Sanjivani Rural Education Society's

Sanjivani College of Engineering, Kopargaon-423603

(An Autonomous Institute Affiliated to Savitribai Phule Pune University, Pune)

NAAC 'A' Grade Accredited, ISO 9001:2015 Certified

Department of Information Technology

(NBA Accredited)

Cryptography and Cyber Security [IT311]

Mrs. Kanchan D. Patil

Assistant Professor

Unit 3: Message Digest & Key Management

- Cryptographic Hash Functions, Applications of Cryptographic Hash Functions- Message Authentication, Digital Signatures, Two Simple Hash Functions, MD5 algorithm, SHA-1 algorithm, Key Management: Introduction, Generations, Distribution, Updation, Digital Certificate, Kerberos 5.0.

Cryptographic Hash Function

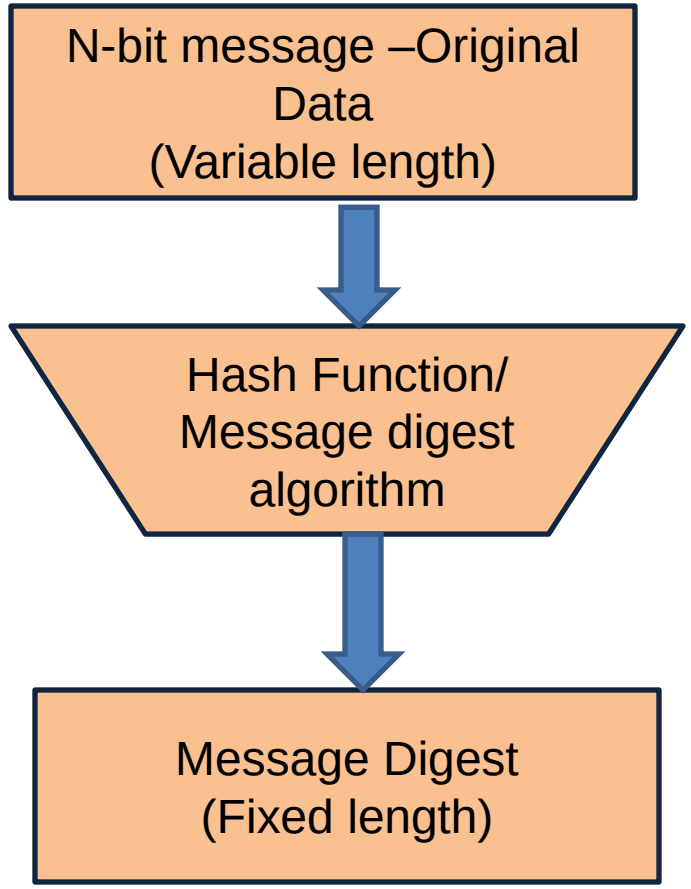
- It is a fingerprint or summary of message
- A hash function is a mathematical function that converts a numerical input value into another compressed numerical value.
- The input to the hash function is of **variable length** but output is always of **fixed length.**
- The value generated as a output is referred **as hash value or message digest**
- It is used to verify the integrity of data (i.e. to ensure that a message has not been tampered with after it leaves the sender but before it reaches the receiver)
- It has a **property** that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random

Cryptographic Hash Function

- A change to any bit or bits in M results, with high probability, in a change to hash code
- The kind of **hash function needed for security applications** is referred to as **cryptographic hash function**
- A Cryptographic hash function is an algorithm for which it is infeasible to find
  - Either a data object that maps to a pre-specified hash result (one-way function)
  - Or two data objects that map to same hash result (collision-free property)

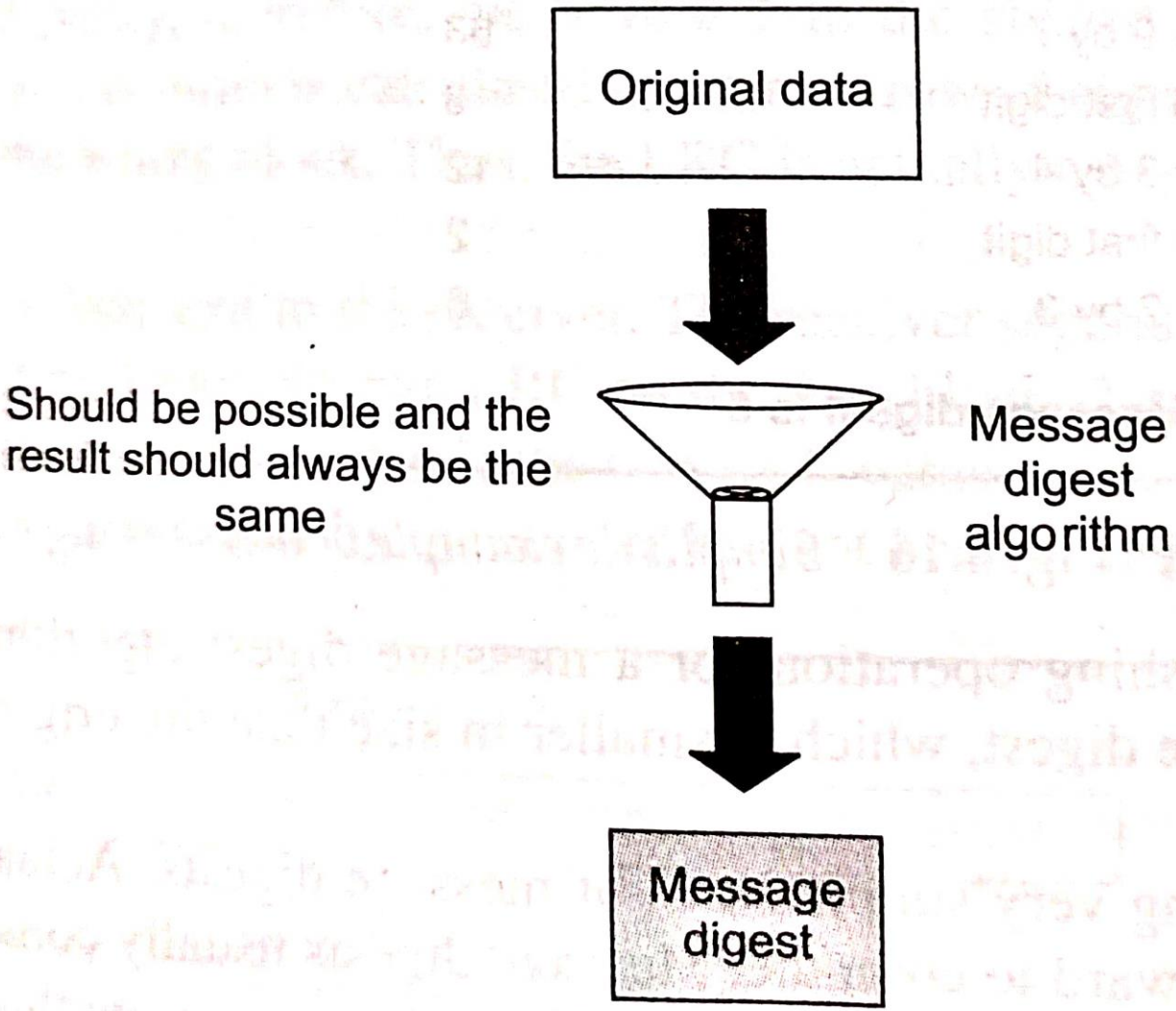
Cryptographic Hash Function

- Hash functions/ Message digests are not so small and straightforward to compute.
- The input is padded out to an integer of multiple of some fixed length and the padding includes the value of the length of the original message in bits
- Message digests usually consist of **128 or more bits**.
- This means that the chance of any two message digests being the same is anything between 0 to at least 2^128
- The message digest length is chosen to be so to ensure the scope for two message digests being the same.



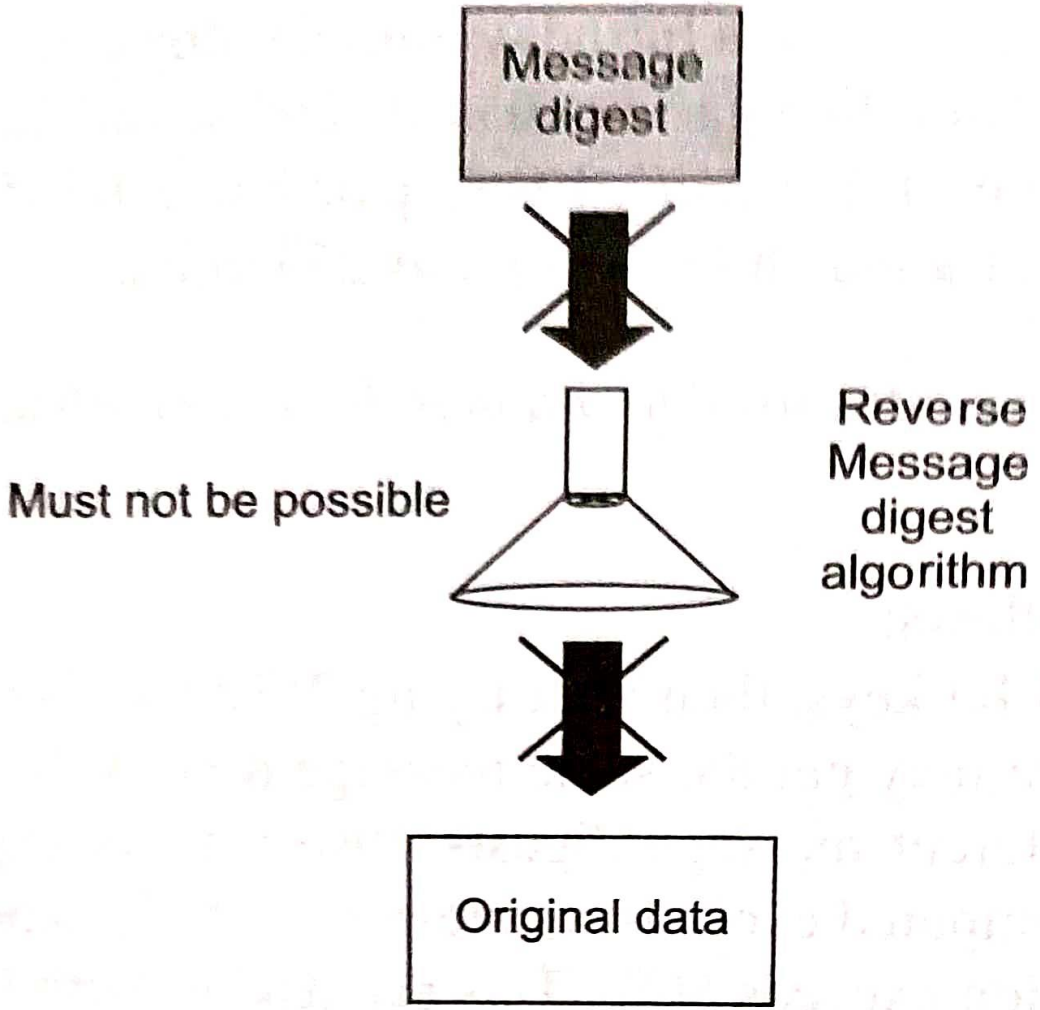
Requirements of Cryptographic Hash Function

- **R1-Efficiency:**
- Given a message, x, it should be very easy to find its corresponding message digest, H(x).
- For a given message, the message digest must always be the same.



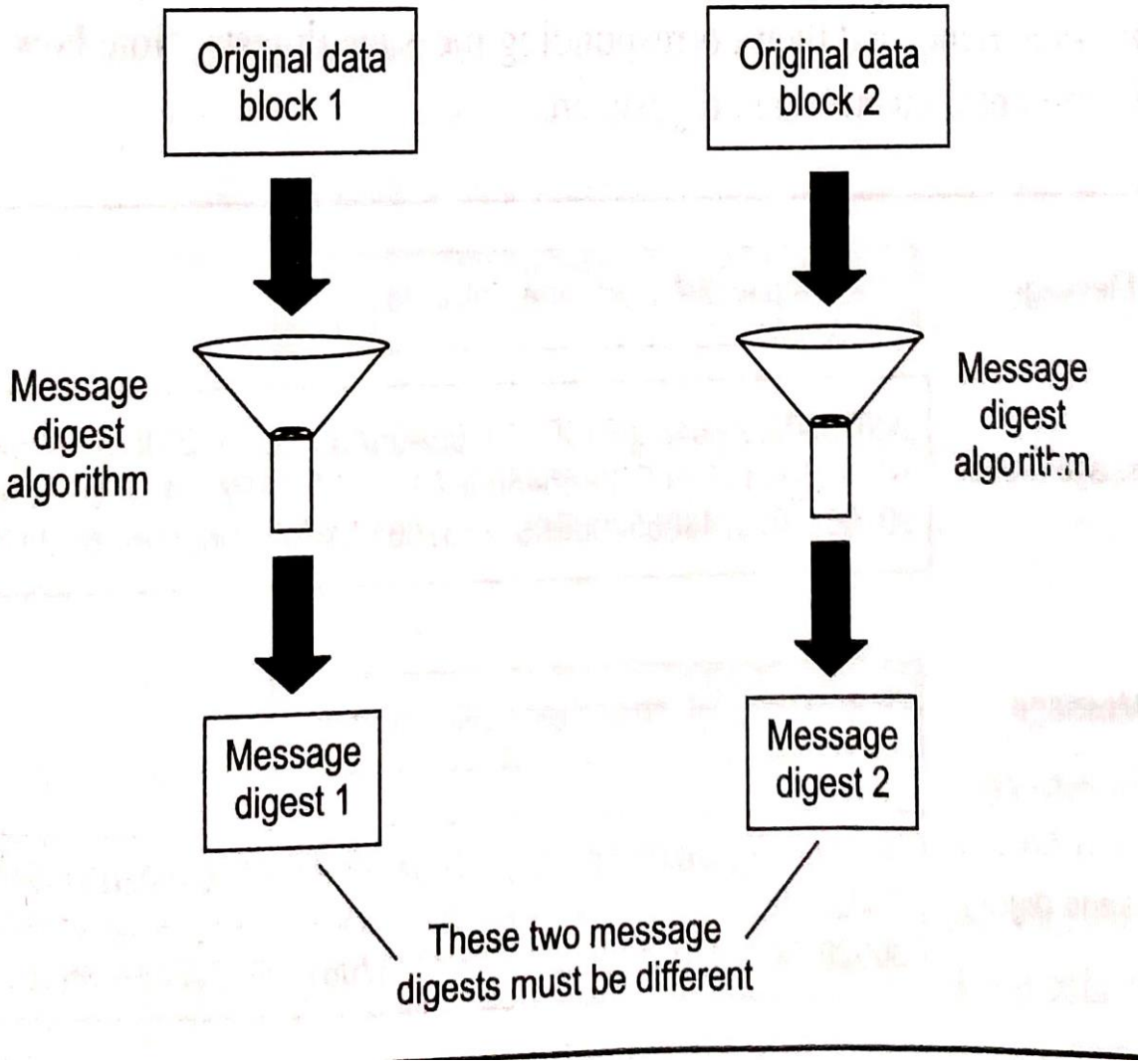
Requirements of Cryptographic Hash Function

- **R2-Pre-Image Resistance (one way property):**
- Given a message digest/hash value h, it should be very difficult to find the original message, y for which the digest such that H(y) = h
- This property protects against an attacker who only has a hash value and is trying to find the input.



Requirements of Cryptographic Hash Function / Message Digests

- **R3-Collision resistant:**
- Given any two messages, if we calculate their message digests, the two message digests must be different.
- Means if we have two messages, x and y, H(x) not equal to H(y)
- If any two messages produce the same message digest, thus violating our principle, it is called at **collision.**



Applications of Cryptographic Hash Function

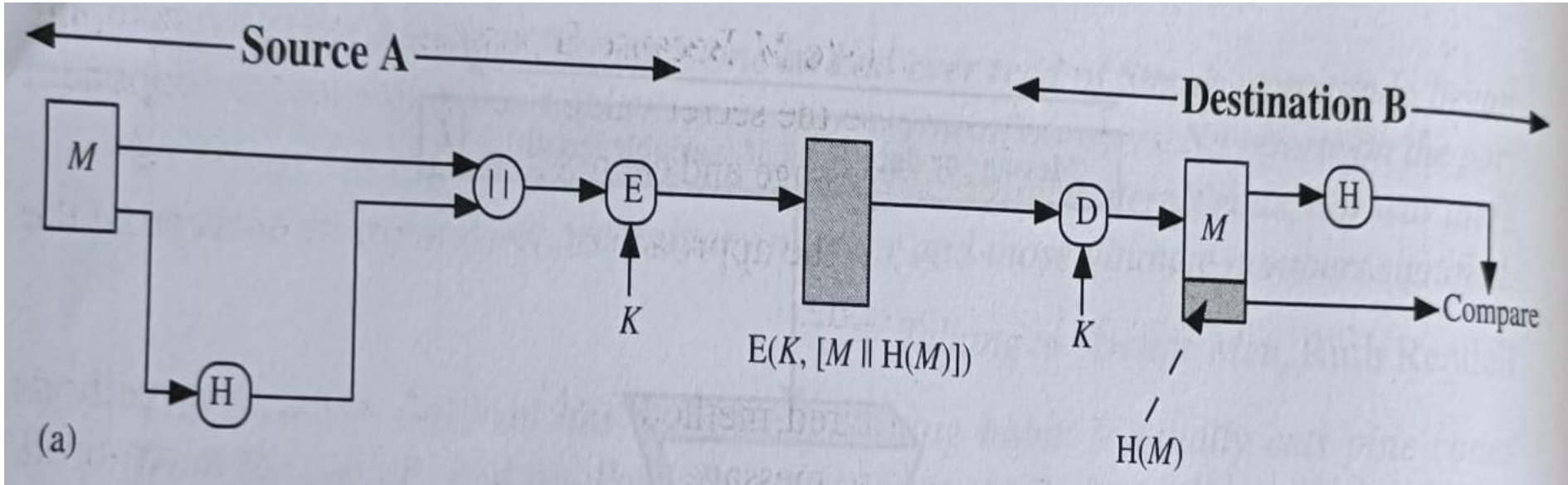
- **Message Authentication:**
- It is a mechanism or service used to verify the integrity of a message.
- Message authentication assures that data received are exactly as sent (i.e., contain no modification, insertion, deletion, or replay).
- In many cases, there is a requirement that the authentication mechanism assures that purported identity of the sender is valid.
- When **a hash function** is used to provide message authentication, the **hash function value** is often referred to as **a message digest.**

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

Applications of Cryptographic Hash Function

- **Message Authentication:**

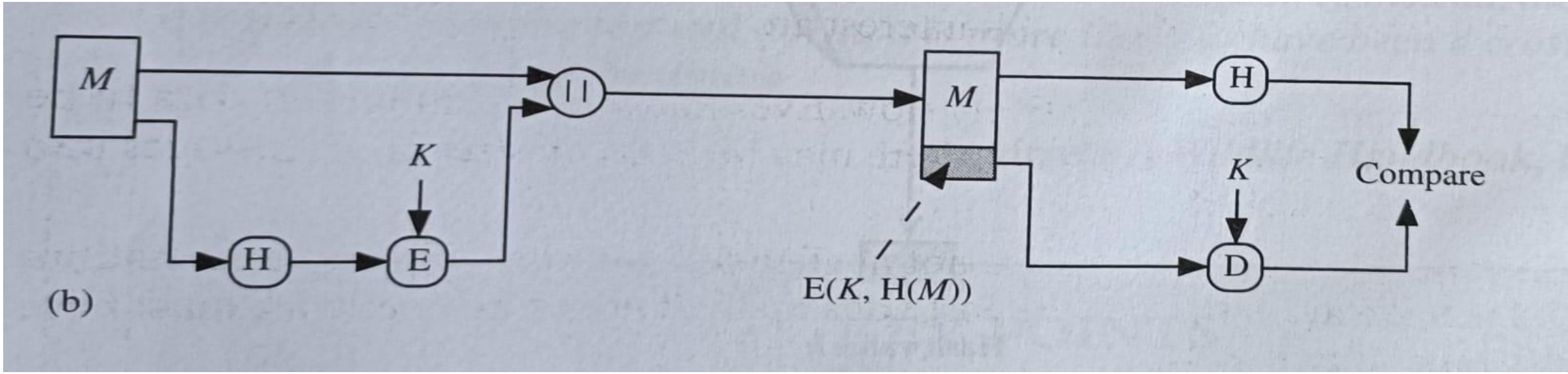
a. The **message plus concatenated hash code** is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure required to achieve **authentication**. Because encryption is applied to the entire message plus hash code, **confidentiality** is also provided.



Applications of Cryptographic Hash Function

- **Message Authentication:**

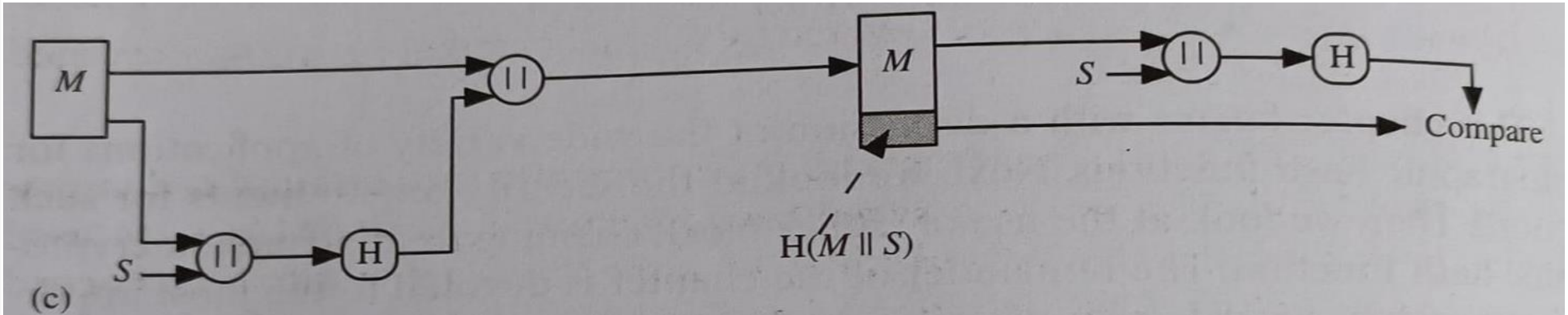
b. Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do **not require confidentiality.**



Applications of Cryptographic Hash Function

- **Message Authentication:**

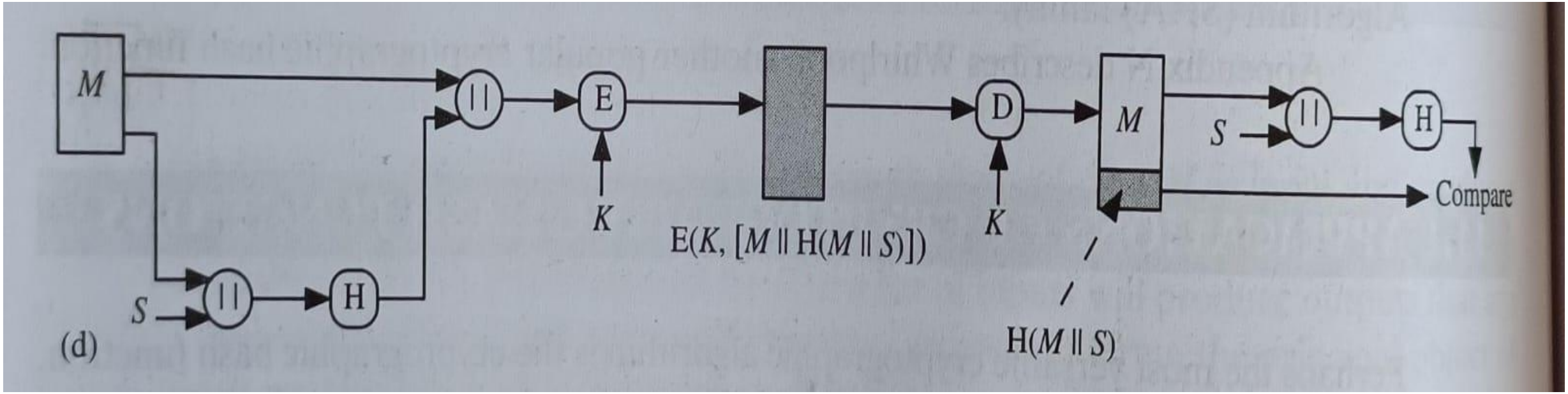
C. It is possible to use a hash function but **no encryption** for message authentication. The technique assumes that the two communicating parties **share a common secret value S**. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because **B possesses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



Applications of Cryptographic Hash Function

- **Message Authentication:**

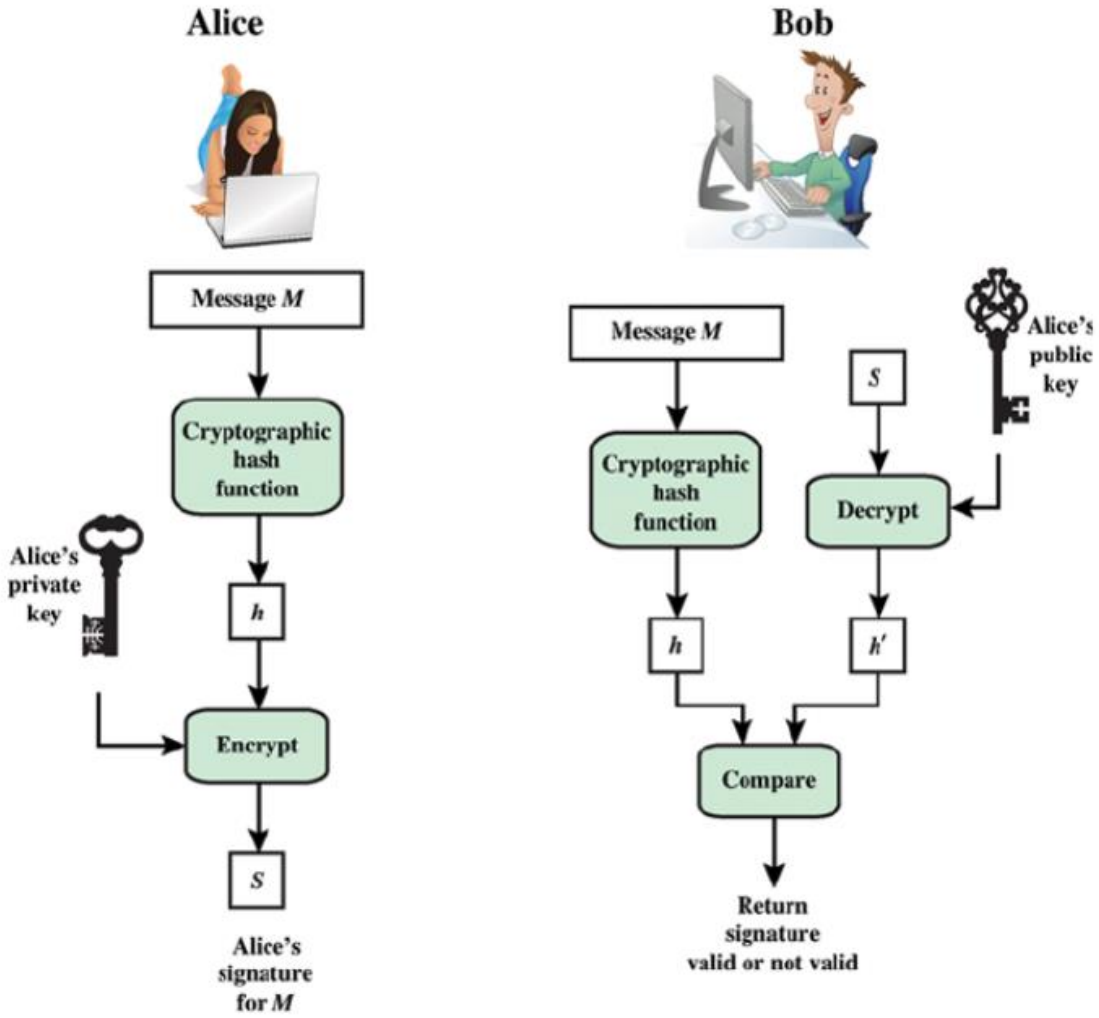
D. Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.



Applications of Cryptographic Hash Function

- **Digital Signature:**

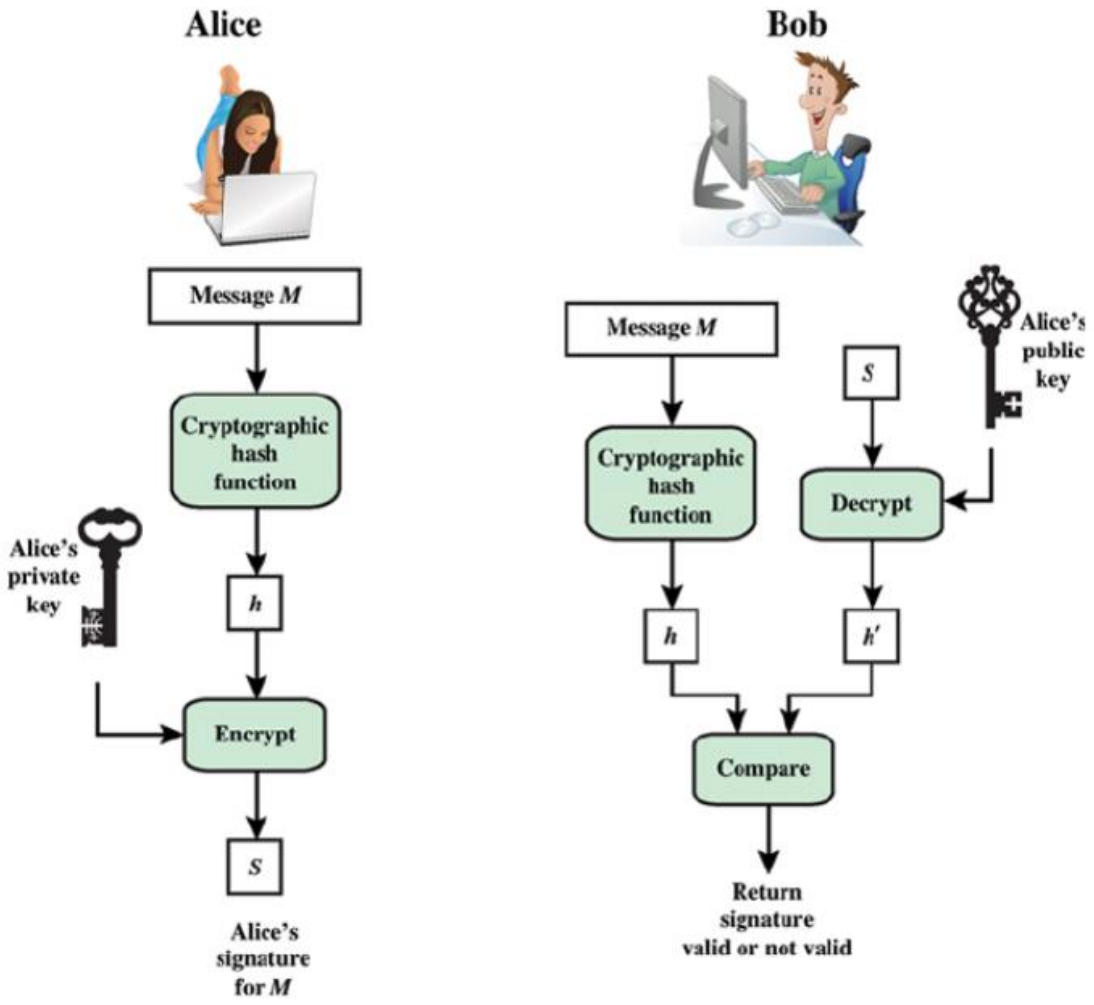
- In the physical world, it is common to use handwritten signatures on handwritten or typed messages. They are used to bind signatory to the message.
- Digital signature is a technique that **binds a person/entity to the digital data**.
- This binding can be independently verified by receiver as well as any third party.



Applications of Cryptographic Hash Function

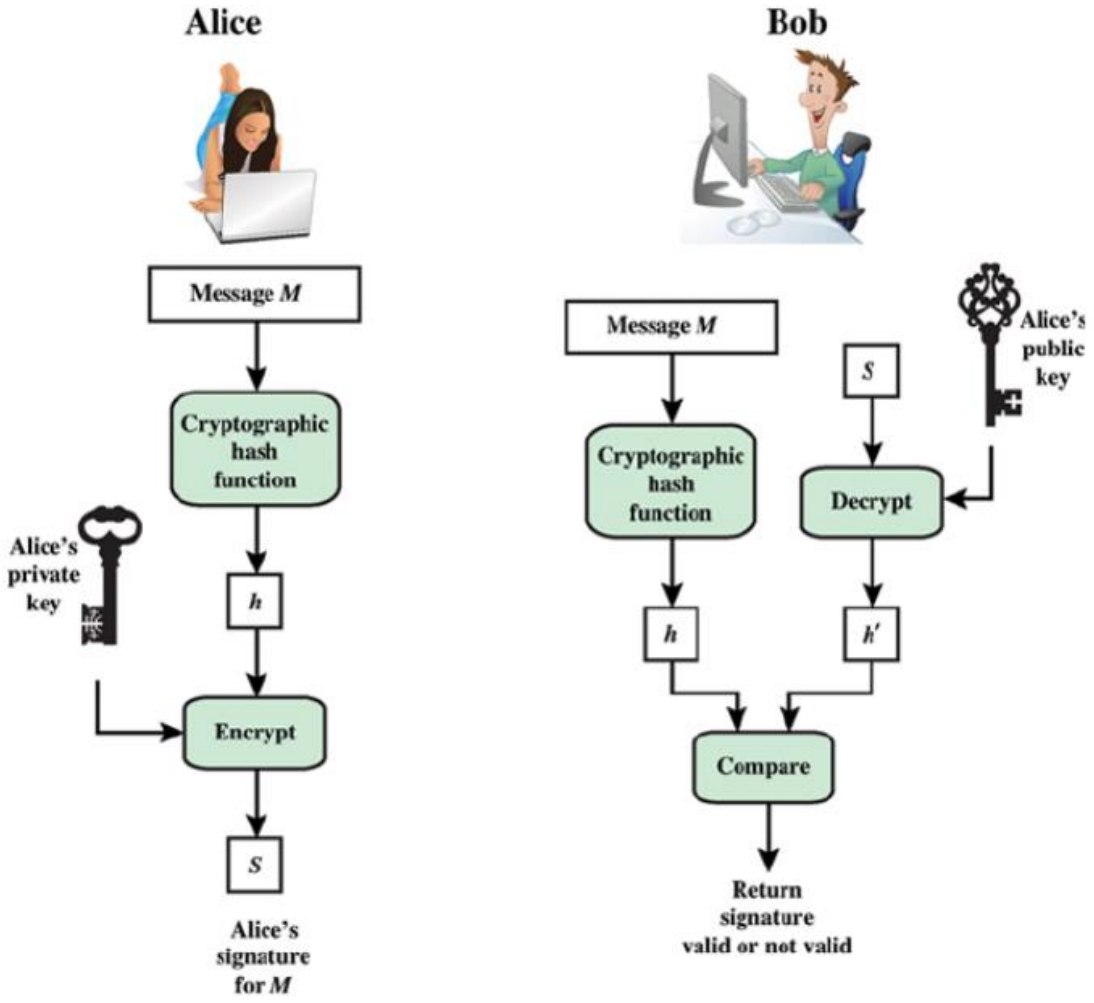
- **Digital Signature:**

- In the case of the digital signature, the hash value of a message is encrypted with a **user's private key**.
- Anyone who knows the **user's public key can verify the integrity of the message that is associated with the digital signature.**
- In this case, an attacker who wishes to alter the message would need to know the **user's private key**.



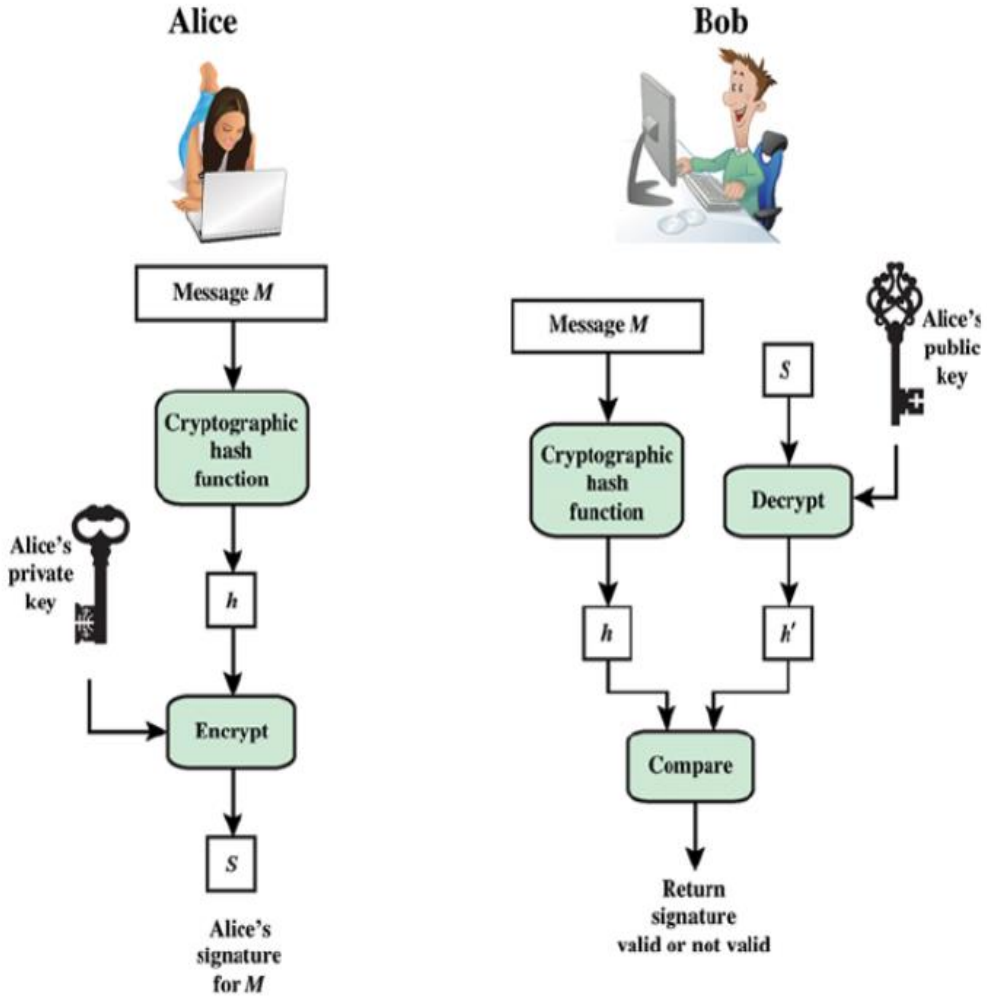
Applications of Cryptographic Hash Function

- **Digital Signature:**
- The **private key** used for signing is referred to as **the signature key** and the **public key** as the **verification key**.
- Signer feeds data to the hash function and generates hash of data.
- Hash value and signature key are then fed to the signature algorithm which produces the digital signature on given hash.
- Signature is appended to the data and then both are sent to the verifier.



Applications of Cryptographic Hash Function

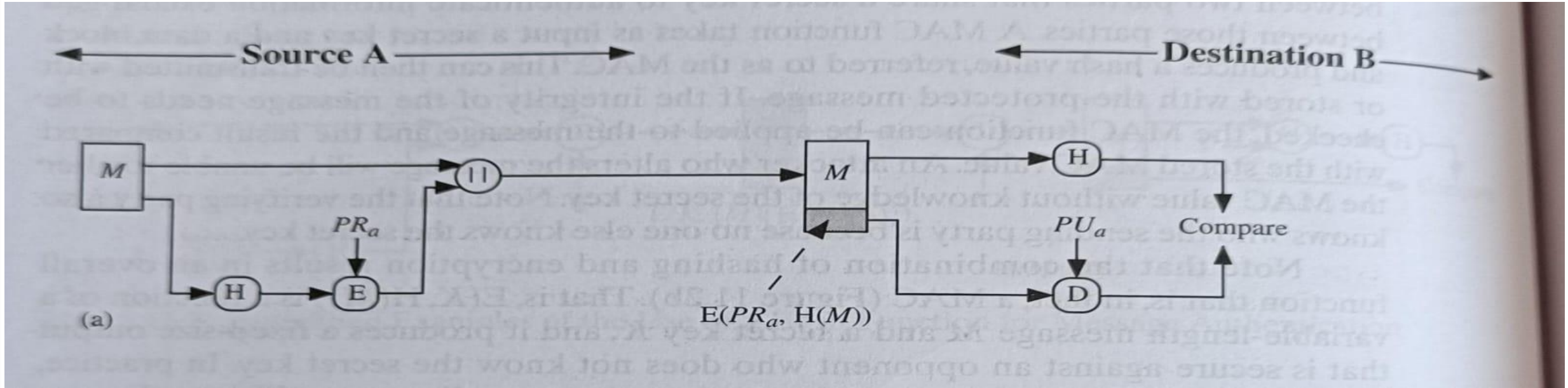
- **Digital Signature:**
- Verifier feeds the digital signature and the verification key into the verification algorithm.
- The verification algorithm gives some value as output.
- Verifier also runs same hash function on received data to generate hash value.
- For verification, this hash value and output of verification algorithm are compared.
- Based on the comparison result, verifier decides whether the digital signature is valid.
- Since digital signature is created by 'private' key of signer and no one else can have this key. the signer cannot repudiate signing the data in future.



Applications of Cryptographic Hash Function

- **Digital Signature:**
- **how a hash code is used to provide a digital signature?**

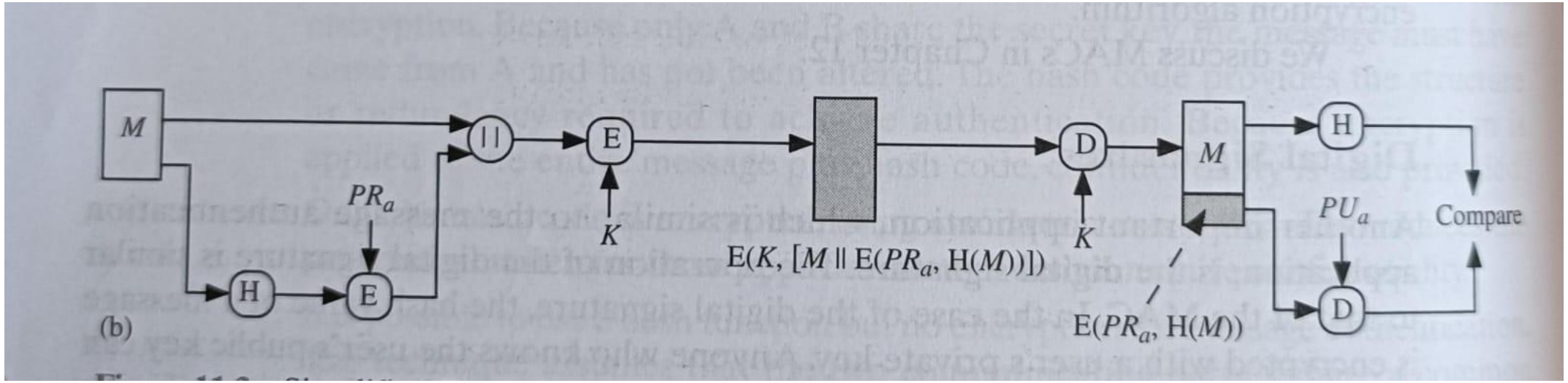
a. The hash code is encrypted with the **sender's private key**. This provides authentication. It also provides a digital signature, because only the sender could have produced the **encrypted hash code**. In fact, this is the essence of the digital signature technique.



Applications of Cryptographic Hash Function

- **Digital Signature:**
- **how a hash code is used to provide a digital signature?**

b. 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.



Two Simple Hash Functions

- To get some feel for the security considerations involved in cryptographic hash functions, we present two simple, insecure hash functions.
- All hash functions operate using the following general principles.
- The input (message, file, etc.) is viewed as a **sequence of n-bit blocks**.
- The input is processed **one block at a time** in an **iterative fashion** to produce an n-bit hash function.
- One of the simplest hash functions is the **bit-by-bit exclusive-OR (XOR) of every block**. This can be expressed as

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

where

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

Two Simple Hash Functions

- One of the simplest hash functions is the **bit-by-bit exclusive-OR (XOR) of every block**. This can be expressed as

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

where

C_i = 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
$\oplus$ = 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

- This operation produces a simple parity for each bit position and is **known as a longitudinal redundancy check.**
- It is reasonably effective for **random data as a data integrity check.**

Two Simple Hash Functions

- To increase the complexity and improve performance, use one-bit circular shift, or rotation, on the hash value after each block is processed.
- The procedure is as follows:

a. Initially set the n-bit hash value to zero

b. Process each successive n-bit block of data

B1.Rotate the current hash value to the **left by one bit**

B2.Perform XOR operation in between block and hash value

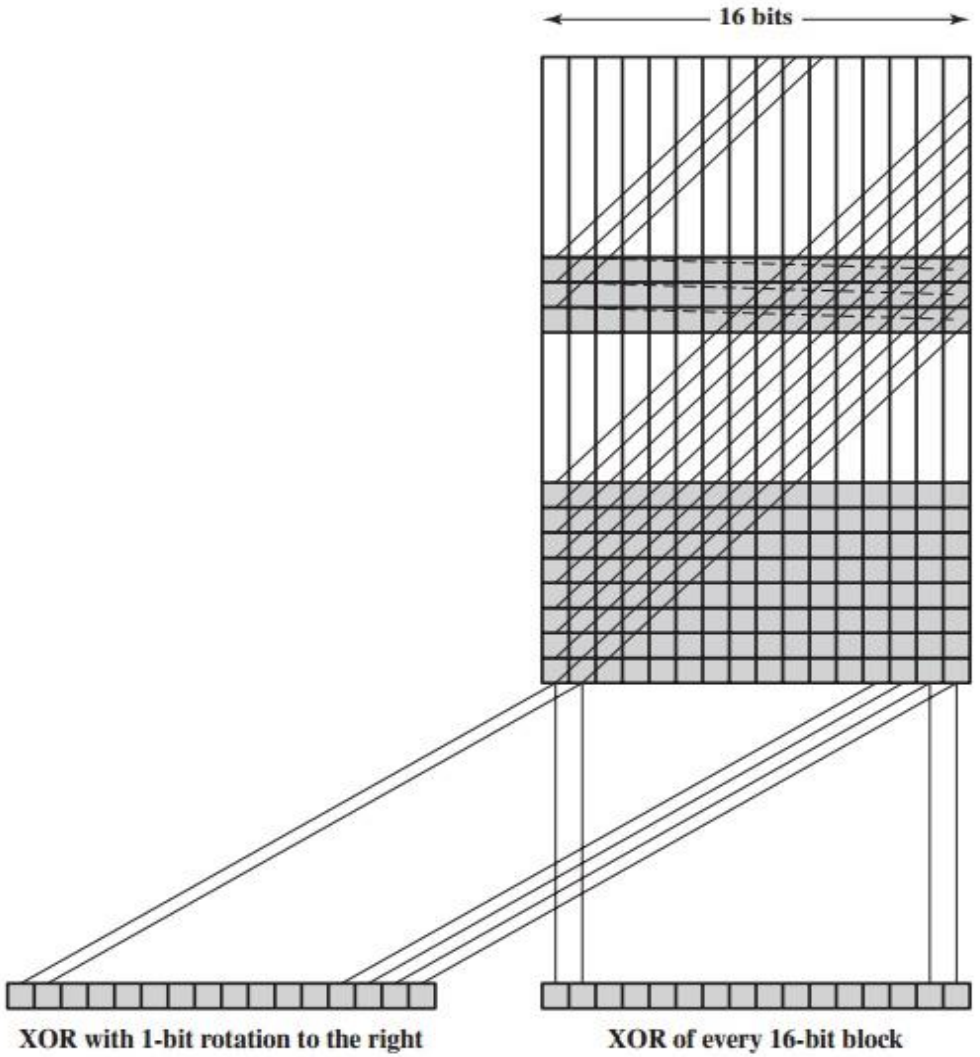


Figure 11.4 Two Simple Hash Functions

Two Simple Hash Functions

- We can define the scheme as follows:
  Given a message M consisting of a sequence of 64-bit blocks X1, X2, ……., XN, define the hash code h = H(M) as the **block-by-block XOR** of all blocks and append the hash code as the final block:

$$h = X_{N+1} = X_1 \otimes X_2 \otimes \text{Á} \otimes X_N$$

- Next, encrypt the entire message plus hash code using CBC mode to produce the encrypted message Y1, Y2, ….. , YN+1.

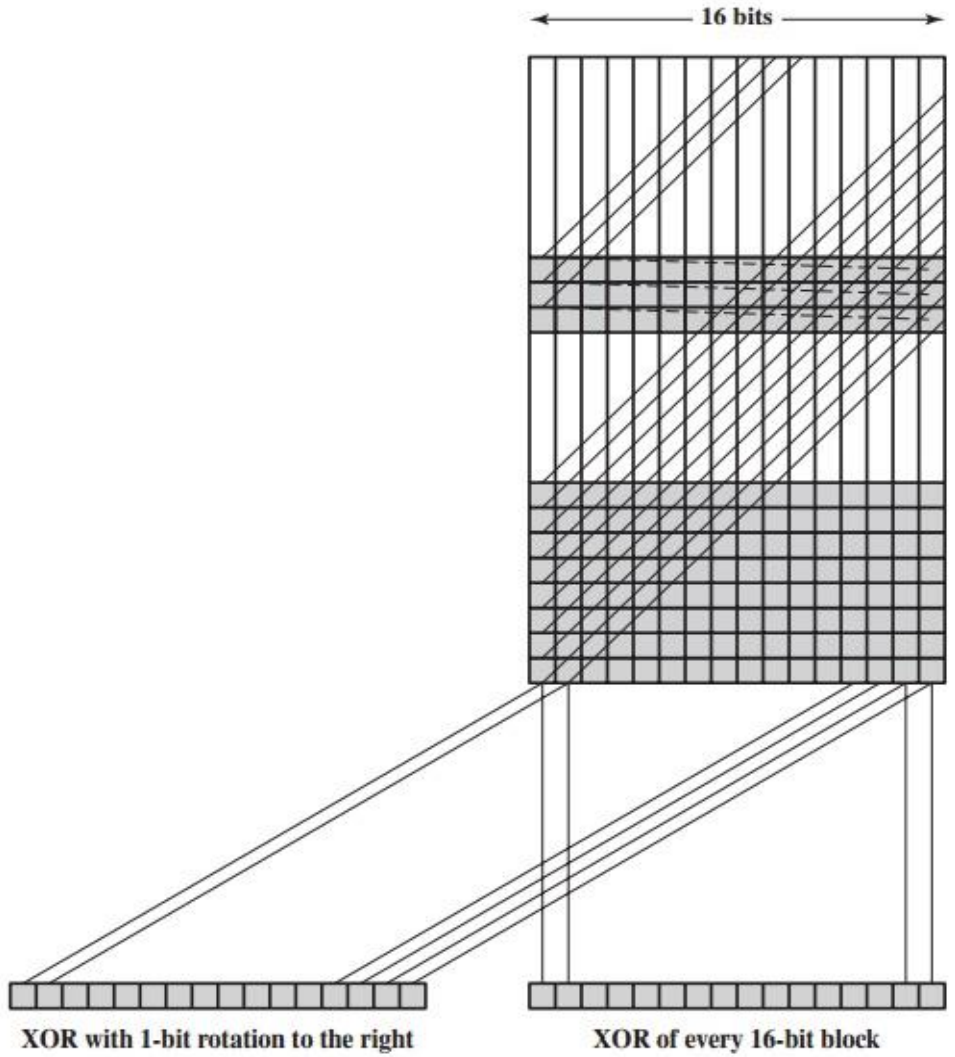


Figure 11.4 Two Simple Hash Functions

Birthday Attack

- A specific type of security attack called as **birthday attack** is used to **detect collisions in the message digest algorithm such as MD5 and SHA1.**
- It is based on principle of Birthday **Paradox.**
- It that if there are 23 people in a room, chances are more than 50% that **any two** of the people will share the same birthday.
- For instance, suppose that we have Alice, Bob and Carol as three of the 23 people in the room.
- Therefore, Alice has 22 possibilities to share a birthday with anyone else Alice will leave if no birthday matching
- Bob has 21 possibilities to share a birthday with anyone else Bob will leave if no birthday matching
- Carol has 20 possibilities to share a birthday with anyone else and so on

Birthday Attack

- If a message digest uses 64-bit keys, then after trying 2^32 transactions, an attacker can expect that for two different messages, we may get the same message digests.
- for a given message, if we compute up to N different message digests, then we can expect the first collision after the number of message digests computed exceeds square-root of N.
- In other words, a collision is expected when the probability of collision exceeds 50%.
- This can lead to birthday attacks.

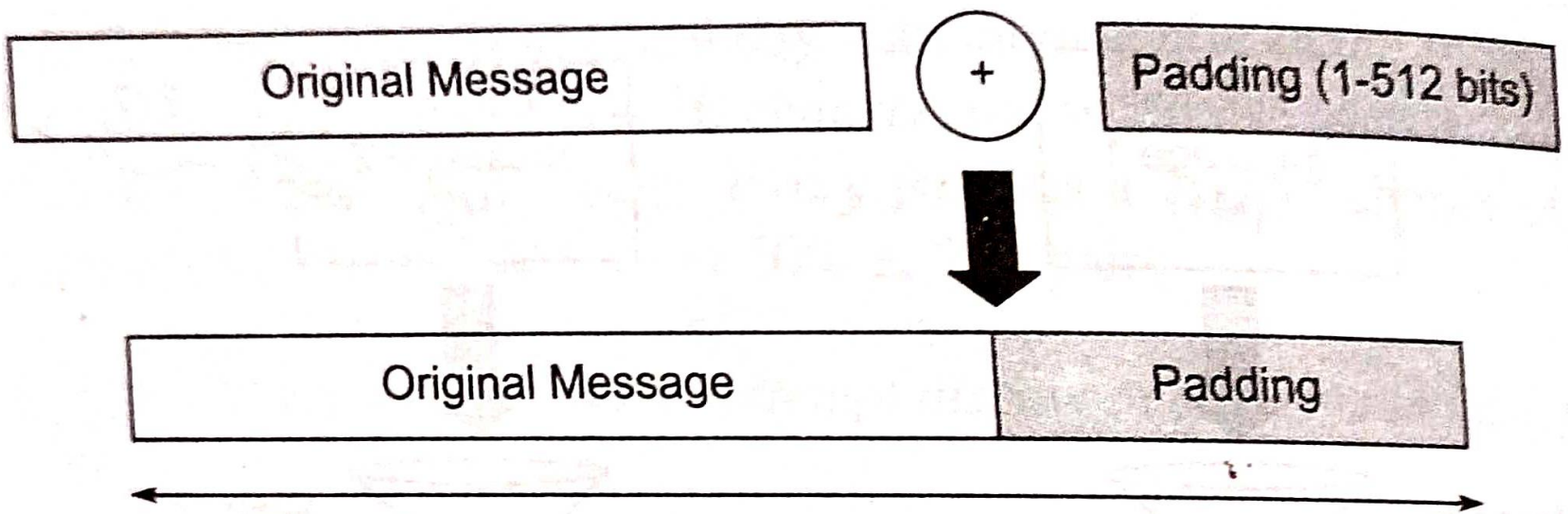
Message Digests –MD5

- Message digest algorithm developed by Ron Rivest
- Series of Message Digests algorithms
- Original was MD
- Then MD2 - weak one
- MD3 - failure
- MD4 – weak
- MD5 – fast and produces 128-bit message digests (four 32-bit blocks)
- Input 512-bit (sixteen 32-bit blocks)

MD5 Working

- **Step 1: Padding**
  - The first step in MDS is to add padding bits to the original message.
  - The aim of this step is to make the length of the original message equal to a value, which is 64 bits less than an exact multiple of 512.
  - **Example**
  - if the length of the original message is 1000 bits, we add a padding of 472 bits to make the length of the message 1472 bits.
  - This is because, if we add 64 to 1472, we get 1536, which is a multiple of 512 (because 1536 = 512 ×3).

MD5 Working



The total length of this should be 64 bits less than a multiple of 512.

For example, it can be 448 bits (448 = 512 – 64) or 960 bits (960 = [2 x 512] – 64) or 1472 (1472 = [3 x 512] – 64), etc.

Note: Padding is always added, even if the original message is already a multiple of 512.

MD5 Working

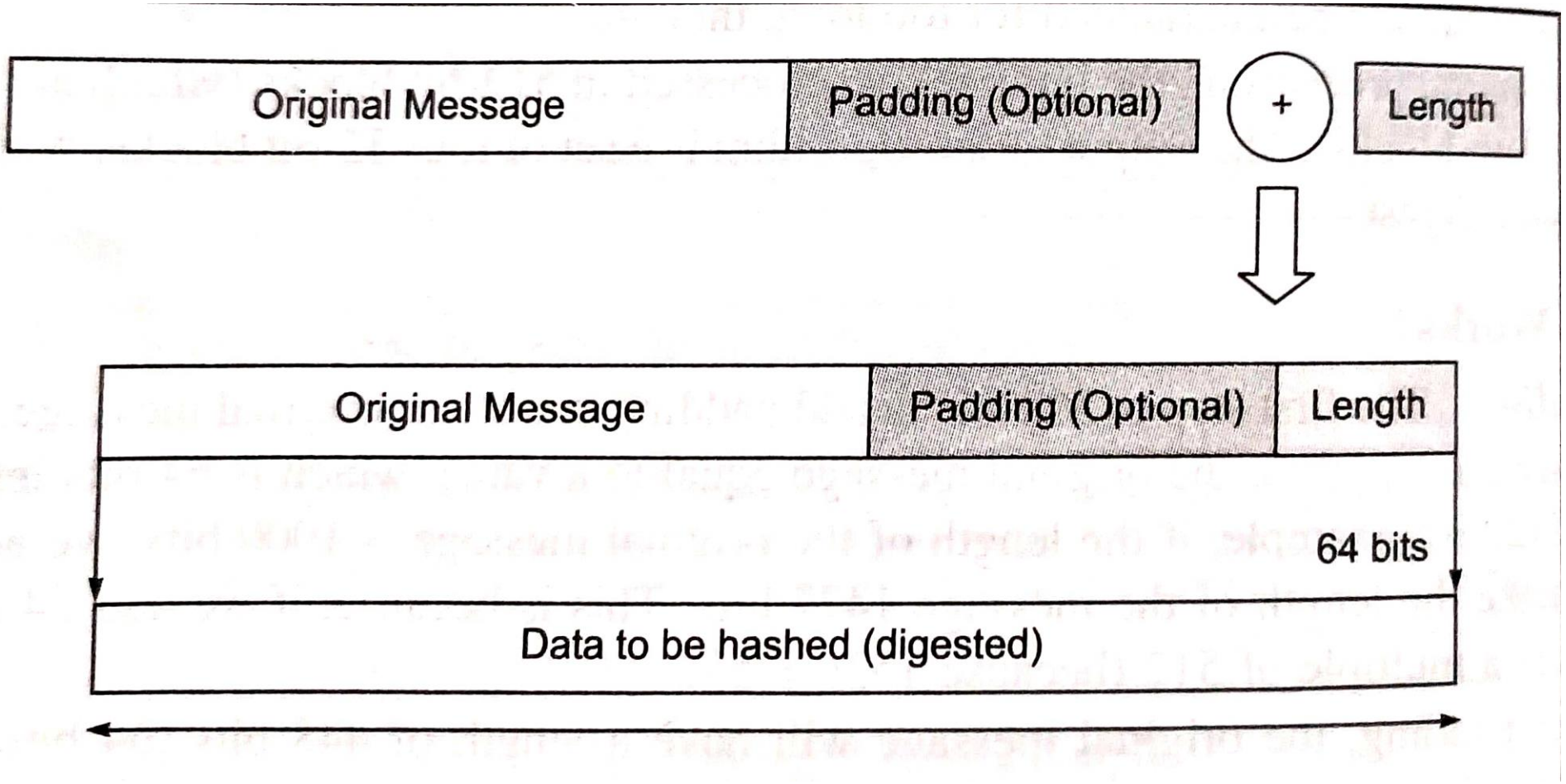
- **Step 1: Padding**
  - After padding, original message will have a length of 448 bits (64 bits less than 512), 960 bits (64 bits less than 1024), 1472 bits (64 bits less than 1536), etc.
  - The padding consists of a **single 1-bit,** followed by as **many 0-bits**, as required.
  - Padding is always added, even if the message length is already 64 bits less than a multiple of 512.
  - Thus, if the message were already of length say 448 bits, we will add a padding of 512 bits to make its length 960 bits. Thus, the padding length is any value between 1 and 512.

MD5 Working

- **Step 2: Append Length**
  - Append length after padding bits are added
  - Calculate the original length of the message and add it to the end of the message
  - The length of the message is calculated, excluding the padding bits (i.e. it is the length before the padding bits were added).
  - **Example**
  - if the original message consisted of 1000 bits and we added a padding of 472 bits to make the length of the message 64 bits less than 1536 (a multiple of 512), the length is considered as **1000** and not 1472
  - This length of the original message is now expressed as a 64-bit value and these 64 bits are appended to the end of the original message + padding.

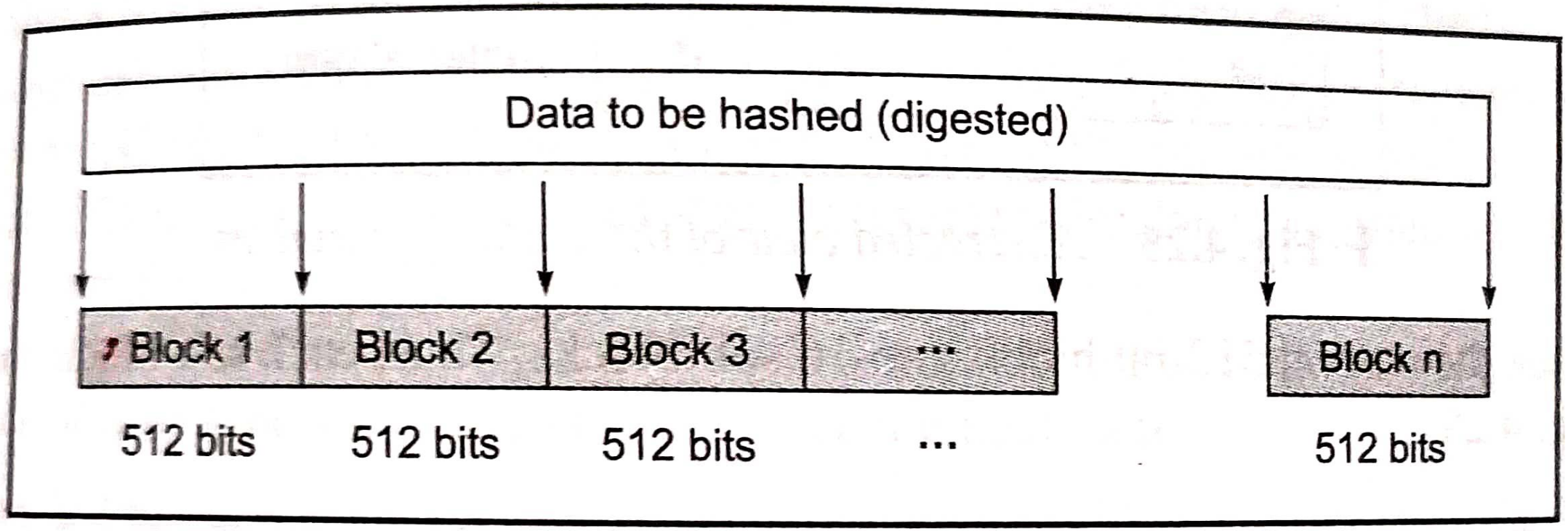
MD5 Working

- **Step 2: Append Length**



MD5 Working

- **Step 3: Divide the input into 512-bit blocks**
  - divide the input message into blocks, each of length 512 bits.



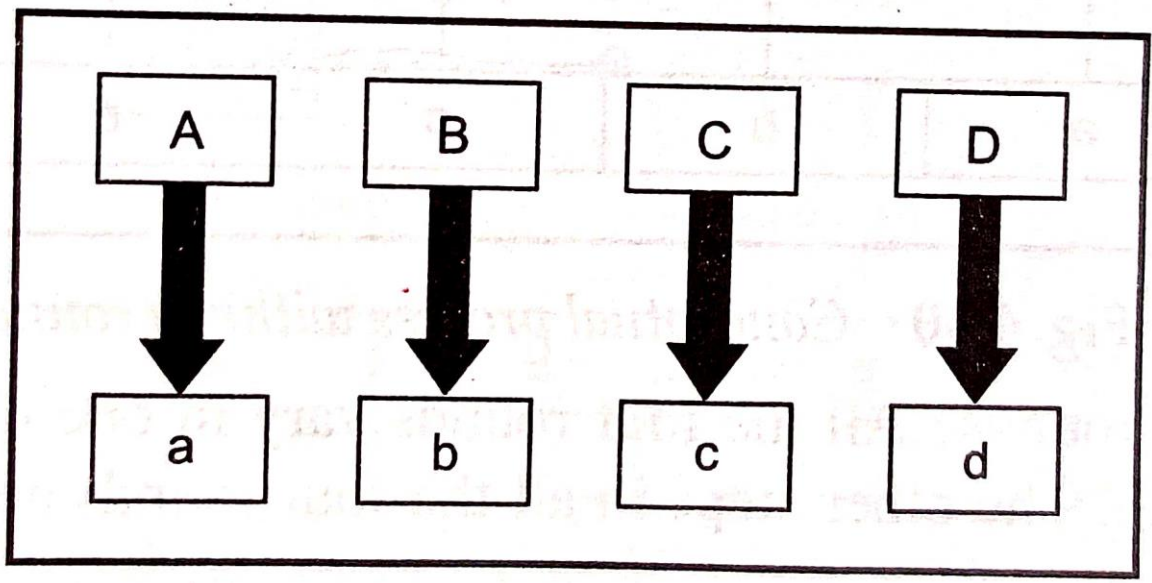
MD5 Working

- **Step 4: Initialize chaining variables**
  - In this step, four variables (called as chaining variables) are initialized.
  - They are called as A, B, C and D.
  - Each of these is a 32-bit number.
  - The initial hexadecimal values of these chaining variables

A	Hex	01	23	45	67
B	Hex	89	AB	CD	EF
C	Hex	FE	DC	BA	98
D	Hex	76	54	32	10

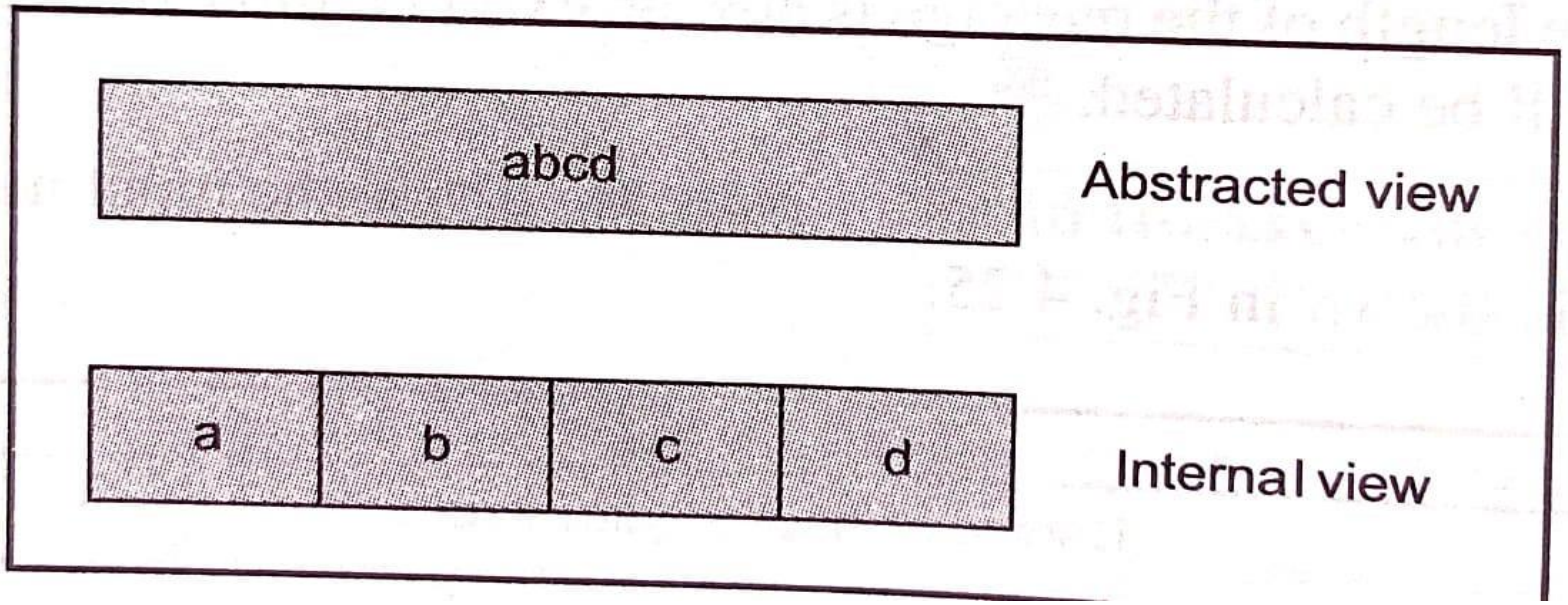
MD5 Working

- **Step 5: Process blocks**
  - After all the initializations, the real algorithm begins.
  - There is a loop that runs for as many 512-bit blocks as are in the message.
  - **Step 5.1:** Copy the four chaining variables into four corresponding variables, a, b, c and d (smaller case). Thus, we now have a = A, b = B, c=C and d = D.



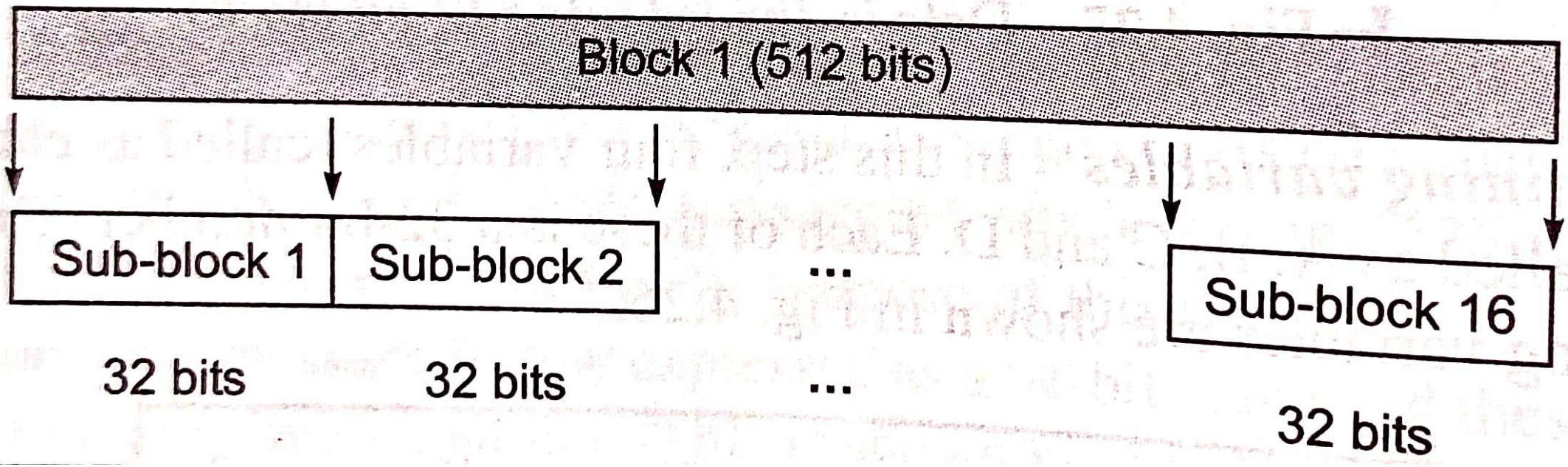
MD5 Working

- **Step 5: Process blocks**
  - **Step 5.1:** The algorithm considers the combination of a, b, c and d as a 128-bit single register (call as abcd). This register (abcd) is useful in the actual algorithm operation for holding intermediate as well as final results.



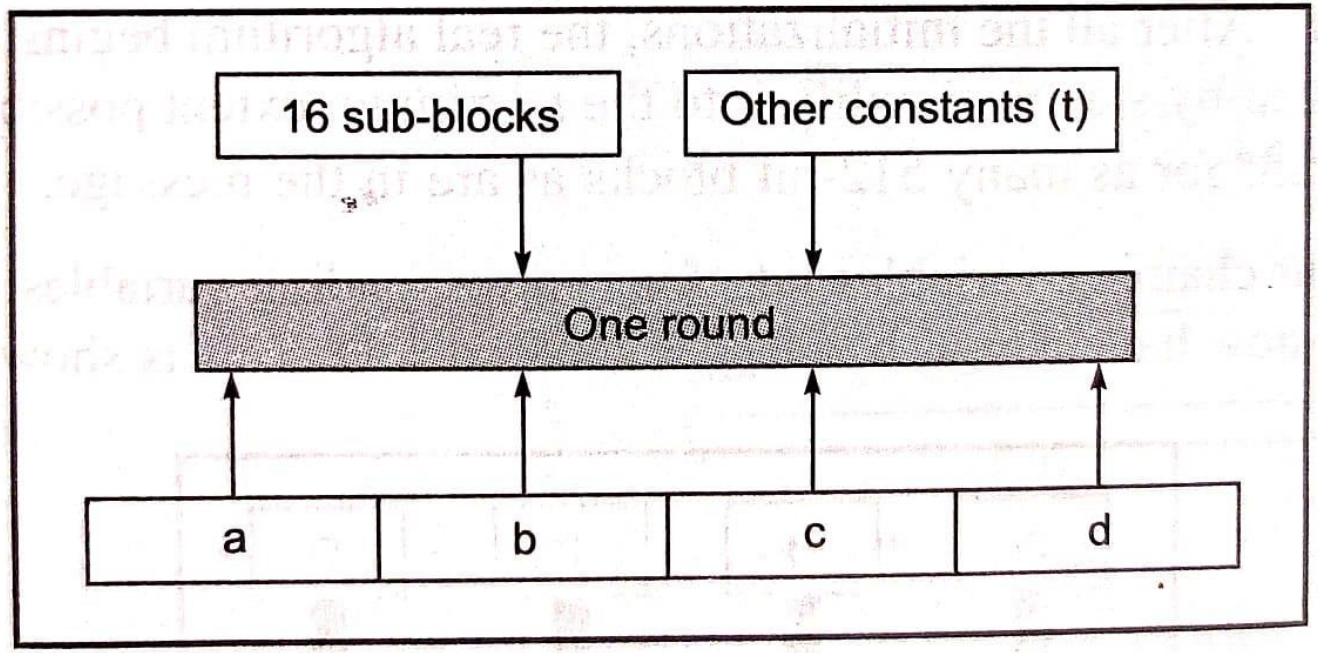
MD5 Working

- **Step 5: Process blocks**
  - **Step 5.2:** Divide the current 512-bit block into 16 sub-blocks. Thus, each sub-block contains 32 bits



MD5 Working

- **Step 5: Process blocks**
  - **Step 5.3:** Now, we have **four rounds**.
  - In each round, we process all the 16 sub-blocks belonging block.
  - The inputs to each round are: (a) all the 16 sub-blocks, (b) the variables a, b, c, d and (c) some constants, designated as t



MD5 Working

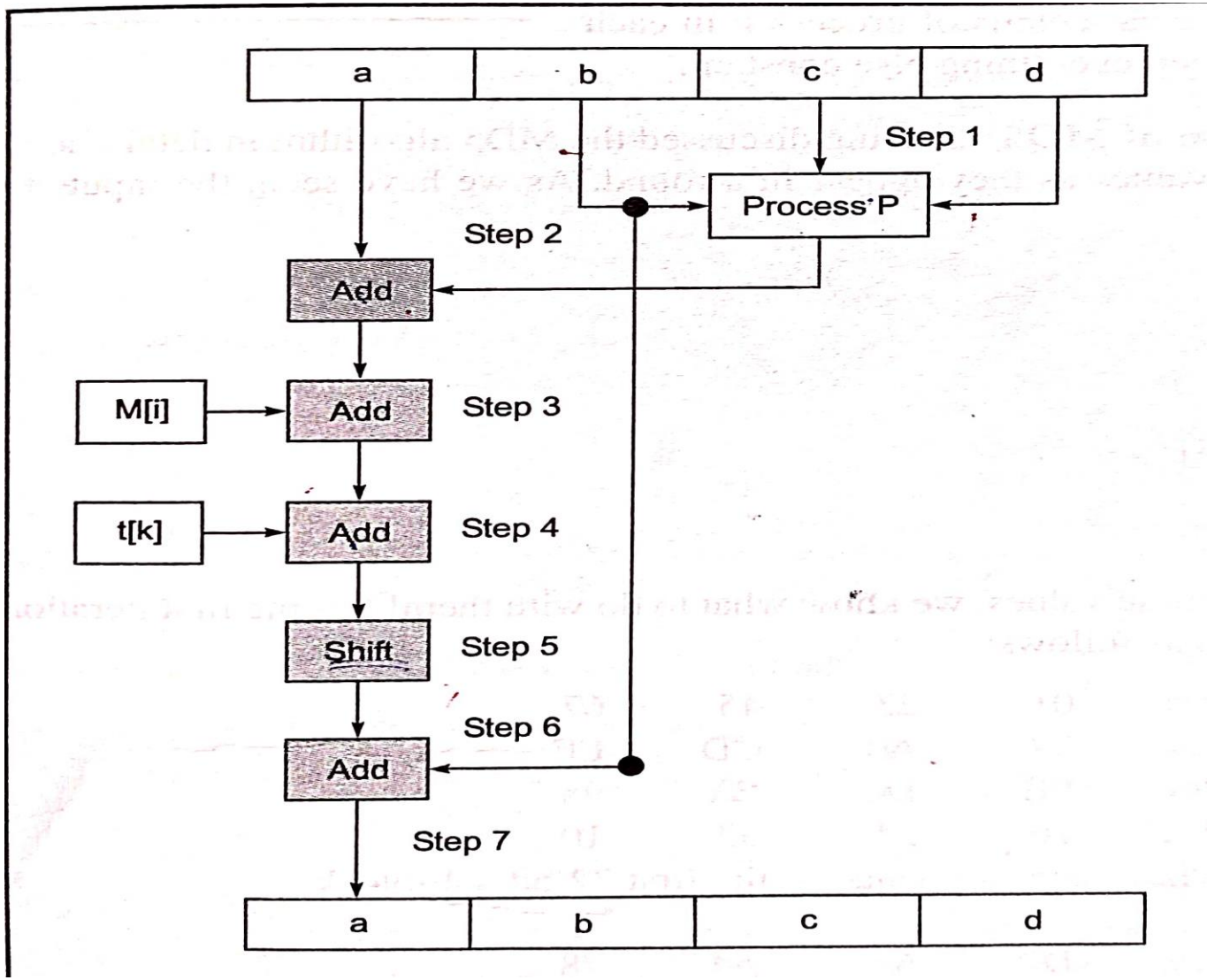
- **Step 5: Process blocks**
  - **Step 5.3:**
  - Step 1 of the four rounds has different processing.
  - The other steps in all the four rounds are the same.
  - In each round, we have 16 input sub-blocks, named M[0], M[1], ..., M[15] or in general, M[i] where i varies from 0 to 15.
  - We process all 16 blocks consists of 32-bits
  - The inputs to each round are:
    - all the 16 sub-blocks
    - the variables a, b, c, d
    - some constants, designated as t

MD5 Working

- **Step 5: Process blocks**
  - **Step 5.3:**
  - T is array of constants
  - It contains 64 elements, with each element consists of 32 bits
  - Lets say, array t as t[1],t[2] ..., t[64] or in general, t[k] where k varies from 1 to 64.
  - As there are 4 rounds, we use 16 out of 64 values of t in each round.

MD5 Working

- **Step 5.3**
- Iterations of all the four rounds is as follows
- In each case, the output of the intermediate as well as the final iteration is copied into the register abcd.
- We have 16 iterations in each round.



MD5 Working

- **Step 5.3**
- A process P is first performed on b, c and d.
- This process P is different in all the four rounds.
- The variable a is added to the output of the process P (i.e. to the register abcd).
- The message sub-block M[i] is added to the output of Step 2 (i.e. to the register abcd).
- The constant t[k] is added to the output of Step 3 (i.e. to the register abcd).
- The output of Step 4 (i.e. the contents of register abcd) is circular-left shifted by s bits. (The value of s keeps changing).

MD5 Working

- **Step 5.3**
- The variable b is added to the output of Step 5 (i.e. to the register abcd).
- The output of Step 6 becomes the new abcd for the next step.
- We can mathematically express a single MDS operation as follows:

a = b + ((a+ Process P (b. c. d) + M[i] + T[k]) <<< s)

Where,

a, b, c, d = Chaining variables

P = A non-linear operation

M[i] = M[q x 16+ i], which is the ith 32-bit word in the qth 512-bit block of the message

t[k] = A constant

<<<s = Circular-left shift by s bits

MD5 Strength

- The attempt of Rivest was to add as much of complexity and randomness as possible to the MD5 algorithm, so that no two message digests produced by MDS on any two different messages are equal.
- MD5 has a property that every bit of the message digest is some function of every bit in the input.
- The possibility that two messages produce the same message digest using MDS is in the order of 2^64 operations.
- Given a message digest, working backwards to find the original message can lead up to 2^128 operations.

Attacks on MD5

- Tom Berson could find two messages that produce the **same message digest for each of the four individual rounds.** However, he could not come up with two messages that produce the same message digest for all the four rounds taken together.
- Den Boer and Bosselaers showed that the execution of MD5 on a single block of 512 bits will produce the **same output for two different values in the chaining variable register abcd.** This is called as **pseudocollision.** However, they could not extend this to a full MD5 consisting of four rounds, each containing 16 steps.
- Dobbertin provided the most serious attack on MDS. Using his attack, the operation of **MD5 on two different 512-bit blocks produces the same 128-bit output.** However, this has not bees generalized to a full message block.

MD5 Example

- Original message: **They are deterministic**
- **Step 1:**
  - Identify ASCII for each character (Hint: A-Z : 65-90 in decimal, a-z: 97-122 in decimal, space: 20 in hex)
  - ASCII for T is 84 in decimal, h is 104 in decimal and so on
- **Step 2:**
  - Convert decimal values into the hex format and then into the binary format **OR** directly convert ASCII into binary format
  - “T” is written as “01010100” in binary
  - A lowercase “h” is “01101000”
  - A lowercase “e” is “01100101”
  - A lowercase “y” is “01111001”
  - The binary code for a space (SP) is “00100000” etc.

MD5 Example

- **Step 3:**

  our input **They are deterministic** is written in binary as:
  01010100 01101000 01100101 01111001 00100000 01100001 01110010 01100101 00100000 01100100 01100101 01110100 01100101 01110010 01101101 01101001 01101110 01101001 01110011 01110100 01101001 01100011

- **Step 4: Padding**
  - Inputs in MD5 are broken up into 512-bit blocks, with padding added to fill up the rest of the space in the block.
  - Our input is 22 characters long including spaces, and each character is 8 bits long, so **total bits = 22 * 8 = 176**

MD5 Example

- **Step 4: Padding**
  - With an input of only 176 bits and a 512-bit block that needs to be filled, we need 336 bits of padding to complete the block.
  - **One block size = 512 bits, reserved length = 64 bits**
  - **Total padding bits required = 512 – 64 – message size = 448 – 176 = 272 bits**
  - After laying out the initial 176 bits of binary that represent our input, the rest of the block is padded with a single one, then enough zeros to bring it up to a length of 448 bits. So: **448 – 1 – 176 = 271**
  - The padding for this block will include a one, then an extra 271

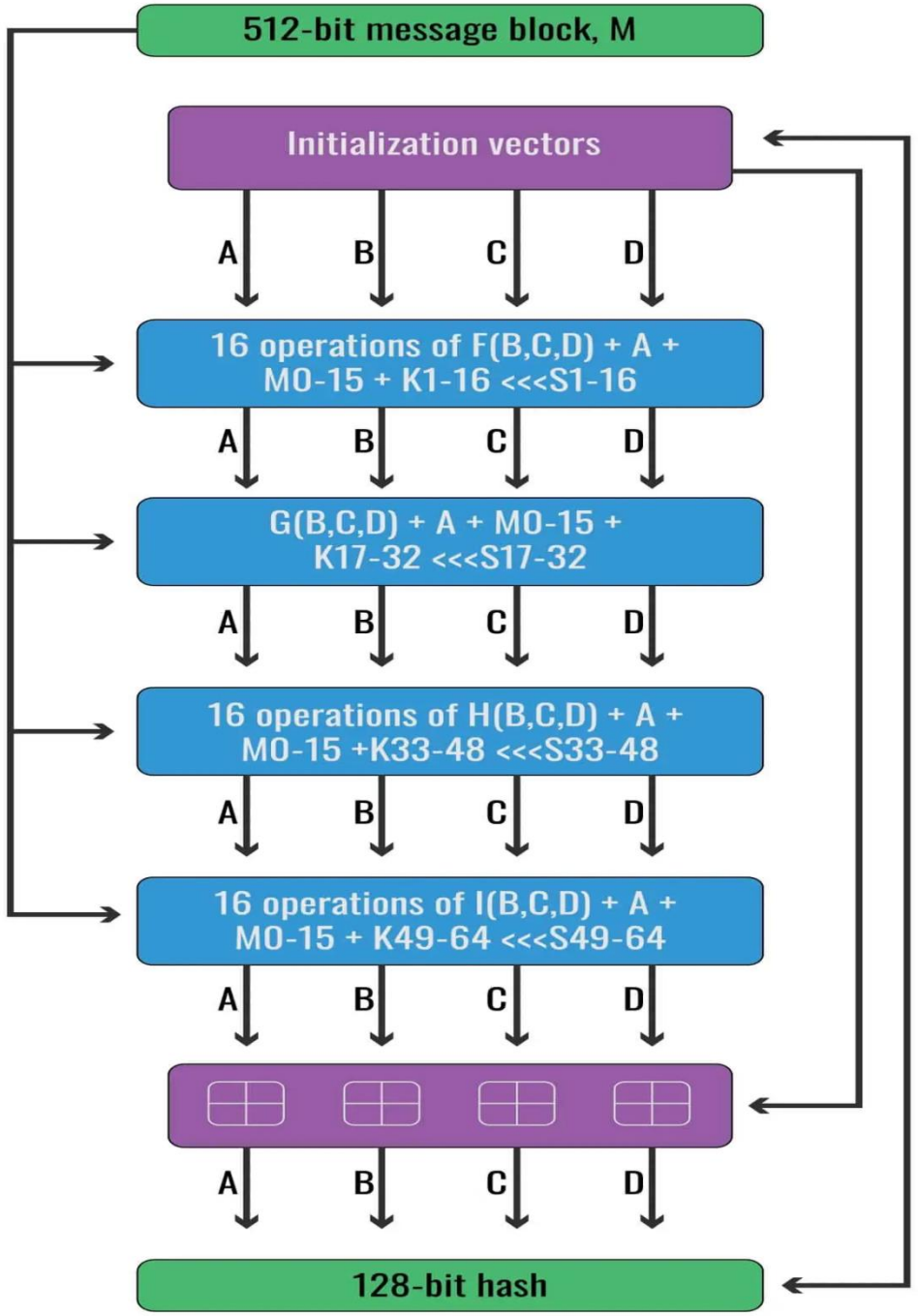
MD5 Example

- **Step 5: Append Length**
  - The reason we only need to pad it up to 448 bits (instead of 512) is because the final 64 bits (512 – 64 = 448) are reserved to display the message's length in binary.
  - In our example, the number **176 is 10110000 in binary.**
  - This forms the very end of the padding scheme, while the **preceding 56 bits** (64 minus the 8 bits that make up 10110000) are all **filled up with zeros.**

  - **00000000 00000000 00000000 00000000 00000000 00000000 00000000 10110000**

MD5 Example

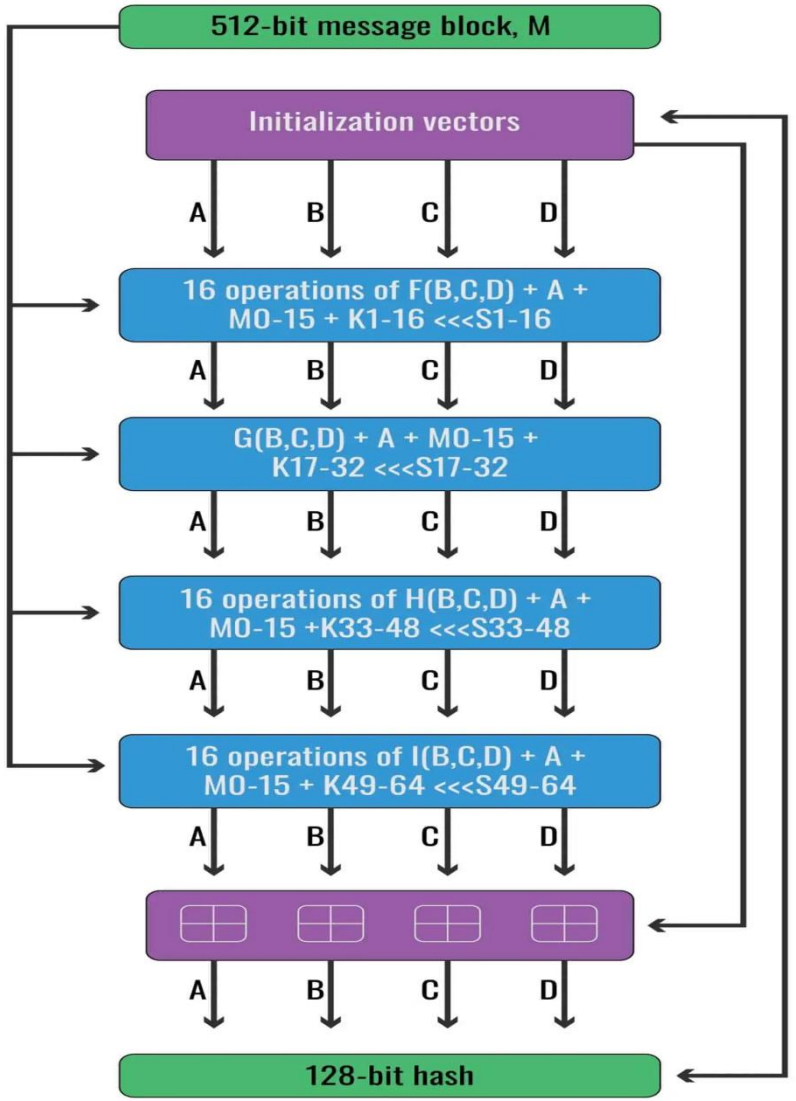
- **Step 6: The input M**
- As we can see four “16 operations” rectangles. **Each of these four rectangles are called rounds**, and each of them are composed of a series of sixteen operations
- This means that **our input, M, is an input in each of these four stages**. However, before it can be used as an input, **our 512-bit M needs to be split into sixteen 32-bit “words”.**
- Each of these words is assigned its own number, ranging from M0 to M15.



MD5 Example

- **Step 6: The input M :** In our example, these 16 words are:

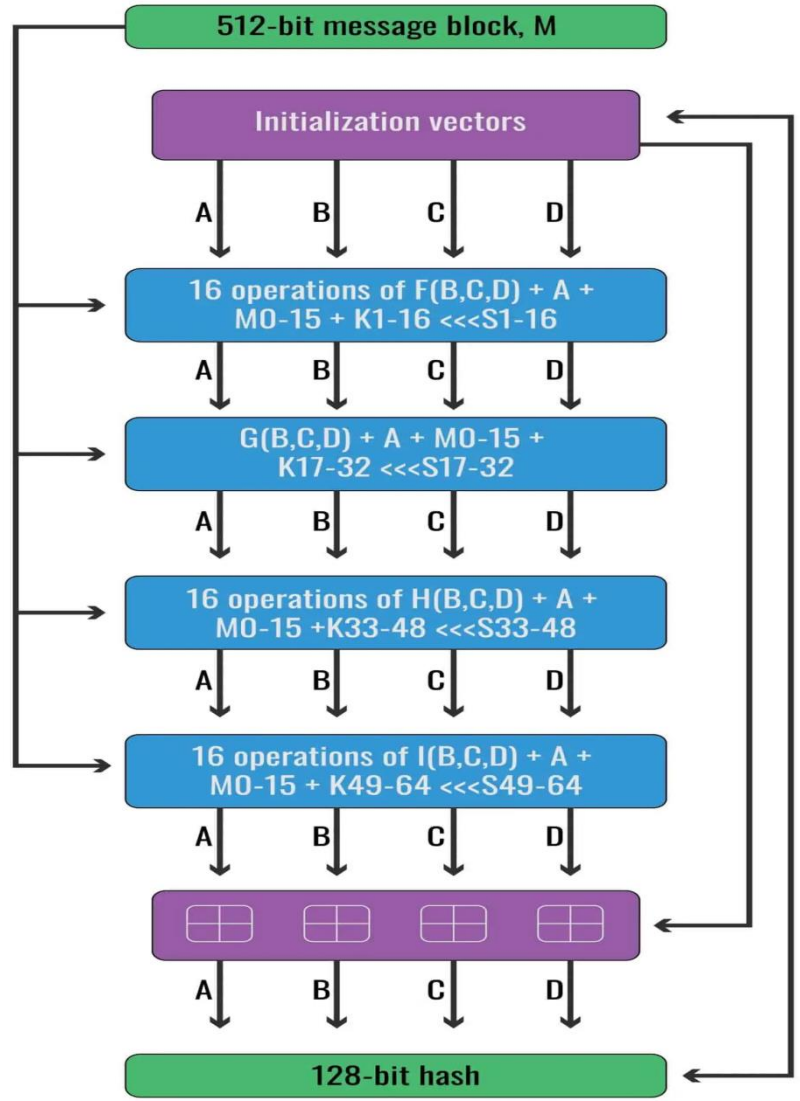
M_0 – 01010100 01101000 01100101 01111001
M_1 – 00100000 01100001 01110010 01100101
M_2 – 00100000 01100100 01100101 01110100
M_3 – 01100101 01110010 01101101 01101001
M_4 – 01101110 01101001 01110011 01110100
M_5 – 01101001 01100011 10000000 00000000
M_6 – 00000000 00000000 00000000 00000000
M_7 – 00000000 00000000 00000000 00000000
M_8 – 00000000 00000000 00000000 00000000
M_9 – 00000000 00000000 00000000 00000000
M_{10} – 00000000 00000000 00000000 00000000
M_{11} – 00000000 00000000 00000000 00000000
M_{12} – 00000000 00000000 00000000 00000000
M_{13} – 00000000 00000000 00000000 00000000
M_{14} – 00000000 00000000 00000000 00000000
M_{15} – 00000000 00000000 00000000 10110000



MD5 Example

- **Step 6: The input M :** In hexadecimal forms, these 16 words are:

M_0 – 54686579
M_1 – 20617265
M_2 – 20646574
M_3 – 65726D69
M_4 – 6E697374
M_5 – 69638000
M_6 – 00000000
M_7 – 00000000
M_8 – 00000000
M_9 – 00000000
M_{10} – 00000000
M_{11} – 00000000
M_{12} – 00000000
M_{13} – 00000000
M_{14} – 00000000
M_{15} – 000000B0



MD5 Example

- **Step 7:**
- Each of these sixteen values act as inputs to the complex set of operations that are represented by each “16 operations of…” rectangle.
- While each of these M inputs are used in every single round, they are added in different orders.
- In the first round, the M inputs are added into the algorithm sequentially, e.g.
  M0, M1, M2… M15.
- In the second round, the M inputs are added in the following order:
  M1, M6, M11, M0, M5, M10, M15, M4, M9, M14, M3, M8, M13, M2, M7, M12
- In the third round, the M inputs are added in this sequence:
  M5, M8, M11, M14, M1, M4, M7, M10, M13, M0, M3, M6, M9, M12, M15, M2
- In the fourth round, the M inputs are added in the following order:
  M0, M7, M14, M5, M12, M3, M10, M1, M8, M15, M6, M13, M4, M11, M2, M9

- **Step 8 onwards: after above steps, constant variables, K1 to K64 are generated and perform the process along with 4 chaining variables A, B, C, D which is complex one. At the end, we get 128-bit message digest as output**

References:

- Atul Kahate,"Cryptography and Network Security", second edition, Tata McGraw Hill
- William Stallings, "Cryptography and Network Security-Principles and practice"