Message Authentication
and
Hash Functions

Security Requirements

1. Disclosure – Release of message to any person or processes not possessing cryptographic key.
2. Traffic analysis – discovery of the pattern of traffic between parties.
3. Masquerade - Insertion of a messages into the network from fraudulent source.
4. Content modification – changes to the content of message
5. sequence modification – any modifications to a sequence of messages between parties including insertion, deletion and reordering
6. Timing modification – delay or replay of messages
7. Source repudiation – denial of transmission of message by source
8. Destination repudiation - denial of receipt of message by destination

1 to 2 – Encryption

2 to 6 – Authentication +

7 to 8 – Digital Signatures +

Primitives → **Service ↓**	**Encryption**	**Hash Function**	**MAC**	**Digital Signature**
Confidentiality	Yes	No	No	No
Integrity	No	Sometimes	Yes	Yes
Authentication	No	No	Yes	Yes
Non Reputation	No	No	Sometimes	Yes

Message Authentication

- Most confusing area of n/w security
- message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- will consider the security requirements
- then three alternative functions used:
  - message encryption
  - message authentication code (MAC)
  - hash function

Message Authentication Functions

- **Message Encryption:**

  Ciphertext of entire message serves as its authenticator

- **Message Authentication Code (MAC):**

  A public function of the message and a secrete key that produces fixed length value serves as the authenticator

- **Hash Function:**

  A public function that maps a message of any length into a fixed hash value, which serves as the authenticator

- **Digital Signature:**

  A public function that calculates a fixed length bit pattern or value of the message of any length involving keys, which serves as the authenticator

Message Authentication Code (MAC)

- generated by an algorithm that creates a small fixed-sized block
  - depending on both message and some key
  - like encryption though ***need not be reversible***
- appended to message as a **signature/fingerprint**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

Message Authentication Code

- MAC is similar to encryption
- But need not be reversible as it is must for decryption
- MAC is many-to-one function
- Function domain consists of messages of some arbitrary length
- Function range consists of all possible MAC’s and all possible keys.
- For n-bit MAC : 2^n possible MAC’s
- For k-bit key : 2^k possible key’s

Message Authentication Code

- e.g. 100-bit message & 10 bit MAC → 2^{100} different messages but 2^{10} different MAC's

- on average, each MAC value is generated by total of

$2^{100}/2^{10} = 2^{90}$ different messages.

- for 5-bit key ; 2^5 =32 different mappings from the set of messages to the set of MAC values.

Internal and External Error Control

F = FCS – Frame check sequence

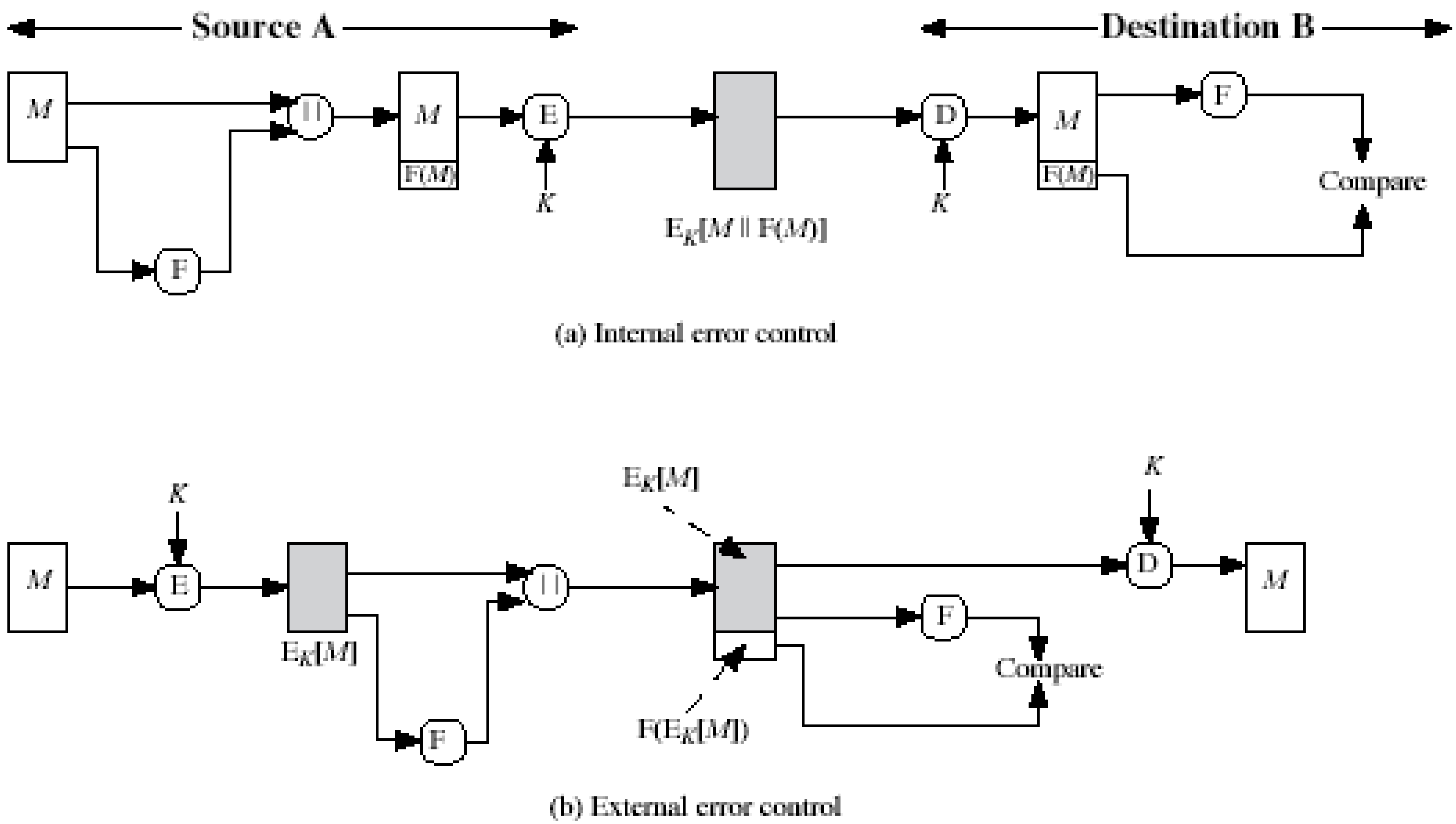
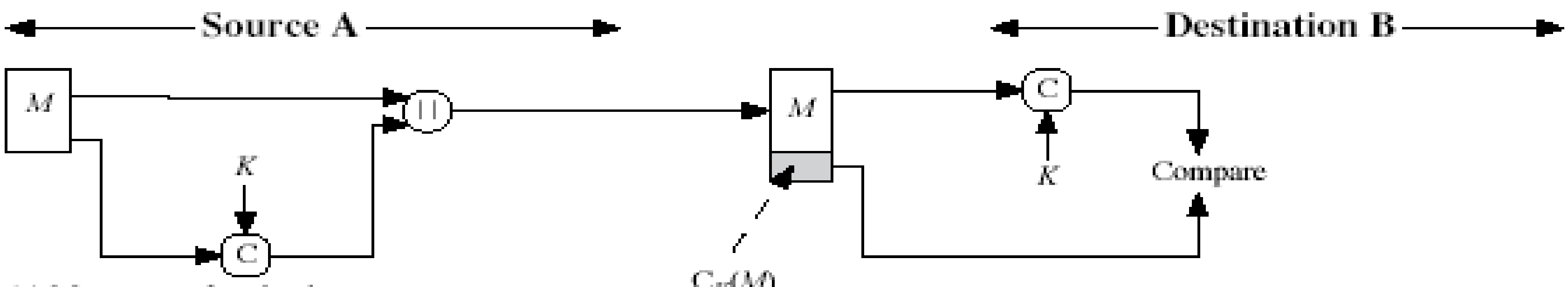


Figure 11.2 Internal and External Error Control

Message Authentication Code



(a) Message authentication

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC.

2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.

3. If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

Message Authentication Code

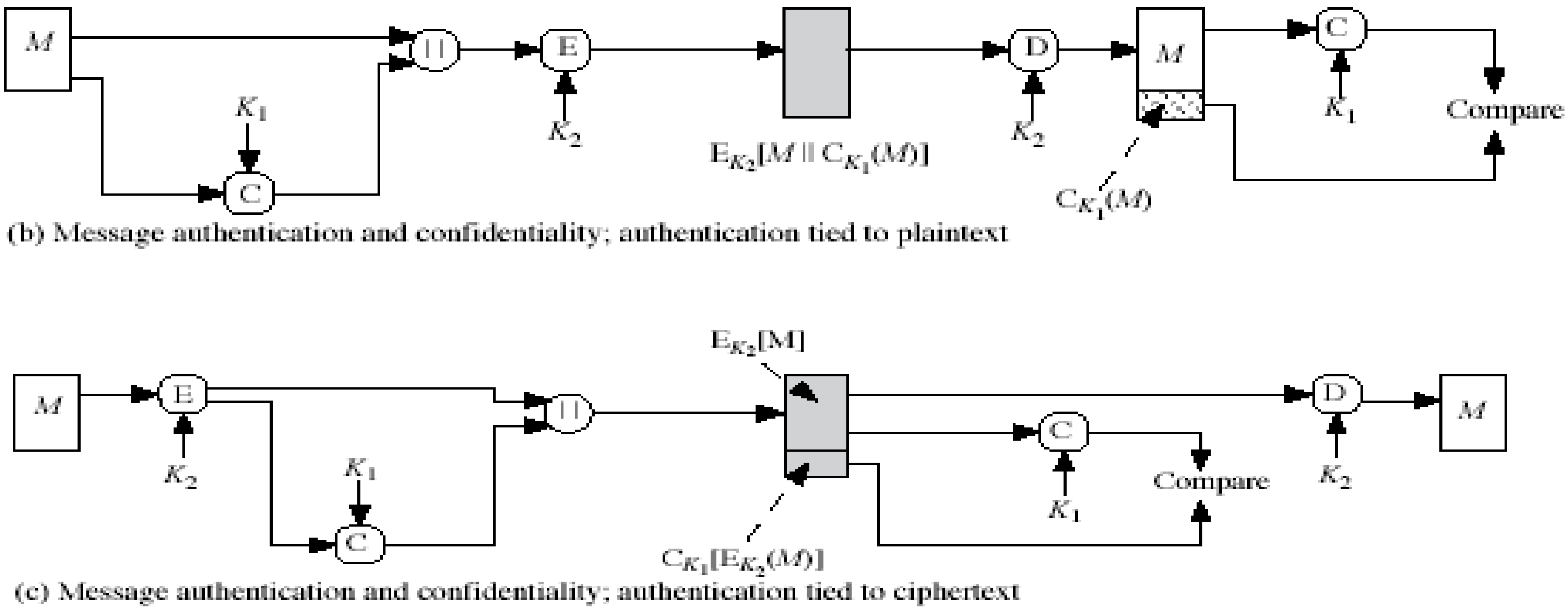


Figure 11.4 Basic Uses of Message Authentication Code (MAC)

Confidentiality can be provided by performing message encryption either after (Figure 1) or before (Figure 2) the MAC algorithm. In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver. However Fig. 1 is preferable.

Message Authentication Code

$A \rightarrow B: M \,\|\, C_K(M)$

- Provides authentication
  - Only A and B share K

(a) Message authentication

$A \rightarrow B: E_{K_2}\left[M \,\|\, C_{K_1}(M)\right]$

- Provides authentication
  - Only A and B share K_1
- Provides confidentiality
  - Only A and B share K_2

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

$A \rightarrow B: E_{K_2}[M] \,\|\, C_{K_1}\left(E_{K_2}[M]\right)$

- Provides authentication
  - Using K_1
- Provides confidentiality
  - Using K_2

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

Message Authentication Code

Applications

- Same message broadcast to number of destinations

- In an exchange in which one side has a heavy load and can not afford time to decrypt all incoming messages-:

→ random /selective checking

- Authentication of computer program, always executed without wasting resources time in decryption

- Architectural flexibility due to separation of authentication (application layer) and confidentiality (transport layer)

- prolong period of protection after decryption within home network

Message Authentication Codes

- as shown the MAC provides confidentiality
- can also use encryption for secrecy
  - generally use separate keys for each
  - can compute MAC either before or after encryption
  - is generally regarded as better done before
- why use a MAC?
  - sometimes only authentication is needed
  - sometimes need authentication to persist longer than the encryption (eg. archival use)
- note that a MAC is not a digital signature

MAC Properties

- a MAC is a cryptographic checksum

  $$\mathtt{MAC} = \mathtt{C}_{\mathtt{K}}(\mathtt{M})$$

  - condenses a variable-length message M
  - using a secret key K
  - to a fixed-sized authenticator
- is a many-to-one function
  - potentially many messages have same MAC
  - but finding these needs to be very difficult

Requirements for MACs

- Taking into account the types of attacks, need the MAC satisfy the following:

1. If an opponent observes M and $C_K(M)$, it should be **computationally infeasible** for the opponent to construct a message M' such that $C_K(M') = C_K(M)$ – ***even not knowing the key***

2. $C_K(M)$ should be **uniformly distributed** in the sense that for randomly chosen messages, M and M', the probability that $C_K(M) = C_K(M')$ is $\mathbf{2^{-n}}$, where n is the number of bits in the MAC. – ***not knowing the key but having access to MAC***

3. Let M' be equal to some **known transformation** on M.
   That is, $M' = f(M)$. For example, f may involve inverting one or more specific bits. In that case, $Pr[C_K(M) = C_K(M')] = 2^{-n}$. ***– known weak spots to match new with old MAC***

- For encryption security lies with key size
- k bit key the cipher text- only attack will require $2^{(k-1)}$ attempts.
- However with MAC entirely different method to find the key
- e.g. if confidentiality is not employed

- opponent has access to plaintext and associated MAC

- For k > n i.e. For key_size > MAC_size

- ROUND 1 - Given: M1, MAC1 = C_k(M1)

Compute: $MAC_i = Ck_i$ (M1) for all 2^k keys

But number of matches = $2^k/2^n = 2^{(k-n)}$

- ROUND 2 Given: M2, MAC2 = Ck(M2)

Compute: $MAC_i = Ck_i$ (M2) for remaining $2^{(k-n)}$ keys

But number of matches = $2^{(k-2xn)}$ ....

- On average R rounds are required if $k = R \times n$
- for 80 bit key and 32 bit MAC
- Round 1 will produce about $2^{(80-32)} = 2^{48}$ possible keys
- Round 2 will produce about $2^{(80-2 \times 32)} = 2^{16}$ possible keys
- Thus, Round 3 should produce only a single key which must be used by the sender.

Using Symmetric Ciphers for MACs

- can use any block cipher chaining mode and use final block as a MAC
- **Data Authentication Algorithm (DAA)** is a widely used MAC based on DES-CBC
  - using IV=0 and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final block as the MAC
    - or the leftmost M bits ($16 \leq M \leq 64$) of final block

    as DAC (Data Authentication Code)
- but final MAC is now too small for security

Using Symmetric Ciphers for MACs

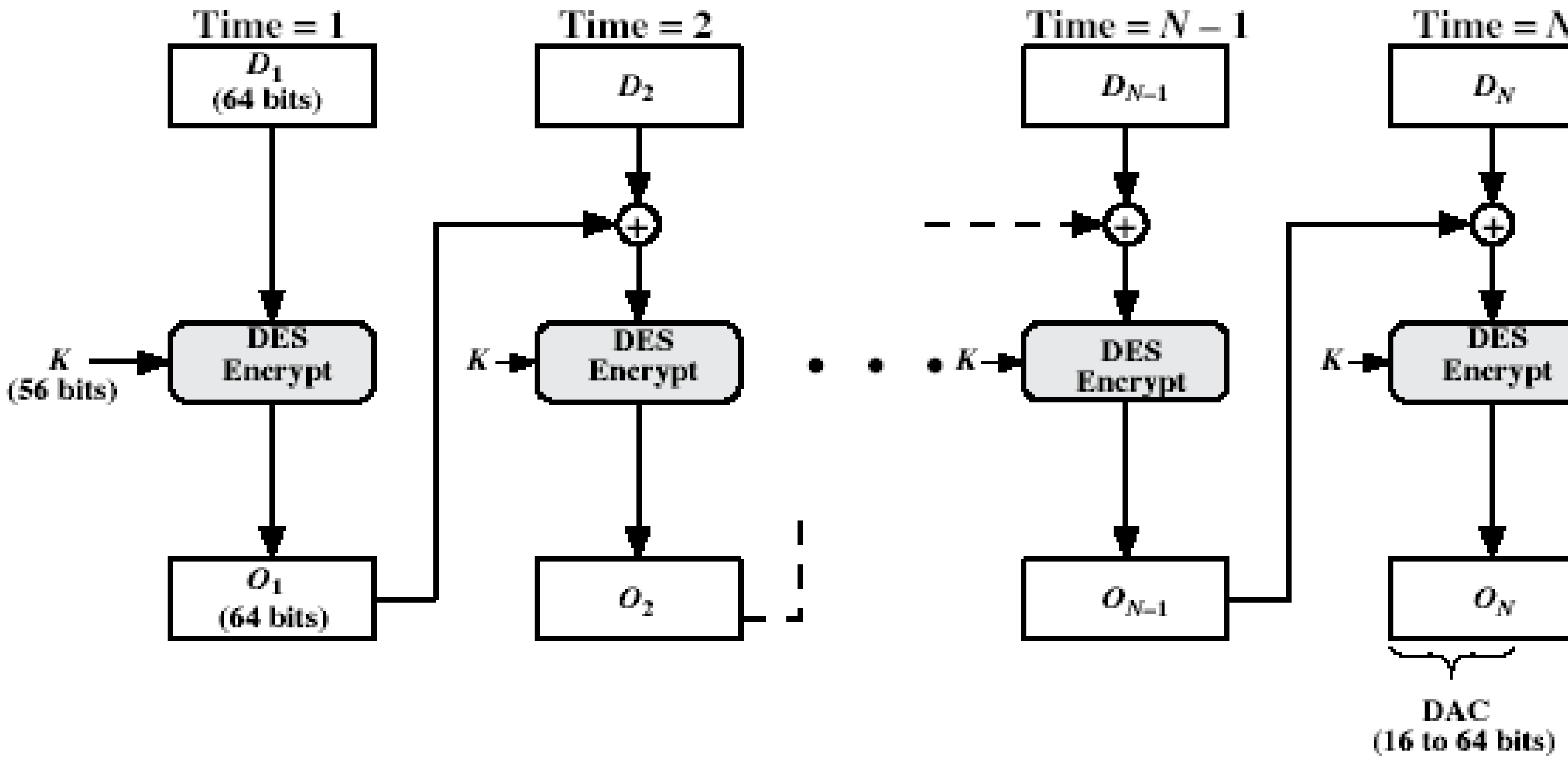


Figure 11.6 Data Authentication Algorithm (FIPS PUB 113)

Hash Functions

- condenses arbitrary message to fixed size
- usually assume that the hash function is public and not keyed
- hash used to detect changes to message
- can use in various ways with message
- most often to create a digital signature

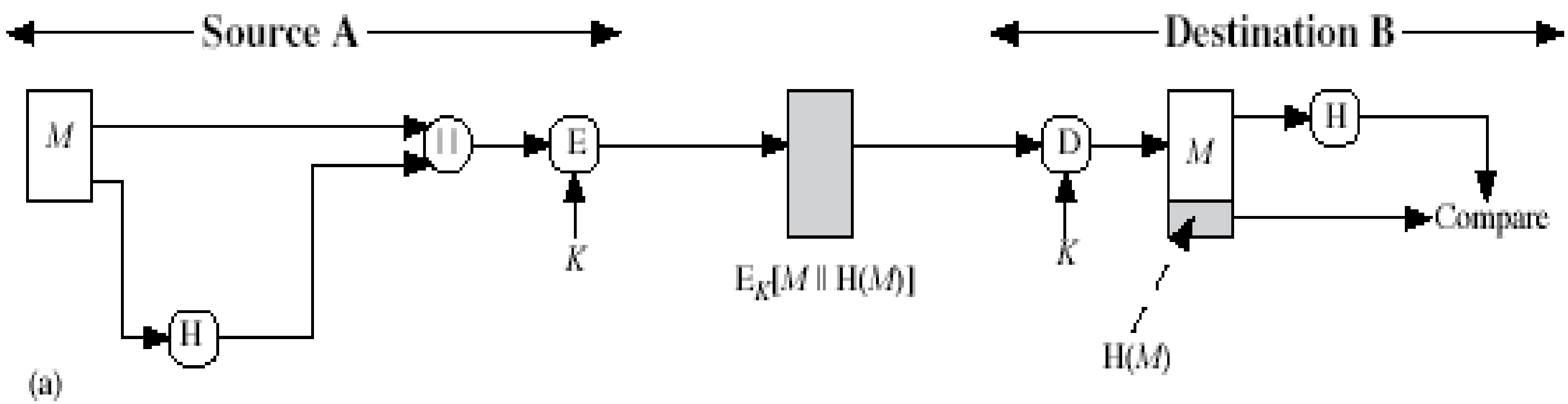
Hash Functions

- A variation on the message authentication code is the one-way hash function.

- As with the message authentication code, a hash function accepts a variable-size message *M* as input and produces a fixed-size output, referred to as a hash code *H(M).*

- Unlike MAC, a hash code does not use a key but is a function only of the input message.

Hash Functions

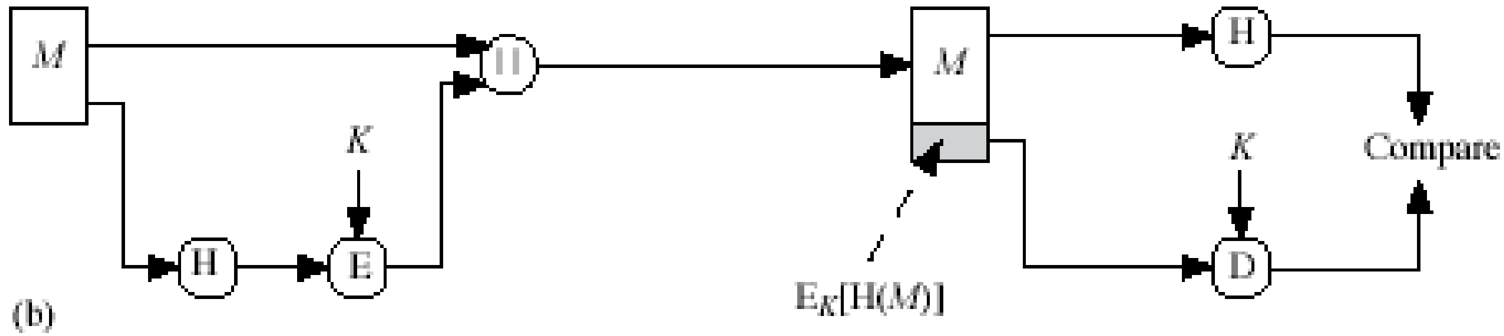


•The message plus concatenated hash code is encrypted using symmetric encryption.(identical in structure to the internal error strategy)

•The same line of reasoning applies: Because only A and B share the secret key, the message must have come from A and has not been altered.

•Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

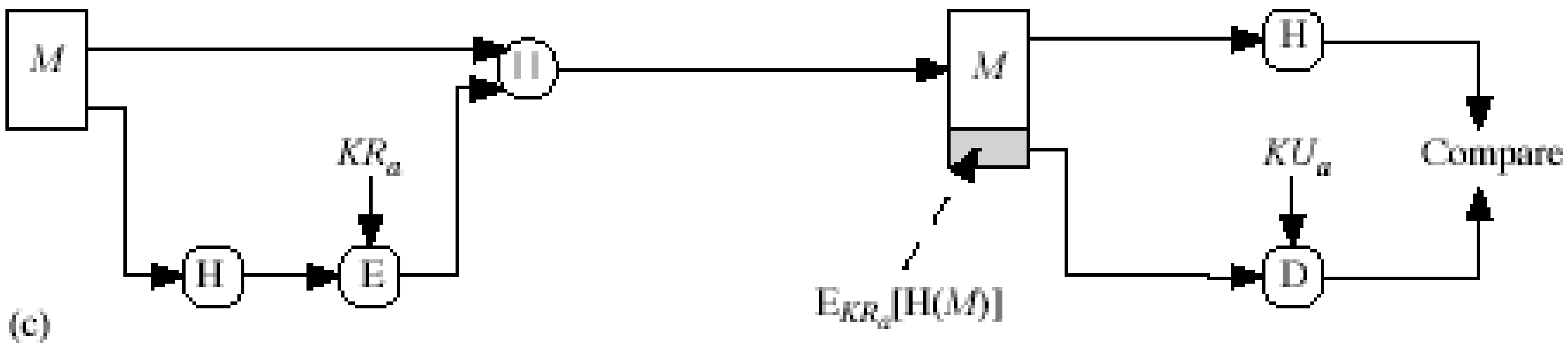
Hash Functions



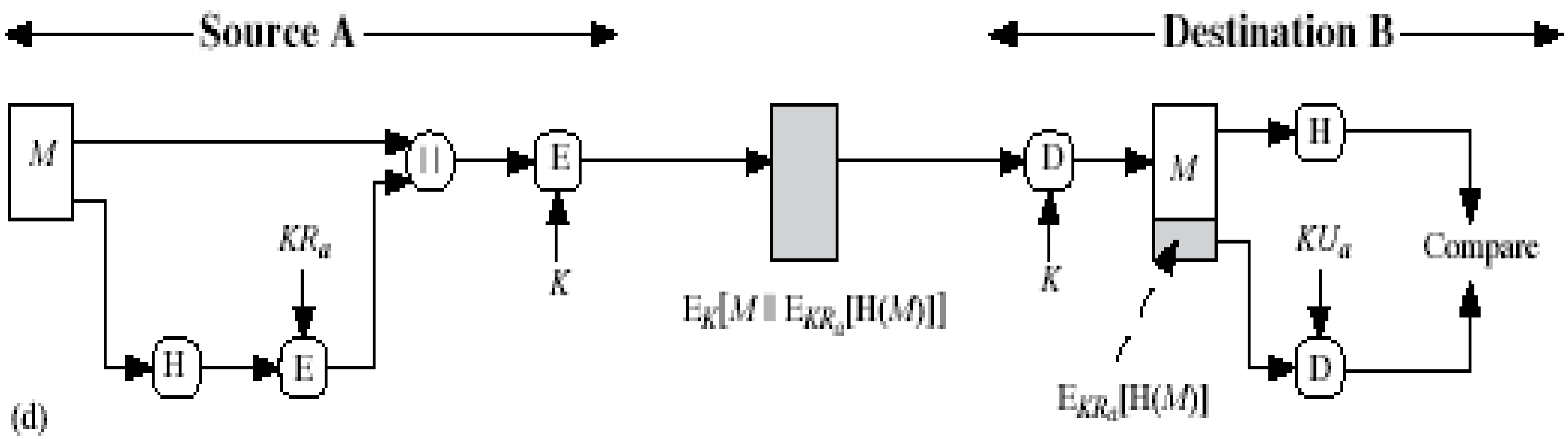
- Only the hash code is encrypted, using symmetric encryption.

- Reduces, the processing burden for those applications that do not require confidentiality.

Hash Functions



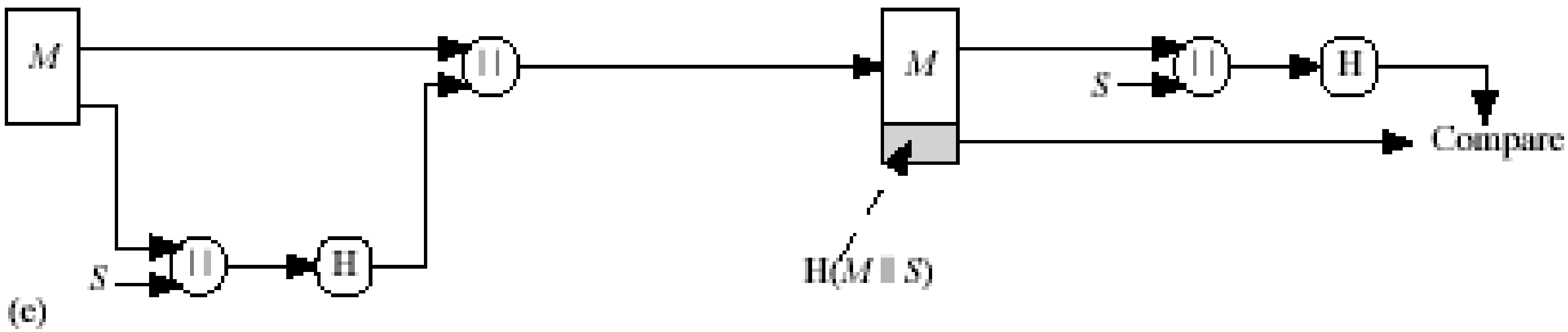
- Only the hash code is encrypted, using public-key encryption and using the sender's private key.
- This provides authentication. It also provides a digital signature, because only the sender could have produce the encrypted hash code.
- This is the essence of the digital signature technique.



• If confidentiality as well as a digital signature is desired, then the message plus the public-key-encrypted hash code can be encrypted using a symmetric secret key.

•This is a common technique.

Hash Functions



•This technique uses a hash function but no encryption for message authentication.

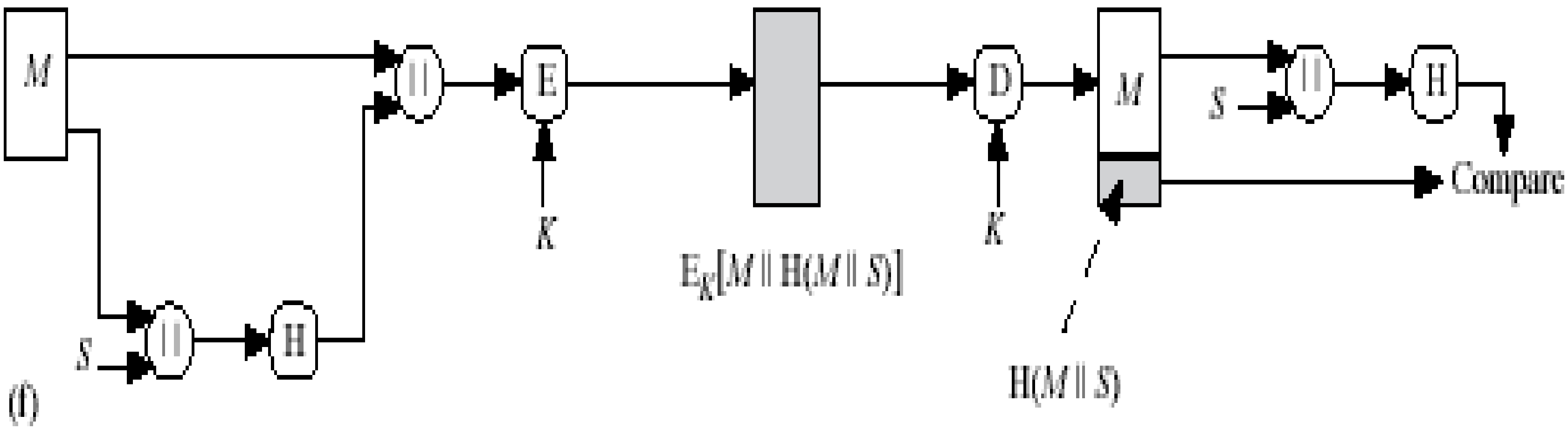
•The technique assumes that the two communicating parties share a common secret value S.

• A computes the hash value over the concatenation of M and S and appends the resulting hash value to M.

• Since B possesses S, it can recompute the hash value to verify.

• Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

Hash Functions



Confidentiality can be added to the approach of (e) by encrypting the entire message plus the hash code.

Table 11.3 Basic Uses of Hash Function H (see Figure 11.5)

$A \rightarrow B: E_K[M \parallel H(M)]$ •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)

Requirements for 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,* making both hardware and software implementations practical.

4. For any given value *h,* it is **computationally infeasible** to find *x* such that *H(x)*= *h.* This is sometimes referred to in the literature as the **one-way property.**

5. For any given block *x,* it is **computationally infeasible** to find *y* ≠ x with 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

- are several proposals for simple functions based on XOR of message blocks
- not secure since can manipulate any message and either not change hash or change hash also
- need a stronger cryptographic function

Simple Hash Functions

Simple hash function can be expressed as -

$C_i = b_{i1}$ Ø b_{i2} ø . . . ø b_{im}

where

Ci = ith bit of the hash code, $1 \le i \le n$,

m = number of n-bit blocks in the input

b_{ij} = ith bit in jth block

Ø= XOR operation

	bit 1	bit 2	• • •	bit n
block 1	b_{11}	b_{21}		b_{n1}
block 2	b_{12}	b_{22}		b_{n2}
	• • •	• • •	• • •	• • •
block m	b_{1m}	b_{2m}		b_{nm}
hash code	C_1	C_2		C_n

Figure 11.7 Simple Hash Function Using Bitwise XOR

Simple Hash Functions

A simple improvement 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.

This has the effect of "randomizing" the input more completely and overcoming any regularities that appear in the input.

Simple Hash Functions

- A technique originally proposed by the National Bureau of Standards used the simple XOR applied to 64-bit blocks of the message and then an encryption of the entire message that used the cipher block chaining (CBC) mode.

- Scheme is as follows:

  Given a message consisting of a sequence of 64-bit blocks X_1, X_2, . . . , X_N, define the hash code C as the block-by-block XOR of all blocks and append the hash code as the final block:

  $$C = X_{N+1} = X_1 \text{ø} X_2 \text{ø} \ldots \text{ø} X_N$$

- *Next, encrypt the entire message plus hash code, using CBC mode to produce the encrypted message $Y_1, Y_2, \ldots, Y_{N+1}$*

Block Ciphers as Hash Functions

- A number of proposals have been made for hash functions based on using a cipher block chaining technique, but without the secret key.

- One of the first such proposals was that of Rabin.

- Divide a message *M* into fixed-size blocks $M_1, M_2, \ldots, M_N$ and use a symmetric encryption system such as DES to compute the hash code G as follows

$$H_0 = \text{initial value}$$
$$H_i = E_{mi}[H_{i-1}]$$
$$G = H_N$$

- Similar to the CBC technique, but in this case there is no secret key.

Hash Functions & MAC Security

- like block ciphers have:
- **brute-force** attacks exploiting
  - strong collision resistance hash have cost $2^{m/2}$
    - have proposal for h/w MD5 cracker
    - 128-bit hash looks vulnerable, 160-bits better
  - MACs with known message-MAC pairs
    - can either attack key space or MAC
    - at least 128-bit MAC is needed for security

Hash Functions & MAC Security

- **cryptanalytic attacks** exploit structure
  - like block ciphers want brute-force attacks to be the best alternative
- have a number of analytic attacks on iterated hash functions

$$CV_0 = IV = \text{initial } n - \text{bit value}$$
$$CV_i = f[CV_{i-1}, M_i];$$
$$H(M)=CV_N$$

  - typically focus on collisions in function f
  - like block ciphers is often composed of rounds
  - attacks exploit properties of round functions