Diyala University
College of Engineering
Computer & Software
Engineering Department

Fourth Year 2012/2013

Cryptographic Data Integrity Algorithms

Chapter 4/Part1

Message Authentication: Hashes and MAC

PRESENTED BY
DR. ALI J. ABBOUD

4. Objectives

- Message Authentication.
- Authentication Functions.
- Message Encryption as Authentication.
- Authentication based on Hash Functions.
- Basic Functions of Hash Functions.
- A Few Simple Hash Functions.
- Birthday Attacks.
- MD5 Hash Algorithm.
- Secure Hash Algorithm (SAH).

4.1 Message Authentication

- Goal here: having received a message one would like to make sure that the message has not been altered on the way
  - Produce a short sequence of bits that depends on the message and on a secret key
  - To authenticate the message, the partner will compute the same bit pattern, assuming he shares the same secret key
- This does not necessarily includes encrypting or signing the message
  - The message can be sent in plain, with the authenticator appended
  - This is not a digital signature: the receiver can produce the same MAC
  - One may encrypt the authenticator with his private key to produce a digital signature
  - One may encrypt both the message and the authenticator
- Possible attacks on message authentication:
  - Content modification
  - Sequence modification – modifications to a sequence of messages, including insertion, deletion, reordering
  - Timing modification – delay or replay messages

4.2 Authentication Functions

- Three types of authentication exist
  - Message encryption – the ciphertext serves as authenticator
  - Message authentication code (MAC) – a public function of the message and a secret key producing a fixed-length value to serve as authenticator
    - This does not provide a digital signature because A and B share the same key
  - Hash function – a public function mapping an arbitrary length message into a fixed-length hash value to serve as authenticator
    - This does not provide a digital signature because there is no key

4.3 Message Encryption as Authentication

- Main idea here: the message must have come from A because the ciphertext can be decrypted using his (secret or public) key
- Also, none of the bits in the message have been altered because an opponent does not know how to manipulate the bits of the ciphertext to induce meaningful changes to the plaintext
- **Conclusion**: encryption (either symmetric or public-key) provides authentication as well as confidentiality
- Some careful considerations are needed here:
  - How does B recognize a meaningful message from an arbitrary sequence of bits?
    - He can apply the decryption key to **any** sequence of bits he receives
  - This is not necessarily easy task if the message is some sort of binary file
  - Immediate idea of attack: send arbitrary bit sequences to disrupt the receiver – he will try to figure out the meaning of that bit sequence
- Defense against this type of attack: add to the message a certain structure such as an error-correcting code (e.g., check-sum bits) and then encrypt the whole file
  - B will detect illegitimate messages because they will not have the required structure

4.3 Message Encryption as Authentication

A → B: $E_K[M]$

- Provides confidentiality
  - —Only A and B share K
- Provides a degree of authentication
  - —Could come only from A
  - —Has not been altered in transit
  - —Requires some formatting/redundancy
- Does not provide signature
  - —Receiver could forge message
  - —Sender could deny message

(a) Symmetric encryption

A → B: $E_{KU_b}[M]$

- Provides confidentiality
  - —Only B has KR_b to decrypt
- Provides no authentication
  - —Any party could use KU_b to encrypt message and claim to be A

(b) Public-key encryption: confidentiality

A → B: $E_{KR_a}[M]$

- Provides authentication and signature
  - —Only A has KR_a to encrypt
  - —Has not been altered in transit
  - —Requires some formatting/redundancy
  - —Any party can use KU_a to verify signature

(c) Public-key encryption: authentication and signature

A → B: $E_{KU_b}\left[E_{KR_a}(M)\right]$

- Provides confidentiality because of KU_b
- Provides authentication and signature because of Kr_a

(d) Public-key encryption: confidentiality, authentication, and signature

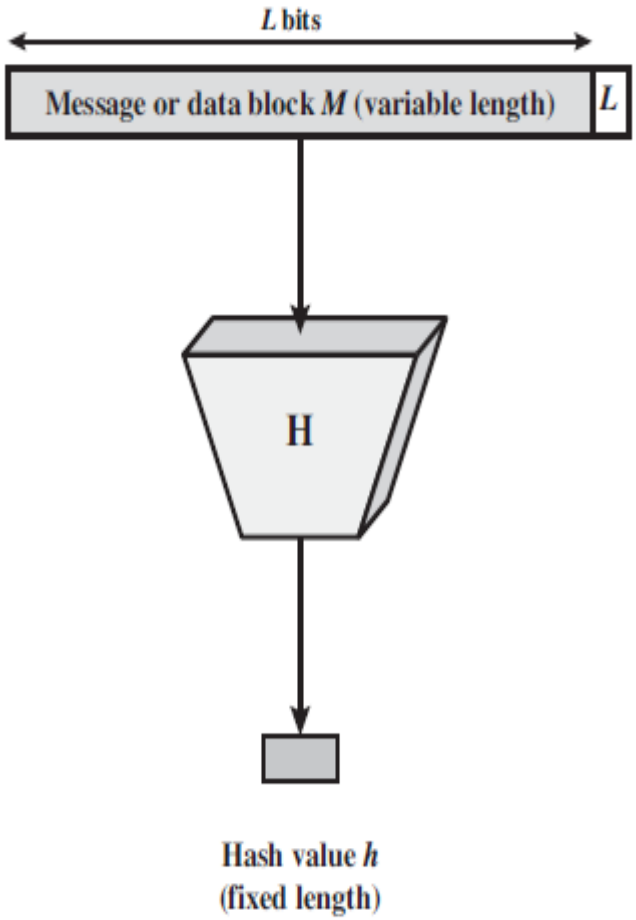
4.3 Message Encryption as Authentication

- Often one needs alternative authentication schemes than just encrypting the message
  - Sometimes one needs to avoid encryption of full messages due to legal requirements
  - Encryption and authentication may be separated in the system architecture
  - If a message is broadcast to several destinations in a network (such as a military control center), then it is cheaper and more reliable to have just one node responsible to evaluate the authenticity – message will be sent in plain with an attached authenticator
  - If one side has a heavy load, it cannot afford to decrypt all messages – it will just check the authenticity of some randomly selected messages
  - *If the message is sent encrypted, it is of course protected over the network. However, once the receiver decrypts the message, it is no longer secure. Using a different type of authentication protects the message also on the local computer*

4.4 Authentication based on hash functions

- A fixed-length hash value h is generated by a function H that takes as input a message of arbitrary length: h=H(M)
  - A sends M and H(M)
  - B authenticates the message by computing H(M) and checking the match
- Requirements for a hash function
  - H can be applied to a message of any size
  - H produces fixed-length output
  - Computationally easy to compute H(M)
  - Computationally infeasible to find M such that H(M)=h, for a given h
  - Computationally infeasible to find M' such that H(M')=H(M), for a given M
  - Computationally infeasible to find M,M' with H(M)=H(M') (to resist to birthday attacks)
- **Note 1: the hash function is not considered secret – some other means are required to protect it**
- Note 2: Hash function plus secrecy (key) gives a MAC – these are called HMACs

4.4 Authentication based on hash functions



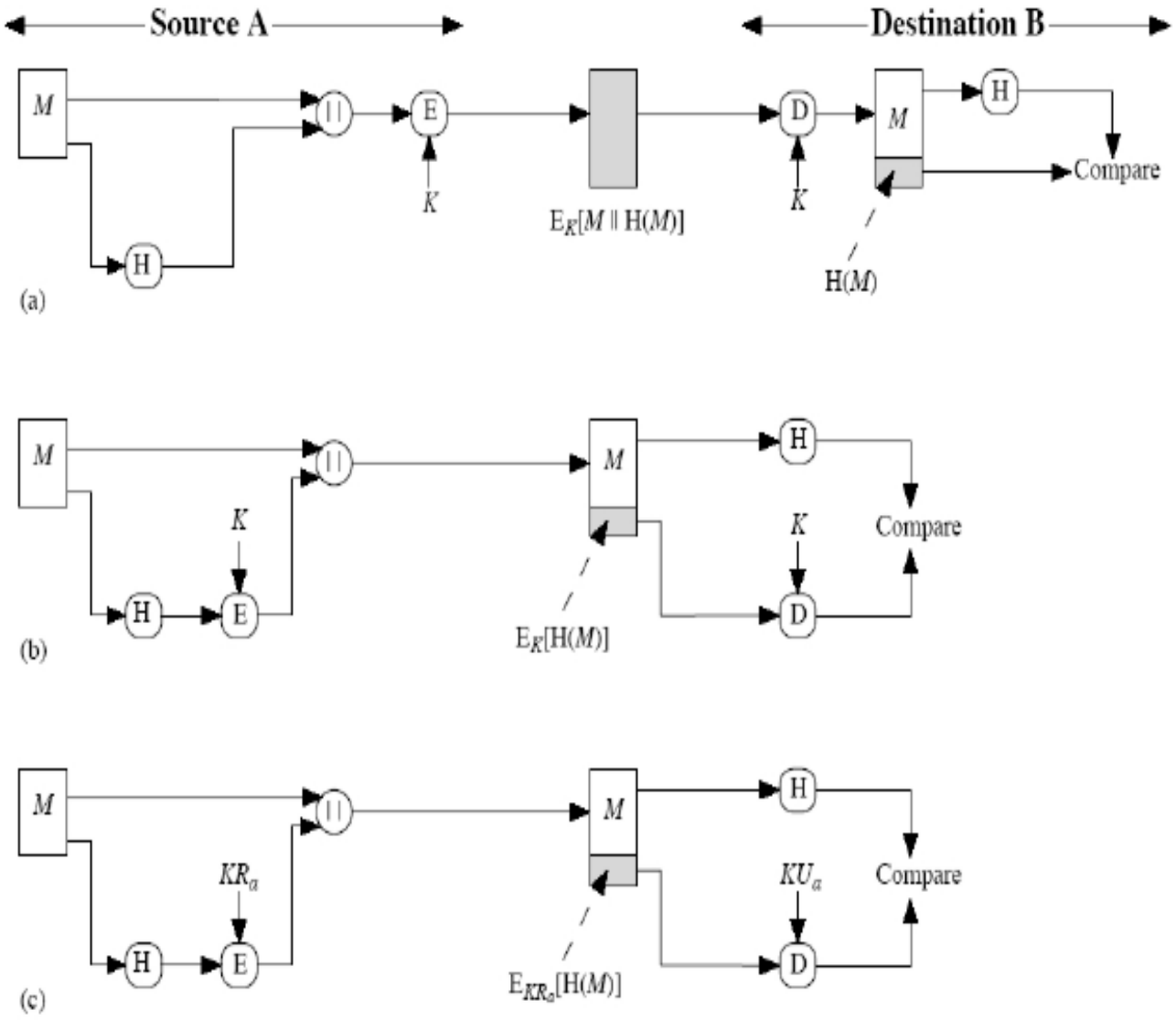
Black Diagram of Cryptographic Hash Function; $h = \mathrm{H}(M)$

4.4.1 Basic Functions of hash functions

a. Classical encryption of message+hash

b. Only the hash value is encrypted

c. As in (b) but with private key (provides digital signature)

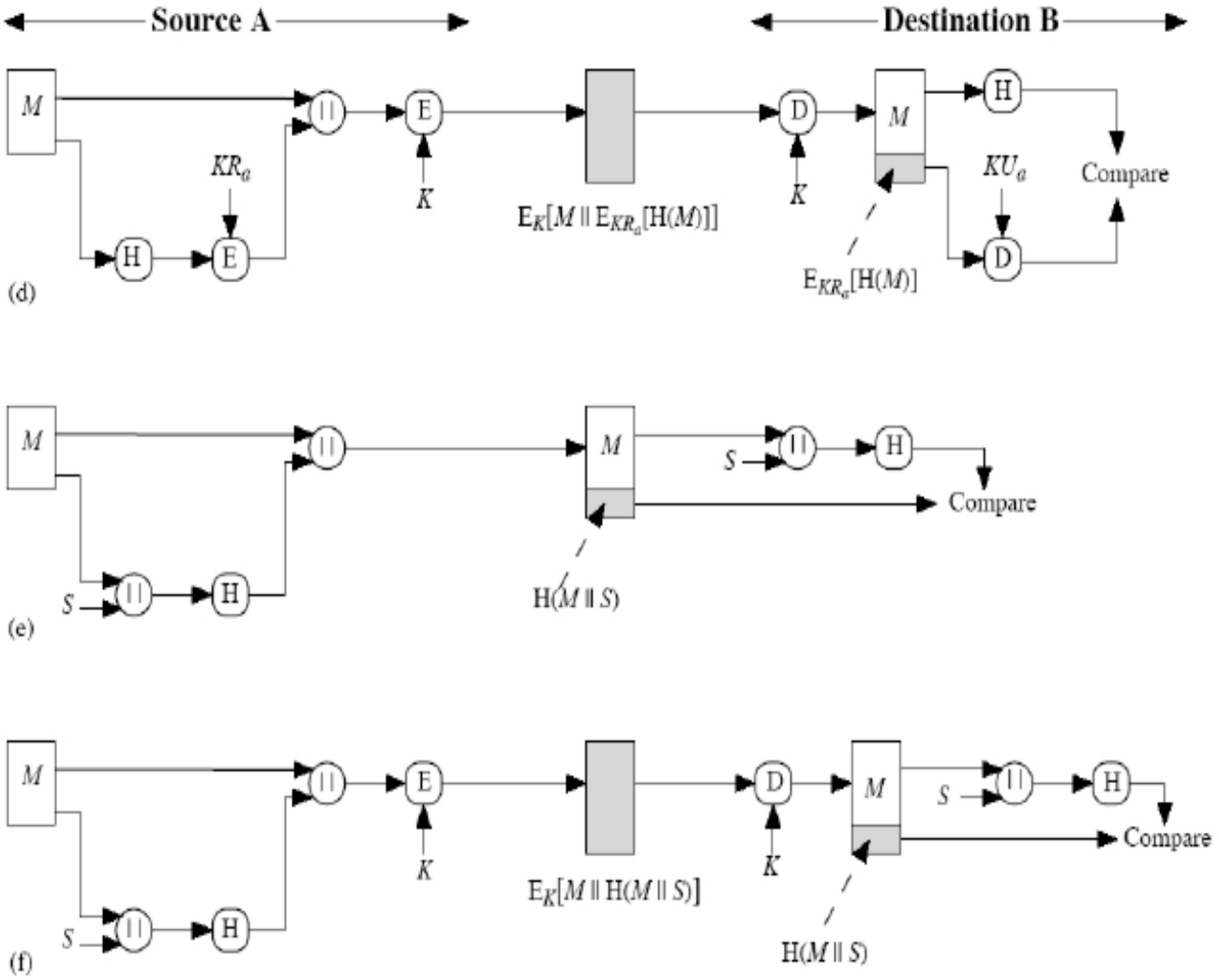


4.4.1 Basic Functions of hash functions

d. Hash is encrypted with an asymmetric system, then a second encryption is applied

e. ***No encryption here but the hash is applied to a message where a secret text S has been appended***

f. As in (e), but with encryption



4.4.1 Basic Functions of hash functions

$A \rightarrow B: E_K[M \parallel H(M)]$ •Provides confidentiality —Only A and B share K •Provides authentication —H(M) is cryptographically protected (a) Encrypt message plus hash code	$A \rightarrow B: E_K[M \parallel E_{KR_a}[H(M)]]$ •Provides authentication and digital signature •Provides confidentiality —Only A and B share K (d) Encrypt result of (c) - shared secret key
$A \rightarrow B: M \parallel E_K[H(M)]$ •Provides authentication —H(M) is cryptographically protected (b) Encrypt hash code - shared secret key	$A \rightarrow B: M \parallel H(M \parallel S)$ •Provides authentication —Only A and B share S (e) Compute hash code of message plus secret value
$A \rightarrow B: M \parallel E_{KR_a}[H(M)]$ •Provides authentication and digital signature —H(M) is cryptographically protected —Only A could create $E_{KR_a}[H(M)]$ (c) Encrypt hash code - sender's private key	$A \rightarrow B: E_K[M \parallel H(M) \parallel S]$ •Provides authentication —Only A and B share S •Provides confidentiality —Only A and B share K (f) Encrypt result of (e)

4.4.2 A few simple hash functions

- Bit-by-bit XOR of plaintext blocks: $h = D_1 \oplus D_2 \oplus \ldots \oplus D_N$
  - Provides a parity check for each bit position
  - Not very effective with text files: most significant bit always 0
  - Attack: to send blocks $X_1, X_2, \ldots, X_{N-1}$, choose $X_N = X_1 \oplus X_2 \oplus \ldots \oplus X_{N-1} \oplus h$
  - It does not help if (only) the hash is sent encrypted!
- Another example: rotated XOR – before each addition the hash value is rotated to the left with 1 bit
  - Better than the previous hash on text files
  - Similar attack
- Another technique: cipher block chaining technique without a secret key
  - Divide message into blocks $D_1, D_2, \ldots, D_N$ and use them as keys in the encryption method (e.g., DES)
  - H_0=some initial value, $H_i = E_{Di}(H_{i-1})$
  - $H = H_N$
  - This can be attacked with the birthday attack if the key is short (as in DES)

4.4.3 Birthday Attacks

- Given at least 23 people, the probability of having two people with the same birthday is more then 0.5
  - it is in fact 0.5005
  - for 30 people it is more than 0.7
  - for 50 people it is more than 0.97
- Related problem: Given two sets X,Y each having k elements from the set {1,2,…,N}, how large should k be so that the probability that X and Y have a common element is more than 0.5?
  - Answer: k should be larger than the square root of N
  - If $N=2^m$, take $k=2^{m/2}$

4.4.3 Birthday Attacks

- Suppose a hash value on 64 bits is used (as the one based on DES)
  - In principle this is secure: given M, to find a message M' with H(M)=H(M'), one has to generate in average 2^{63} messages M'
- A different much more effective attack is possible
  - A is prepared to sign the document by appending its hash value (on *m* bits) and then encrypting the hash code with its private key
  - E will generate $2^{m/2}$ variations of the message M and computes the hash values for all of them
  - E also generates $2^{m/2}$ variations of the message M' that she would really like to have A authenticating and computes the hash values for all of them
  - By the birthday paradox, the probability that the two sets of hash values have one element in common is more than 0.5 – she finds M and M' with the same hash values (messages expressing totally different things!)
  - E will offer M to A for hashing and then signing
  - E will send instead M' with the signature A has produced
  - E breaks the protocol although she does not know A's private key!
  - Level of effort for the hash based on DES: 2^{33}

4.4.3 Birthday Attacks

{This letter is / I am writing} to introduce {you to / to you} {Mr. / } Alfred {P. / } Barton, the {new / newly appointed} {chief / senior} jewelry buyer for {our / the} Northern {European / Europe} {area / division}. He {will take /has taken} over {the / } responsibility for {all / the whole of} our interests in {watches and jewelry / jewelry and watches} in the {area/region}. …

- Complexity of the attack
  - Compute the two lists of messages: each requires an effort on the scale of $2^{m/2}$
  - Sort them: again an effort on the scale of $2^{m/2}$
  - Comparing two sorted tables can be done in linear time!

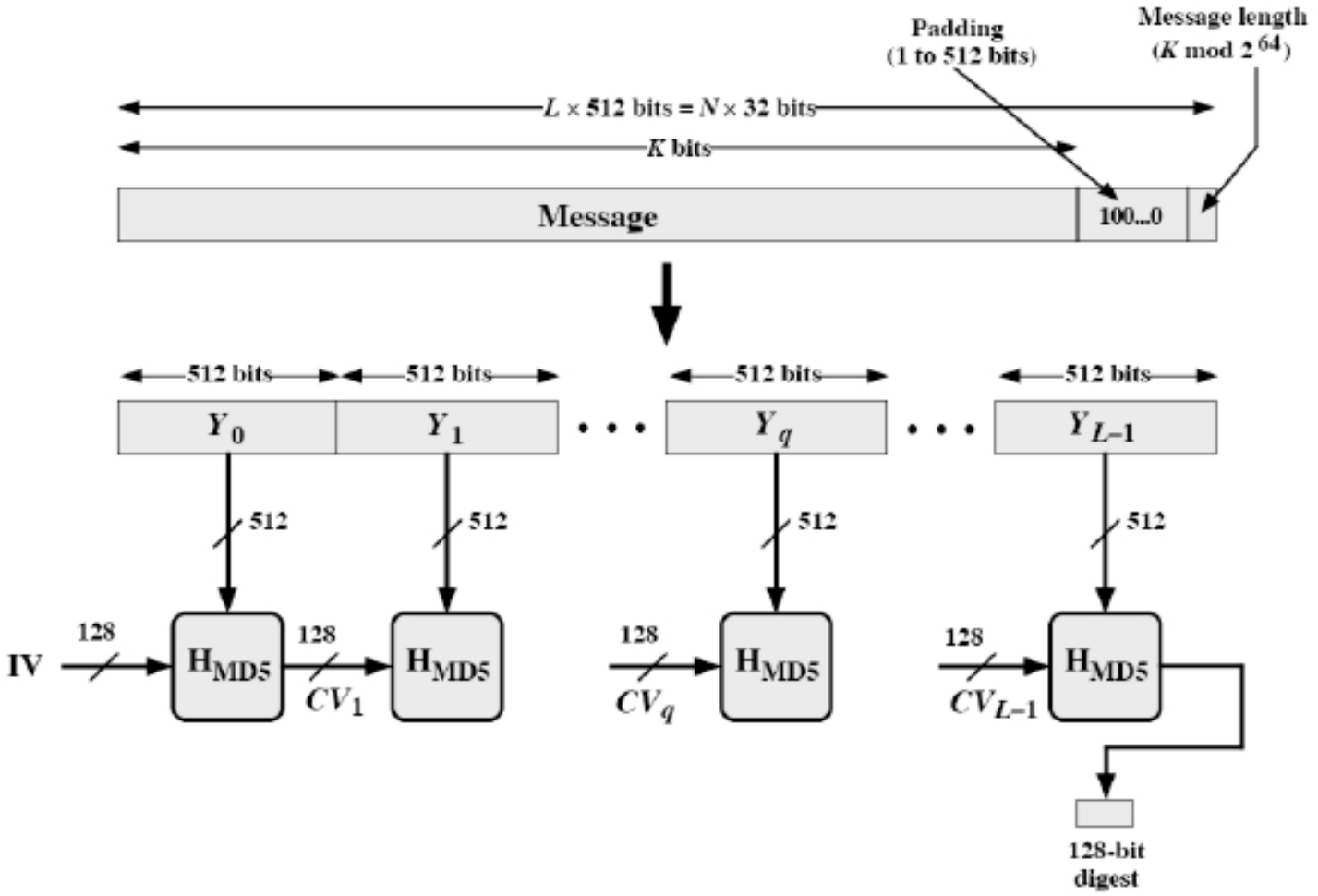
Two popular hash algorithms

- MD5
- SHA-1

4.4.4 MD5 Hash Algorithm

- Most popular hash algorithm until very recently – concerns for its security were raised and was proposed to be replaced by SHA-1, SHA-2
- Developed by Rivest at MIT
- For a message of arbitrary length, it produces an output of 128 bits
  - Processes the input in blocks of 512 bits
- Idea:
  - Start by padding the message to a length of 448 bits modulo 512 – padding is always added even if the message is of required length; the length of the message is added on 64 bits so that altogether the length is a multiple of 512 bits
  - Several rounds, each round takes a block of 512 bits from the message and mixes it thoroughly with a 128 bit buffer that was the result of the previous round
  - The last content of the buffer is the hash value

4.4.4 MD5 Hash Algorithm



4.4.4 MD5 Hash Algorithm

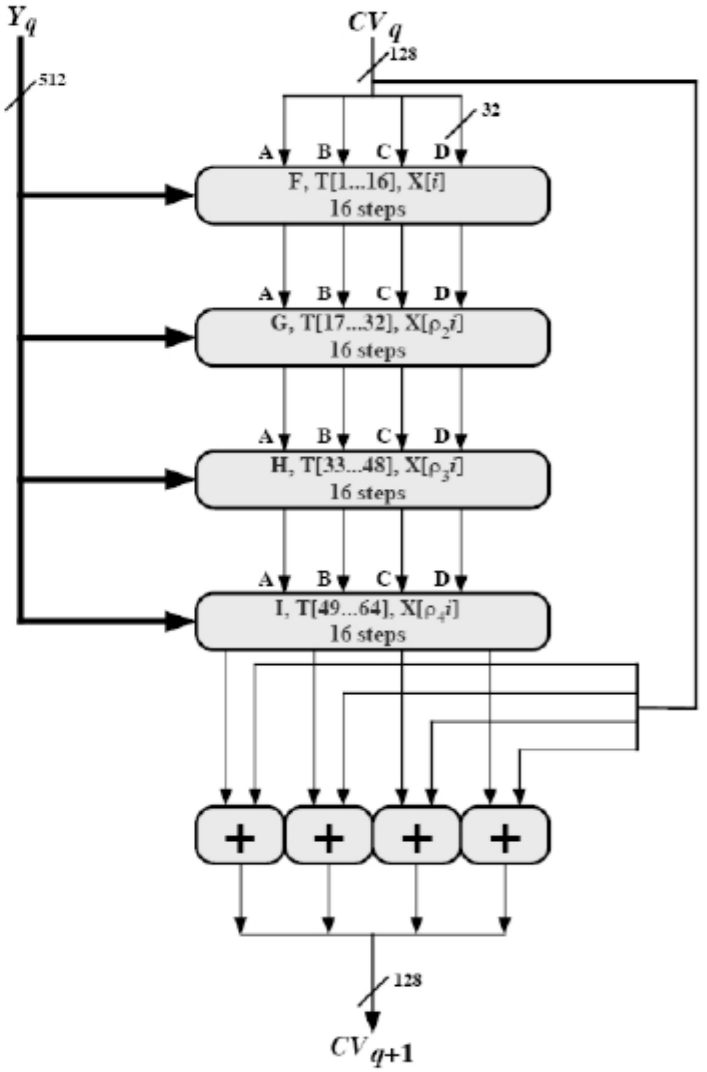
MD5 – the algorithm

1. Padding: add a bit 1 followed by the necessary number of bits 0
2. Append length – the length is represented on 64 bits
   - If the length is larger than 2^{64}, take the 64 least representative bits
3. Initialize MD buffer with the following 4 values, all on 32 bits: A=0x01234567, B=0x89ABCDEF, C=0xFEDCBA98, D=0x76543210
4. Process each message block of 512 bits in 4 rounds
   - Each round takes as input the 512 bits in the input and the content of the buffer ABCD and updates the buffer ABCD (details on the next slide)
   - The four words A,B,C,D in the output of the 4^{th} round are added modulo 2^{32} to the corresponding words A,B,C,D of the input to the first round
5. Output: the 128 bits in the buffer ABCD after the last round

4.4.4 MD5 Hash Algorithm

MD5 processing of a single 512-bit block

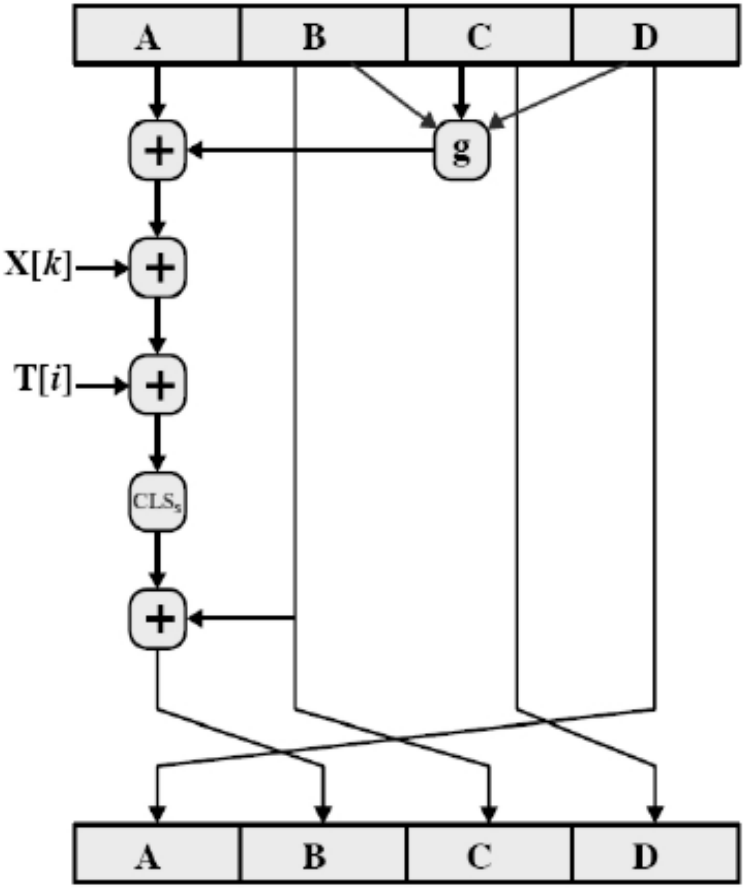
- Each round has 16 steps
- T is a table
- F,G,H,I are Boolean functions (tables) on B,C,D (bit-by-bit operations)
- X has the current 32 bits of the message
  - The message has 512 bits, i.e., 16 blocks of 32 bits
  - Each of the 16 blocks is used exactly once in each round
  - Round 1: used in consecutive order
  - Round 2: used in the order (1+5i) mod 16, i=0,…,15
  - Round 3: used in the order (5+3i) mod 16, i=0,…,15
  - Round 4: used in the order 7i mod 16, i=0,…,15



4.4.4 MD5 Hash Algorithm

One single step in MD5

- All operations here are on blocks of 32 bits
- T is a table
- g is one of the functions F,G,H,I (bit-wise function)
- X has the current 32 bits of the message
- CLS_s is a circular left shift (rotation) with s bits
- "+" is addition modulo 2^{32}



4.4.4 MD5 Hash Algorithm

Table T and truth table of F,G,H,I

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] = 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

4.4.4 MD5 Hash Algorithm

Strength of MD5

- Every bit of the output is a function of all bits of the input
- Rivest's conjecture:
  - As strong as it can be for a 128-bit hash: birthday attack on the order of 2^{64} and finding a message with a given digest is on the order of 2^{128}
- Vulnerabilities found in 1996, then after 10 years a number of other weaknesses reported, most serious in 2008
  - 2008: fake certification of SSL was demonstrated based on MD5
  - currently classified as cryptographically weak
- Used in HMAC

4.4.5 Secure Hash Algorithm (SHA)

Secure Hash Algorithm (SHA)

- Developed by NSA and adopted by NIST in FIPS 180-1 (1993)
  - SHA-1 specified in RFC 3174 – contains a C code implementation
- Part of a family of 3 hashes: SHA-0, SHA-1, SHA-2
  - SHA-1 most widely used
  - recommendations that SHA-2 should be used because of a potential math weakness in SHA-1
  - current competition for a new hash standard to be concluded in December 2012
- Design based on MD4 (previous version of MD5)
- Takes as input any message of length up to 2^{64} bits and gives a 160-bit message digest
- Same structure as MD5, with block length of 512 bits and buffer of 160 bits

4.4.5 Secure Hash Algorithm (SHA)

SHA-512 Logic

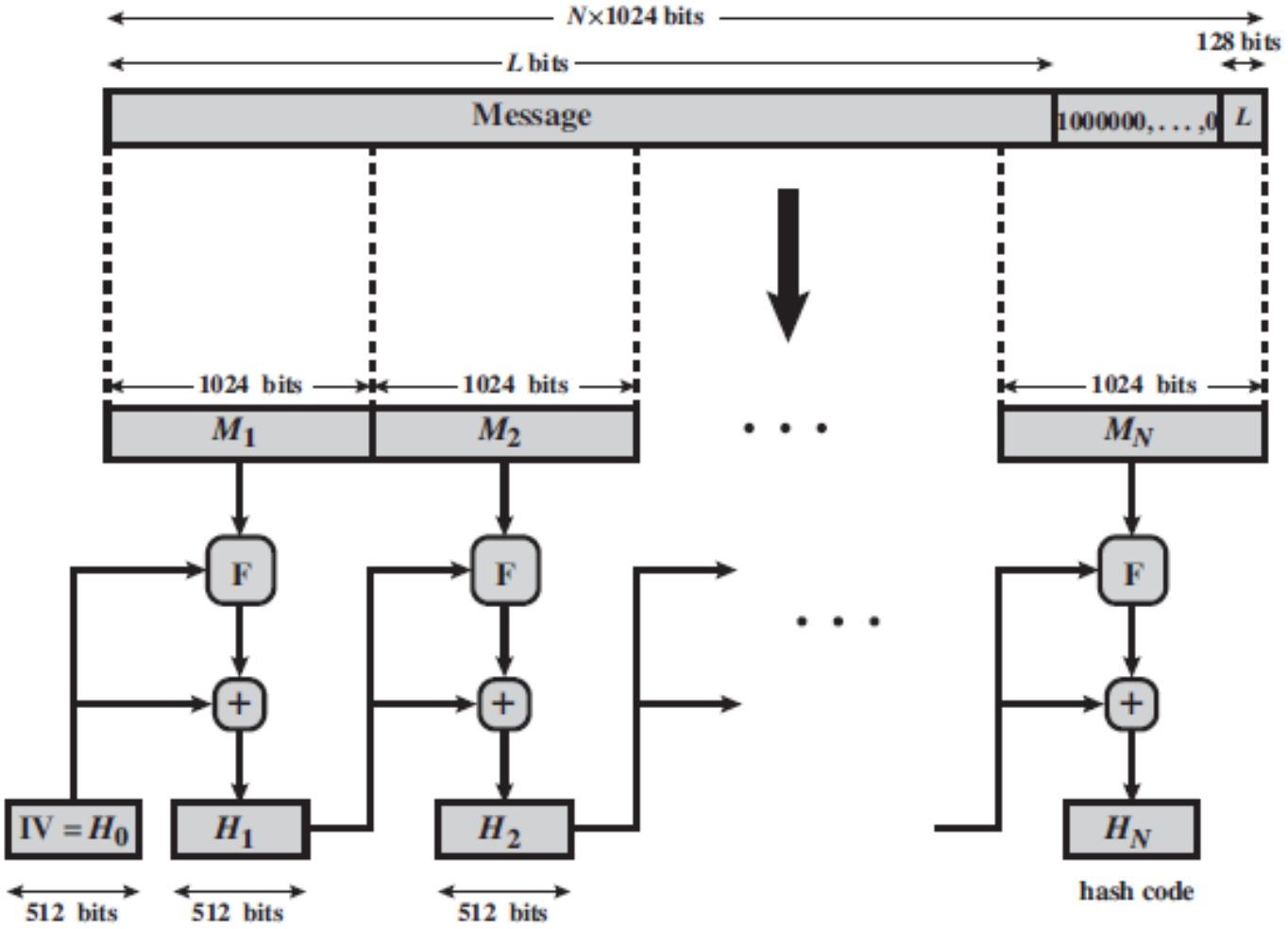
The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks.

Table Comparison of SHA Parameters

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80

Note: All sizes are measured in bits.

4.4.5 Secure Hash Algorithm (SHA)



+ = word-by-word addition mod 2^{64}

Message Digest Generation Using SHA-512

4.4.5 Secure Hash Algorithm (SHA)

Step 1 **Append padding bits.** The message is padded so that its length is congruent to 896 modulo 1024 [length $\equiv$ 896(mod 1024)]. 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 1024. The padding consists of a single 1 bit followed by the necessary number of 0 bits.

Step 2 **Append length.** A block of 128 bits is appended to the message. This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message (before the padding).

The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length. the expanded message is represented as the sequence of 1024-bit blocks $M_1, M_2, \ldots, M_N$, so that the total length of the expanded message is $N \times 1024$ bits.

Step 3 **Initialize hash buffer.** A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h). These registers are initialized to the following 64-bit integers (hexadecimal values):

```
a = 6A09E667F3BCC908   e = 510E527FADE682D1
b = BB67AE8584CAA73B   f = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B   g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1   h = 5BE0CD19137E2179
```

4.4.5 Secure Hash Algorithm (SHA)

These values are stored in **big-endian** format, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

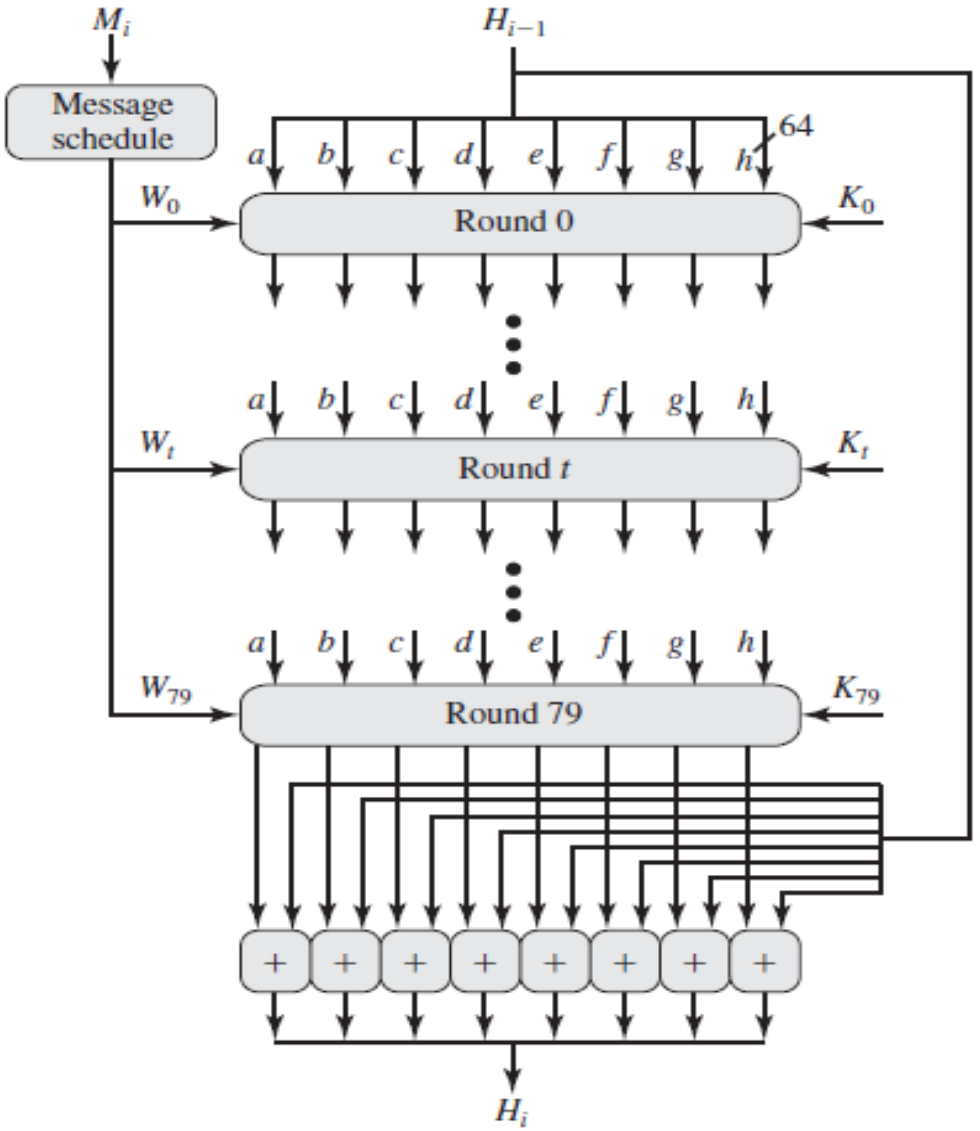
Step 4 Process message in 1024-bit (128-word) blocks. The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F

Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}. Each round t makes use of a 64-bit value W_t, derived from the current 1024-bit block being processed (M_i). These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant K_t, where $0 \leq t \leq 79$ indicates one of the 80 rounds. These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. The constants provide a "randomized" set of 64-bit patterns, which should eliminate any regularities in the input data. Table 11.4 shows these constants in hexadecimal format (from left to right).

The output of the eightieth round is added to the input to the first round (H_{i-1}) to produce H_i. The addition is done independently for each of the eight words in the buffer with each of the corresponding words in H_{i-1}, using addition modulo 2^{64}.

Step 5 Output. After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.

4.4.5 Secure Hash Algorithm (SHA)



SHA-512 Processing of a Single 1024-Bit Block

4.4.5 Secure Hash Algorithm (SHA)

We can summarize the behavior of SHA-512 as follows:

$$H_0 - \text{IV}$$

$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$

$$MD = H_N$$

where

IV = initial value of the abcdefgh buffer, defined in step 3

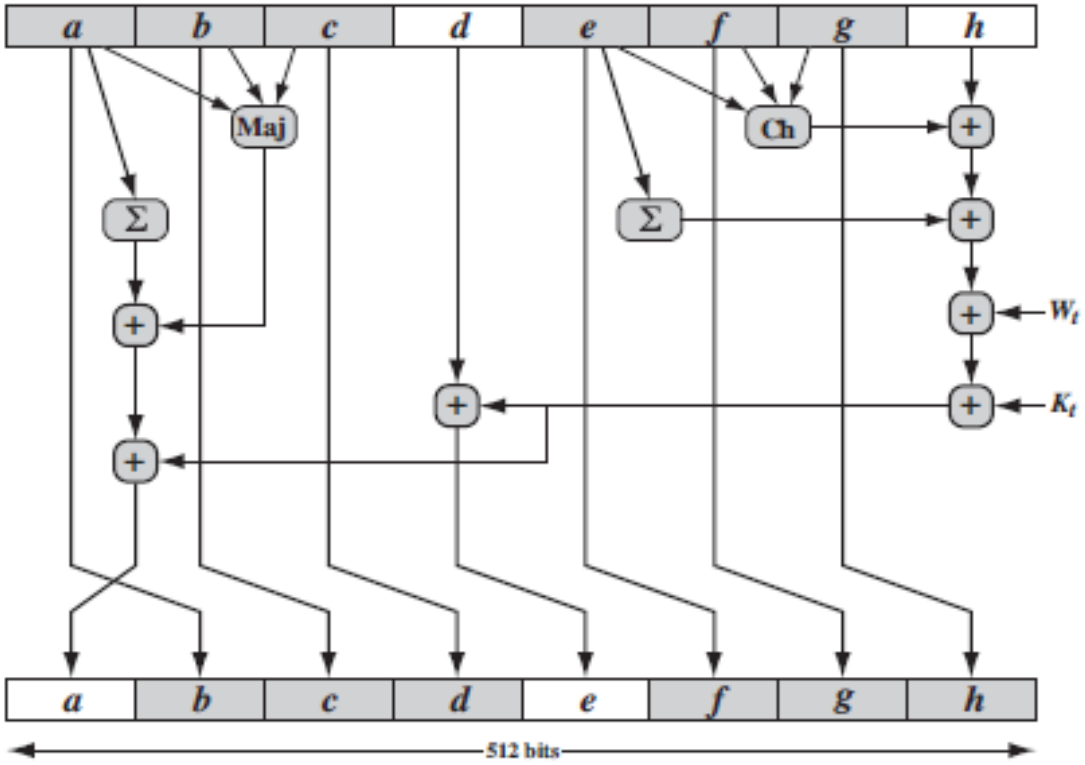
abcdefgh_i = the output of the last round of processing of the ith message block

N = the number of blocks in the message (including padding and length fields)

SUM_{64} = addition modulo 2^{64} performed separately on each word of the pair of inputs

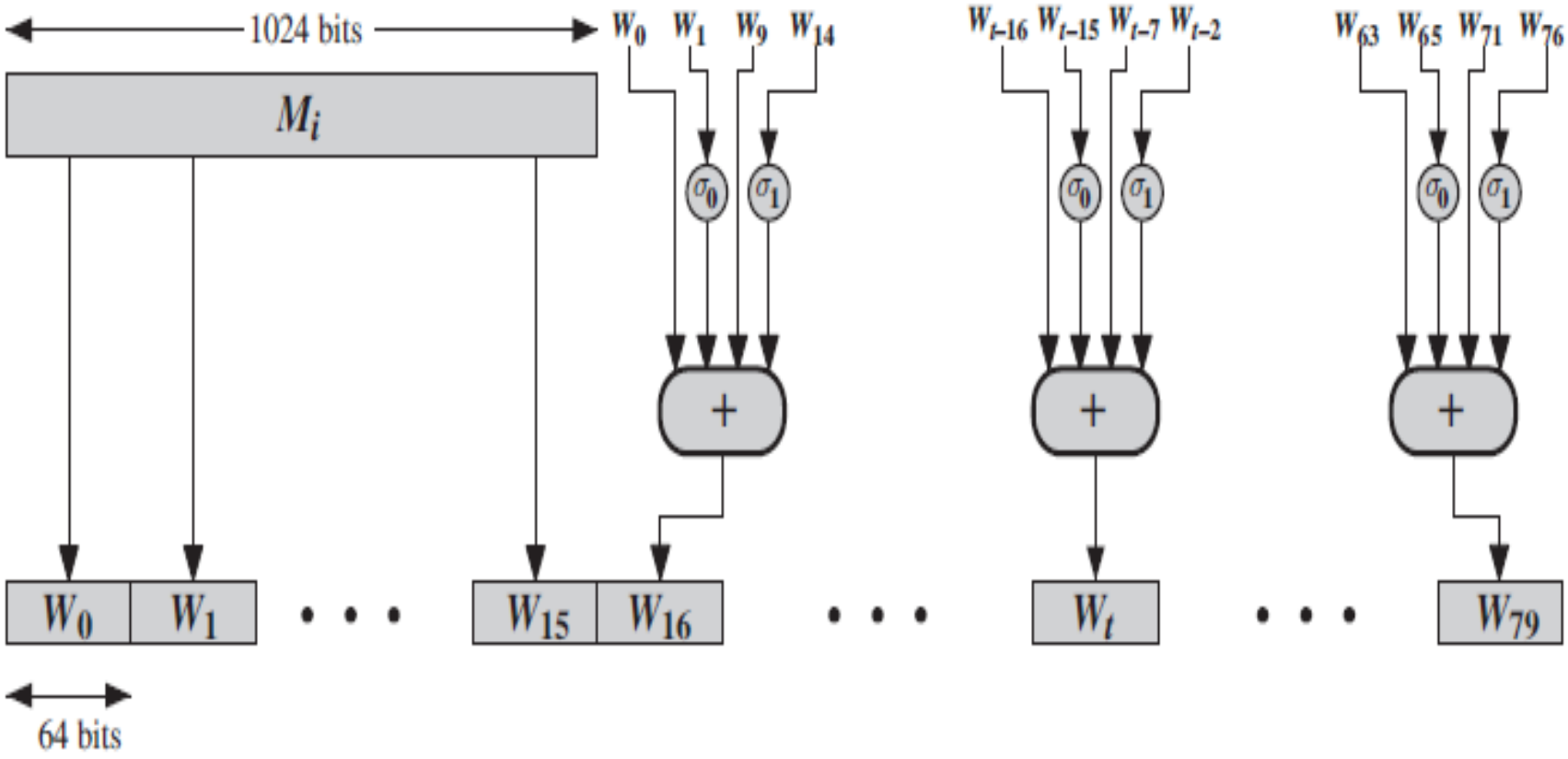
MD = final message digest value

4.4.5 Secure Hash Algorithm (SHA)



Elementary SHA-512 Operation (single round)

4.4.5 Secure Hash Algorithm (SHA)



Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

4.4.5 Secure Hash Algorithm (SHA)

Example

We include here an example based on one in FIPS 180. We wish to hash a one-block message consisting of three ASCII characters: "abc", which is equivalent to the following 24-bit binary string:

```
01100001 01100010 01100011
```

Recall from step 1 of the SHA algorithm, that the message is padded to a length congruent to 896 modulo 1024. In this case of a single block, the padding consists of 896 − 24 = 872 bits, consisting of a "1" bit followed by 871 "0" bits. Then a 128-bit length value is appended to the message, which contains the length of the original message (before the padding). The original length is 24 bits, or a hexadecimal value of 18. Putting this all together, the 1024-bit message block, in hexadecimal, is

```
6162638000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000018
```

This block is assigned to the words W0, . . . ,W15 of the message schedule, which appears as follows.

$$W_0 = 6162638000000000 \qquad W_5 = 0000000000000000$$
$$W_1 = 0000000000000000 \qquad W_6 = 0000000000000000$$
$$W_2 = 0000000000000000 \qquad W_7 = 0000000000000000$$
$$W_3 = 0000000000000000 \qquad W_8 = 0000000000000000$$
$$W_4 = 0000000000000000 \qquad W_9 = 0000000000000000$$

4.4.5 Secure Hash Algorithm (SHA)

$W_{10} = 0000000000000000$ $W_{13} = 0000000000000000$
$W_{11} = 0000000000000000$ $W_{14} = 0000000000000000$
$W_{12} = 0000000000000000$ $W_{15} = 0000000000000018$

As indicated in Figure 11.12, the eight 64-bit variables, a through h, are initialized to values $H_{0,0}$ through $H_{0,7}$. The following table shows the initial values of these variables and their values after each of the first two rounds.

a	6a09e667f3bcc908	f6afceb8bcfcddf5	1320f8c9fb872cc0
b	bb67ae8584caa73b	6a09e667f3bcc908	f6afceb8bcfcddf5
c	3c6ef372fe94f82b	bb67ae8584caa73b	6a09e667f3bcc908
d	a54ff53a5f1d36f1	3c6ef372fe94f82b	bb67ae8584caa73b
e	510e527fade682d1	58cb02347ab51f91	c3d4ebfd48650ffa
f	9b05688c2b3e6c1f	510e527fade682d1	58cb02347ab51f91
g	1f83d9abfb41bd6b	9b05688c2b3e6c1f	510e527fade682d1
h	5be0cd19137e2179	1f83d9abfb41bd6b	9b05688c2b3e6c1f

4.4.5 Secure Hash Algorithm (SHA)

Note that in each of the rounds, six of the variables are copied directly from variables from the preceding round.

The process continues through 80 rounds. The output of the final round is

```
73a54f399fa4b1b2 10d9c4c4295599f6 d67806db8b148677 654ef9abec389ca9
d08446aa79693ed7 9bb4d39778c07f9e 25c96a7768fb2aa3 ceb9fc3691ce8326
```

The hash value is then calculated as

$H_{1,0}$ = 6a09e667f3bcc908 + 73a54f399fa4b1b2 = ddaf35a193617aba

$H_{1,1}$ = bb67ae8584caa73b + 10d9c4c4295599f6 = cc417349ae204131

$H_{1,2}$ = 3c6ef372fe94f82b + d67806db8b148677 = 12e6fa4e89a97ea2

$H_{1,3}$ = a54ff53a5f1d36f1 + 654ef9abec389ca9 = 0a9eeee64b55d39a

$H_{1,4}$ = 510e527fade682d1 + d08446aa79693ed7 = 2192992a274fc1a8

$H_{1,5}$ = 9b05688c2b3e6c1f + 9bb4d39778c07f9e = 36ba3c23a3feebbd

$H_{1,6}$ = 1f83d9abfb41bd6b + 25c96a7768fb2aa3 = 454d4423643ce80e

$H_{1,7}$ = 5be0cd19137e2179 + ceb9fc3691ce8326 = 2a9ac94fa54ca49f

The resulting 512-bit message digest is

```
ddaf35a193617aba cc417349ae204131 12e6fa4e89a97ea2 0a9eeee64b55d39a
2192992a274fc1a8 36ba3c23a3feebbd 454d4423643ce80e 2a9ac94fa54ca49f
```

End of Chapter 4/Part1