Cryptography and Network Security

Third Edition

by William Stallings

Lecture slides by Lawrie Brown

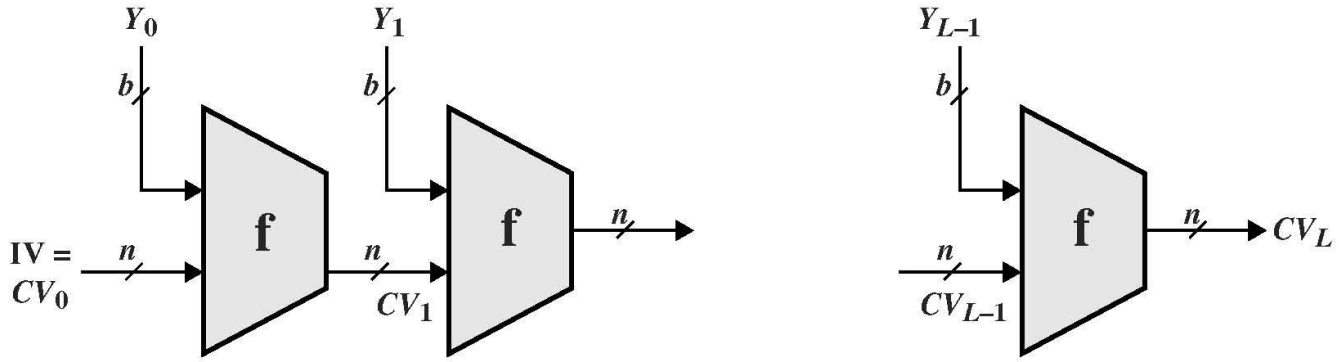
Chapter 12 – Hash Algorithms

Each of the messages, like each one he had ever read of Stern's commands, began with a number and ended with a number or row of numbers. No efforts on the part of Mungo or any of his experts had been able to break Stern's code, nor was there any clue as to what the preliminary number and those ultimate numbers signified.

***—Talking to Strange Men,* Ruth Rendell**

Hash Algorithms

- see similarities in the evolution of hash functions & block ciphers
  - increasing power of brute-force attacks
  - leading to evolution in algorithms
  - from DES to AES in block ciphers
  - from MD4 & MD5 to SHA-1 & RIPEMD-160 in hash algorithms
- likewise tend to use common iterative structure as do block ciphers



IV = Initial value
CV = chaining variable
Y_i = *i*th input block
f = compression algorithm
L = number of input blocks
n = length of hash code
b = length of input block

Figure 11.10 General Structure of Secure Hash Code

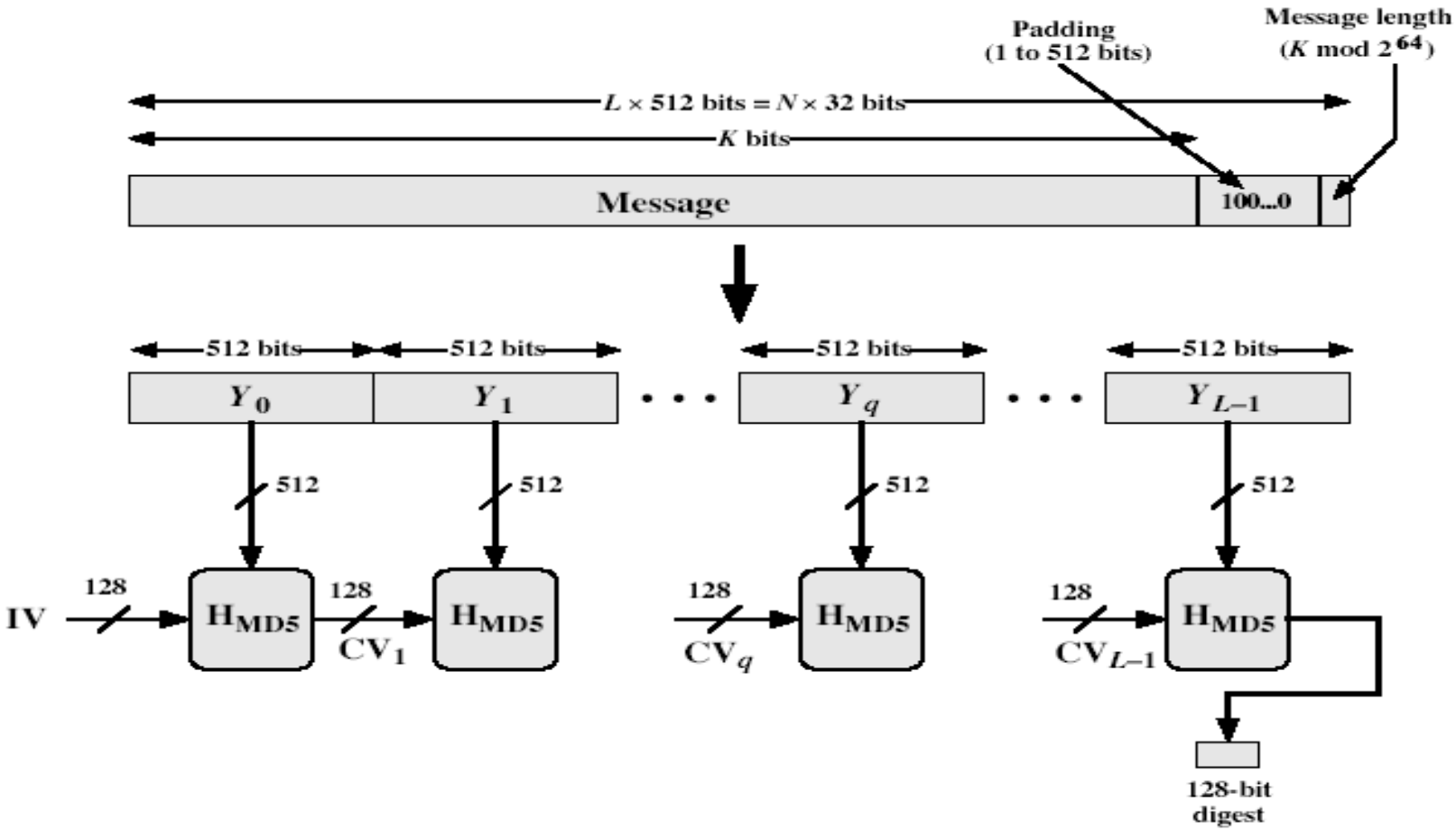
MD5

- designed by Ronald Rivest (the R in RSA)
- latest in a series of MD2, MD4
- produces a 128-bit hash value
- until recently was the most widely used hash algorithm
  - in recent times have both brute-force & cryptanalytic concerns
- specified as Internet standard RFC1321

MD5 Overview

1. pad message so its length is 448 mod 512
   - Padding of 1-512 bits is always used.
   - Padding: 1000….0
2. append a 64-bit length value to message
   - Generate a message with 512L bits in length
3. initialise 4-word (128-bit) MD buffer (A,B,C,D)
4. process message in 16-word (512-bit) blocks:
5. output hash value is the final buffer value

MD5 Overview



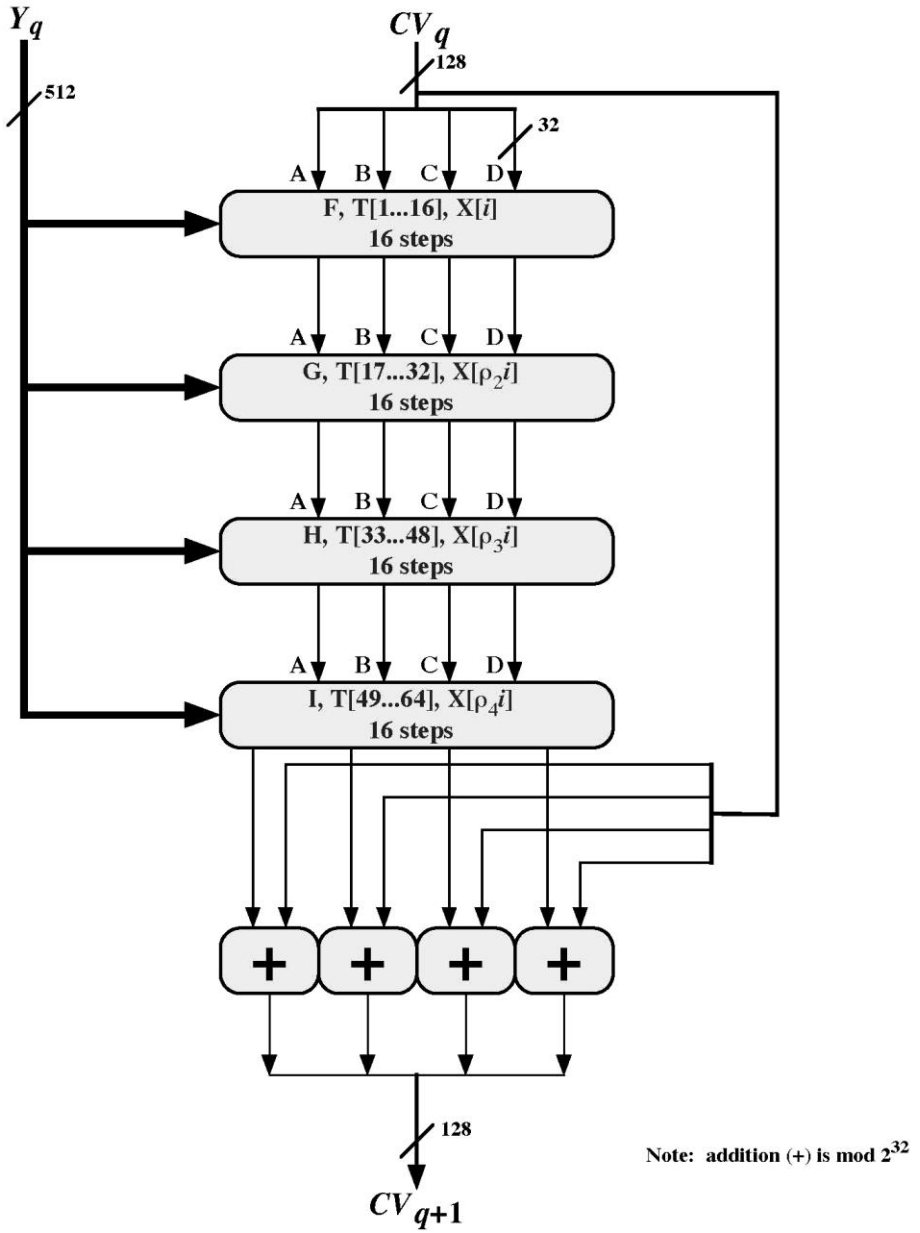


Figure 12.2 MD5 Processing of a Single 512-bit Block

MD5 Compression Function

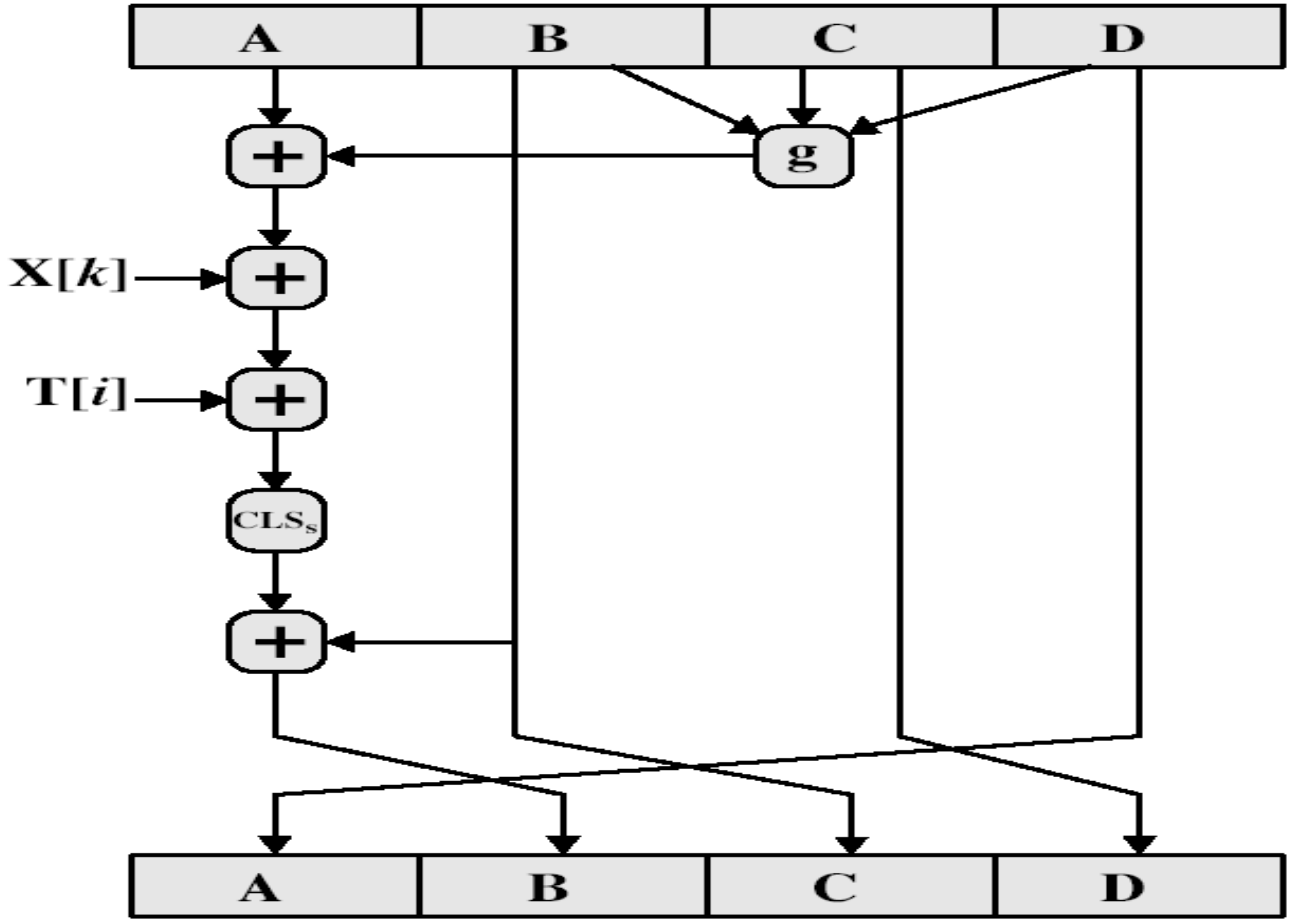


Table 12.1 Key Elements of MD5

(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

(b) 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] = F57COFAF	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

```
For q = 0 to (N/16) - 1 do
    /* Copy block q into X. */
    For j = 0 to 15 do
        Set X[j] to  M[q*16 + j].
    end /* of loop on j */

    /* Save A as AA, B as BB, C as CC, and
    D as DD. */
    AA = A
    BB = B
    CC = C
    DD = D

    /* Round 1. */
    /* Let [abcd  k  s  i] denote the operation
    a = b + ((a + F(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD     0     7     1]
    [DABC     1    12     2]
    [CDAB     2    17     3]
    [BCDA     3    22     4]
    [ABCD     4     7     5]
    [DABC     5    12     6]
    [CDAB     6    17     7]
    [BCDA     7    22     8]
    [ABCD     8     7     9]
    [DABC     9    12    10]
    [CDAB    10    17    11]
    [BCDA    11    22    12]
    [ABCD    12     7    13]
    [DABC    13    12    14]
    [CDAB    14    17    15]
    [BCDA    15    22    16]

    /* Round 2. */
    /* Let [abcd  k  s  i] denote the operation
    a = b + ((a + G(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD     1     5    17]
    [DABC     6     9    18]
    [CDAB    11    14    19]
    [BCDA     0    20    20]
    [ABCD     5     5    21]
    [DABC    10     9    22]
    [CDAB    15    14    23]
    [BCDA     4    20    24]
    [ABCD     9     5    25]
    [DABC    14     9    26]
    [CDAB     3    14    27]
    [BCDA     8    20    28]
    [ABCD    13     5    29]
    [DABC     2     9    30]
    [CDAB     7    14    31]
    [BCDA    12    20    32]

    /* Round 3. */
    /* Let [abcd  k  s  i] denote the operation
    a = b + ((a + H(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD     5     4    33]
    [DABC     8    11    34]
    [CDAB    11    16    35]
    [BCDA    14    23    36]
    [ABCD     1     4    37]
    [DABC     4    11    38]
    [CDAB     7    16    39]
    [BCDA    10    23    40]
    [ABCD    13     4    41]
    [DABC     0    11    42]
    [CDAB     3    16    43]
    [BCDA     6    23    44]
    [ABCD     9     4    45]
    [DABC    12    11    46]
    [CDAB    15    16    47]
    [BCDA     2    23    48]

    /* Round 4. */
    /* Let [abcd  k  s  i] denote the operation
    a = b + ((a + I(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD     0     6    49]
    [DABC     7    10    50]
    [CDAB    14    15    51]
    [BCDA     5    21    52]
    [ABCD    12     6    53]
    [DABC     3    10    54]
    [CDAB    10    15    55]
    [BCDA     1    21    56]
    [ABCD     8     6    57]
    [DABC    15    10    58]
    [CDAB     6    15    59]
    [BCDA    13    21    60]
    [ABCD     4     6    61]
    [DABC    11    10    62]
    [CDAB     2    15    63]
    [BCDA     9    21    64]

    /* Then increment each of the four registers by the
    value it had before this block was started. */
    A = A + AA
    B = B + BB
    C = C + CC
    D = D + DD

end /* of loop on q */
```

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
  - note this updates 1 word only of the buffer
  - after 16 steps each word is updated 4 times
- 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
- The point of all this complexity:
  - To make it difficult to generate collisions

Strength of MD5

- Every hash bit is dependent on all message bits
- Rivest conjectures security is as good as possible for a 128 bit hash
  - Given a hash, find a message: $O(2^{128})$ operations
  - No disproof exists yet
- known attacks are:
  - Berson 92 attacked any 1 round using differential cryptanalysis (but can't extend)
  - Boer & Bosselaers 93 found a pseudo collision (again unable to extend)
  - Dobbertin 96 created collisions on MD compression function for one block, cannot expand to many blocks
  - Brute-force search now considered possible

Secure Hash Algorithm (SHA-1)

- SHA was designed by NIST & NSA in 1993, revised 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- produces 160-bit hash values
- now the generally preferred hash algorithm
- based on design of MD4 with key differences

SHA Overview

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 5-word (160-bit) buffer (A,B,C,D,E) to
   (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. process message in 16-word (512-bit) chunks:
   - expand 16 words into 80 words by mixing & shifting
   - use 4 rounds of 20 bit operations on message block & buffer
   - add output to input to form new buffer value
5. output hash value is the final buffer value

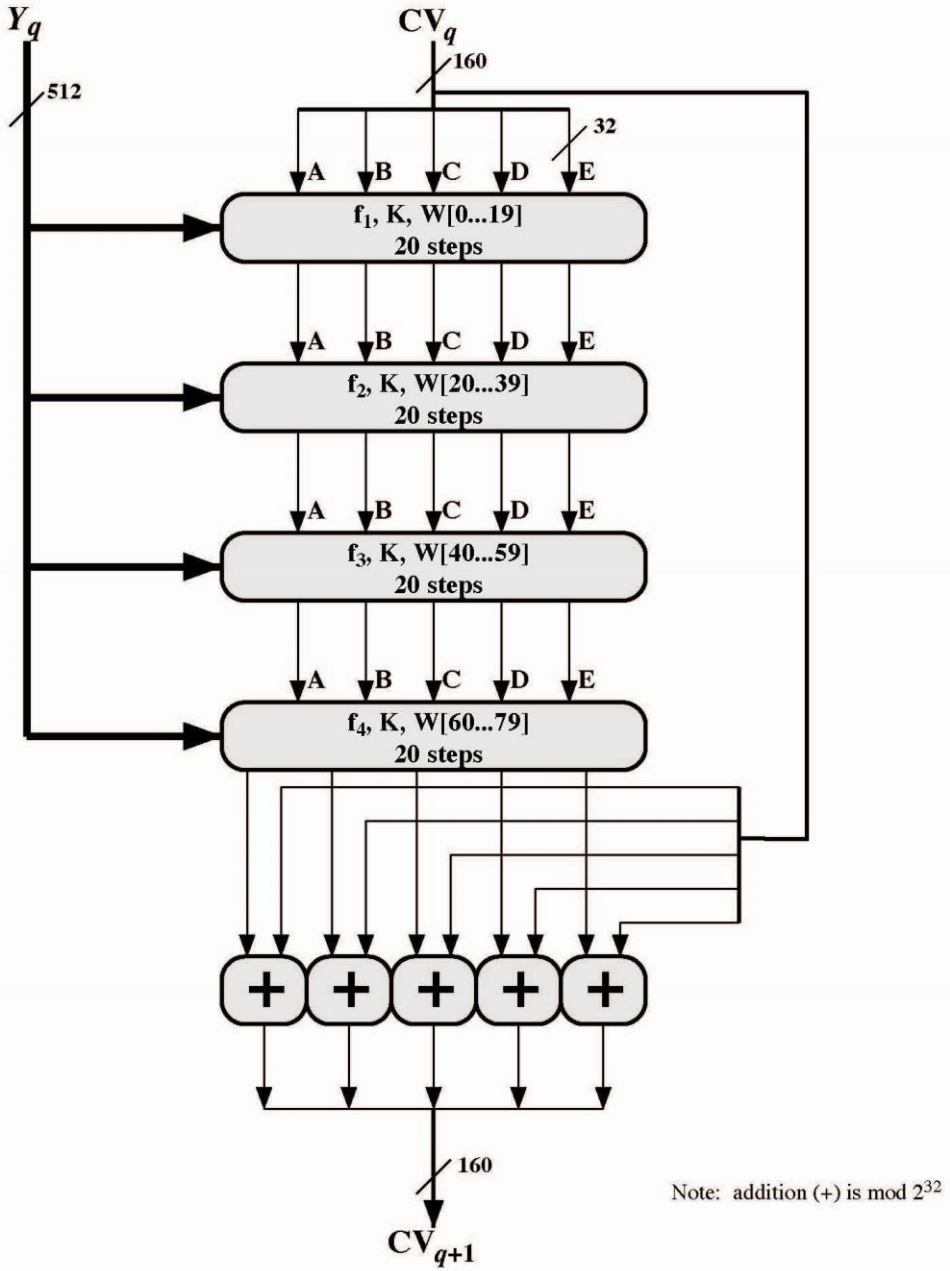
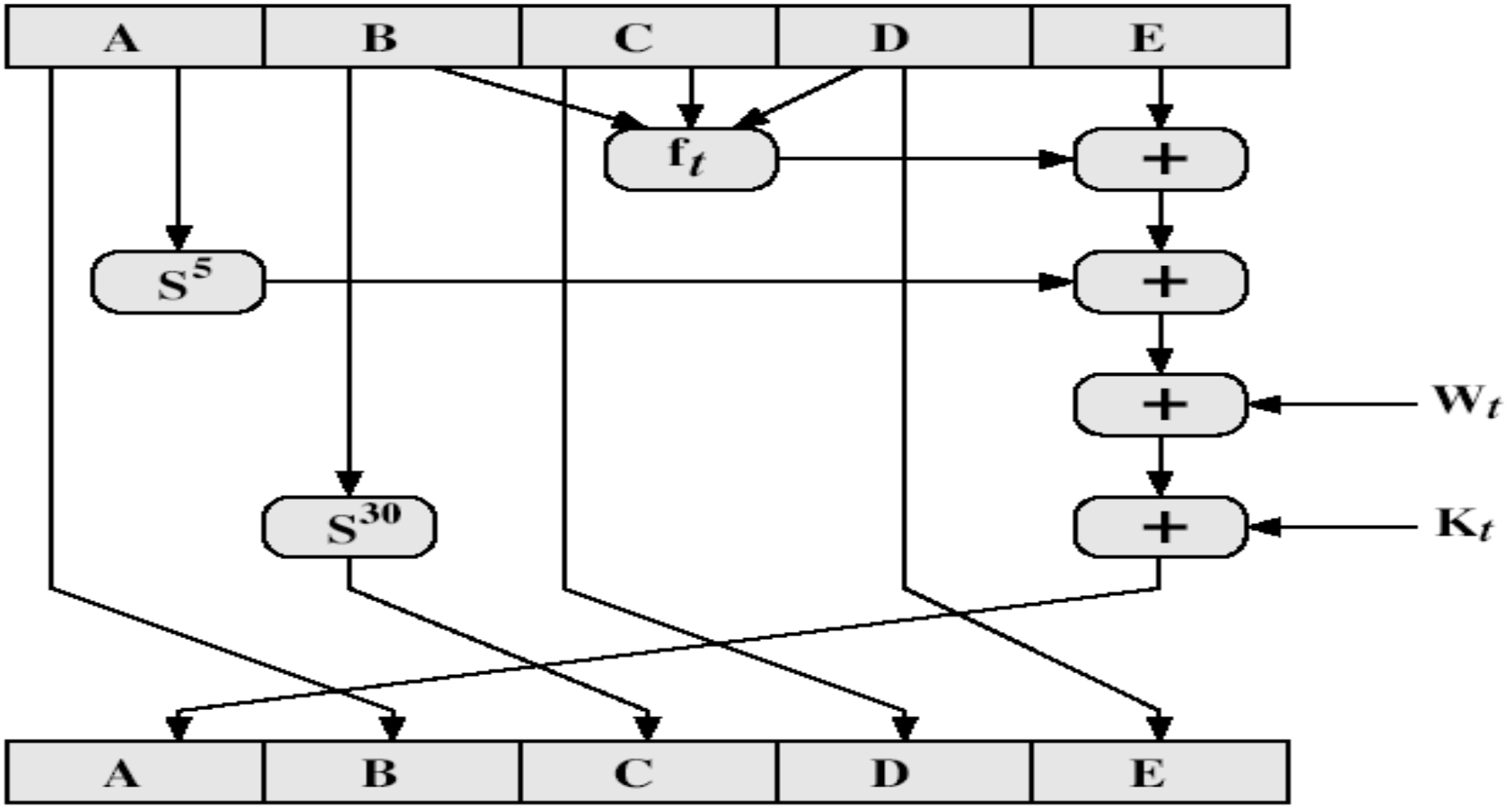


Figure 12.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

SHA-1 Compression Function



Logical functions for SHA-1

Table 12.2 Truth Table of Logical Functions for SHA-1

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

SHA-1 Compression Function

- each round has 20 steps which replaces the 5 buffer words thus:

  $(A,B,C,D,E) \leftarrow (E+f(t,B,C,D)+(A<<5)+W_t+K_t),A,(B<<30),C,D)$

- ABCDE refer to the 5 words of the buffer
- t is the step number
- $f(t,B,C,D)$ is nonlinear function for round
- W_t is derived from the message block
- K_t is a constant value (P359)

Creation of 80-word input

- Adds redundancy and interdependence among message blocks

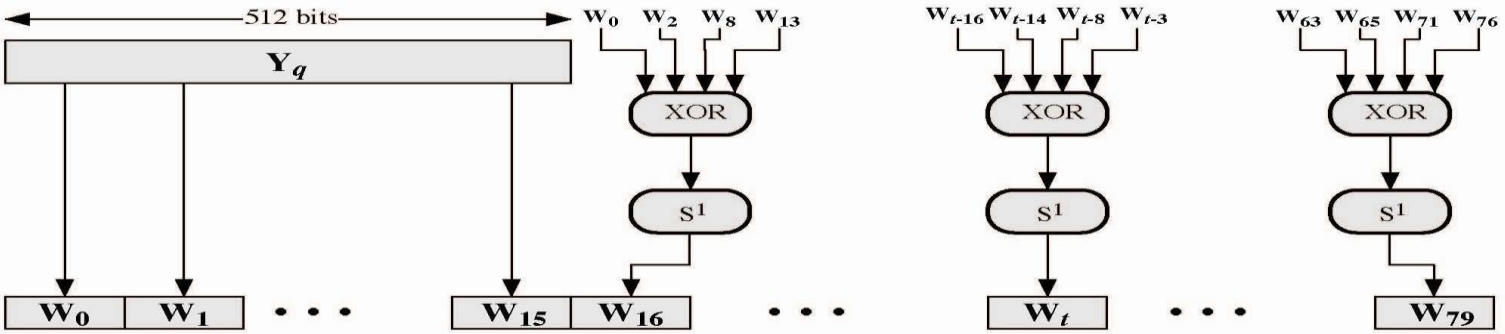


Figure 12.7 Creation of 80-word Input Sequence for SHA-1 Processing of Single Block

SHA-1 verses MD5

- brute force attack is harder (160 vs 128 bits for MD5)
- not vulnerable to any known attacks (compared to MD4/5)
- a little slower than MD5 (80 vs 64 steps)
- both designed as simple and compact
- optimised for big endian CPU's (SUN) vs MD5 for little endian CPU’s (PC)

Revised Secure Hash Standard

- NIST have issued a revision FIPS 180-2
- adds 3 additional hash algorithms
- SHA-256, SHA-384, SHA-512
  - Different lengths of hash bits
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1

Table 12.3 Comparison of SHA Properties

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	80	80	80
Security	80	128	192	256

Notes: 1. All sizes are measured in bits.
2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a workfactor of approximately $2^{n/2}$.

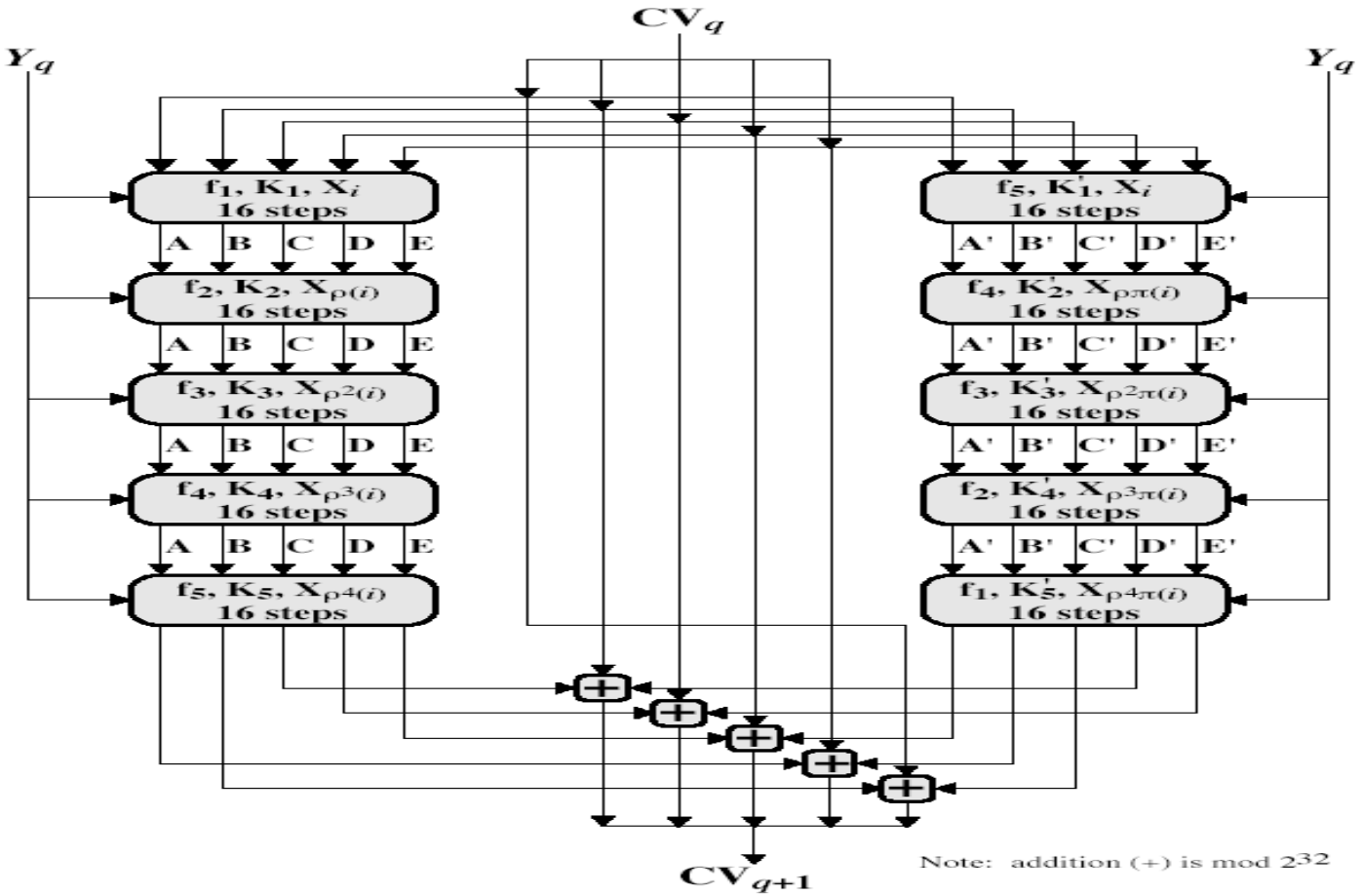
RIPEMD-160

- RIPEMD-160 was developed in Europe as part of RIPE project in 96
- by researchers involved in attacks on MD4/5
- initial proposal strengthen following analysis to become RIPEMD-160
- somewhat similar to MD5/SHA
- uses 2 parallel lines of 5 rounds of 16 steps
- creates a 160-bit hash value
- slower, but probably more secure, than SHA

RIPEMD-160 Overview

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 5-word (160-bit) buffer (A,B,C,D,E) to
   (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. process message in 16-word (512-bit) chunks:
   - use 10 rounds of 16 bit operations on message block & buffer – in 2 parallel lines of 5
   - add output to input to form new buffer value
5. output hash value is the final buffer value

RIPEMD-160 Round



RIPEMD-160 Compression Function

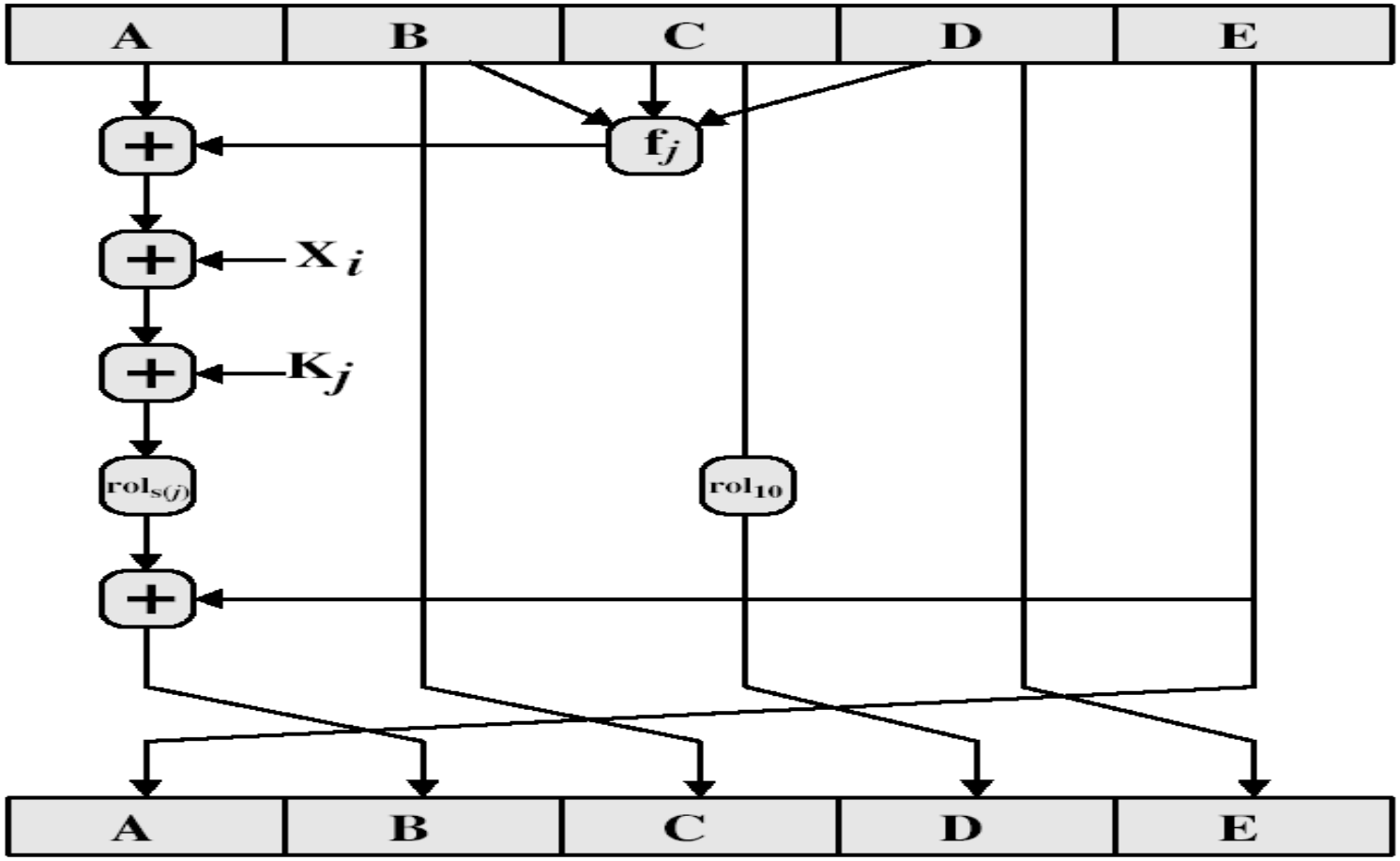


Table 12.7 Elements of RIPEMD-160

(a) Permutations of Message Words

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	$\rho\pi$	$\rho^2\pi$	$\rho^3\pi$	$\rho^4\pi$

(b) Circular Left Shift of Message Words (Both Lines)

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

RIPEMD-160 Design Criteria

- use 2 parallel lines of 5 rounds for increased complexity
- for simplicity the 2 lines are very similar
  - Different Ks
  - Different order of fs
  - Different ordering of Xi
- step operation very close to MD5
  - Rotate C by 10 bit to avoid a known MD5 attack

RIPEMD-160 Design Criteria

- permutation varies parts of message used
  - Two words close in one round are far apart in the next
  - Two words close in the left line will be at least 7 positions apart in the right line
- circular shifts designed for best results
  - Shifts larger than 5 (<5 is considered weak)
  - Different amount for the five rounds
  - Total shifts for each word in five rounds not divisible by 32
  - Not too many shift constants should be divisible by 4

RIPEMD-160 verses MD5 & SHA-1

- brute force attack harder (160 like SHA-1 vs 128 bits for MD5)
- not vulnerable to known attacks to MD4/5
  - Double lines considered more secure than SHA-1
  - Still little is know for the design principles for them
- slower than MD5 (more steps)
- all designed as simple and compact
- SHA-1 optimised for big endian CPU's vs RIPEMD-160 & MD5 optimised for little endian CPU’s

What is more secure?

- Longer messages lead to more collision per hash value
  - Is it more secure to use shorter messages?
  - Need to consider the scenarios
    - Known message, find out a collision message
    - Find out a collision pair using birthday attack
- Uniform distribution assumption

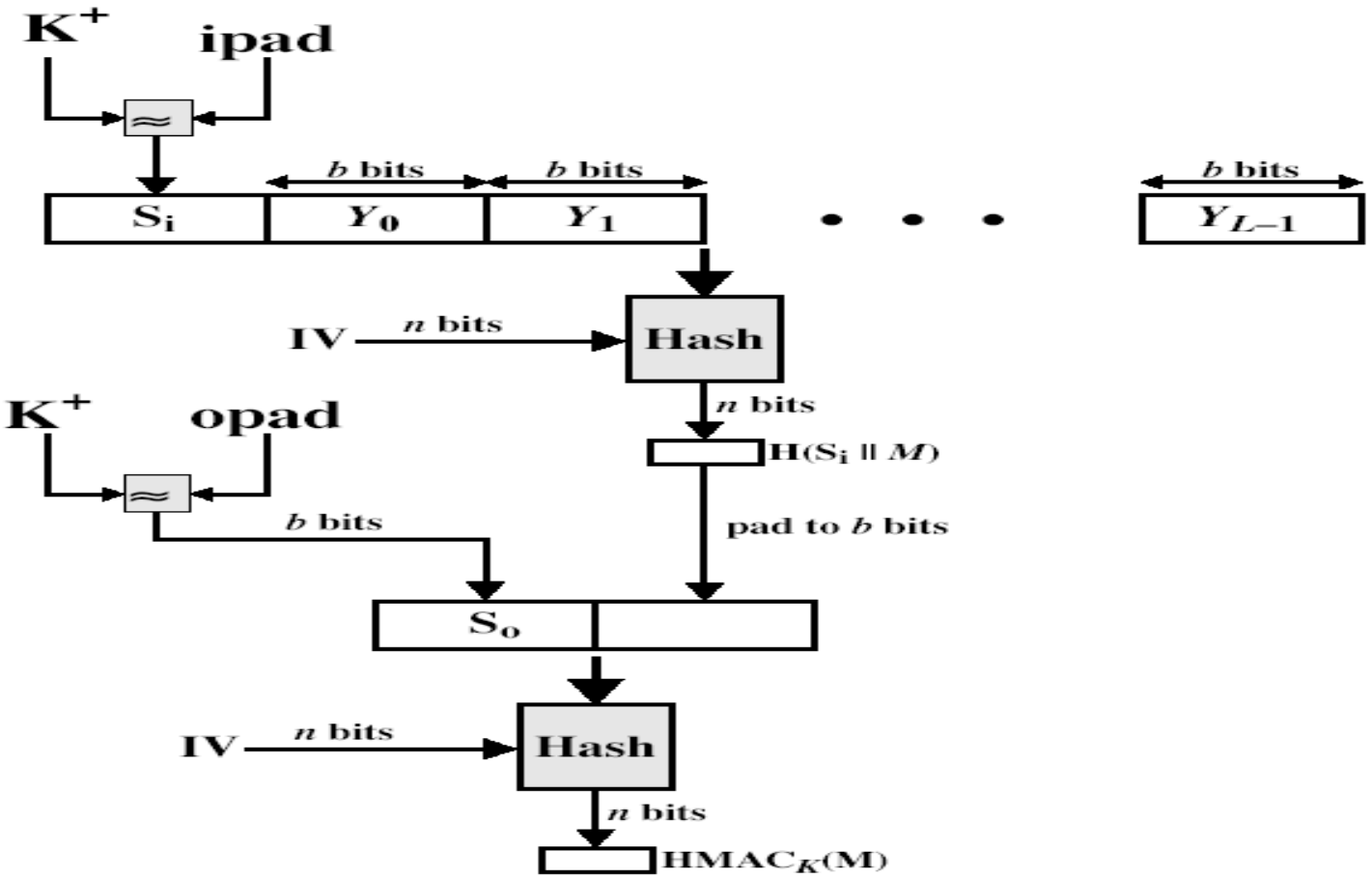
Keyed Hash Functions as MACs

- have desire to create a MAC using a hash function rather than a block cipher
  - because hash functions are generally faster
  - not limited by export controls unlike block ciphers
- hash includes a key along with the message
- led to development of HMAC

HMAC Requirements

- Blackbox use of hash without modification
- Not much overhead than original hash
- Easy to replace the hash module
  - Easy to upgrade security

HMAC Overview



HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:

$$HMAC_K = Hash[(K^+ \text{ XOR opad}) \,||\, Hash[(K^+ \text{ XOR ipad})||M)]]$$

- where K^+ is the key padded out to size
- and opad, ipad are specified padding constants
- overhead is just 3 more hash calculations than the message needs alone
- any of MD5, SHA-1, RIPEMD-160 can be used

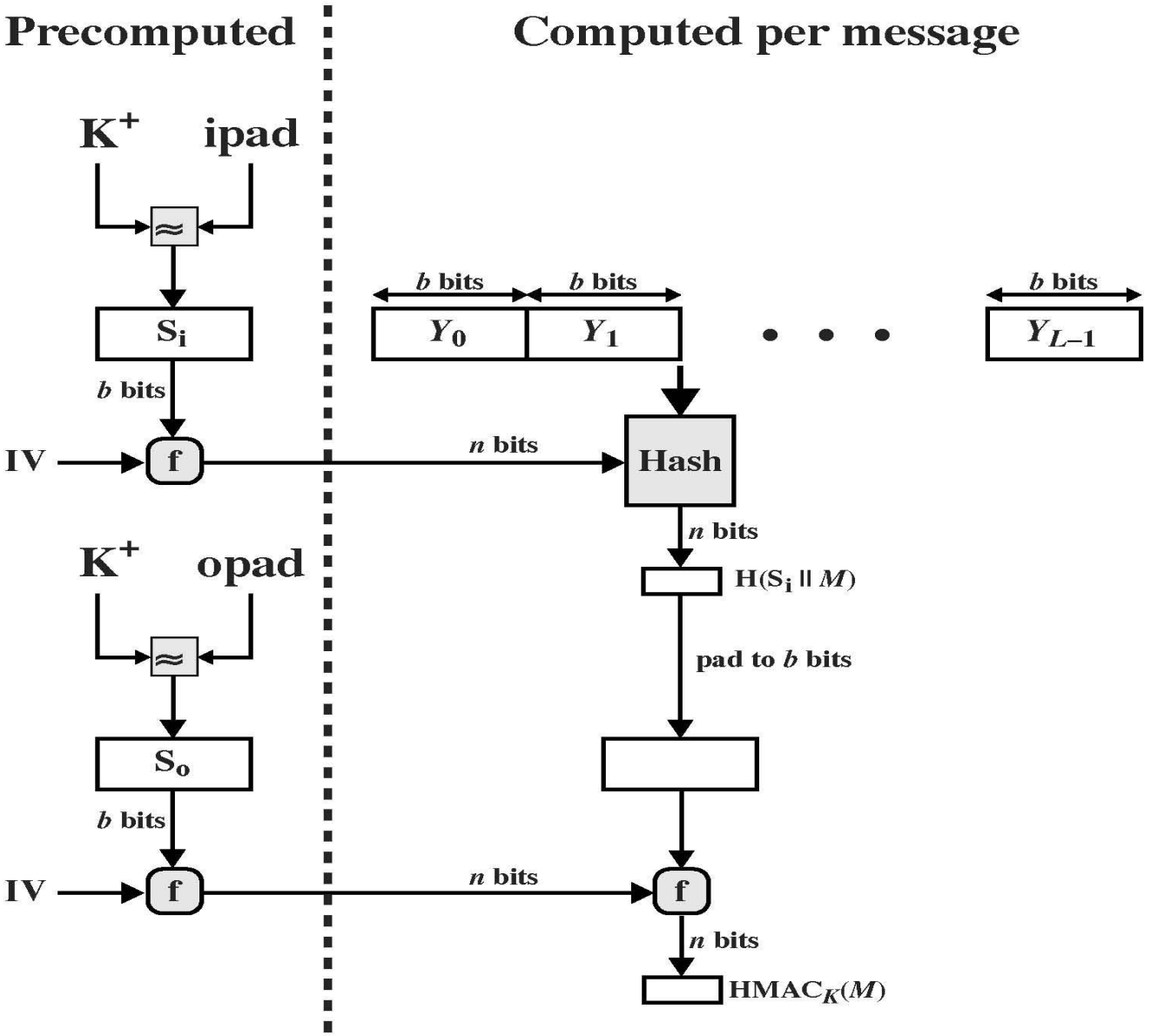


Figure 12.11 Efficient Implementation of HMAC

Summary

- have considered:
  - some current hash algorithms:
    - MD5, SHA-1, RIPEMD-160
  - HMAC authentication using a hash function