INFORMATION SECURITY (2170709)

MESSAGE AUTHENTICATION AND HASH FUNCTIONS

Authentication Requirements

- When communication takes place across a network, some types of attacks can be considered:
  - Disclosure
    - Message contents are released to any person who has not appropriate cryptographic key.
  - Traffic Analysis
    - Traffic is discovered between communicating parties
  - Masquerade
    - Fraudulent source can insert messages into network
    - Fraudulent recipient can give fraudulent acknowledgements of message receipt.

Authentication Requirements

- Content Modification
  - Changes to the contents of the message
  - Insert/update/delete
- Sequence Modification
  - Message sequence is modified
- Timing Modification
  - A sequence of messages can be delayed or replayed
- Source Repudiation
  - Transmission of message is denied by the source
- Destination Repudiation
  - Reception of message is denied by the destination

Authentication Requirements

- Message authentication is a procedure to verify that received messages come from intended source and have not been altered.
- It should also verify sequencing and timeliness.

Authentication Functions

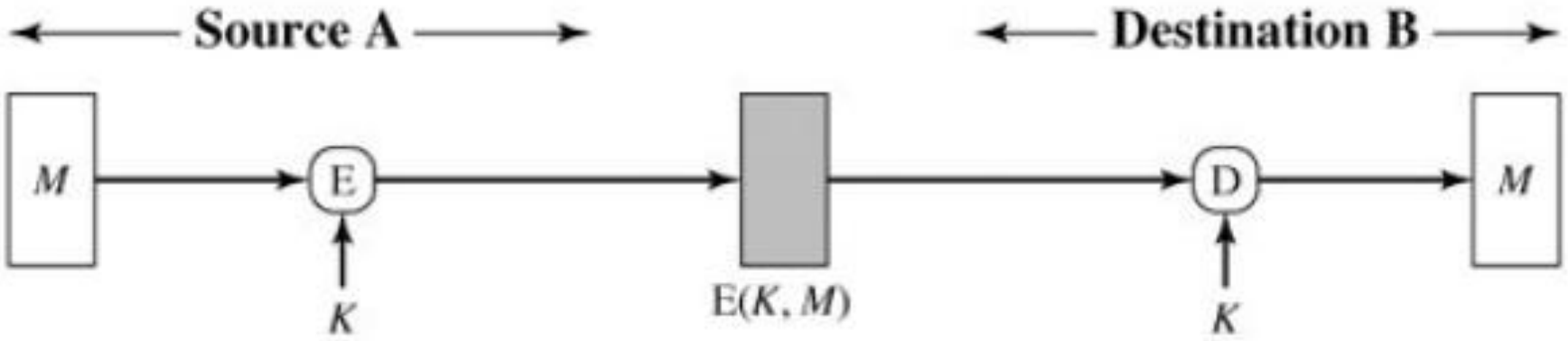
- For any message authentication, there must be some sort of function that produces an ***Authenticator***.
- Authenticator is a value that is used to authenticate a message.
- Three different classes used to generate authentication functions that produce authenticator are:
  - Message Encryption
  - Message Authentication Code (MAC)
  - Hash Function

Authentication Functions

- 1. Message Encryption
  - The ciphertext itself serves as an authenticator
- 2. Message Authentication Code (MAC)
  - A public function and a secret key are used to produce a fixed-length value that serves as an authenticator
- 3. Hash Function
  - A public function is used to map a variable size message into a fixed-length hash value which serves as an authenticator

Authentication Functions: Message Encryption

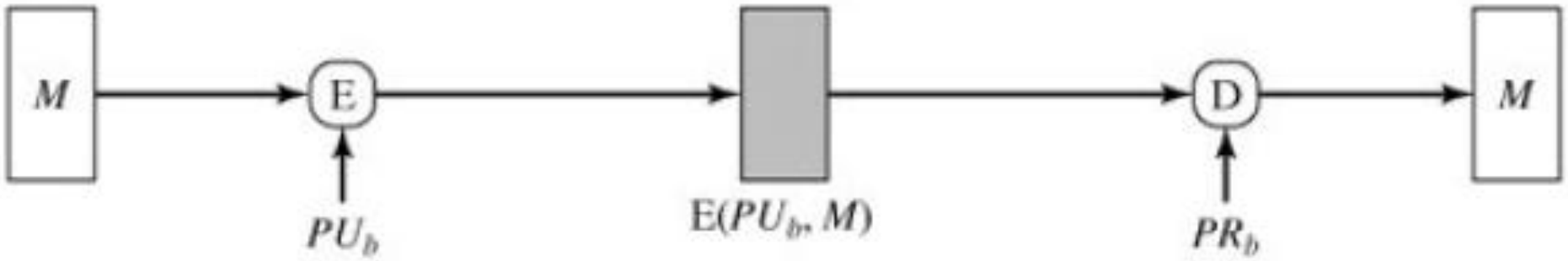
- (a) Symmetric Key Encryption



(a) Symmetric encryption: confidentiality and authentication

Message Encryption

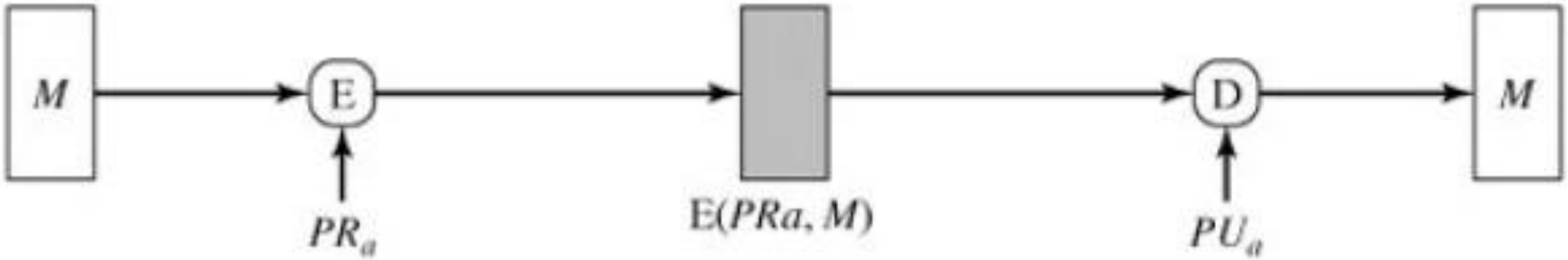
- (b) Public Key Encryption



(b) Public-key encryption: confidentiality

Message Encryption

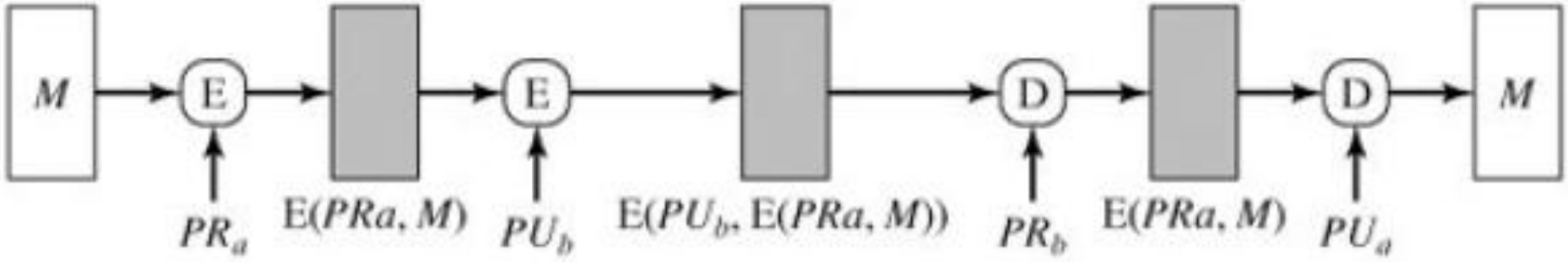
- (b) Public Key Encryption



(c) Public-key encryption: authentication and signature

Message Encryption

- (b) Public Key Encryption



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

Authentication Functions: Message Authentication Code (MAC)

- A public function and a secret key are used to generate a small fixed-length value, known as *MAC* or *Cryptographic Checksum*.

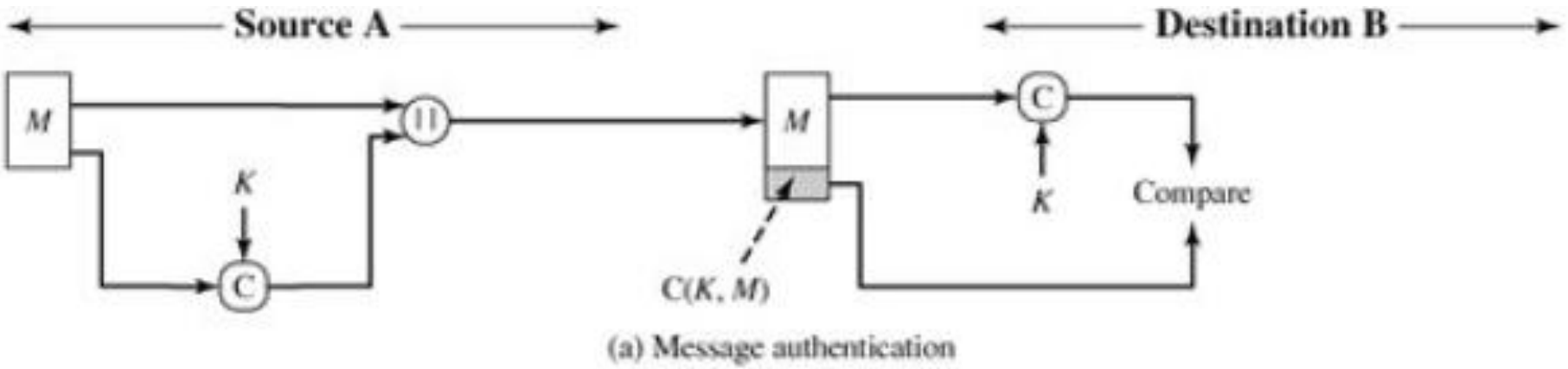
- When *A* has a message to send to *B*, it calculates the value of MAC by:

$$MAC = C_k(M)$$

Where M= input message
C = MAC Function
K = Shared Secret Key
MAC = Message Authentication Code

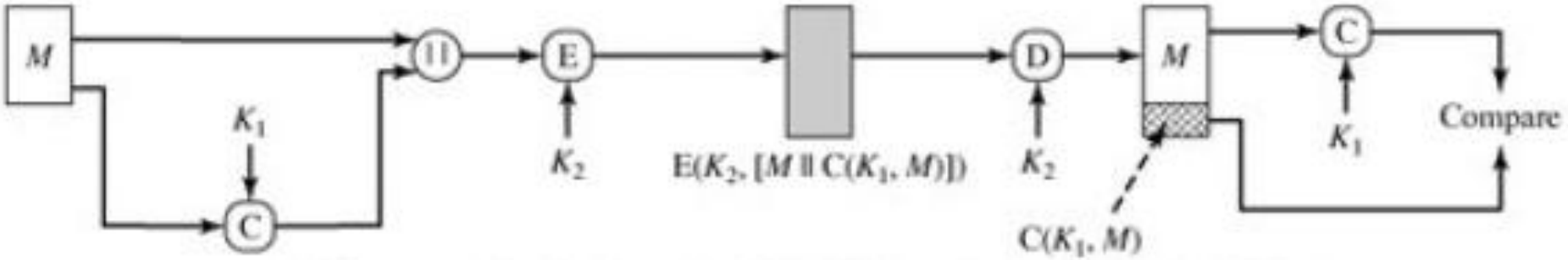
- For any message M, a unique MAC value is generated. Changing a single bit of message M leads to change in many bits of the value of MAC.

MAC

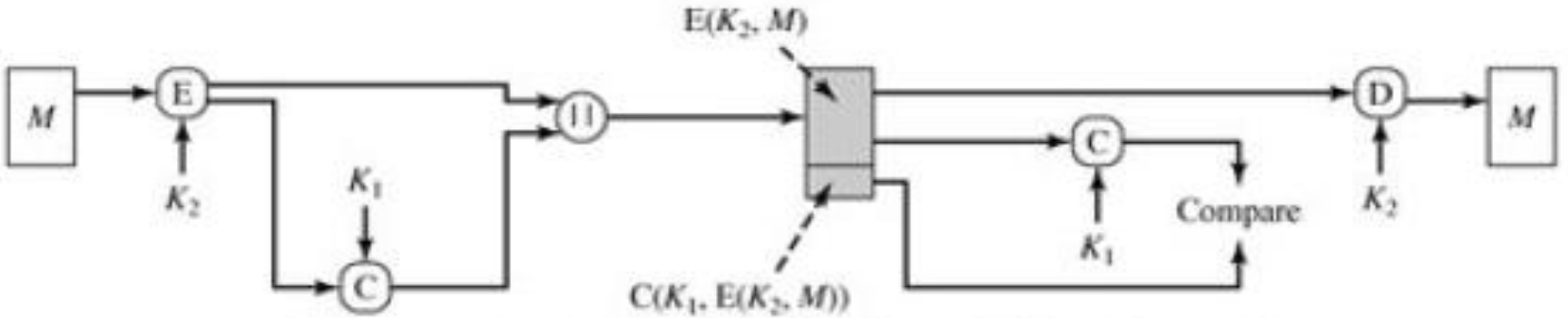


(a) Message authentication

MAC



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

Message Authentication Code (MAC)

- When *A* has a message to send to *B*, it calculates the value of MAC by:

$$MAC = C_k(M)$$

Where M= variable length message
C = MAC Function
K = Shared Secret Key

- Brute force attack is possible.

- For a ciphertext-only attack, given a ciphertext, the opponent would perform $P_i = D(K_i, C)$ for all possible key values K_i.

Requirements of MAC

- If an opponent observes M and $C(K, M)$, it is computationally infeasible to construct a message M' such that $C(K, M') = C(K, M)$
- $C(K, M)$ should be uniformly distributed. For randomly chosen messages M and M', the probability that $C(K, M') = C(K, M)$ is 2^{-n} (n: MAC size)
- Let M' is some known transformation on M. M' = f(M). Then Pr[$C(K, M') = C(K, M)$] is 2^{-n}.

MAC based on Block Ciphers

- Following two MACs are based on the use of a block cipher mode of operation.
  1. MAC based on DES (DAA)
  2. Cipher-based MAC (CMAC)

MAC based on DES (DAA)

- Data Authentication Algorithm (DAA), based on DES, has been one of the most widely used MACs.
- DAA can be defined as using Cipher Block Chaining (CBC) mode of operation of DES with an Initialization Vector (IV) of 0.
- The data/message to be authenticated are grouped into 64-bit blocks: D_1, D_2,…,D_N

MAC based on DES (DAA)

- Using the DES encryption algorithm, E, and a secret key, K, a *data authentication code (DAC)* is calculated as follows:

$$O_1 = E(K, D_1)$$

$$O_2 = E(K, [D_2 \oplus O_1])$$

$$O_3 = (K, [D_3 \oplus O_2])$$

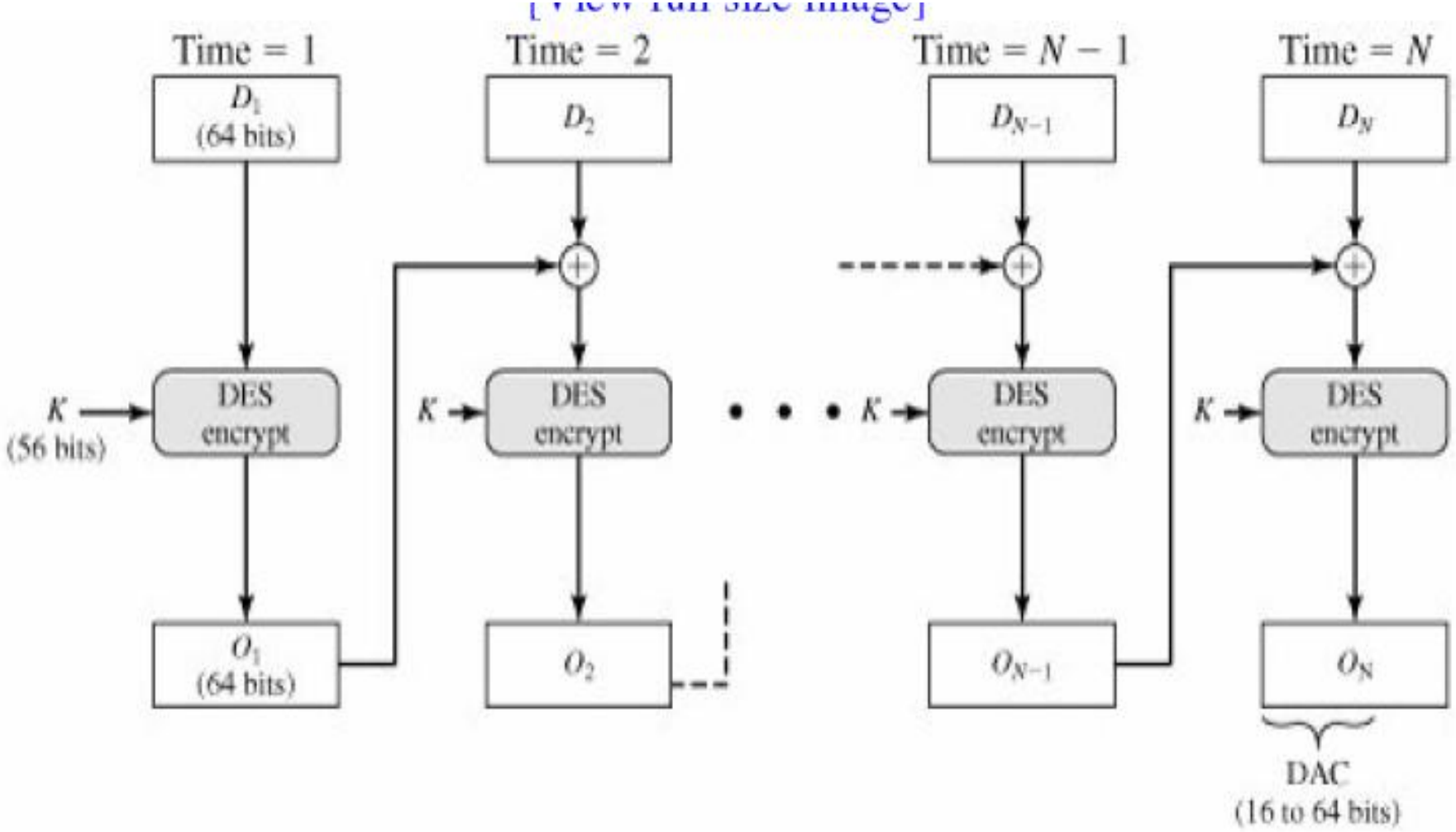
•

•

•

$$O_N = E(K, [D_N \oplus O_{N1}])$$

MAC based on DES (DAA)



MAC based on DES (DAA)

- The DAC consists of either the entire block O_N or the leftmost M bits of the block, with 16 <= M <= 64.

Cipher-based MAC (CMAC)

- In DAA, only messages of one fixed length of *mn* bits are processed, where *n* is the cipher block size and *m* is a fixed positive integer.

- In CMAC, The two n-bit keys could be derived from the encryption key.
- **Cipher-based Message Authentication Code (CMAC)** mode of operation is used with AES and triple DES.

- The message is an integer multiple *n* of the cipher block length *b*. For AES, *b* = 128, and for triple DES, *b* = 64.

- The message is divided into *n* blocks $(M_1, M_2, .. , M_n)$.
- The algorithm makes use of a ***k-bit*** encryption key ***K*** and a ***b-bit*** constant, $\boldsymbol{K_1}$.

- For AES, the key size k is 128, 192, or 256 bits; for triple DES, the key size is 112 or 168 bits.

Cipher-based MAC (CMAC)

- CMAC is calculated as follows

$$
\begin{aligned}
C_1 &= \mathrm{E}(K, M_1) \\
C_2 &= \mathrm{E}(K, [M_2 \oplus C_1]) \\
C_3 &= \mathrm{E}(K, [M_3 \oplus C_2]) \\
&\cdot \\
&\cdot \\
&\cdot \\
C_n &= \mathrm{E}(K, [M_n \oplus C_{n-1} \oplus K_1]) \\
T &= \mathrm{MSB}_{Tlen}(C_n)
\end{aligned}
$$

where

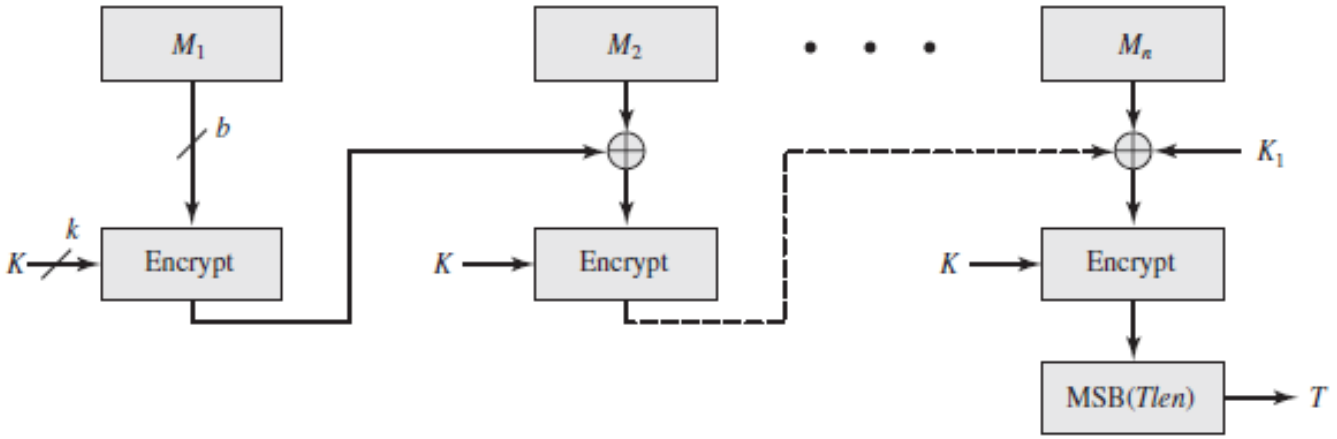
$$
\begin{aligned}
T &= \text{message authentication code, also referred to as the tag} \\
Tlen &= \text{bit length of T} \\
\mathrm{MSB}_s(X) &= \text{the } s \text{ leftmost bits of the bit string } X
\end{aligned}
$$

Cipher-based MAC (CMAC)

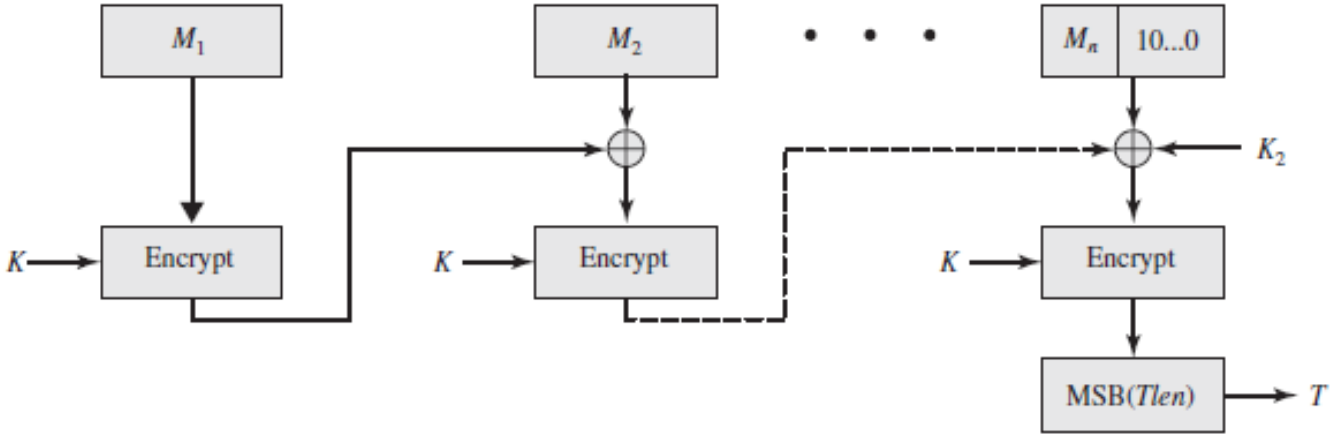
- If the message is not an integer multiple of the cipher block length b, then the final block is padded to the right (least significant bits) with a 1 and as many 0s as necessary so that the final block is also of length b.

- The CMAC operation then proceeds as before, except that a different b-*bit* key K_2 is used instead of K_1.

Cipher-based MAC (CMAC)



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

Figure 12.8 Cipher-Based Message Authentication Code (CMAC)

Authentication Functions: Hash Functions

- A Hash Function accepts a variable size message *M* as input and produces a fixed-length value as output, known as a Hash Code *H*(*M*)/Message Digest/Hash Value.
- Unlike MAC, a hash code does not use a secret key, but a hash function *H* only.
- A change to any bit or bits of original message results in a change to the hash code.
- A Hash value h is generated by a function H as
  - h=H(M)

where M: variable length message
H(M): Fixed length Hash Value

Hash Functions

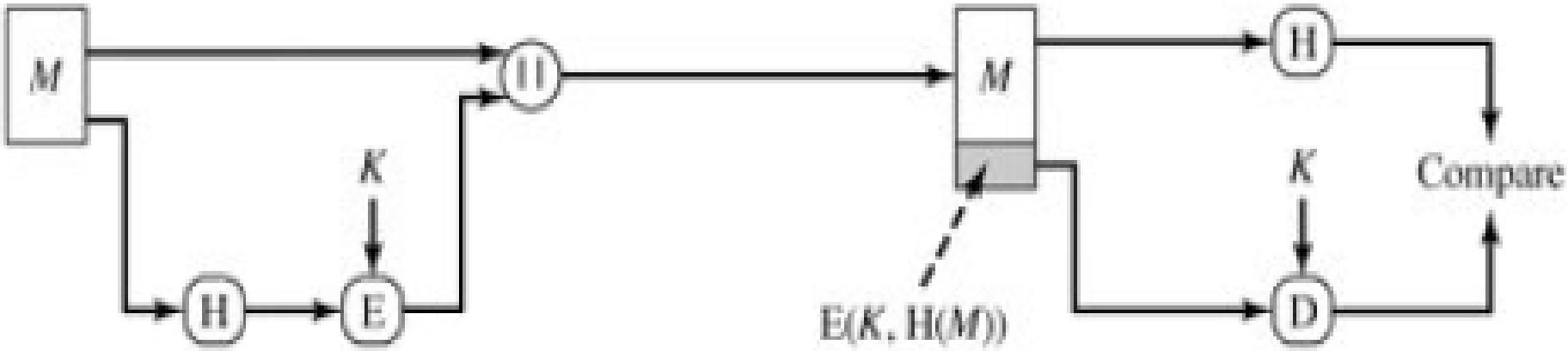


Fig (b). Authentication

Hash Functions

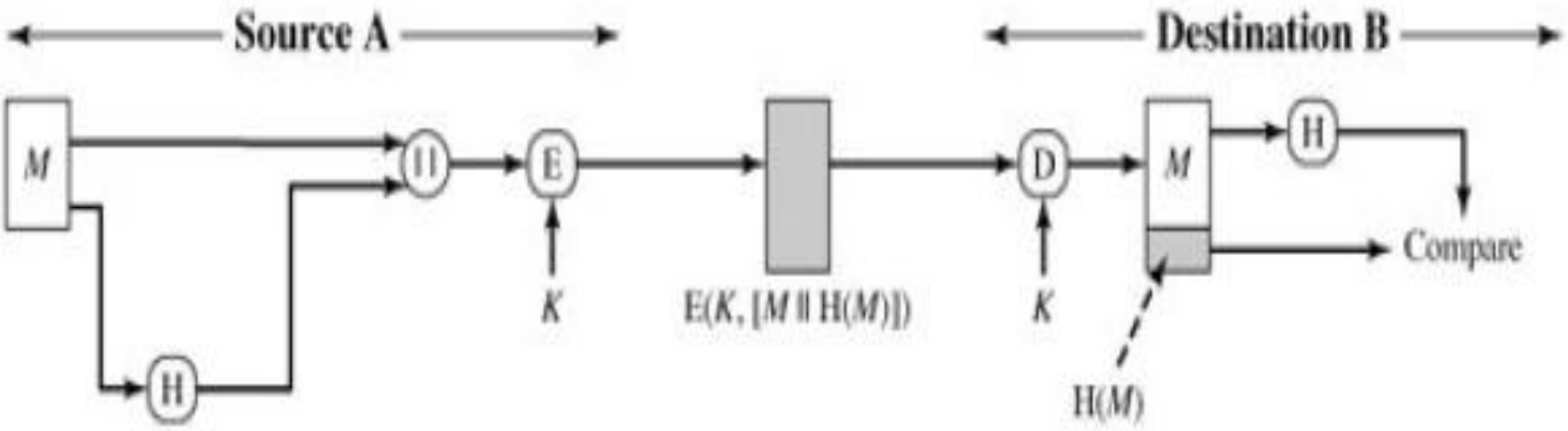


Fig (a). Authentication & Confidentiality

Hash Functions

Fig (c). Authentication & Signature

Hash Functions

Fig (d). Confidentiality & Digital Signature

Hash Functions

- A Hash value *h* is generated by a function *H* as
  - *h=H(M)*

  where *M*: variable length message
  H(M): Fixed length Hash Value

- ***Avalanche Effect***: A slight change in an input string should cause the hash value to change drastically.

Requirements of Hash Functions

1. *H* can be applied to a block of data of any size.
2. *H* produces a fixed-length output.
3. *H(x)* is relatively easy to compute for any given x.
4. For any given value *h*, it is computationally infeasible to find message *x* such that *H(x) = h*. This is sometimes referred to as the **one-way property**.

Requirements of Hash Functions

5. For any given message block x, it is computationally infeasible to find $y \neq x$ such that $H(y) = H(x)$. This is sometimes referred to as **weak collision resistance.**

6. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$. This is sometimes referred to as **strong collision resistance.**

Simple Hash Functions

- For a hash function, input is viewed as a sequence of n-bit blocks.

- One of the simplest hash functions is bit-by-bit exclusive-OR of every block.

$$C_i = b_{i1} \oplus b_{i1} \oplus \ldots \oplus b_{im}$$

C_i = ith bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input

b_{ij} = ith bit in jth block

$\oplus$ = XOR operation

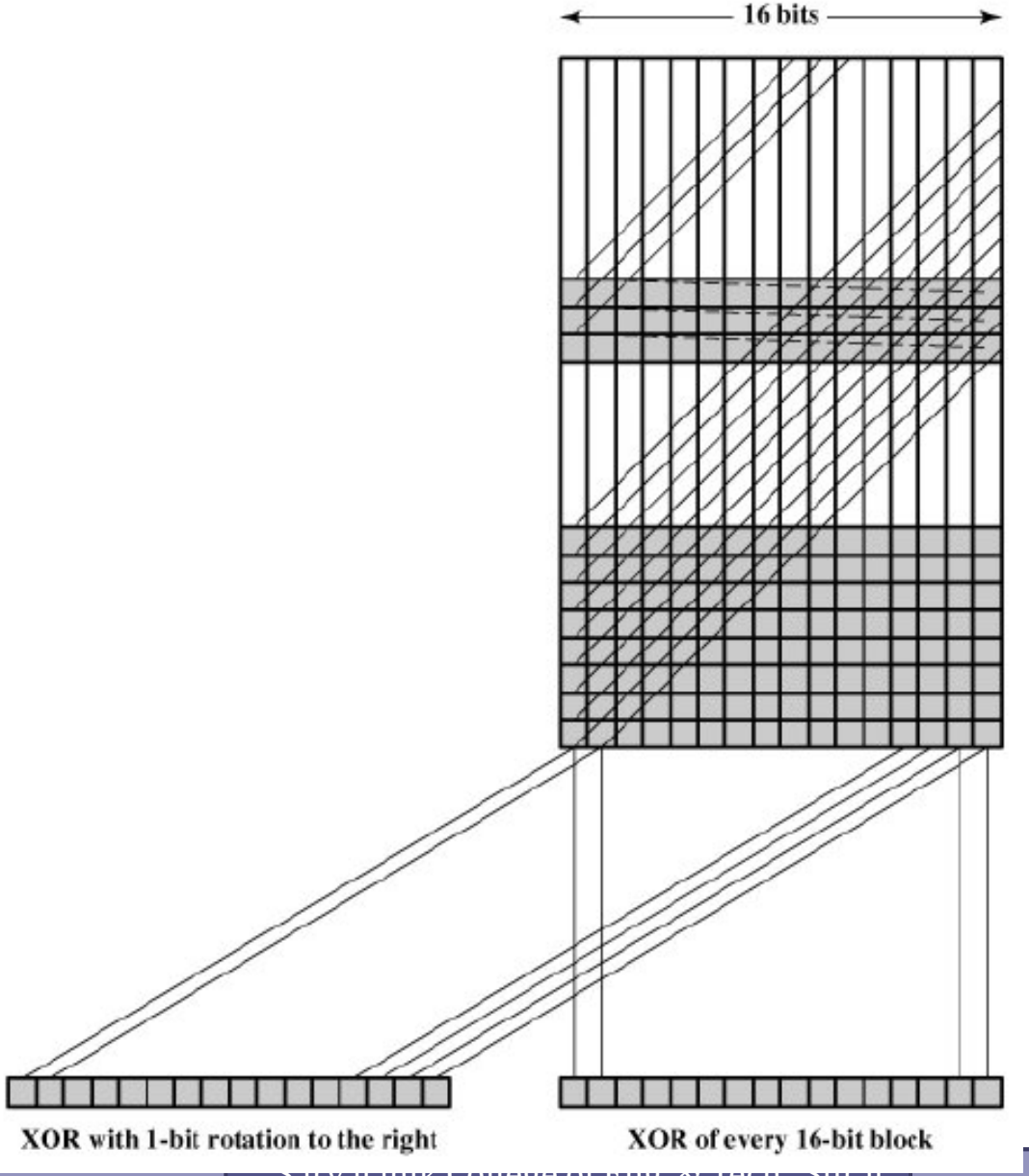
Simple Hash Functions

- With more predictably formatted data, the simple hash function is less effective.

- A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows:

1. Initially set the n-bit hash value to zero.
2. Process each successive n-bit block of data as follows:
   - a. Rotate the current hash value to the left by one bit
   - b. XOR the block into the hash value

Simple Hash Functions



Hash functions based on CBC

- One of the proposals for CBC based Hash was that of Rabin.
- Divide a message M into fixed-size blocks $M_1, M_2, .. , M_N$ and use a symmetric encryption system such as DES to compute the hash code G as

$$H_0 = initial\ value$$
$$H_i = E(M_i, H_{i-1})$$
$$G = H_N$$

- As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.

Birthday Attacks

- Suppose that a 64-bit hash code is used.
- If an encrypted hash code h is transmitted with the corresponding unencrypted message M, then an opponent would need to find an M' such that $H(M') = H(M)$ to substitute another message and fool the receiver.
- On average, the opponent would have to try about 2^{63} messages to find one that matches the hash code of the intercepted message.

Birthday Attacks

- However, a different sort of attack is possible, based on the birthday paradox

1. The source A, is prepared to "sign" a message by appending the appropriate m-bit hash code and encrypting that hash code with A's private key.

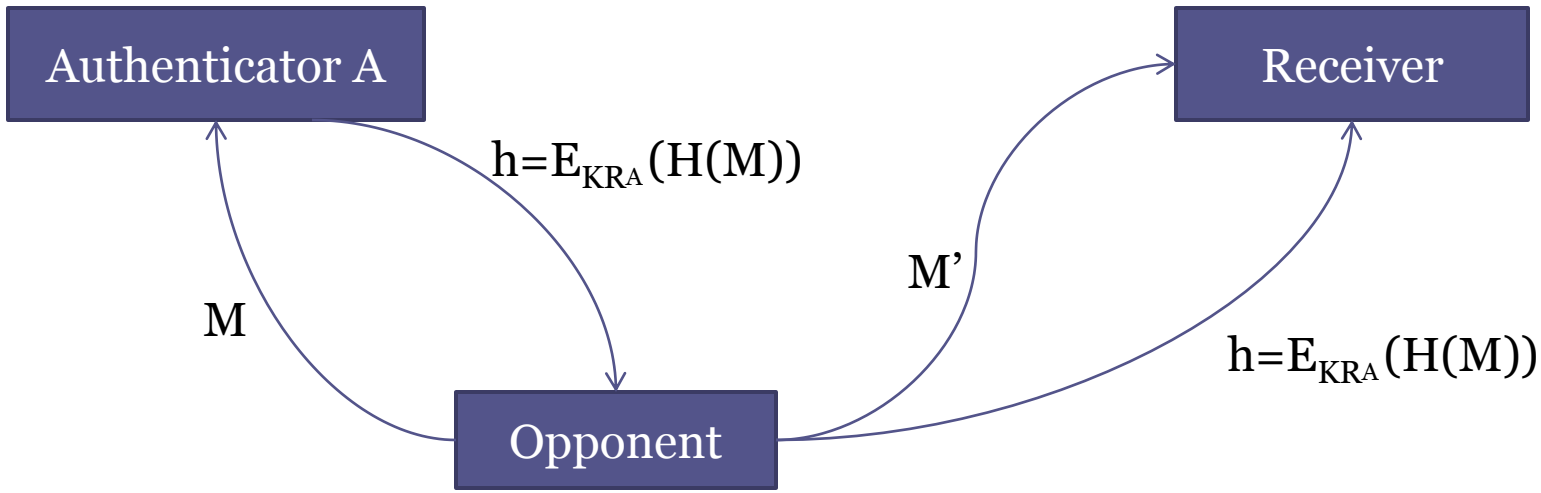
2. The opponent generates $2^{m/2}$ variations on the message, all of which convey essentially the same meaning.

Birthday Attacks

3. The two sets of messages are compared to find a pair of messages that produces the same hash code. The probability of success, by the birthday paradox, is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made.

4. The opponent offers the valid variation (M) to A for signature. This signature can then be attached to the fraudulent variation (M') for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

Birthday Attacks



Birthday Attacks

[View full size image]

Dear Anthony,

{This letter is / I am writing} to introduce {you to / to you} {Mr. / --} Alfred {P. / --}

Barton, the {new / newly appointed} {chief / senior} jewellery 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 jewellery / jewellery and watches}

in the {area / region}. Please {afford / give} him {every / all the} help he {may need / needs}

to {seek out / find} the most {modern / up to date} lines for the {top / high} end of the

market. He is {empowered / authorized} to receive on our behalf {samples / specimens} of the

Birthday Attacks

- Thus, if a 64-bit hash code is used, the level of effort required is only on the order of 2^{32}
- The generation of many variations that convey the same meaning is not difficult.
- For example, the opponent could insert a number of "space-space-backspace" character pairs between words throughout the document.

Security of MAC and Hash

- We can group attacks on hash functions and MACs into two categories:
  - Brute-force attacks and
  - Cryptanalysis

Brute-Force Attack

- Hash Functions
- The strength of a hash function against brute-force attacks depends on the length of the hash code produced by the algorithm.

Brute-Force Attack

- Hash Functions
- There are three desirable properties:
- For any given value h, it is computationally infeasible to find x such that H(x) = h. This is sometimes referred to as the **one-way property**.

- For any given block x, it is computationally infeasible to find y ≠ x such that H(y) = H(x). This is sometimes referred to as **weak collision resistance.**

- It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as **strong collision resistance.**

Brute-Force Attack

- Hash Functions
- For a hash code of length n, the level of effort required is proportional to the following:

One way	2^n
Weak collision resistance	2^n
Strong collision resistance	$2^{n/2}$

- A $10 million collision search machine for MD5, which has a 128-bit hash length, could find a collision in 24 days. Thus a 128-bit code may be viewed as inadequate.

- With a hash length of 160 bits, the same search machine would require over four thousand years to find a collision.

Brute-Force Attack

- Message Authentication Code
- A brute-force attack on a MAC is a more difficult undertaking because it requires known message-MAC pairs.

- We need to state the desired security property of a MAC algorithm, which can be expressed as follows:

- *Computation resistance*: Given one or more text-MAC pairs $[x_i, C(K, x_i)]$, it is computationally infeasible to compute any text-MAC pair $[x, C(K, x)]$ for any new input $x \neq x_i$.

- There are two attacks possible: attack the key space or attack the MAC value

Brute-Force Attack

- Message Authentication Code
- The level of effort for brute-force attack on MAC can be expressed as $\min(2^k, 2^n)$
- It would appear reasonable to require that the key length and MAC length satisfy a relationship such as min(k, n) >= N, where N is perhaps in the range of 128 bits.

Cryptanalysis

- Hash Function

- The hash algorithm involves repeated use of a compression function (*f*), that produces an n-bit output.

- Cryptanalysis of hash function focuses on the internal structure of function *f* and is based on attempts to find efficient techniques for producing collisions for a single execution of *f*.

MD5 Message Digest Algorithm

- MD5 algorithm takes as input a message of arbitrary length and produces as output a 128-bit message digest (hash code).

- The input message is processed in 512-bit blocks.

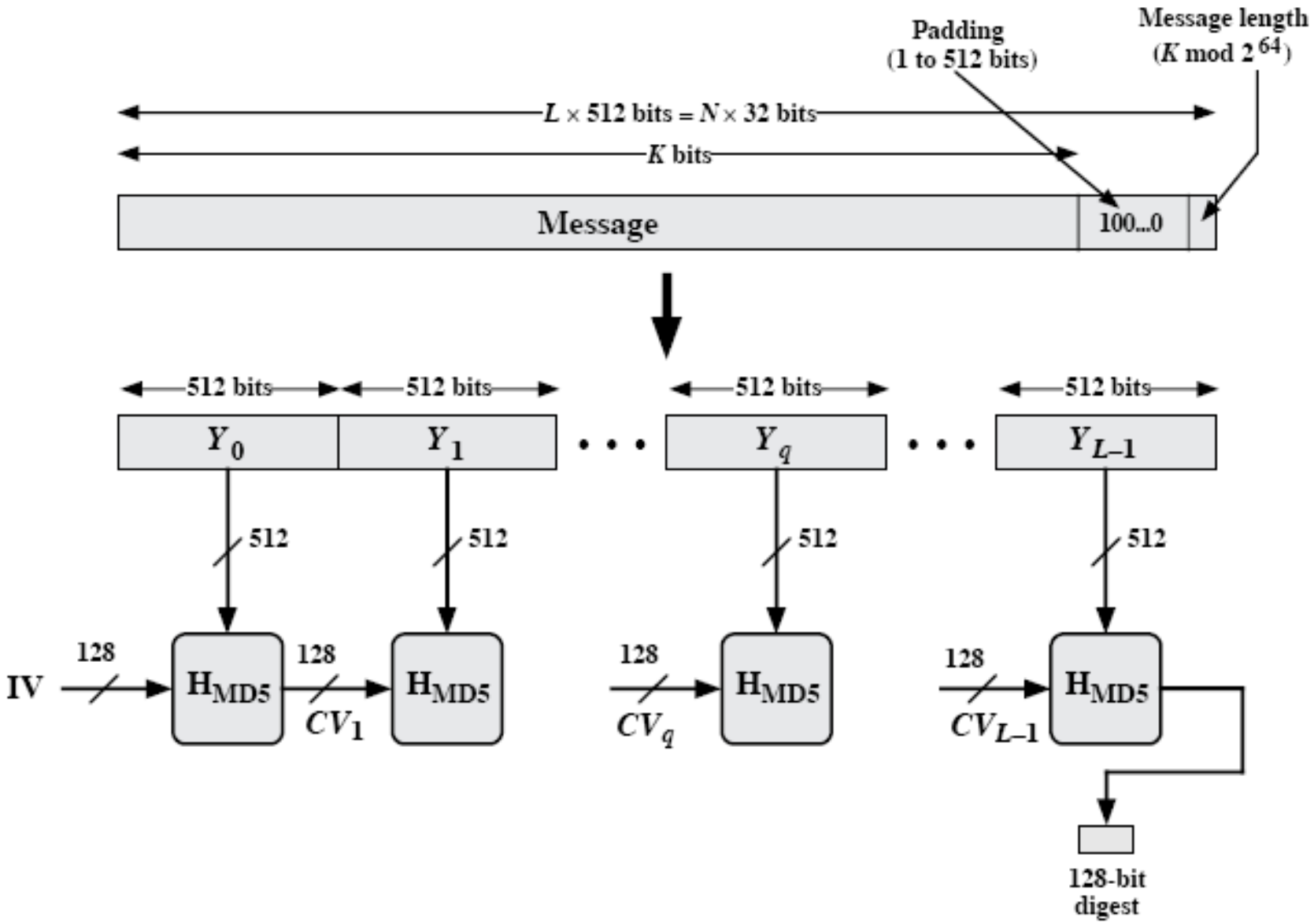


Figure 12.1 Message Digest Generation Using MD5

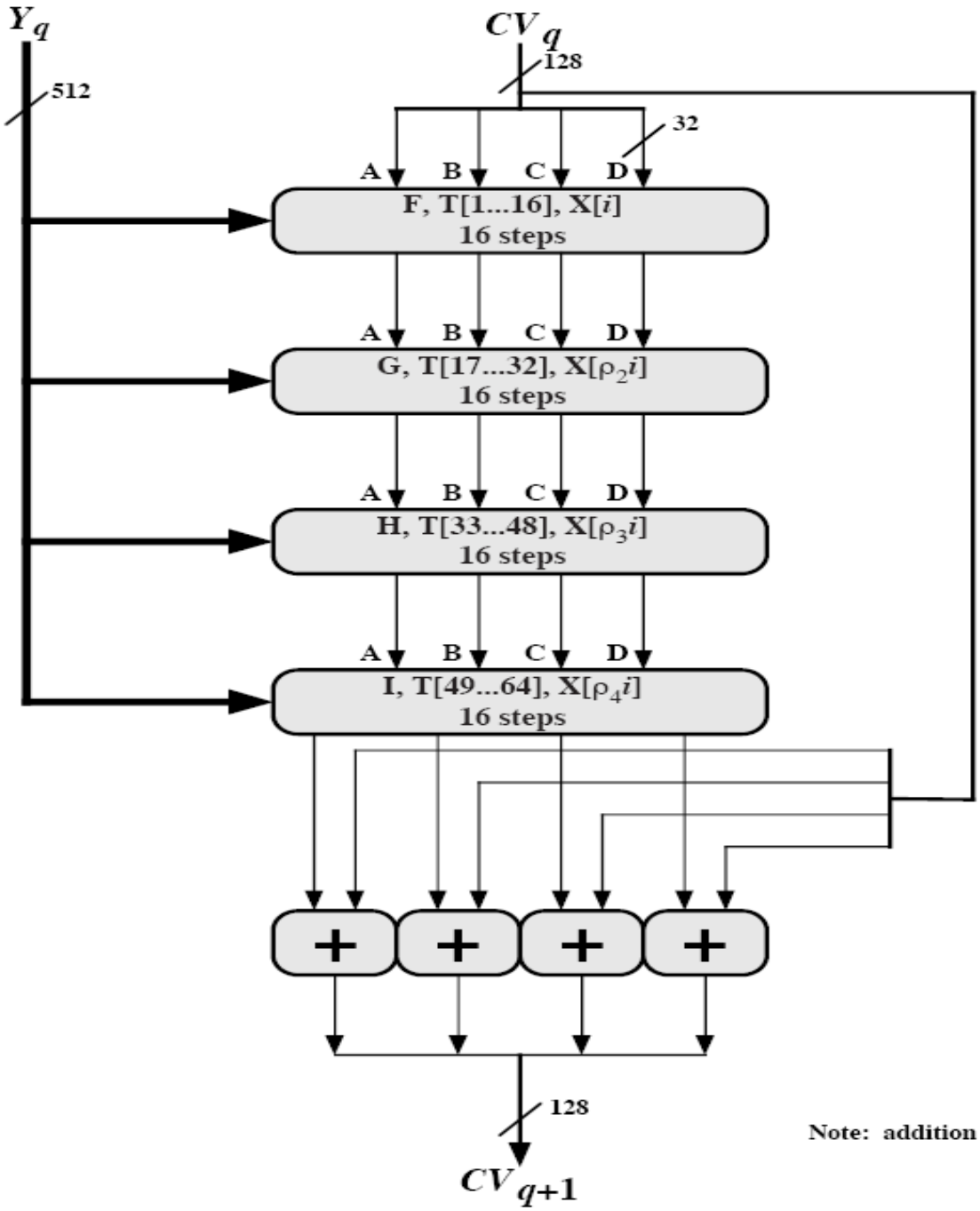


Figure 12.2 MD5 Processing of a Single 512-bit Block

MD5 Message Digest Algorithm

- Step 1: Append Padding bits
  - Padding bits are added to the original message such that message length is an integer multiple of 448 (512-64)
  - Padding bits are added in a way that a single 1 bit is followed by necessary number of 0 bits.

MD5 Message Digest Algorithm

- Step 2: Append Message Length
  - A 64-bit representation of the length of the original message is appended to the result of step 1.
  - Now the total message bits are in integer multiple of 512.
    - (original message + padding bits + 64-bit message length)

MD5 Message Digest Algorithm

- Step 3: Initialize MD Buffer
  - A 128-bit buffer is used to hold intermediate and final results of the hash function.
  - The buffer can be represented as four 32-bit registers (A, B, C, D).
  - The initialization values in each register appears as
    - Register A: Word A: 01 23 45 67
    - Register B: Word B: 89 AB CD EF
    - Register C: Word C: FE DC BA 98
    - Register D: Word D: 76 54 32 10

MD5 Message Digest Algorithm

- Step 4: Process message in 512-bit blocks
  - The module H_{MD5} functions in four rounds
  - Each round consists of 16 steps for processing
  - The two inputs are CV_q chaining variable value of q^{th} step (initially, Initial Value IV of ABCD buffer) and Y_q (q^{th} 512-bit block) and the output is CV_{q+1} (chaining variable)

MD5 Message Digest Algorithm

- Step 4: Process message in 512-bit blocks
  - Each round uses a different function, referred to as F, G, H and I
  - Each round takes as input the current 512-bit block (Y_q) and 128-bit buffer value and updates the contents of the buffer.
  - The output to the fourth round is added to the input to the first round (CV_q) to produce CV_{q+1}.

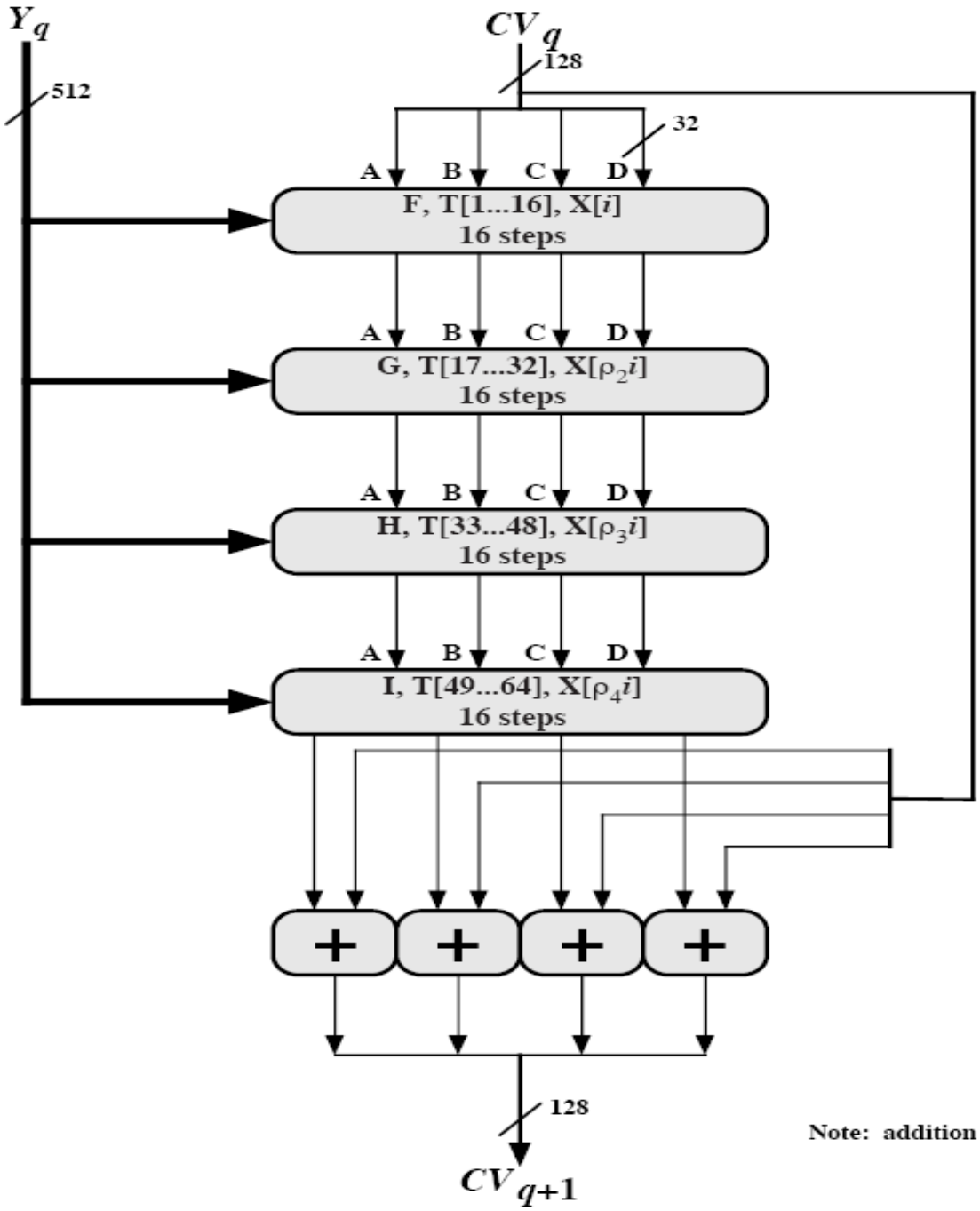


Figure 12.2 MD5 Processing of a Single 512-bit Block

MD5 Message Digest Algorithm

- Step 5: Output
  - After all L 512-bit blocks have been processed, the output from L^{th} stage is the 128-bit message digest.

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

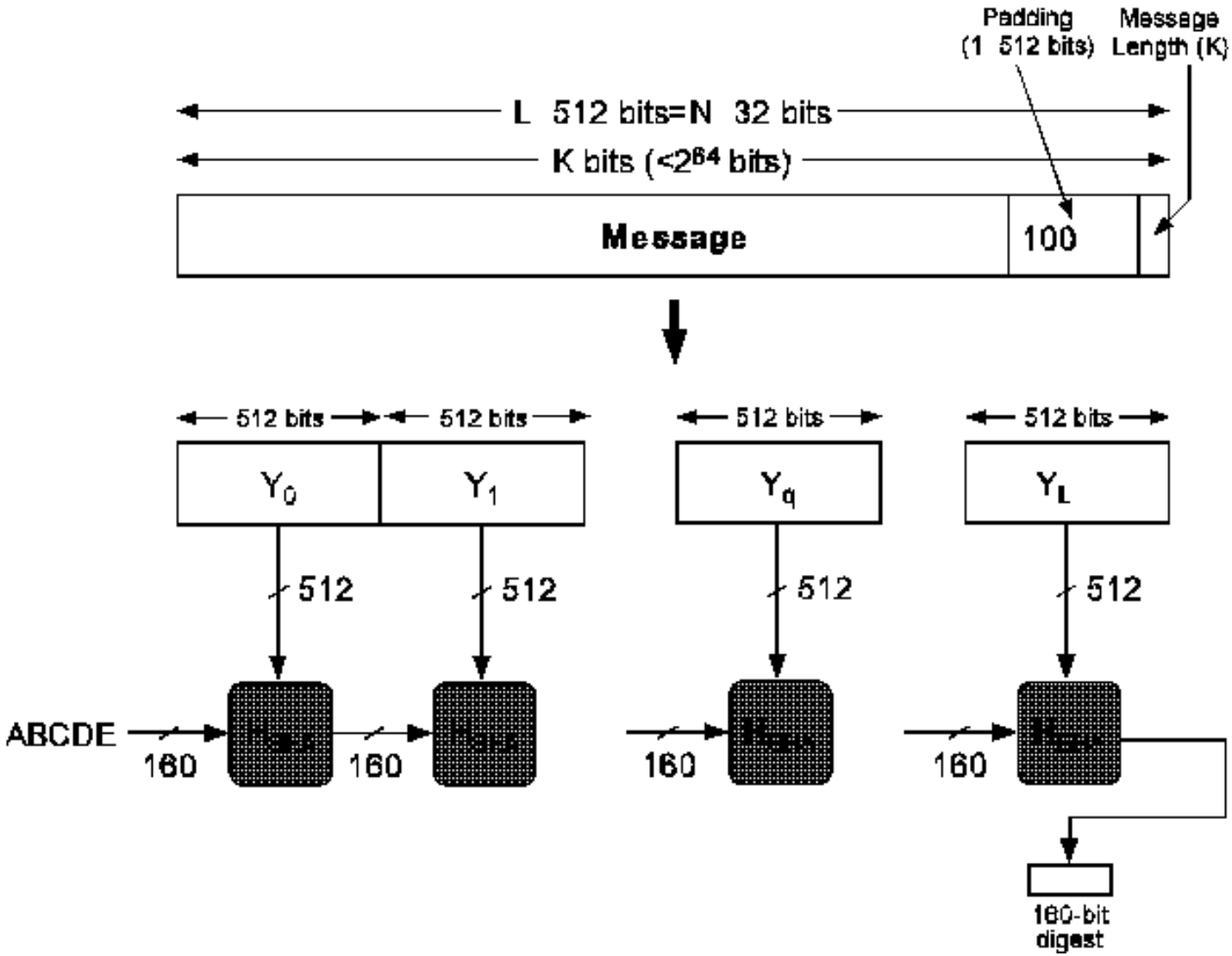
where
- IV = Initial Value of ABCD buffer
- Y_q = q^{th} 512-bit block
- L = total number of blocks in message
- CV_q = Chaining Variable
- RF_x = Round function of X
- MD = Message Digest
- SUM_{32} = Addition modulo 2^{32}

SHA-1 Secure Hash Algorithm

- 4 versions of SHA
  - SHA-1 : 160-bit hash value
  - SHA-256 : 256-bit hash value
  - SHA-384 : 384-bit hash value
  - SHA-512 : 512-bit hash value

- SHA-1 algorithm takes as input a message of maximum length < 2^{64} bits and produces as output a 160-bit message digest (hash code).

- The input message is processed in 512-bit blocks.


Padding
(1 512 bits)
Message
Length (K)
L 512 bits=N 32 bits
K bits (<2^64 bits)
Message
100
512 bits
512 bits
512 bits
512 bits
Y0
Y1
Yq
YL
512
512
512
512
ABCDE
160
160
160
160
160-bit
digest

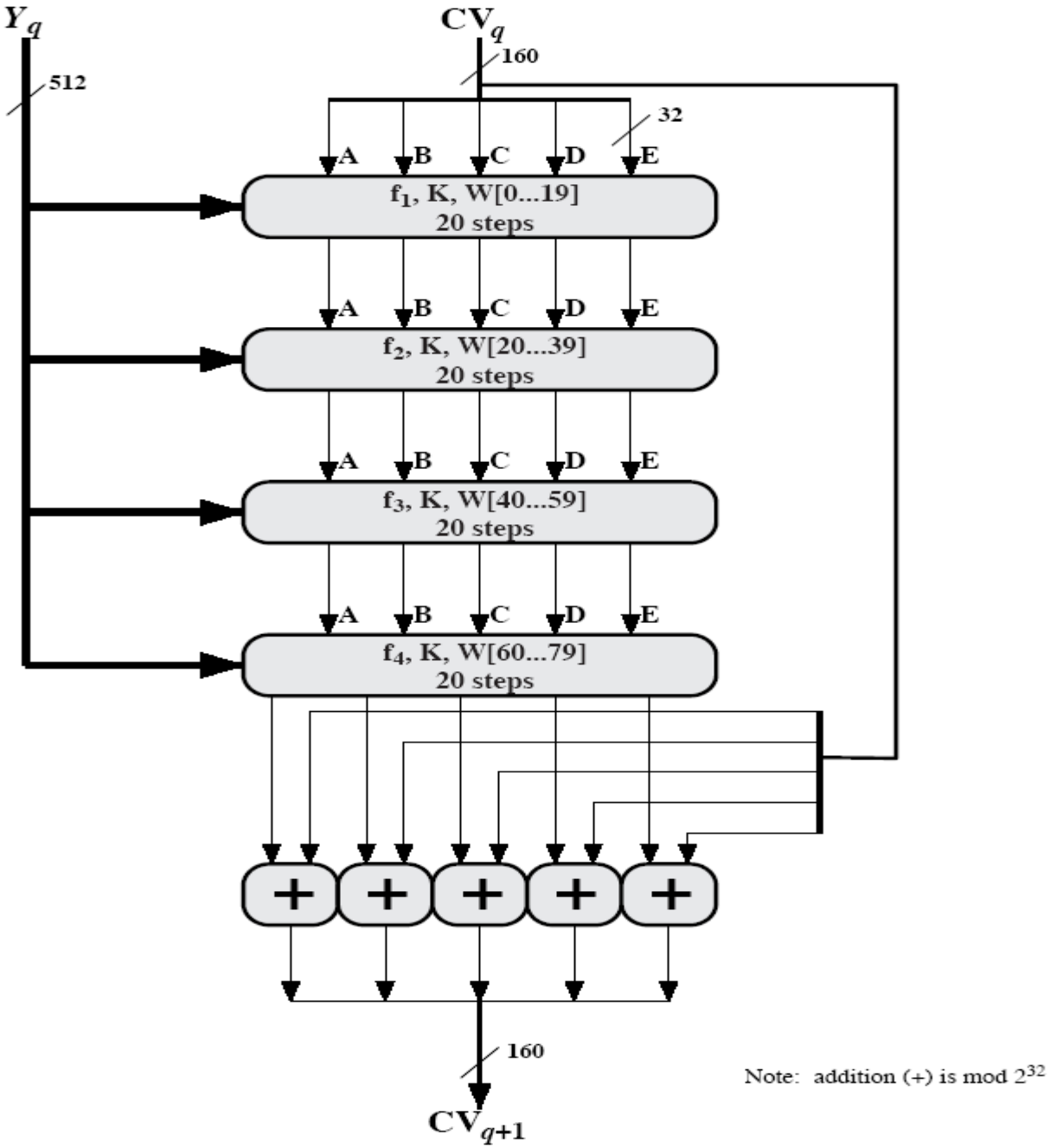


Figure 12.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

SHA-1 Secure Hash Algorithm

- Step 1: Append Padding bits
  - Padding bits are added to the original message such that message length is an integer multiple of 448 (512-64)
  - Padding bits are added in a way that a single 1 bit is followed by necessary number of 0 bits.

SHA-1 Secure Hash Algorithm

- Step 2: Append Message Length
  - A 64-bit representation of the length of the original message is appended to the result of step 1.
  - Now the total message bits are in integer multiple of 512.
    - (original message + padding bits + 64-bit message length)

SHA-1 Secure Hash Algorithm

- Step 3: Initialize MD 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).
  - The initialization values in each register appears as
    - Register A: Word A: 76 54 32 10
    - Register B: Word B: EF CD AB 89
    - Register C: Word C: 98 BA DC FE
    - Register D: Word D: 10 32 54 76
    - Register E: Word E: C3 D2 E1 F0

SHA-1 Secure Hash Algorithm

- Step 4: Process message in 512-bit blocks
  - SHA-1 algorithm consists of a module that uses four rounds for processing.
  - Each round consists of 20 steps for processing
  - The two inputs are CV_q chaining variable of q^{th} step (initially, Initial Value IV of ABCDE buffer) and Y_q (q^{th} 512-bit block) and the output is CV_{q+1} (chaining variable)

SHA-1 Secure Hash Algorithm

- Step 4: Process message in 512-bit blocks
  - Each round uses different function, referred to as f_1, f_2, f_3 and f_4
  - Each round takes as input the current 512-bit block (Y_q) and 160-bit buffer value and updates the contents of the buffer.
  - The output to the fourth round is added to the input to the first round (CV_q) to produce CV_{q+1}.

SHA-1 Secure Hash Algorithm

- Step 5: Output
  - After all L 512-bit blocks have been processed, the output from L^{th} stage is the 160-bit message digest.

$$CV_0 = IV$$
$$CV_{q+1} = SUM_{32}[CV_q, ABCDE_q]$$
$$MD = CV_L$$

where IV = Initial Value of ABCDE buffer
Y_q = q^{th} 512-bit block
L = total number of blocks in message
CV_q = Chaining Variable
MD = Message Digest
$ABCDE_q$=output of last round
SUM_{32} = Addition modulo 2^{32}

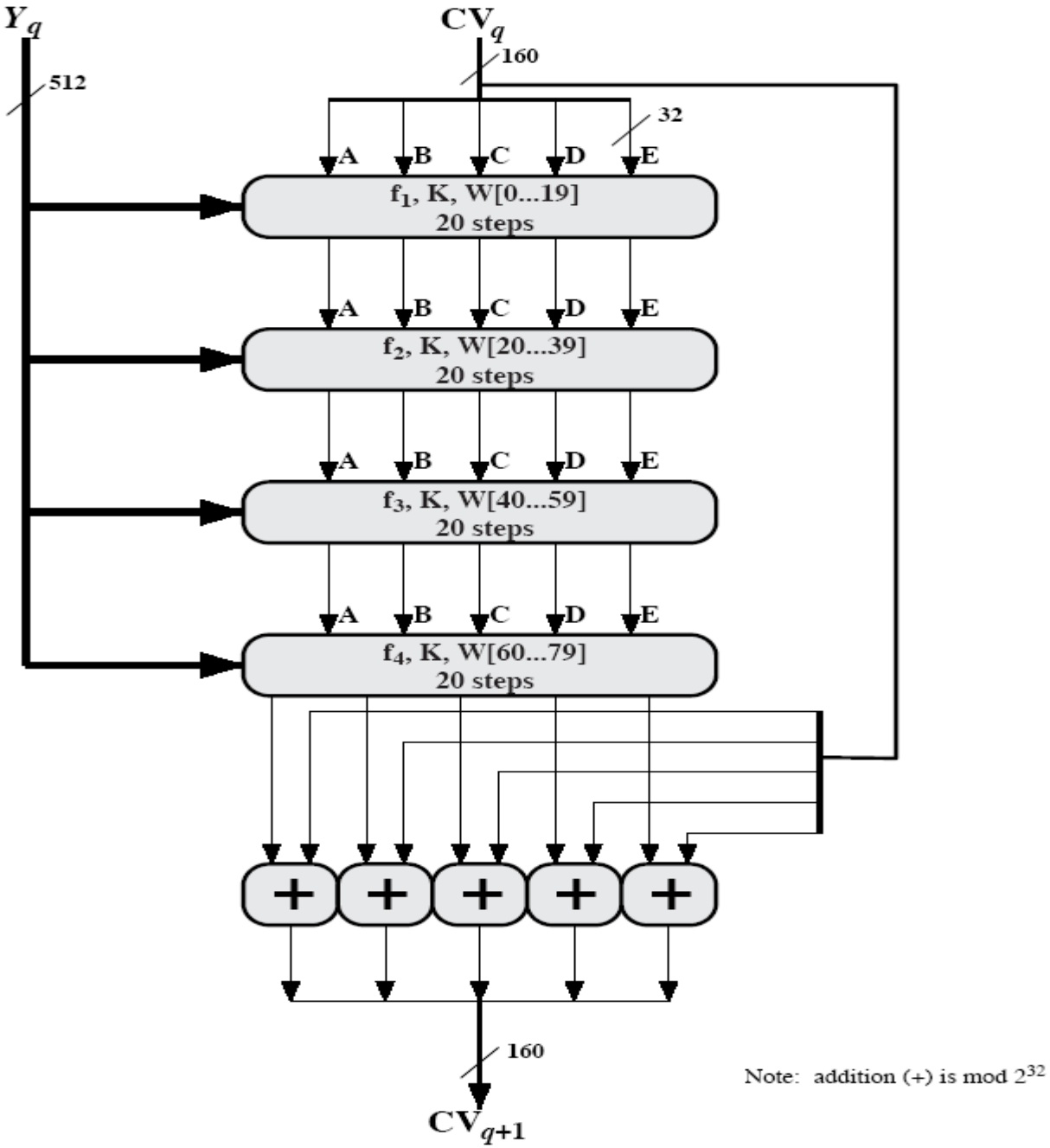


Figure 12.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

MAC based on Hash Functions: HMAC (Hash-based MAC)

- Hash-based Message Authentication Code (HMAC) is an encrypted message digest.

- Uses a shared secret key between two parties rather than public key methods.

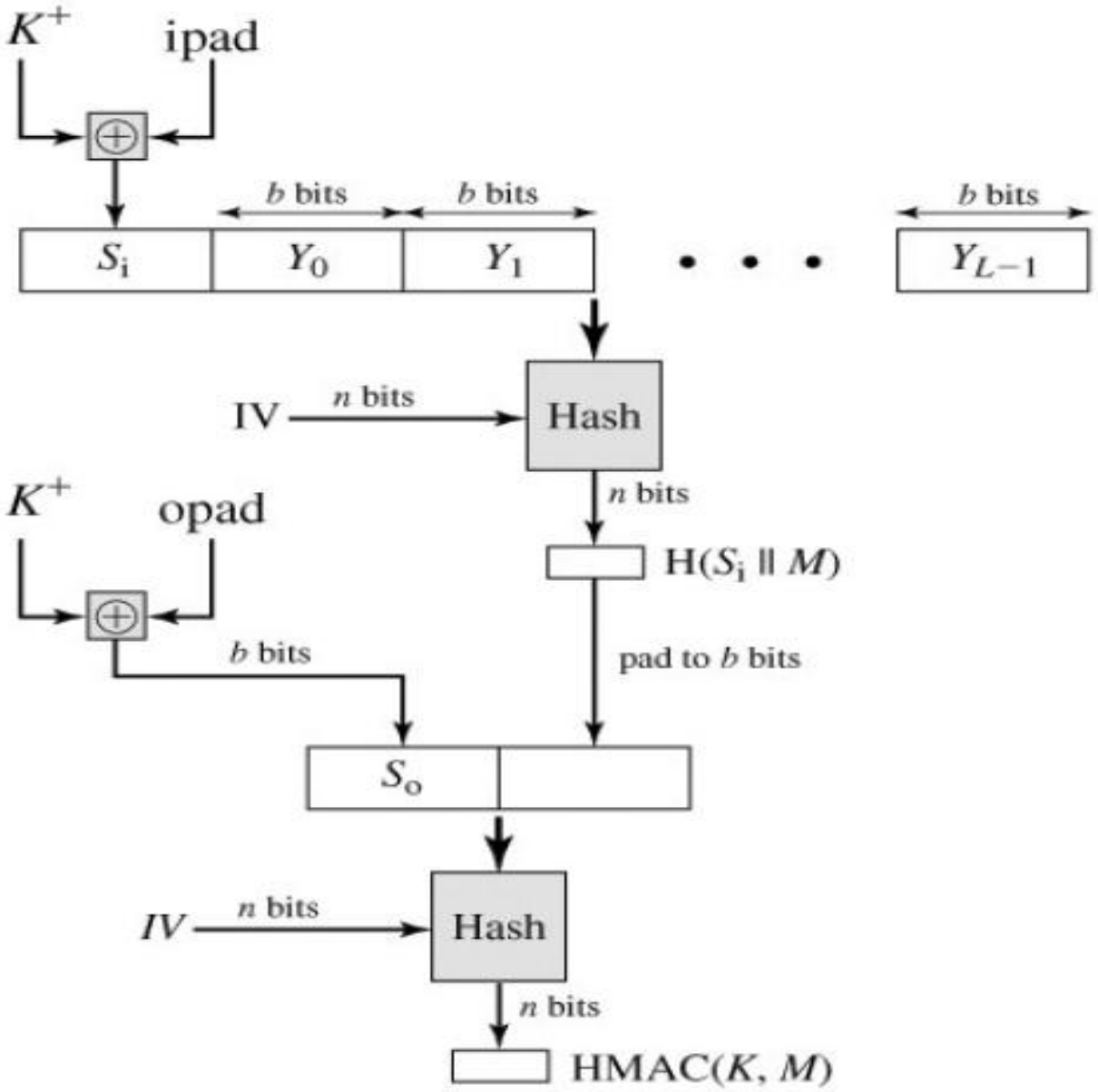
HMAC (Hash-based MAC)

- Define the following terms:
- H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
- IV = initial value input to hash function
- M = message input to HMAC
- Y_i = i^{th} block of M, 0 <= i <= (L-1)
- L = number of blocks in M
- b = number of bits in a block

HMAC (Hash-based MAC)

- n = length of hash code produced by embedded hash function
- K = secret key; recommended length is >= n
- K^+ = K padded with zeros on the left so that the result is b bits in length
- ipad = 00110110 (36 in hexadecimal) repeated b/8 times
- opad = 01011100 (5C in hexadecimal) repeated b/8 times

HMAC (Hash-based MAC)



HMAC (Hash-based MAC)

- Then HMAC can be expressed as follows:

$$\mathrm{HMAC}(K,M) = H[(K^+ \oplus \mathrm{opad}) \| H[(K^+ \oplus \mathrm{ipad}) \| M]]$$

1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and b = 512 then K will be appended with 44 zero bytes).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o
6. Append the hash result from step 4 to S_o
7. Apply H to the stream generated in step 6 and output the result.

HMAC (Hash-based MAC)

- A more efficient implementation of HMAC is possible.

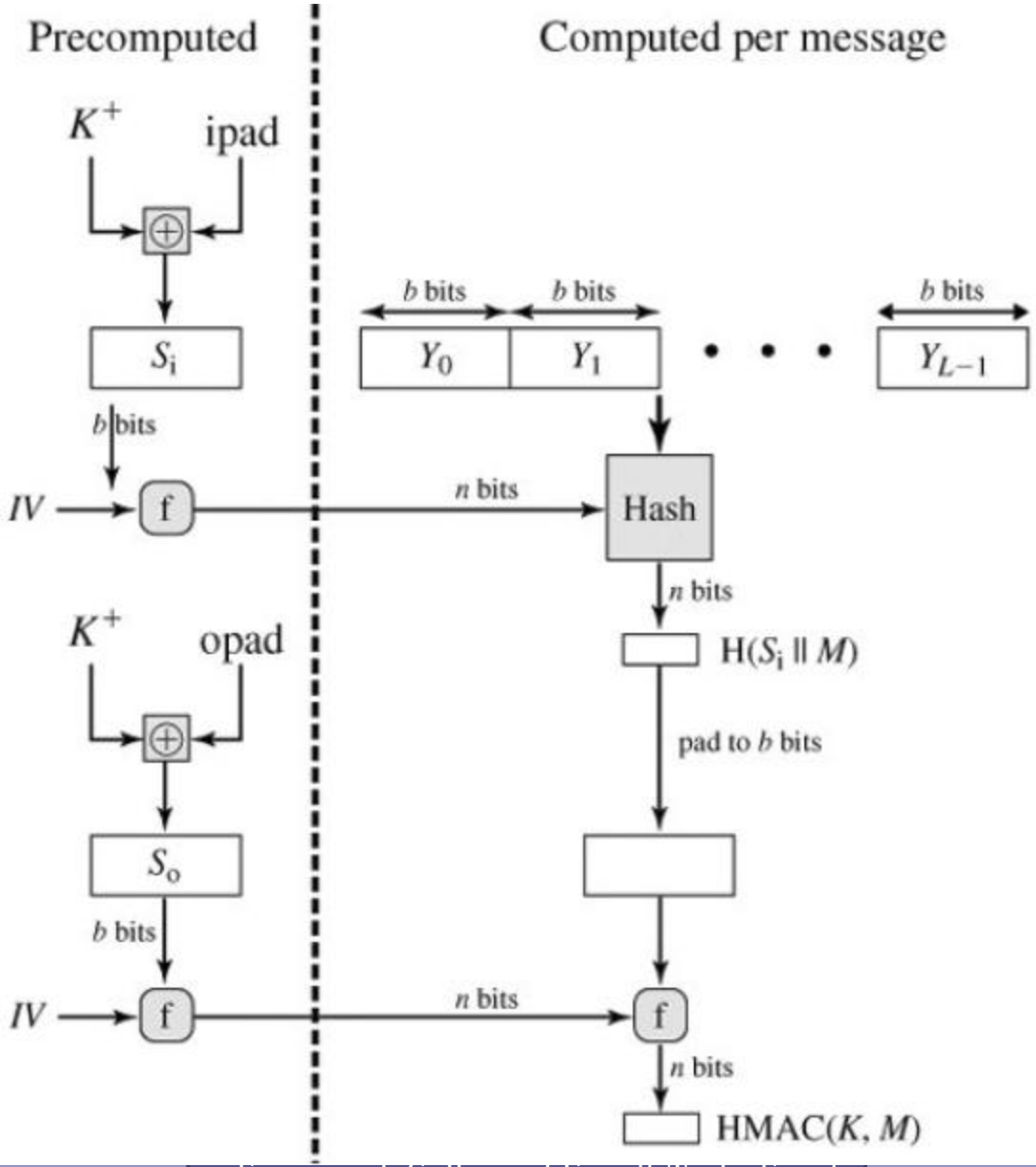
- Two quantities are pre computed:

$$f(IV, (K^+ \oplus ipad))$$

$$f(IV, (K^+ \oplus opad))$$

- 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 (Hash-based MAC)



Security of HMAC

- Attacking HMAC requires
  - Brute-force attack on a key used
  - Birthday attack

- Choose a hash function based on speed versus security constraints.

END