MODULE IV

- Authentication requirements- Authentication functions- Message authentication codes- Hash functions- SHA -1, MD5, Security of Hash functions and MACs- Authentication protocols-Digital signatures-Digital signature standards.

AUTHENTICATION REQUIREMENTS

- Disclosure
- Traffic analysis
- Masquerade
- Content modification
- Sequence modification
- Timing modification
- Source repudiation
- Destination repudiation

Disclosure

- Release of message contents to any person or process not possessing the appropriate cryptographic key.

Traffic analysis

- Discovery of the pattern of traffic between parties.
- In a connection oriented application, the frequency and duration of connections could be determined.
- In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.

Masquerade:

- Insertion of messages into the network from a fraudulent source.
- includes the creation of messages by an opponent that are purported to come from an authorized entity.
- Also included are fraudulent acknowledgments of message receipt or non receipt by someone other than the message recipient.

Content modification

- Changes to the contents of a message, including insertion, deletion, transposition, and modification.

Sequence modification

- Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

Timing modification

- Delay or replay of messages.
- In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed.
- In a connectionless application, an individual message (e.g:- datagram) could be delayed or replayed.

- **Source repudiation**
  - Denial of transmission of message by source.
- **Destination repudiation:**
  - Denial of receipt of message by destination.
  - Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered.
  - Message authentication may also verify sequencing and timeliness.

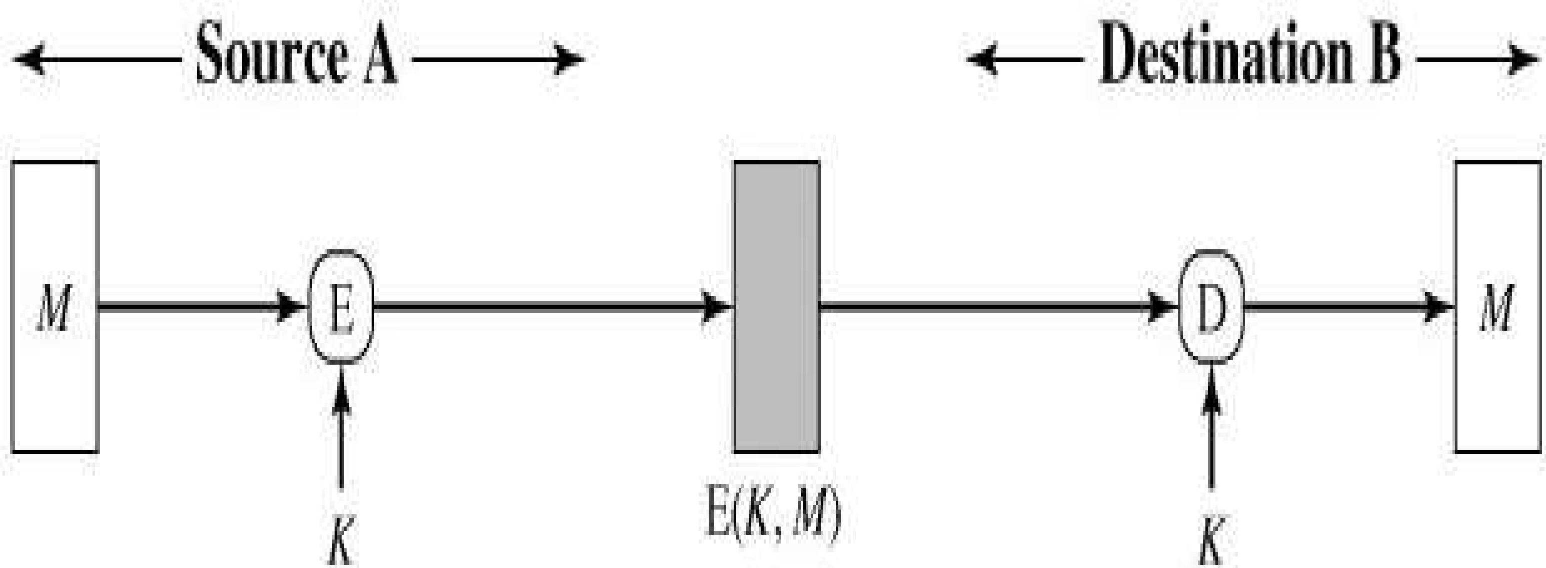
AUTHENTICATION FUNCTIONS

- The types of functions that may be used to produce an authenticator may be grouped into three classes
  - Message encryption
  - Message authentication code (MAC)
  - Hash function:

(1) **Message encryption -** The cipher text of the entire message serves as its authenticator.

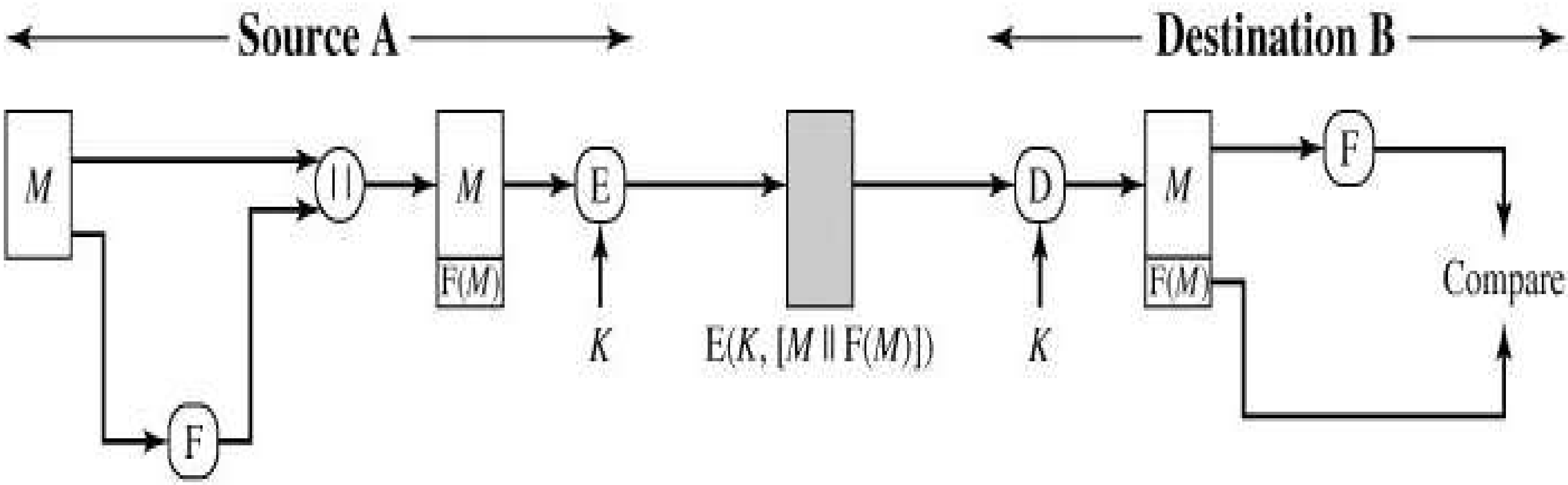
- Message encryption by itself can provide a measure of authentication.
- The analysis differs for symmetric and public-key encryption schemes.
  - Symmetric Encryption
  - Public-Key Encryption: Confidentiality
  - Public-Key Encryption: Authentication
  - Public-Key Encryption: Confidentiality and authentication

Symmetric Encryption



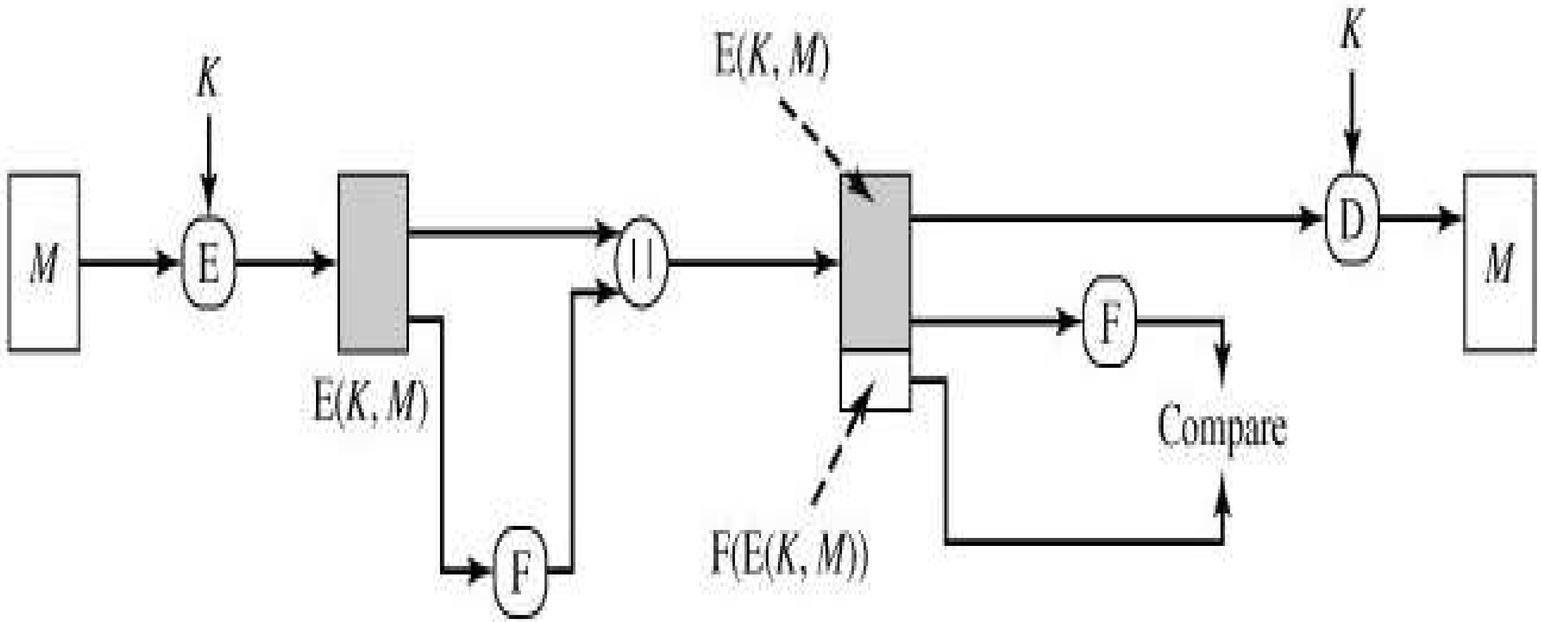
(a) Symmetric encryption: confidentiality and authentication

Internal Error Control - Append an error-detecting code, or frame check sequence (FCS) or checksum with the message



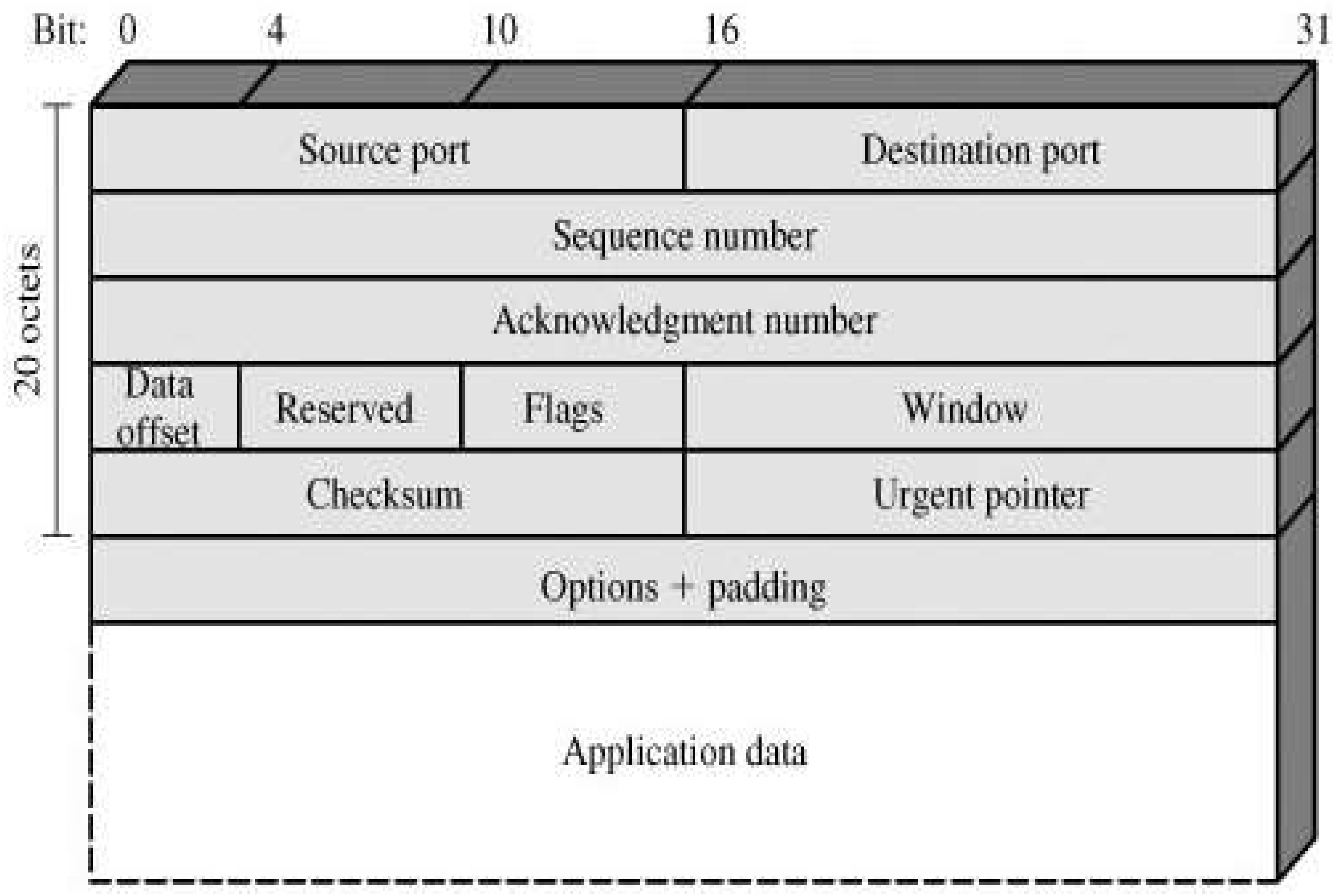
(a) Internal error control

External error control

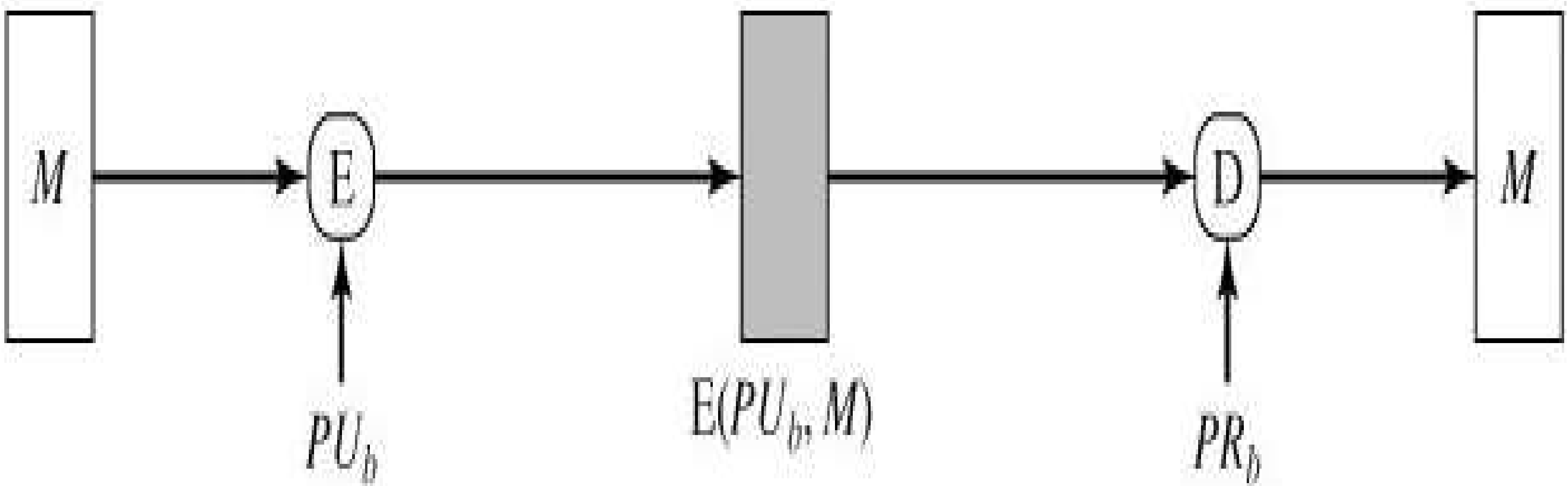


(b) External error control

TCP segment

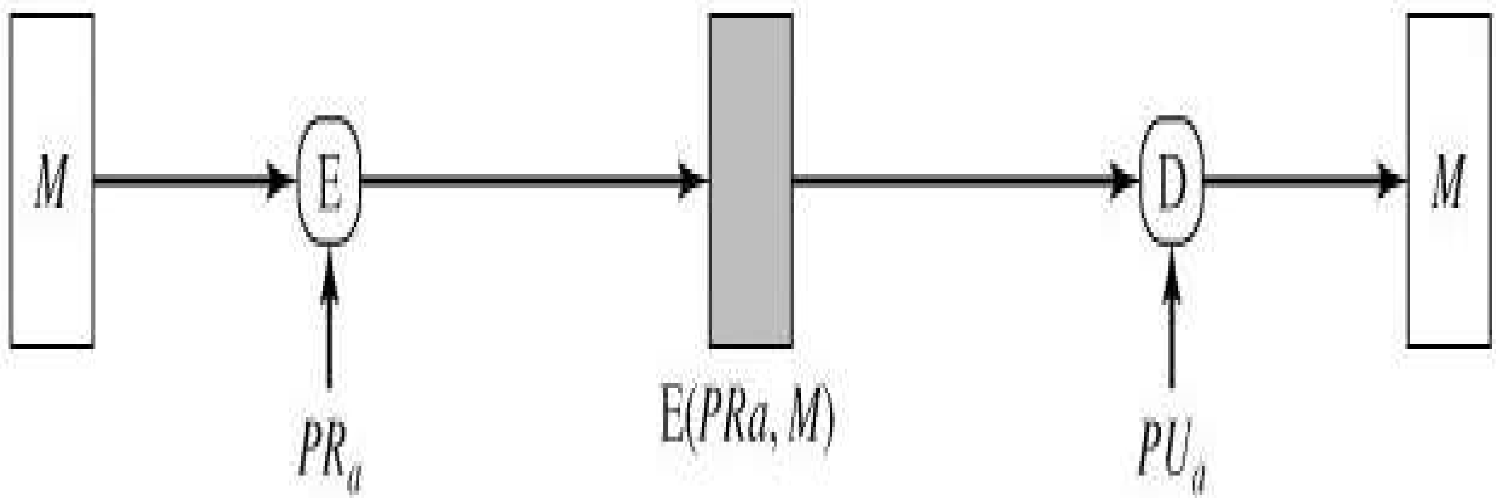

Bit: 0
4
10
16
31
20 octets
Source port
Destination port
Sequence number
Acknowledgment number
Data offset
Reserved
Flags
Window
Checksum
Urgent pointer
Options + padding
Application data

2. Public-Key Encryption: Confidentiality



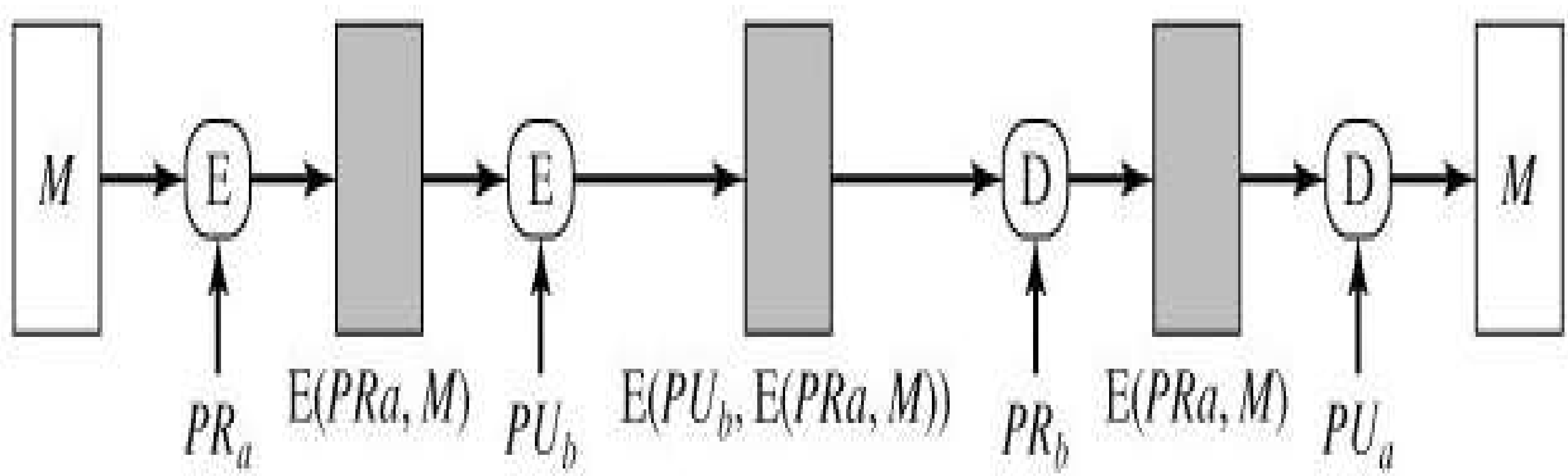
(b) Public-key encryption: confidentiality

3. Public-Key Encryption: Authentication



(c) Public-key encryption: authentication and signature

4. Public-Key Encryption: Confidentiality and authentication



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

(ii) Message authentication code (1)

- An authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum or MAC** that is appended to the message.
- two communicating parties, say A and B, share a common secret key *K.*

Message authentication code (2)

- When A has a message to send to B, it calculates the MAC as a function of the message and the key:

$$\textbf{MAC} = \textbf{C}(\boldsymbol{K}, \boldsymbol{M})$$

Where ***M = input message***

C = MAC function

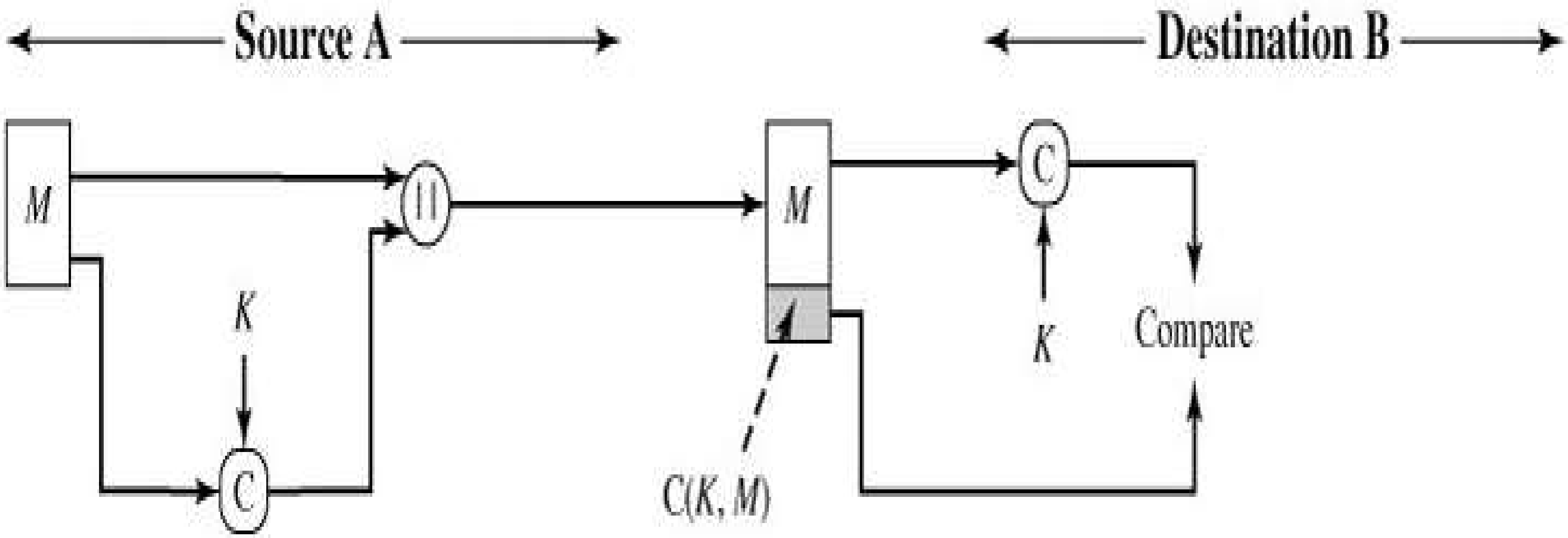
K = shared secret key

MAC = message authentication code

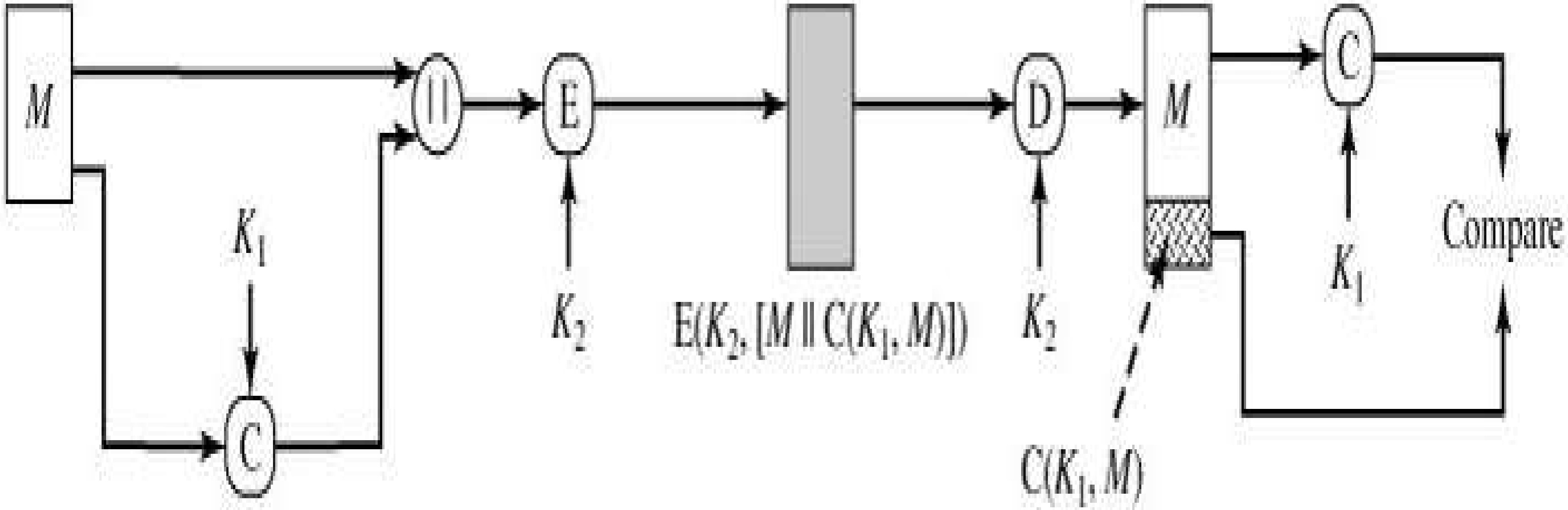
Message authentication code (3)

- message plus MAC are transmitted to the intended recipient.
- recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC.
- received MAC is compared to the calculated MAC as shown in the figure

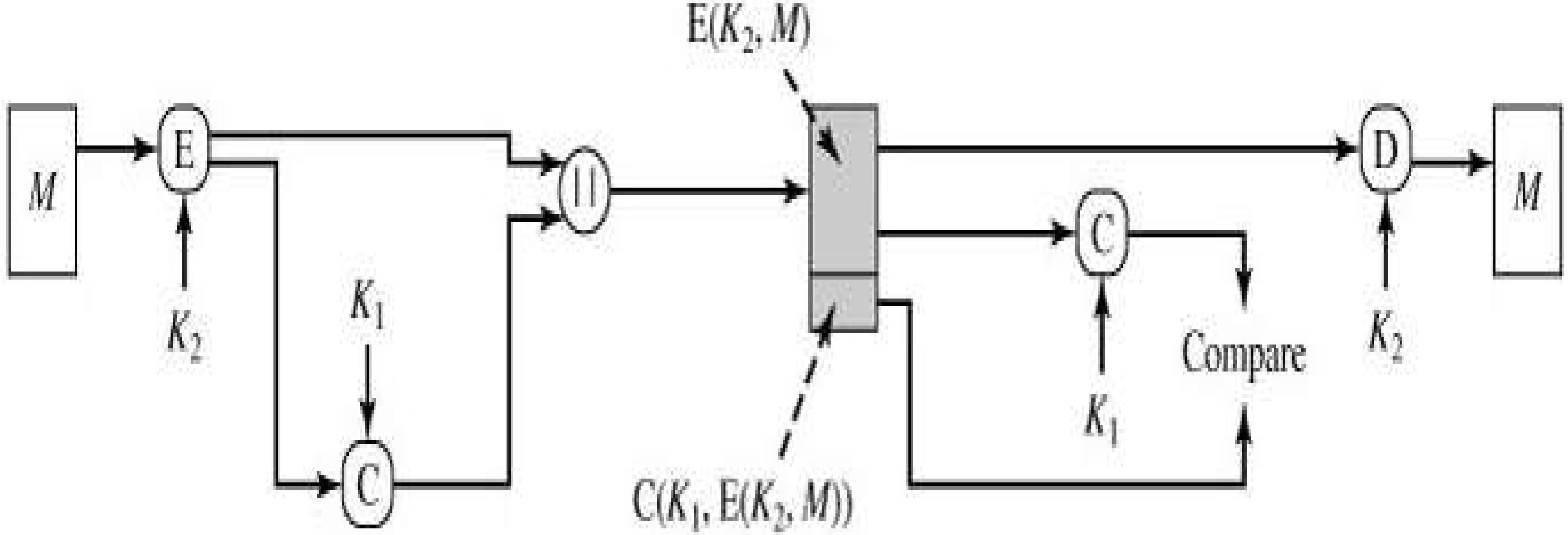
Message authentication code (4)



(a) Message authentication



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



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

(iii) Hash Function

- 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 a MAC, a hash code does not use a key but is a function only of the input message.
- The hash code is also referred to as a **message digest or hash value.**
- The hash code is a function of all the bits of the message and provides an error-detection capability:
- A change to any bit or bits in the message results in a change to the hash code.

HASH FUNCTIONS

- A hash value h is generated by a function H of the form **h = H(M),** where

  M is a variable-length message

  H(M) is the fixed-length hash value.

- The hash value is appended to the message at the source at a time when the message is assumed or known to be correct.
- The receiver authenticates that message by re computing the hash value.
- Because the hash function itself is not considered to be secret, some means is required to protect the hash value.

Requirements for a Hash Function(1)

- The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data.
- To be useful for message authentication, a hash function H must have the following properties:

Requirements for a Hash Function(2)

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.

Requirements for a Hash Function(3)

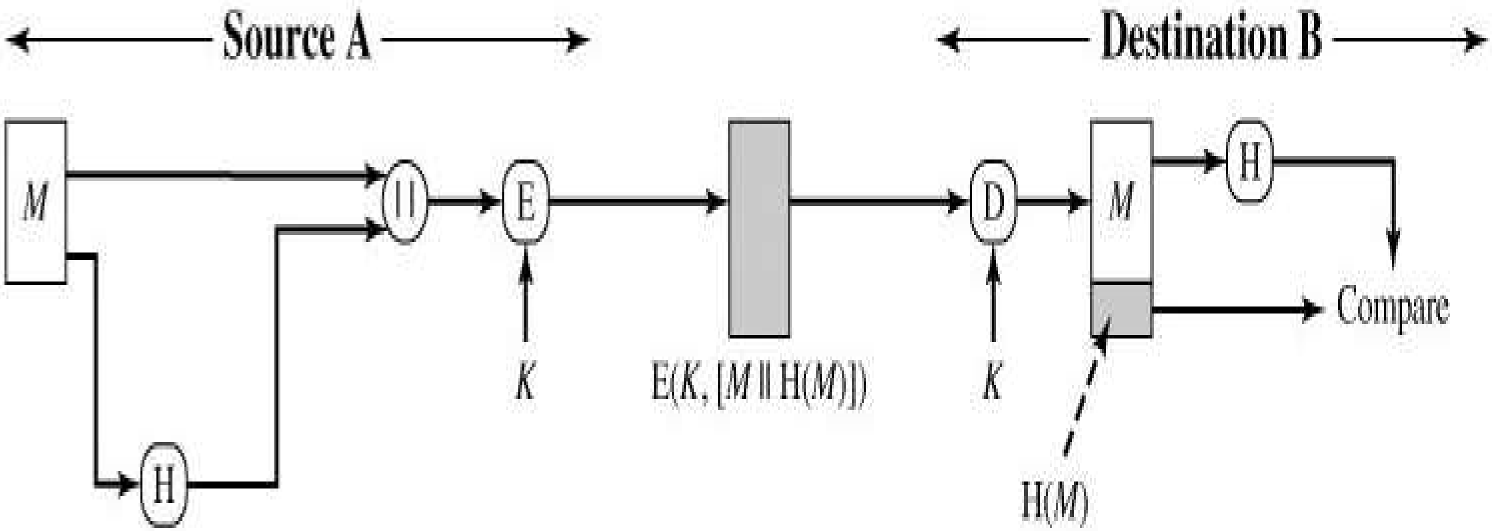
- For any given block x, it is computationally infeasible to find y not equal to x such that H(y) = H(x). This is sometimes referred to as **weak collision resistance.**
- It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as **strong collision resistance.**

- The variety of ways in which a hash code can be used to provide message authentication, as follows:

a)

- The message plus concatenated hash code is encrypted using symmetric encryption.
- This is identical in structure to the internal error control 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.

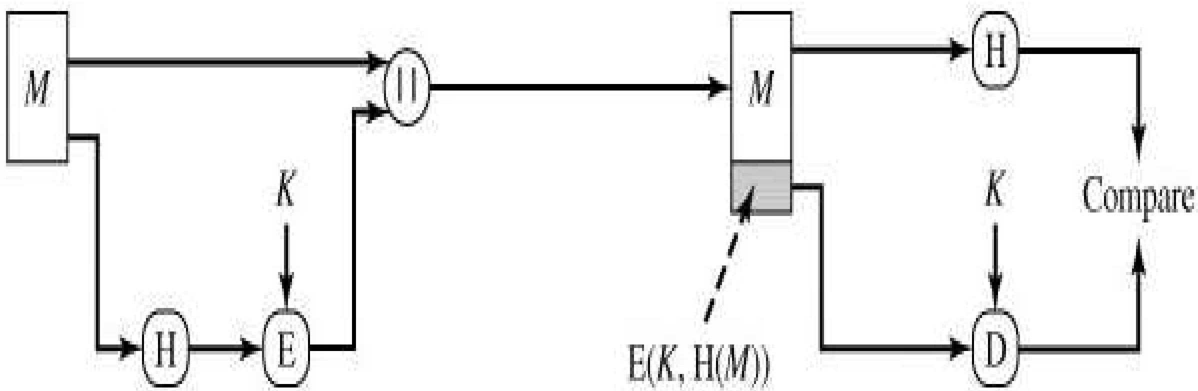
- The hash code provides the structure or redundancy required to achieve authentication.
- Because encryption is applied to the entire message plus hash code, confidentiality is also provided.



(a)

b)

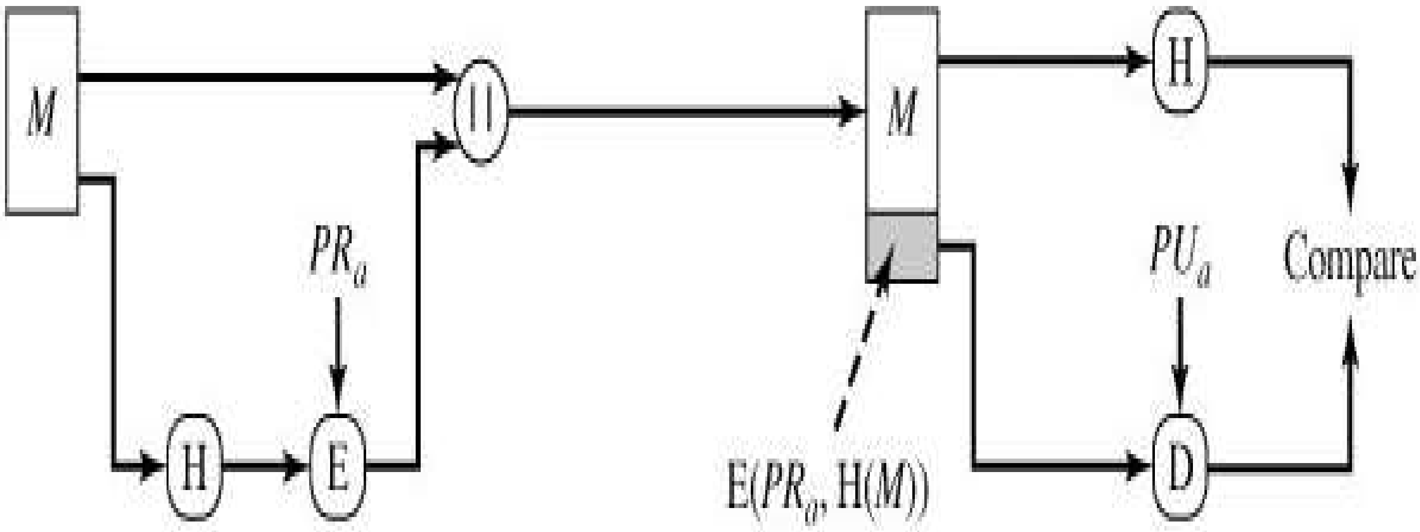
- Only the hash code is encrypted, using symmetric encryption.
- This reduces the processing burden for those applications that do not require confidentiality.



(b)

c)

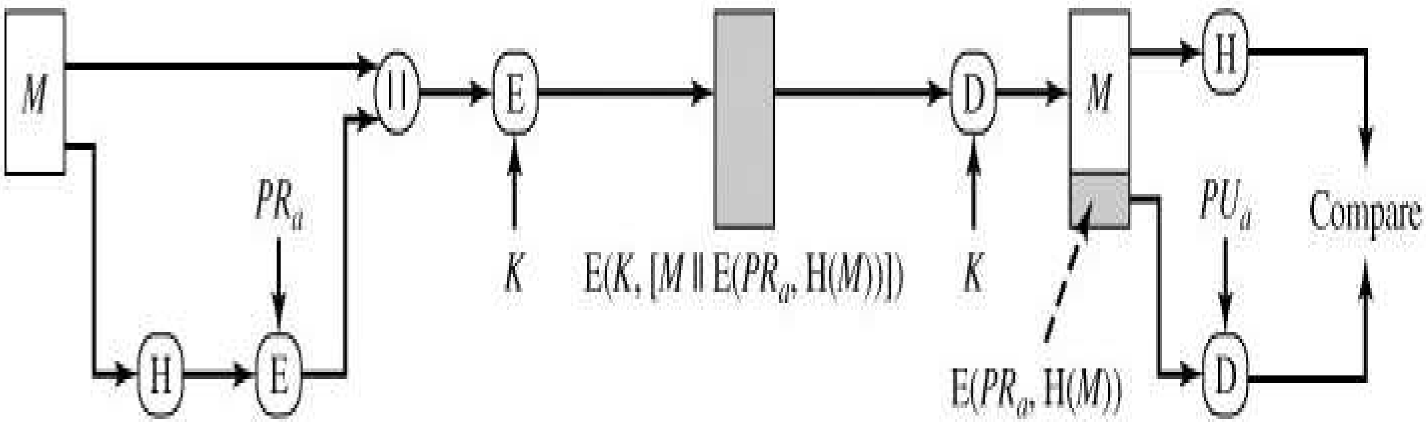
- Only the hash code is encrypted, using public-key encryption and using the sender's private key. As with (b), this provides authentication.
- It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.



(c)

d)

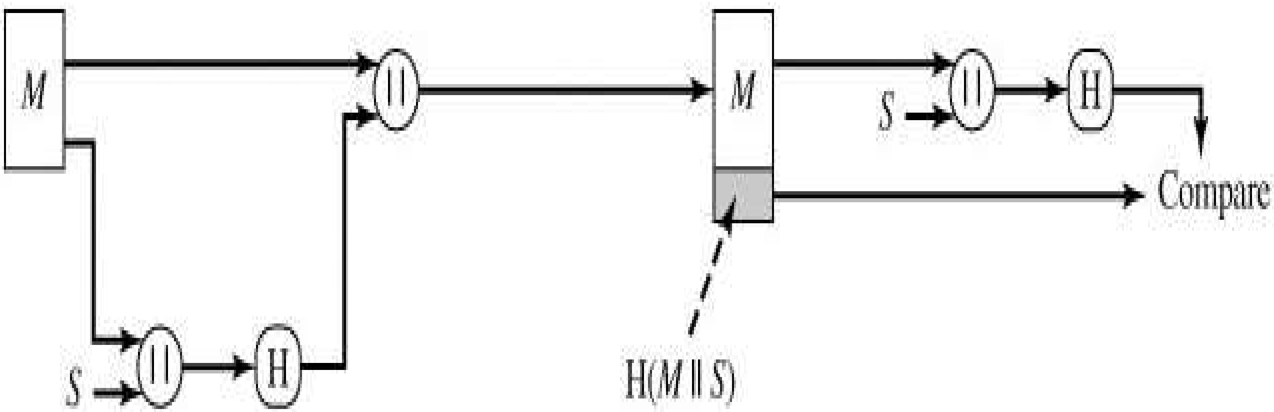
- If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.



(d)

e)

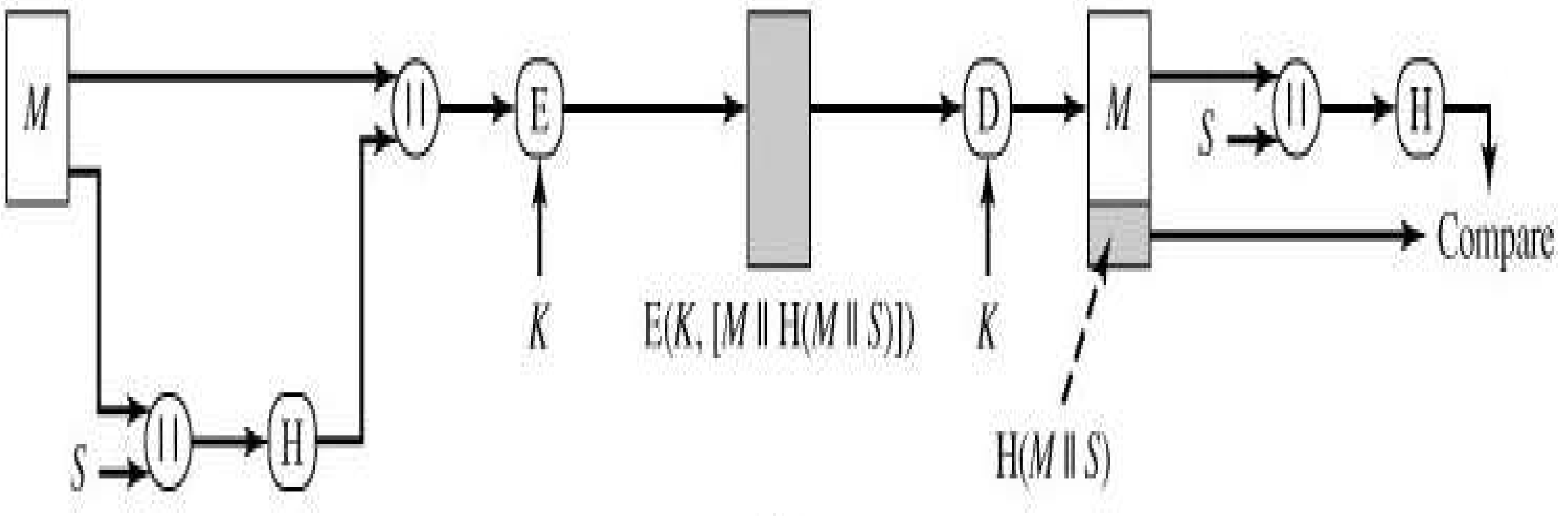
- It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S. A computes the hash value over the
- concatenation of M and S and appends the resulting hash value to M.
- Because B possesses S, it can re compute 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.



(e)

f)

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



(f)

SECURE HASH ALGORITHM

- developed by the National Institute of Standards and Technology (NIST).
- SHA-1 produces a hash value of 160 bits. With hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512.

SHA-512 Logic

- The algorithm takes as input a message with a maximum length of less than 2128 bits
- and produces as output a 512-bit message digest.
- input is processed in 1024-bit blocks.

SHA-1 Logic

- The algorithm takes as input a message with a maximum length of less than 2^{64} bits
- and produces as output a 160-bit message digest.
- input is processed in 512-bit blocks.

Processing steps:

Step 1:Append padding bits

Step 2: Append length

Step 3: Initialize MD buffer

Step 4: Process message in 512-bit blocks

Step 5: Output

Step 1:Append padding bits

- The message is padded so that its length is congruent to 448 modulo 512
- 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 512.
- The padding consists of a single 1-bit followed by the necessary number of 0-bits.

Step 2: Append length

- A block of 64 bits is appended to the message.
- This block is treated as an unsigned 64-bit integer (most significant byte first) and contains the length of the original message (before the padding).

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).
- These registers are initialized to 32-bit integers (hexadecimal values):
- These values are stored in big-endian format, which is the most significant byte of a word in the low-address (leftmost) byte position

Step 4: Process message in 512-bit (16-word) blocks

- The heart of the algorithm is a module that consists of four rounds of processing of 20 steps each
- this module is labeled f in figure.
- Each round takes as input the 512-bit block being processed (Y_q) & the 160 bit buffer value value A B C D E , and updates the contents of the buffer.
-

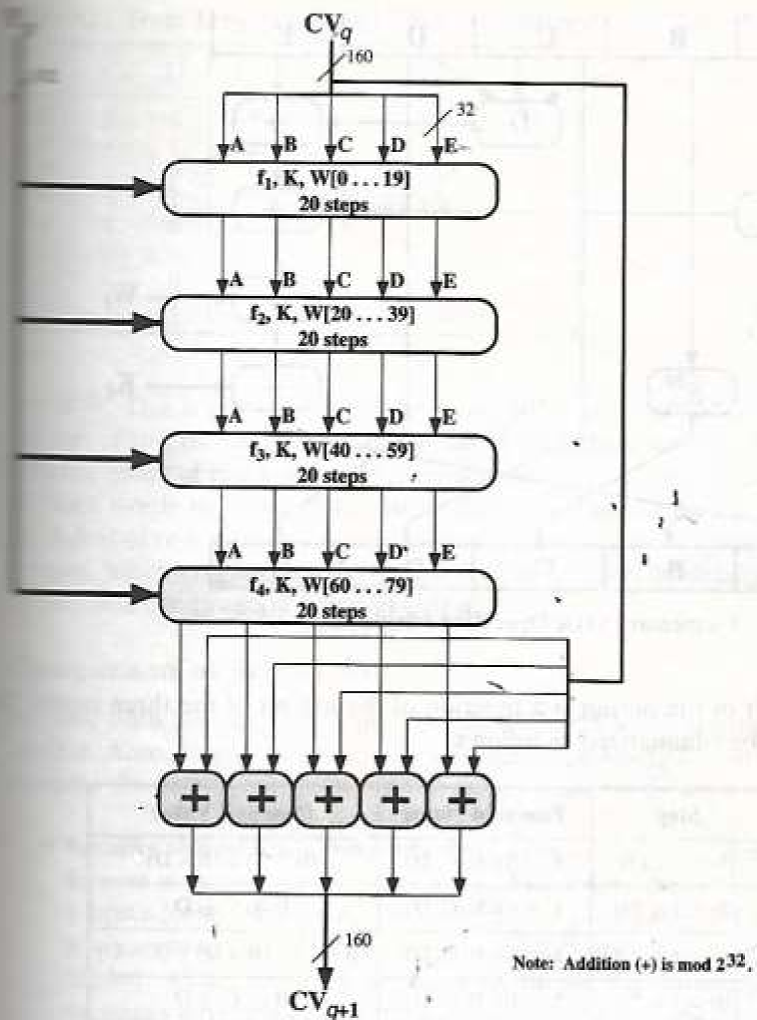


Figure 9.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 compression function).

- 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 steps across 4 rounds.
- Output of the fourth round is added to the input to the first round (CV_{q+1})
- Addition is done independently for each of the five words in the buffer with each of the corresponding words in Cv_q ,using addition modulo 2^{32}

- 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 264

Step 5: Output

- After all L 512-bit blocks have been processed; the output from the Lth stage is the 160-bit message digest.

- IV = initial value of the ABCDE buffer, defined in step 3
- $abcde_q$ = the output of the last round of processing of the q^{th} message block
- N = the number of blocks in the message (including padding and length fields)
- SUM_{32} = Addition modulo 2^{32} performed separately on each word of the pair of inputs
- MD = final message digest value

We can summarize the behavior of SHA-1 as follows:

$$
\begin{aligned}
CV_0 &= IV \\
CV_{q+1} &= SUM_{32}(CV_q, ABCDE_q) \\
MD &= CV_L
\end{aligned}
$$

Elementary SHA Operation (single step)

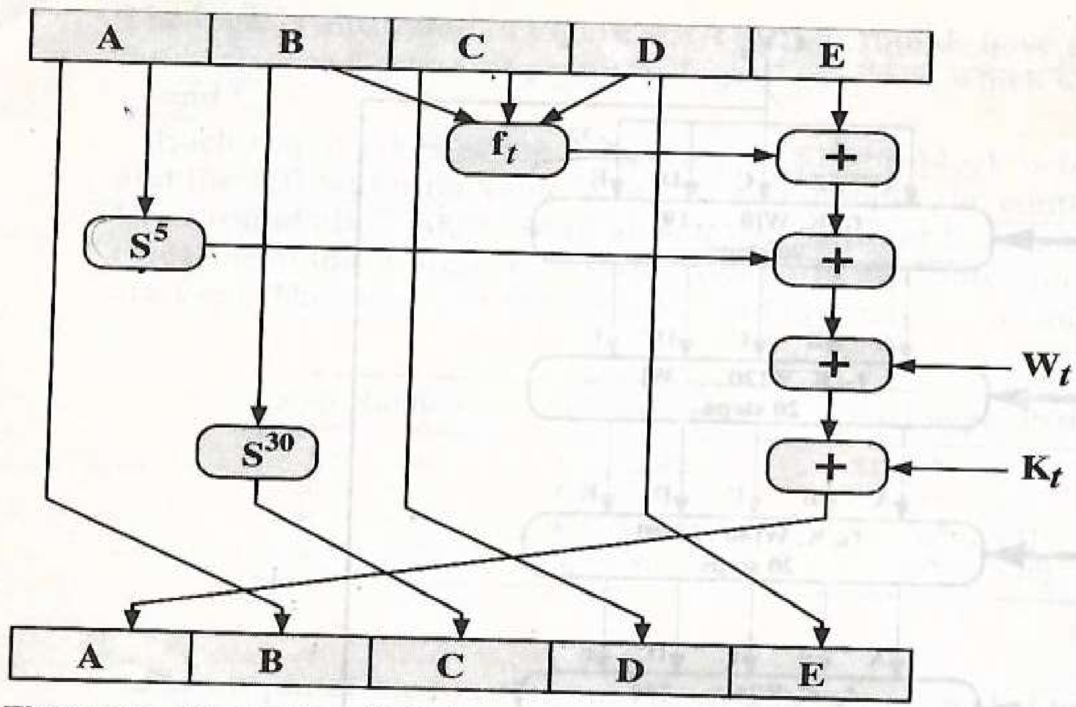


Figure 9.6 Elementary SHA Operation (single step).

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

SHA-1 compression function

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

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

t = step number; $0 \le t \le 79$

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 current 512-bit input block

K_t = an additive constant; four distinct values are used, as defined previously

$+$ = addition modulo 2^{32}

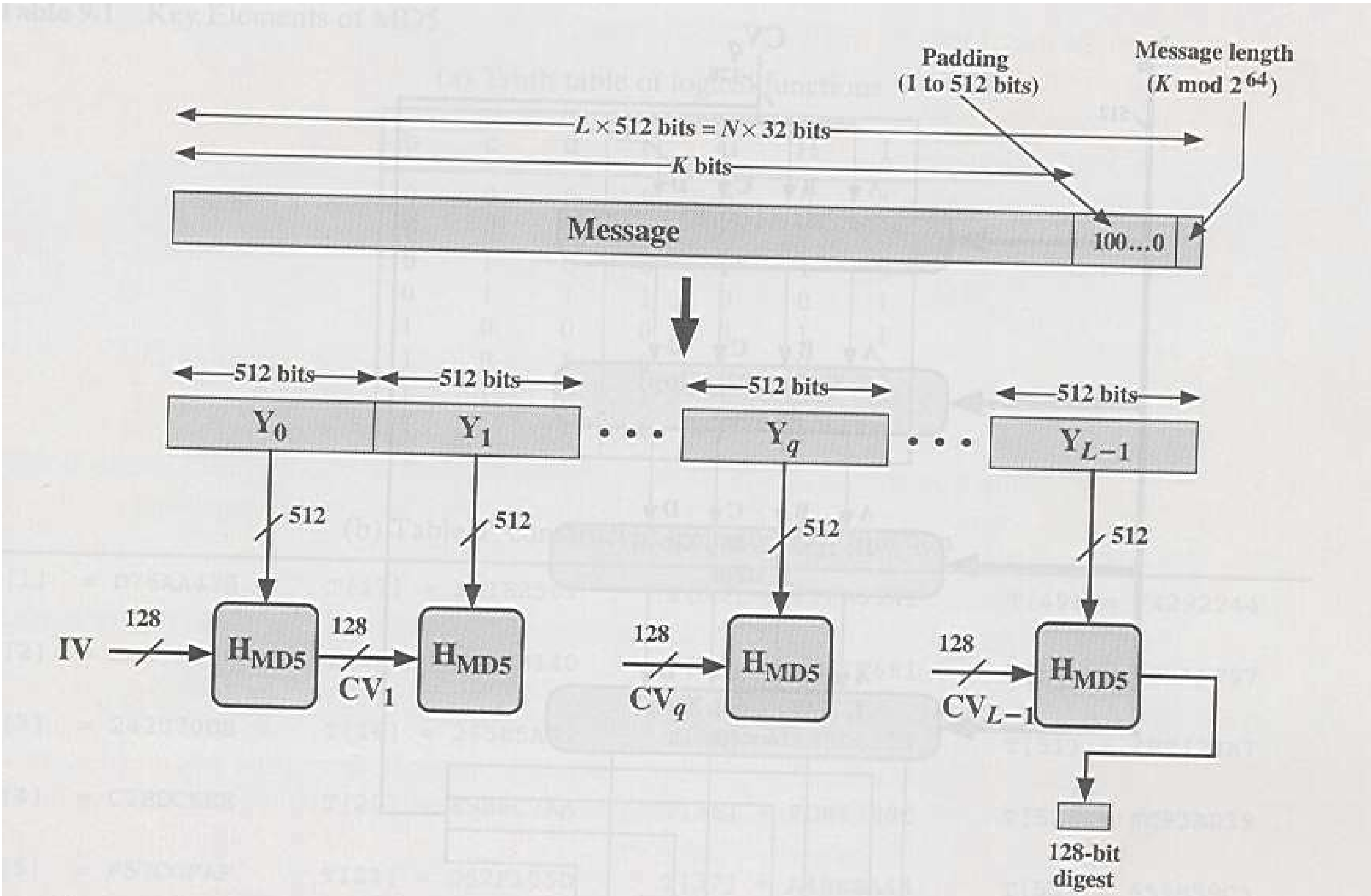
MD5 message digest algorithm

- developed by Ron Rivest
- most widely used secure hash algorithm.
- takes as input a message of arbitrary length and produces as output a 128-bit message digest.
- input is processed in 512-bit blocks

Step 1: Appending padding bits.

- The message is padded so that its length in bits is congruent to 448 modulo 512
- length of the padded message is 64 bits less than an integer multiple of 512 bits.
- Padding is always is added, even if the message is already of the desired length.
- padding consists of a single 1-bit followed by the necessary number of 0-bits.

Message digest generation using MD5



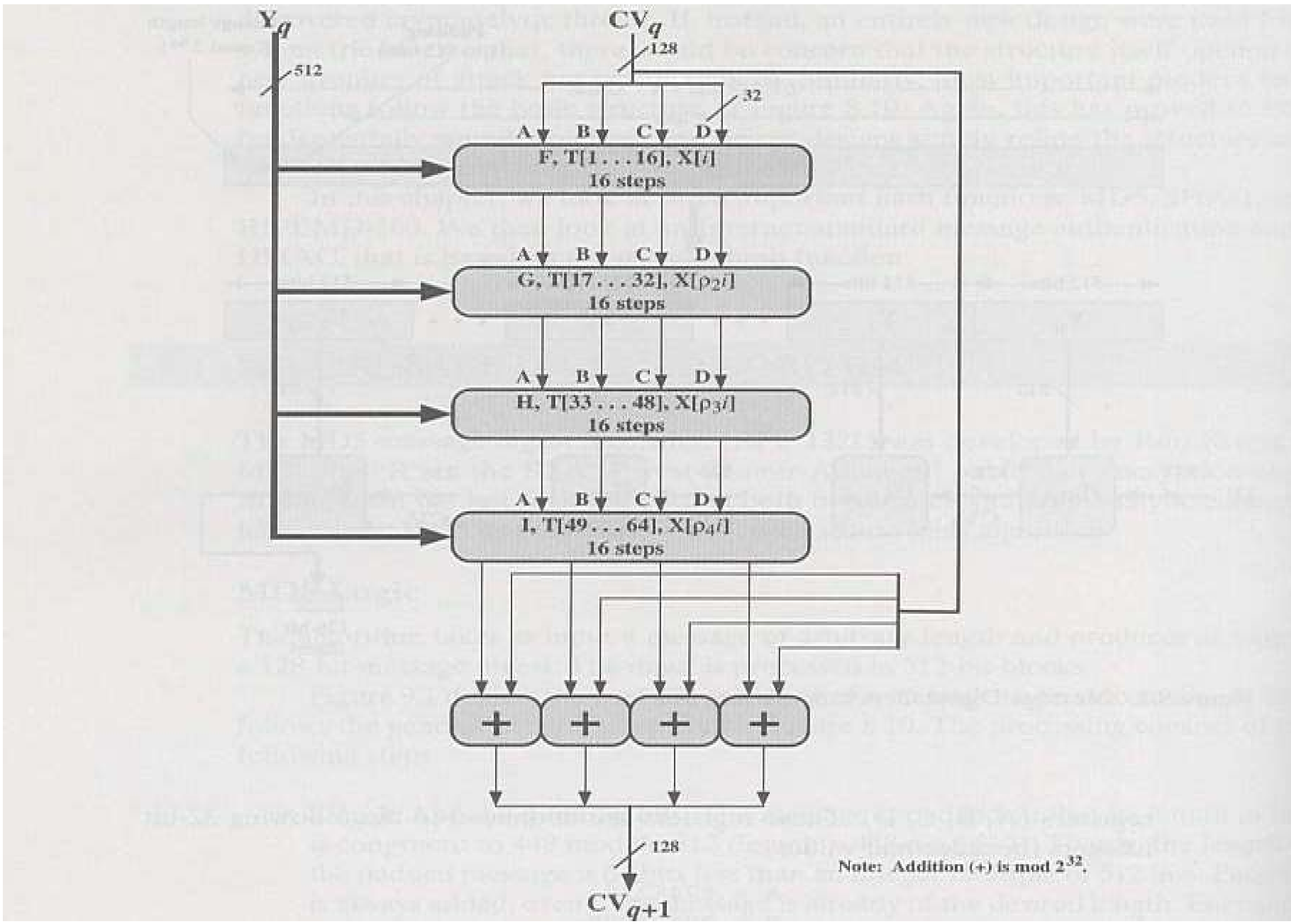
Step 2: Append length

- A 64-bit representation of the length in bits of the original message
- If the original length is greater than 264, then only the low-order 64 bits of the length are used.
- field contains the length of the original message, modulo 2^{64}
- expanded message is represented as the sequence of 512-bit blocks $Y_0, Y_1, \ldots Y_{L-1}$ so that the total length of the expanded message is $L \times 512$ *bits.*

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).
- These values are stored in little-endian format, which is the least significant byte of a word in the low-address byte position.

MD5 processing of a single 512-bit block (MD5 compression function)



Step 4: Process message in 512-bit (16-word) blocks.

- The heart of the algorithm is a compression algorithm that consists of four “rounds” of processing; this module is labeled H_{MD5}.
- *The four rounds have the similar structure,* but each uses a different primitive logical function, referred to as *F, G, H, and I in the specification.*
- Each round takes as input the current 512-bit block being processed (Y_q) *and the 128-bit buffer value ABCD and* updates the contents of the buffer.

Step 5: Output

- After all *L 512-bit blocks have been* processed, the output from the L^{th} *stage is the 128-bit* message digest

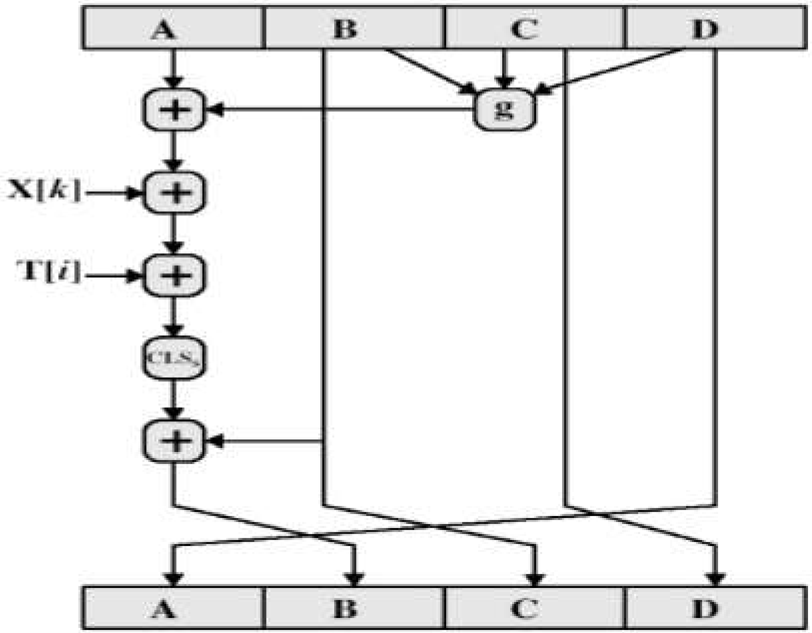
MD5 Compression Function

- 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

MD5 Compression Function



SECURITY OF HASH FUNCTIONS AND MACS

- Attacks on hash functions and MACs is grouped into two categories:

    1. Brute-force attacks

    2. Cryptanalysis.

- The nature of brute-force attacks differs somewhat for hash functions and MACs.

1. Brute force attack on Hash Functions

- The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm.
- There are three desirable properties of hash functions:
    - **One-way property**
    - **Weak collision resistance**
    - **Strong collision resistance**

Properties of hash functions

- **One-way property:** For any given code h, it is computationally infeasible to find x such that H(x) = h.
- **Weak collision resistance:** For any given block x, it is computationally infeasible to find y≠ x with H(y) = H(x).
- **Strong collision resistance:** It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).

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 $2^{n/2}$

- If strong collision resistance is required, then the value $2^{n/2}$ determines the strength of the hash code against brute-force attacks.

2. Message Authentication Codes

- A brute-force attack on a MAC is a more difficult undertaking because it requires known message - MAC pairs.
- To proceed, 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 $x \neq x_i$.

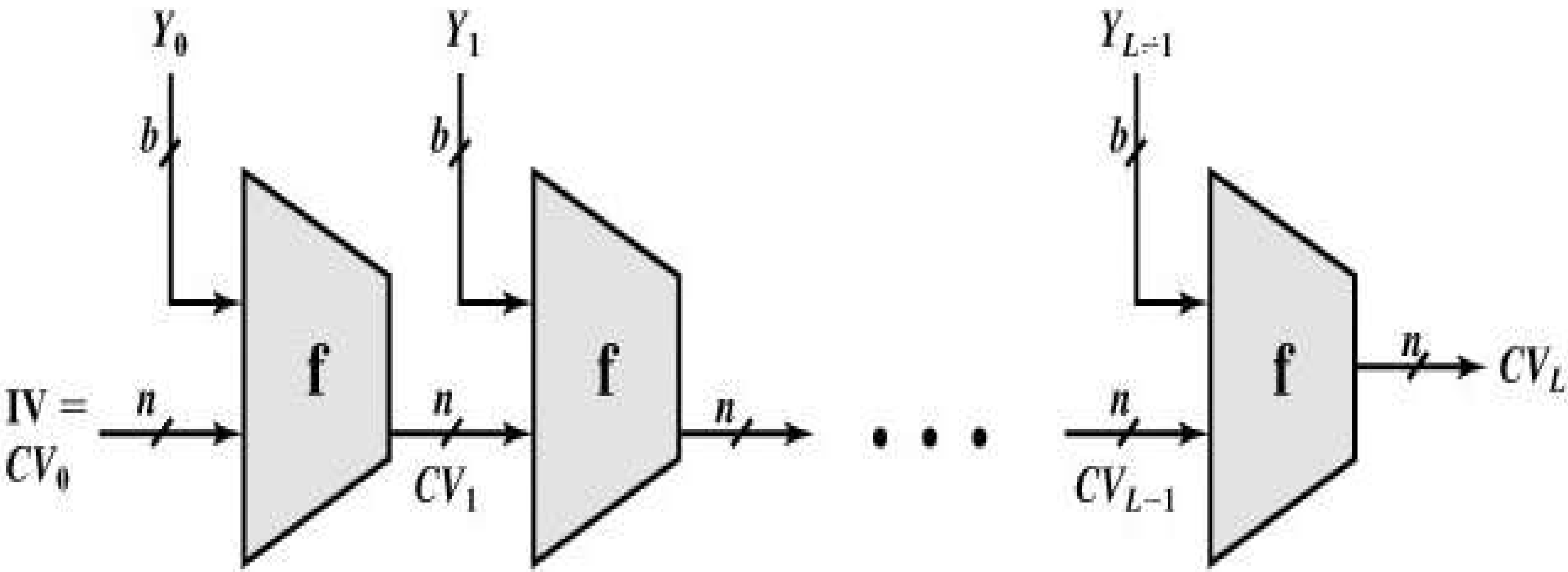
- The attacker would like to come up with the valid MAC code for a given message x. There are two lines of attack possible:
    - **Attack the key space**
    - **Attack the MAC value**

2. **Cryptanalysis**

- The way to measure the resistance of a hash or MAC algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack.
- That is, an ideal hash or MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

a. Cryptanalysis on Hash Functions

- The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each.
- If necessary, the final block is padded to b bits.
- The final block also includes the value of the total length of the input to the hash function.
- The inclusion of the length makes the job of the opponent more difficult.



IV = Initial value
CV_i = Chaining variable
Y_i = ith input block
f = Compression algorithm
L = Number of input blocks
n = Length of hash code
b = Length of input block

- The hash algorithm involves repeated use of a **compression function**, f that takes two inputs (an n-bit input from the previous step, called the chaining variable, and a b-bit block) and produces an n-bit output.
- At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm.
- The final value of the chaining variable is the hash value.

- Often, b>n; hence the term compression.
- The hash function can be summarized as follows:

$$CV_o = IV = \text{initial n-bit value}$$

$$CV_i = f(CV_{i1}, Y_{i1})\ 1 \leq I \leq L$$

$$H(M) = CV_L$$

- where the input to the hash function is a message M consisting of the blocks Y_o, Y_1,..., Y_{L1}.

- Cryptanalysis of hash functions focuses on the internal structure of f and is based on attempts to find efficient techniques for producing collisions for a single execution of f.
- Once that is done, the attack must take into account the fixed value of IV.
- The attack on f depends on exploiting its internal structure.

Message Authentication Codes

- There is much more variety in the structure of MACs than in hash functions
- so it is difficult to generalize about the cryptanalysis of MACs.

DIGITAL SIGNATURES

Requirements

- Message authentication protects two parties who exchange messages from any third party.
- However, it does not protect the two parties against each other.
- Several forms of dispute between the two are possible.
- For example, suppose that John sends an authenticated message to Mary, using one of the schemes. Consider the following disputes that could arise:

1. Mary may forge a different message and claim that it came from John
2. John can deny sending the message.

- In situations where there is not complete trust between sender and receiver, something more than authentication is needed.
- The solution to this problem is the digital signature.
- The digital signature is analogous to the handwritten signature.

Properties of Digital Signature:

- It must verify the author and the date and time of the signature.
- It must authenticate the contents at the time of the signature.
- It must be verifiable by third parties, to resolve disputes.

Thus, the digital signature function includes the authentication function.

Requirements for a digital signature (1)

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information unique to the sender, to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.

Requirements for a digital signature (2)

- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy of the digital signature in storage.

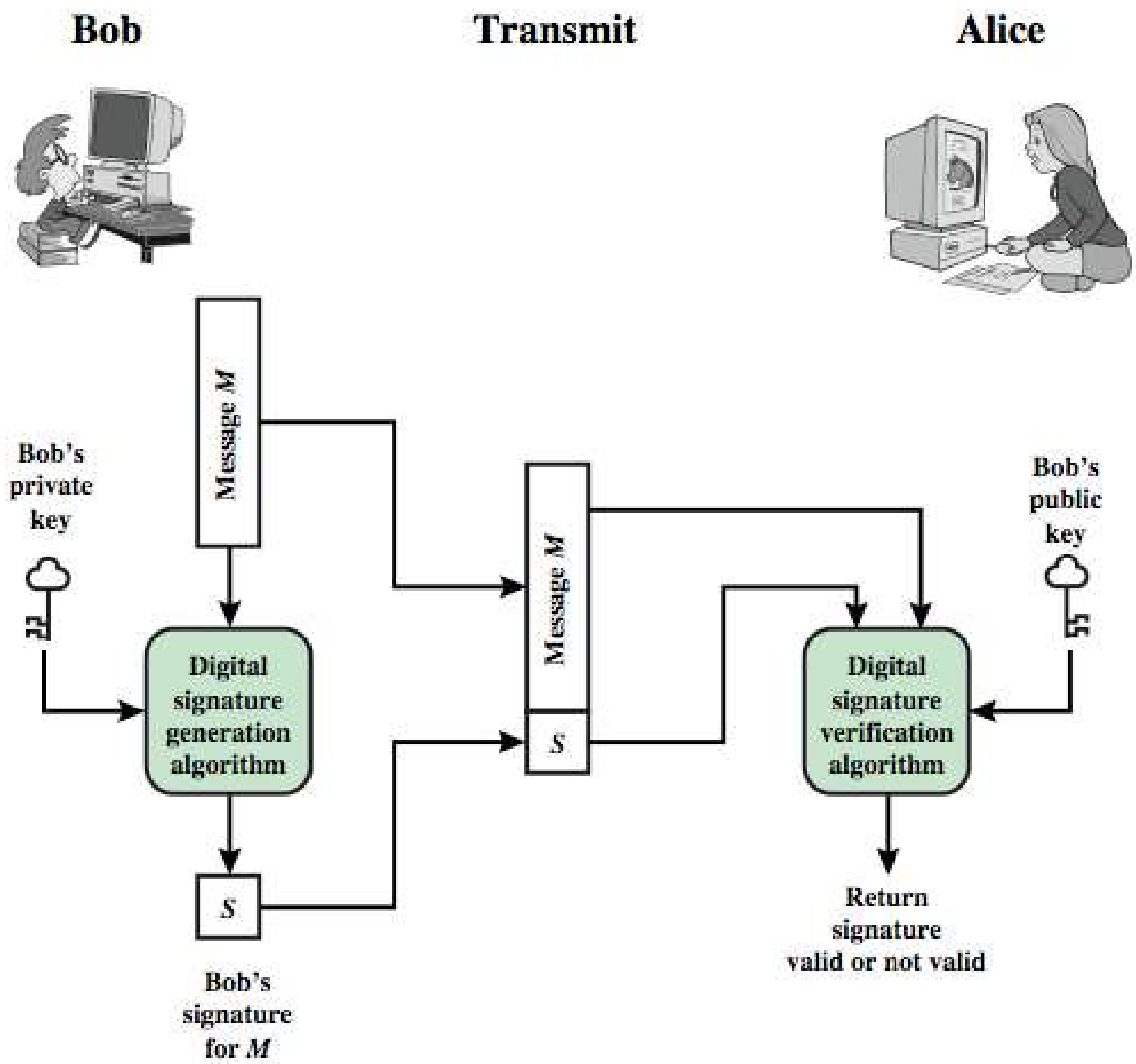
Variety of approaches has been proposed for the digital signature function.
These approaches fall into two categories:

- **Direct Digital Signature**
- **Arbitrated Digital Signature**

Direct Digital Signatures

- involve only sender & receiver
- assumed receiver has sender’s public-key
- digital signature made by sender signing entire message or hash with private-key
- can encrypt using receivers public-key
- important that sign first then encrypt message & signature
- security depends on sender’s private-key

Digital Signature Model



Arbitrated Digital Signatures

- involves use of an arbiter who
  - validates any signed message
  - then dated and sent to recipient
- **requires suitable level of trust in arbiter**
- can be implemented with either private or public-key algorithms
- arbiter may or may not see message

Authentication Protocols

- used to convince parties of each others identity and to exchange session keys
- may be one-way or mutual
- key issues are
  - **confidentiality** – to protect session keys
  - **timeliness** – to prevent replay attacks
- published protocols are often found to have flaws and need to be modified

Mutual authentication

- Such protocols enable communicating parties to satisfy themselves mutually about each others identity & to exchange session keys

Examples of replay attacks

- **Simple replay** - opponent simply copies a message & replays it later
- **Repetition that can be logged**: - opponent can replay a timestamped message within the valid time window.
- Repetition that cannot be detected
- Backward replay without modification

1. Using Symmetric Encryption

- Can use a two-level hierarchy of keys
- usually with a trusted Key Distribution Center (KDC)
  - each party shares own master key with KDC
  - KDC generates session keys used for connections between parties
  - master keys used to distribute these to them

Needham-Schroeder Protocol (1)

- Used by 2 parties who both trust a common key server
- original third-party key distribution protocol
- for session between A & B mediated by KDC
- protocol overview is:
  1. **A->KDC:** $ID_A \,||\, ID_B \,||\, N_1$
  2. **KDC -> A:** $E_{Ka}[Ks \,||\, ID_B \,||\, N_1 \,||\, E_{Kb}[Ks||ID_A]\,]$
  3. **A -> B:** $E_{Kb}[Ks||ID_A]$
  4. **B -> A:** $E_{Ks}[N_2]$
  5. **A -> B:** $E_{Ks}[f(N_2)]$

Needham-Schroeder Protocol(2)

- used to securely distribute a new session key for communications between A & B
- but is vulnerable to a replay attack if an old session key has been compromised
  – then message 3 can be resent convincing B that is communicating with A
- modifications to address this require:
  – timestamps
  – using an extra nonce

2. Using Public-Key Encryption

- have a range of approaches based on the use of public-key encryption
- need to ensure have correct public keys for other parties
- using a central Authentication Server (AS)
- various protocols exist using timestamps or nonces

Denning Authentication Server Protocol

- Denning presented the following:
  1. **A -> AS: ID_A || ID_B**
  2. **AS -> A: $E_{PRas}[ID_A || PU_a || T]$ || $E_{PRas}[ID_B || PU_b || T]$**
  3. **A -> B: $E_{PRas}[ID_A || PU_a || T]$ || $E_{PRas}[ID_B || PU_b || T]$ || $E_{PUb}[E_{PRa}[K_s || T]]$**
- Note that to avoid the risk of exposure by the AS session key is chosen by A, hence AS need not be trusted to protect it
- timestamps prevent replay attacks but require synchronized clocks

One-Way Authentication

- The recipient wants some assurance that the message is from the alleged sender. One-Way Authentication addresses these requirements.
- Required when sender & receiver communicate in connectionless mode (eg. email)
- Have header in clear text so can be delivered by email systems
- May want contents of body protected & sender authenticated

One-Way Authentication Using Public-Key Approaches

- have seen some public-key approaches
- if confidentiality is major concern, can use:

  A->B: $E_{PUb}[Ks]$ || $E_{Ks}[M]$

  – has encrypted session key, encrypted message
- if authentication needed use a digital signature with a digital certificate:

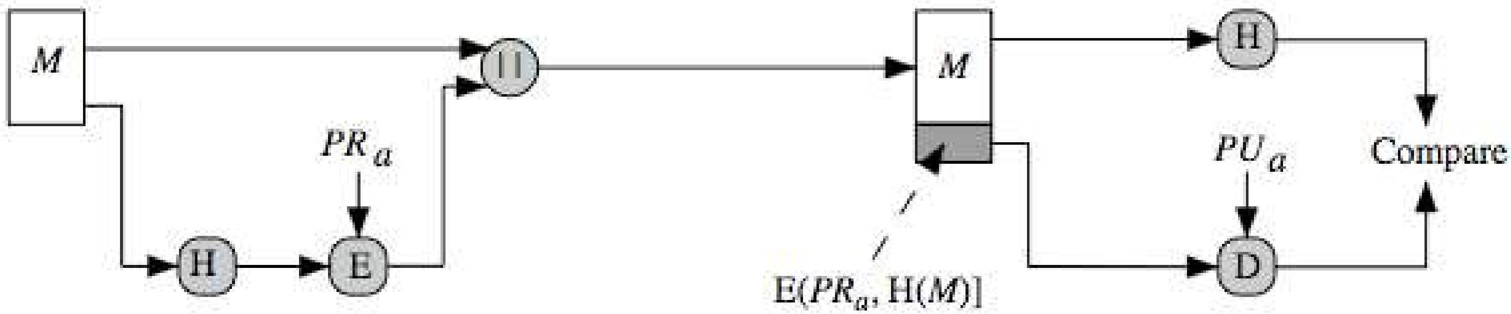
  A->B: M || $E_{PRa}[H(M)]$ || $E_{PRa}[T||ID_A||PU_a]$

  – with message, signature, certificate

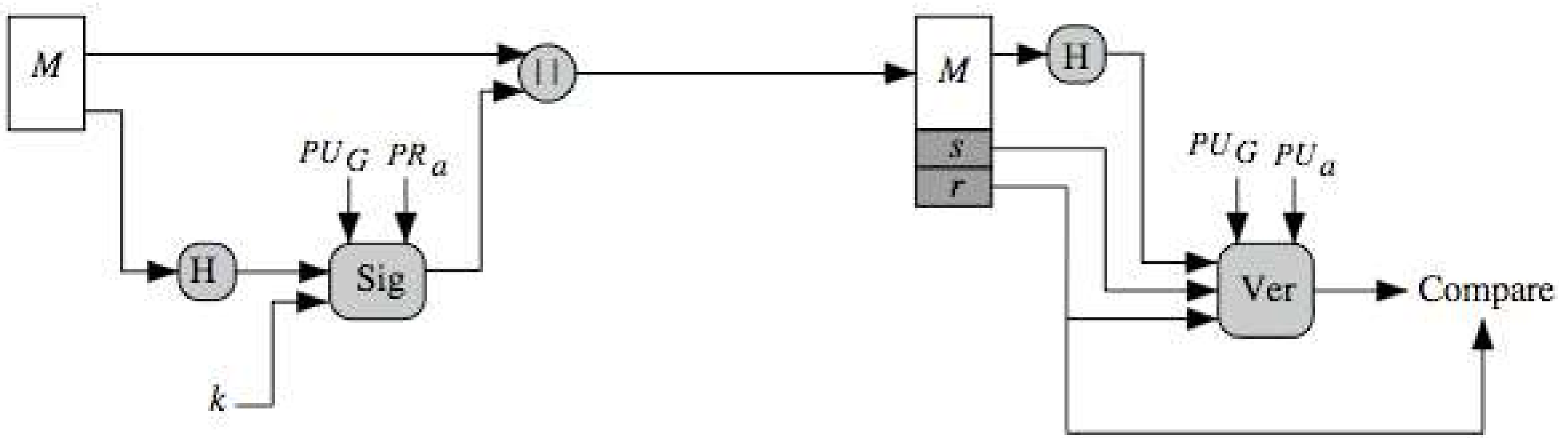
Digital Signature Standard (DSS)

- US Govt approved signature scheme
- designed by NIST & NSA in early 90's
- revised in 1993, 1996 & then 2000
- uses the SHA hash algorithm
- DSS is the standard, DSA is the algorithm
- 2 approaches
    - RSA approach (SHA, RSA)
    - DSS or DSA Approach

DSS vs RSA Signatures



(a) RSA Approach



(b) DSS Approach

Global Public-Key Components

p	prime number where $2^{L\ 1} < p < 2^{L}$ for $512 \leq L \leq 1024$ and L a multiple of 64; i.e., bit length of between 512 and 1024 bits in increments of 64 bits
q	prime divisor of (p 1), where $2^{159} < q < 2^{160}$; i.e., bit length of 160 bits
g	$= h^{(p\ 1)/q} \bmod p$, where h is any integer with $1 < h < (p\ 1)$ such that $h^{(p\ 1)/q} \bmod p > 1$

User's Private Key

x	random or pseudorandom integer with $0 < x < q$

User's Public Key

y	$= g^{x} \bmod p$

User's Per-Message Secret Number

k	= random or pseudorandom integer with $0 < k < q$

Signing

r	$= (g^{k} \bmod p) \bmod q$
s	$= [k^{-1} (H(M) + xr)] \bmod q$

Signature = (r, s)

DSA Key Generation

- have shared global public key values (p,q,g):
  - choose 160-bit prime number q
  - choose a large prime p with $2^{L-1} < p < 2^L$
    - where L= 512 to 1024 bits and is a multiple of 64
    - such that q is a 160 bit prime divisor of $(p-1)$
  - choose $g = h^{(p-1)/q}$
    - where $1<h<p-1$ and $h^{(p-1)/q} \bmod p > 1$
- users choose private & compute public key:
  - choose random private key: $x<q$
  - compute public key: $y = g^x \bmod p$

DSA Signature Creation

- to **sign** a message M the sender:
  - generates a random signature key k, k<q
  - nb. k must be random, be destroyed after use, and never be reused
- then computes signature pair:

$$r = (g^k \bmod p) \bmod q$$

$$s = [k^{-1}(H(M) + xr)] \bmod q$$

- sends signature (r,s) with message M

DSA Signature Verification

- having received M & signature (r,s)
- to **verify** a signature, recipient computes:

$$w = s^{-1} \bmod q$$

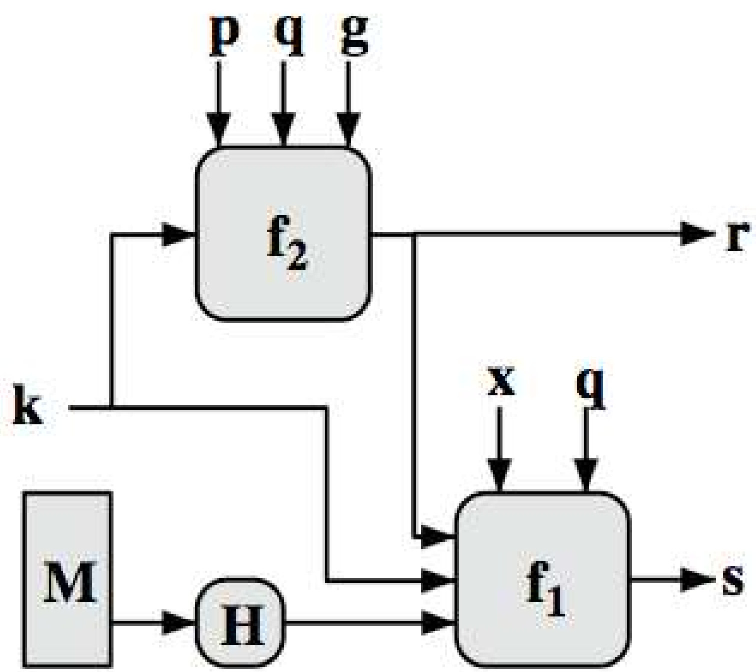
$$u1 = [H(M)w]\bmod q$$

$$u2 = (rw)\bmod q$$

$$v = [(g^{u1}\ y^{u2})\bmod p\ ]\bmod q$$

- if $v=r$ then signature is verified

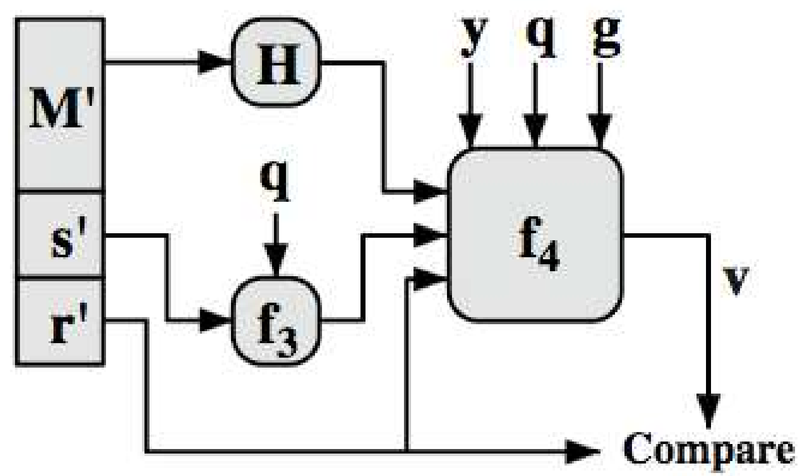
DSS Overview



$$s = f_1(H(M), k, x, r, q) = (k^{-1}\,(H(M) + xr)) \bmod q$$

$$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$$

(a) Signing



$$w = f_3(s', q) = (s')^{-1} \bmod q$$

$$v = f_4(y, q, g, H(M'), w, r')$$

$$= ((g^{(H(M')w) \bmod q}\, y^{r'w \bmod q}) \bmod p) \bmod q$$

(b) Verifying