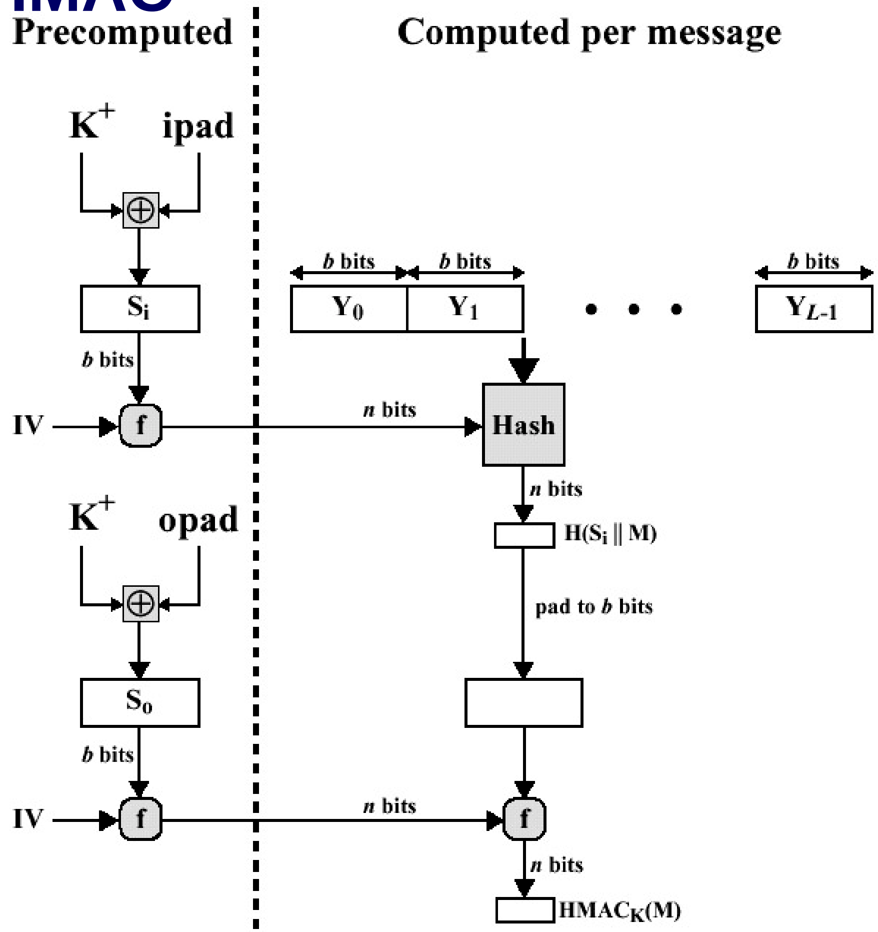
- Two quantities are precomputed:

  f(IV, (K$^+$ ⊕ ipad))

  f(IV, (K$^+$ ⊕ opad))

  where

  f(CV, block) is the compression function for the hash function, which takes as arguments a chaining variable of n bits and a block of b bits and produces a chaining variable of n bits



HMAC (cont.)

Security of HMAC

- The security of a MAC function in generally expressed in terms of:
  - The probability of successful forgery with a given amount of time spent by the forger
  - The probability of successful forgery with a given number of message-MAC pairs created with the same key
- It is proved that the probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded function:
  1. The attacker is able to compute an output of the compression function even with an IV that is random, secret, and unknown
     - The compression function is equivalent to the hash function applied to a message consisting of a single b-bit block
     - A brute-force attack on the key requires an order of 2^n operations

HMAC (cont.)

Security of HMAC (cont.)

2. The attacker finds collisions in the hash function even when the IV is random and unknown
   - Equivalent to looking for two messages M and M' that produce the same hash: H(M) = H(M')
   - The birthday attack requires an order of $2^{n/2}$ operations for a hash length of n
     - For 128-bit MD5, an attacker can off line choose 2^{64} blocks of message and compute the hash code
     - When attacking HMAC, the attacker cannot generate message/code pairs off line since the attacker does not know K