Chapter 4
Message Authentication and Digital Signatures

- By

Jyoti Tryambake

Message Authentication

- A message, file, document, or other collection of data is said to be authentic when it is genuine and came from its alleged source.
- Message authentication is a procedure that allows communicating parties to verify that received message is authentic.
- The important aspects are
  - To verify that the contents of the message have not been altered
  - The source is authentic.
  - To verify a message's timeliness and sequence relative to other messages flowing between two parties.

Message Authentication Techniques

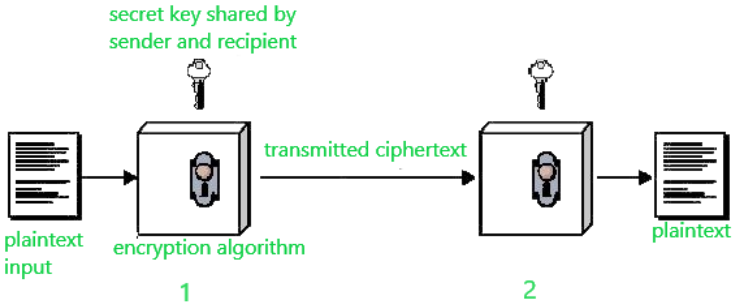
- Encryption
- MAC – Message Authentication Code - fixed length code
- Hash Function – H(M) – fixed length code

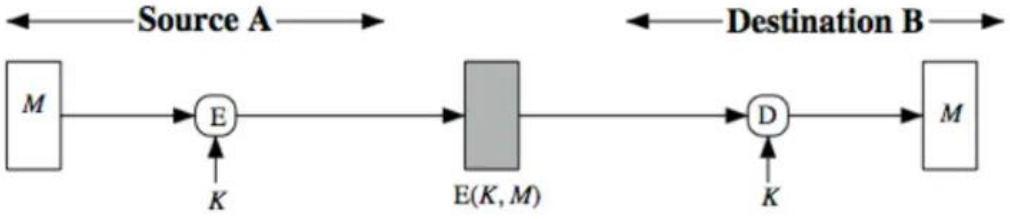
Message Authentication Techniques (cont.)

- **Authentication using Conventional Encryption**

  Symmetric Encryption -

  - Assume only sender and receiver share a key
  - Single key for both encryption and decryption
  - the sender encrypts plaintext using the receiver's secret key, which can be later used by the receiver to decrypt the ciphertext.





(a) Symmetric encryption: confidentiality and authentication

Message Authentication Techniques (cont.)

- **Authentication using Conventional Encryption**

  Asymmetric Encryption

  - Public and private keys

Message Authentication Techniques (cont.)

- **Authentication using Conventional Encryption**

  Asymmetric

  i. (A-sender) Message -> E (Public key of B)-> Cipher-> D(Private key of B) -> (B - receiver) Message –

  Authentication - ✗, Confidentiality - ✓

  ii. (A-sender) Message -> E (Private key of A)-> Cipher-> D(Public key of A) -> (B - receiver) Message

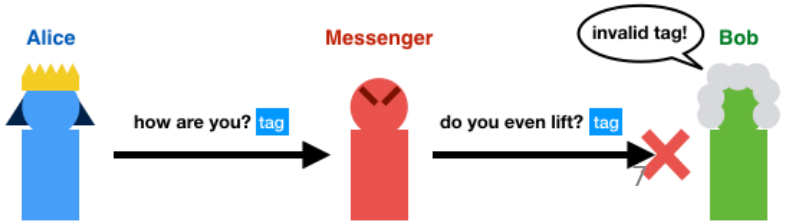
  Authentication - ✓, Confidentiality - ✗

  iii. (A-sender) Message -> E (Private key of A)-> Cipher1-> E(Public key of B) -> Cipher2-> D(Private key of B)-> Decipher 1 -> D(Public key of A)-> (B - receiver) Message

  Authentication - ✓, Confidentiality - ✓

Message Authentication Techniques (cont.)

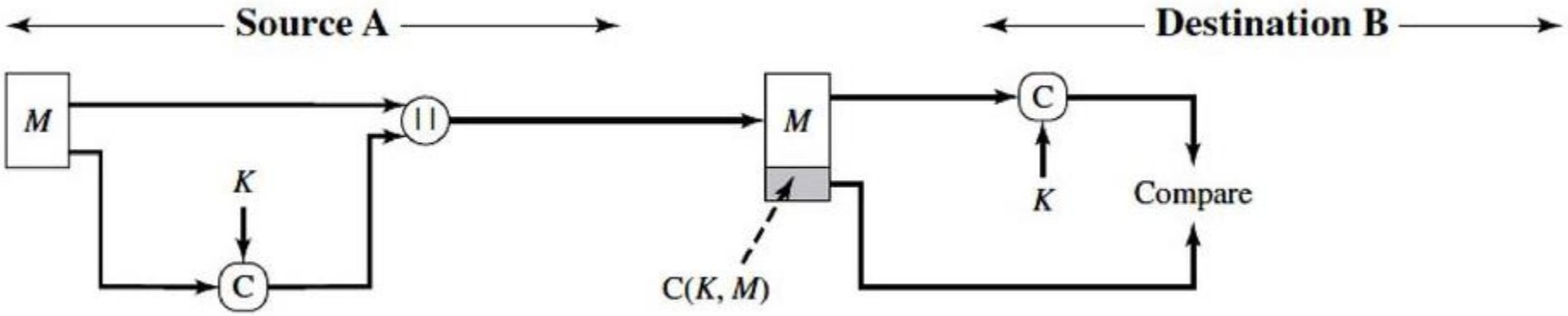
- **Authentication without Message Encryption**
  - An authentication tag is generated and appended to each message
  - The algorithm uses it to verify whether or not the ciphertext and/or associated data have been modified.
  - If either the ciphertext or associated data has been modified, then the procedure that re-computes the validation tag on the receiving end will end up generating a different tag. The algorithm will check the re-computed tag against the tag that was bundled with the ciphertext and associated data (which collectively can be referred to as a "cryptogram").
  - If the tags don't match, that means some part of the ciphertext and/or associated data have been modified.



Message Authentication Techniques (cont.)

Message Authentication Code(MAC)

- MAC algorithm is a symmetric key cryptographic technique to provide message authentication.
- For establishing MAC process, the sender and receiver share a Symmetric key K.
- Essentially, a MAC is an encrypted checksum generated on the underlying message that is sent along with a message to ensure message authentication.

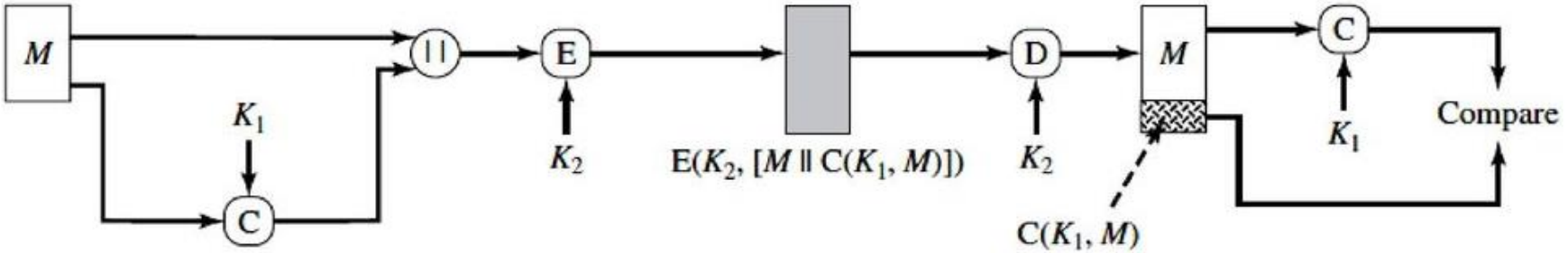


(a) Message authentication

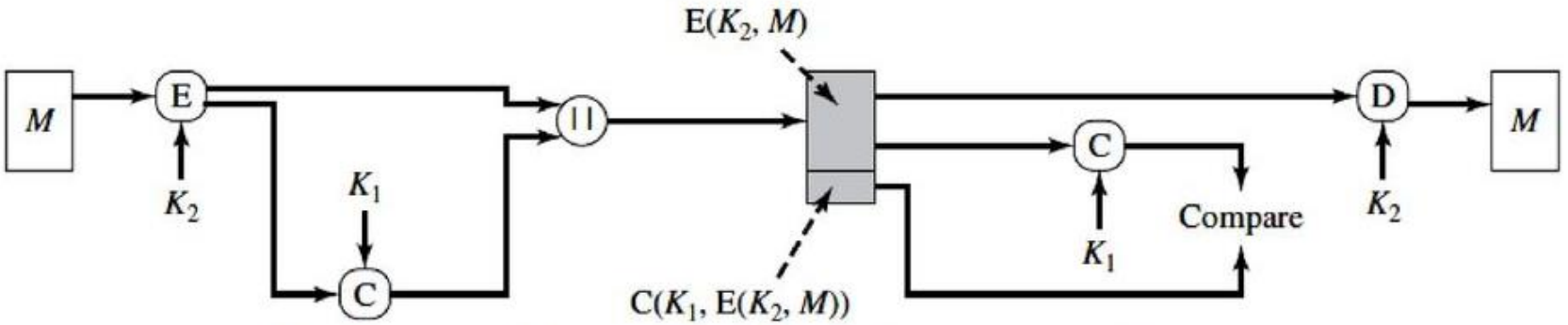
Message Authentication Code(MAC) Process:

- The sender uses MAC algorithm, inputs the message and the secret key K and produces a MAC value.
- Similar to hash, MAC function also compresses an arbitrary long input into a fixed length output.
- The sender forwards the message along with the MAC.
- On receipt of the message and the MAC, the receiver feeds the received message and the shared secret key K into the MAC algorithm and re-computes the MAC value.
- The receiver now checks equality of freshly computed MAC with the MAC received from the sender. If they match, then the receiver accepts the message and assures himself that the message has been sent by the intended sender.
- If the computed MAC does not match the MAC sent by the sender, the receiver cannot determine whether it is the message that has been altered or it is the origin that has been falsified.
- As a bottom-line, a receiver safely assumes that the message is not the genuine.
- No confidentiality but assures message origin authentication.

MAC with Confidentiality and Authentication



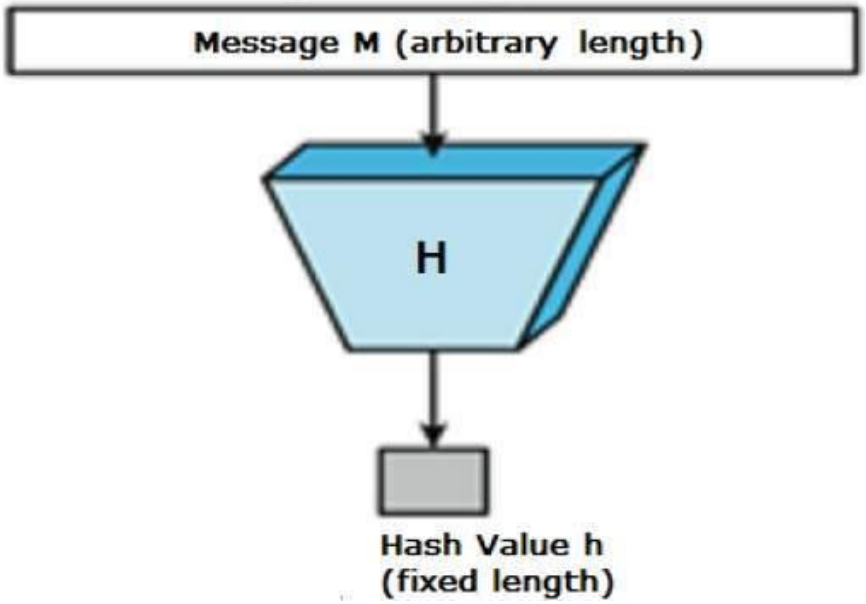
(b) Message authentication and confidentiality; authentication tied to plaintext



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

Hash Function

- 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 arbitrary length but output is always of fixed length.
- Values returned by a hash function are called message digest or simply hash values

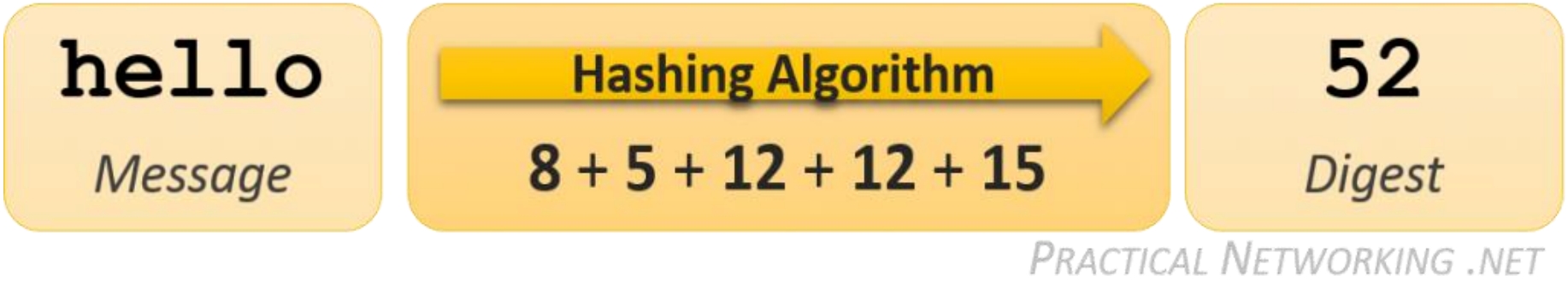


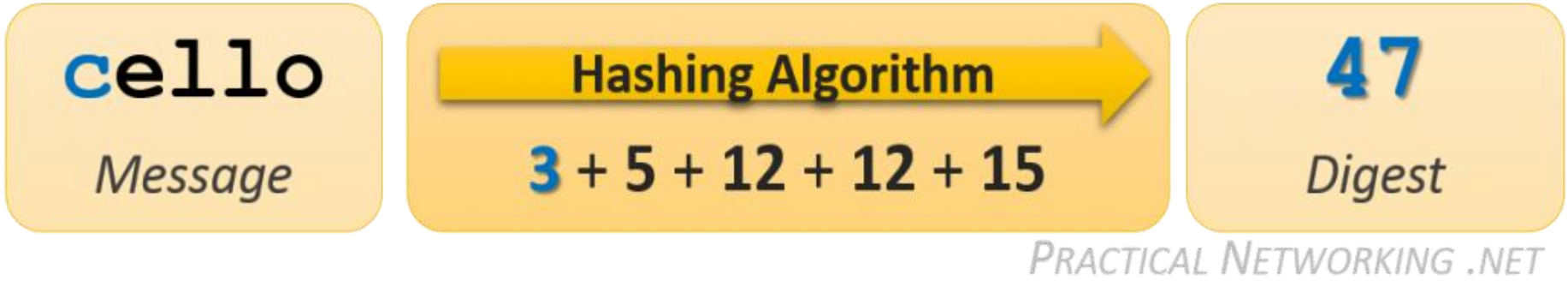
Hash Function

- When hash function provides security , is called as cryptographic hash functions.
- Hash function protects integrity of the message.
- If the encryption process is applied on message with hash function, it provides authentication and confidentiality.

Hash Function

- Example





Features of Hash Functions

Fixed Length Output (Hash Value)

- Hash function coverts data of arbitrary length to a fixed length. This process is often referred to as hashing the data.
- In general, the hash is much smaller than the input data, hence hash functions are sometimes called compression functions.
- Since a hash is a smaller representation of a larger data, it is also referred to as a digest.
- Hash function with n bit output is referred to as an n-bit hash function.
- Popular hash functions generate values between 160 and 512 bits.

Features of Hash Functions

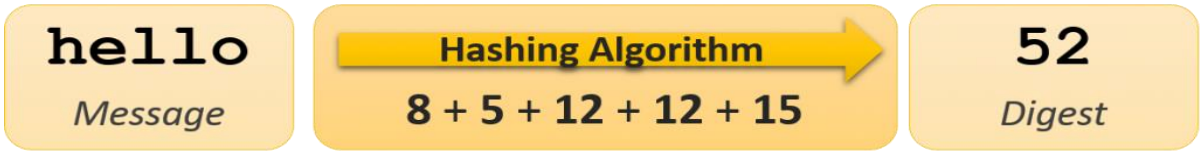
Efficiency of Operation

- Generally for any hash function h with input x, computation of h(x) is a fast operation.
- Computationally hash functions are much faster than a symmetric encryption.

Hash Function Properties

It is mathematically impossible to extract the original message from the digest.

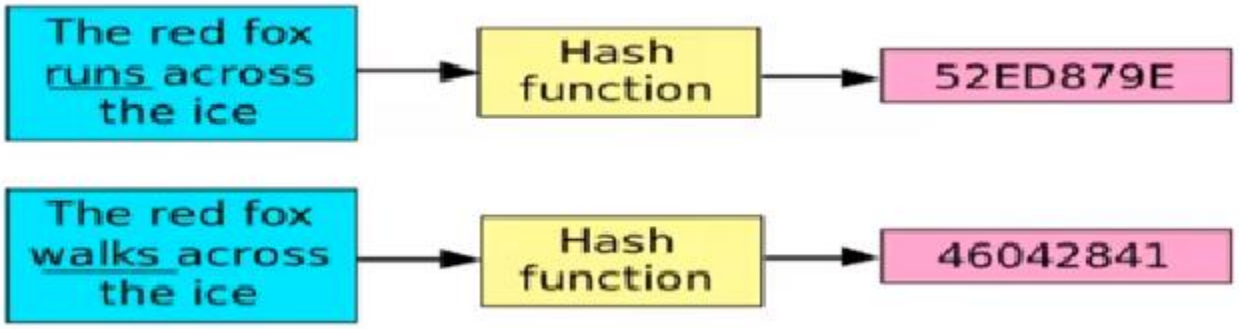
- Hashing is sometimes referred to as one-way encryption: the message can be encrypted but is impossible to decrypt. This is accomplished using one-way functions within the hashing algorithm.
- It is impossible to derive ‘hello’ knowing only a resulting digest of ’52’. Mostly because there could be thousands of messages that result in the identical digest.



Hash Function Properties (cont.)

A slight change to the original message causes a drastic change in the resulting digest.

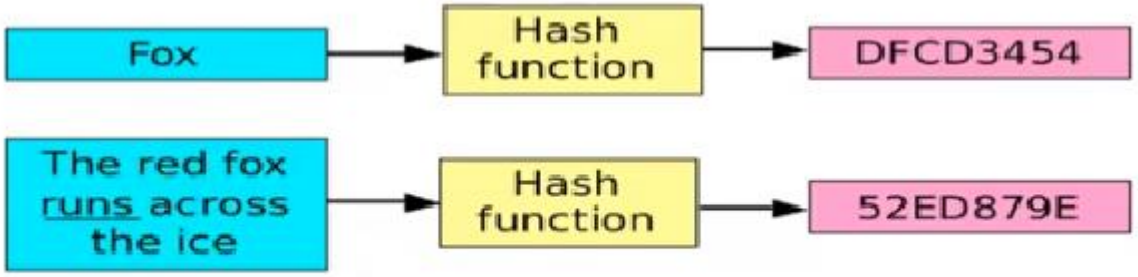
- Any minor modification – even as small as changing a single character – to the original Message should greatly alter the computed digest. This is sometimes referred to as the Avalanche effect.
- If for two different messages, message digest in case is similar then this term is known as Collision.



Hash Function Properties (cont.)

The result of the hashing algorithm is always the same length.

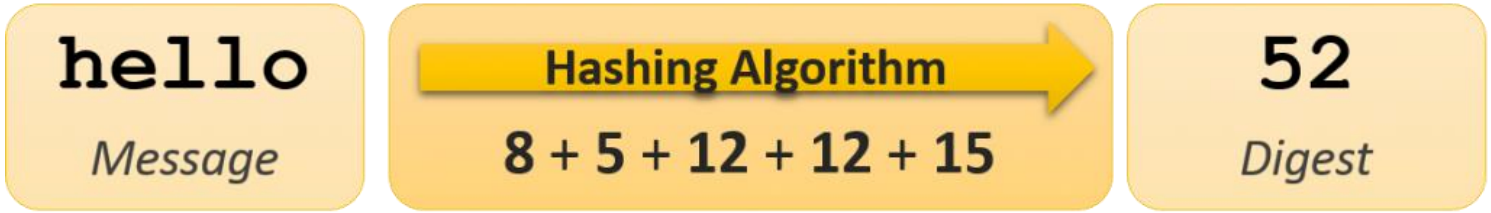
- It is vital for the resulting Digest to not provide any hints or clues about the original Message – including its length. A digest should not grow in size as the length of the Message increases.



Hash Function Properties (cont.)

It is infeasible to construct a message which generates a given digest.

- As per example below, if given the digest of 52 , it would not be overly difficult to generate a list of words that might have been the original message.



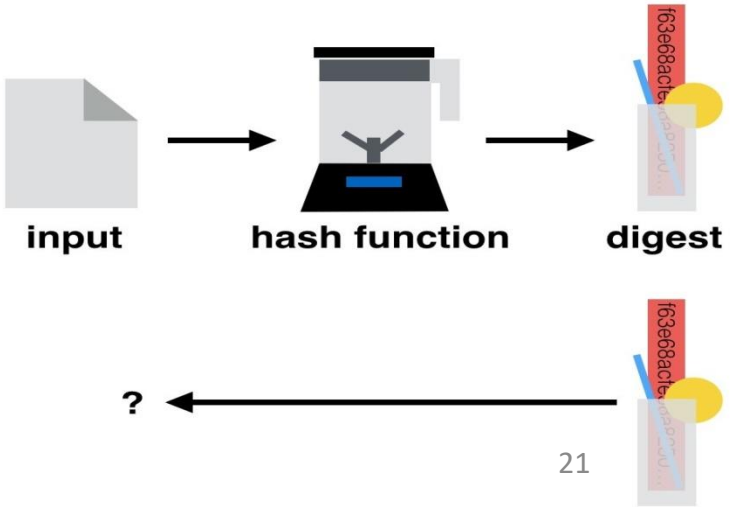
Cryptographic Hash Function

- Hash functions used for Security applications known as Cryptographic Hash Functions.
- Two important properties:
- It is computationally infeasible to find either
  - A data object that maps to a pre-specified hash result (the one-way property)
  - Two data objects that map to the same hash result (the collision-free property)

Cryptographic Hash Function Requirements and Security

Pre-Image Resistance

- Computationally hard to reverse a hash function.
- In other words, if a hash function h produced a hash value z, then it should be a difficult process to find any input value x that hashes to z.
- This property protects against an attacker who only has a hash value and is trying to find the input.



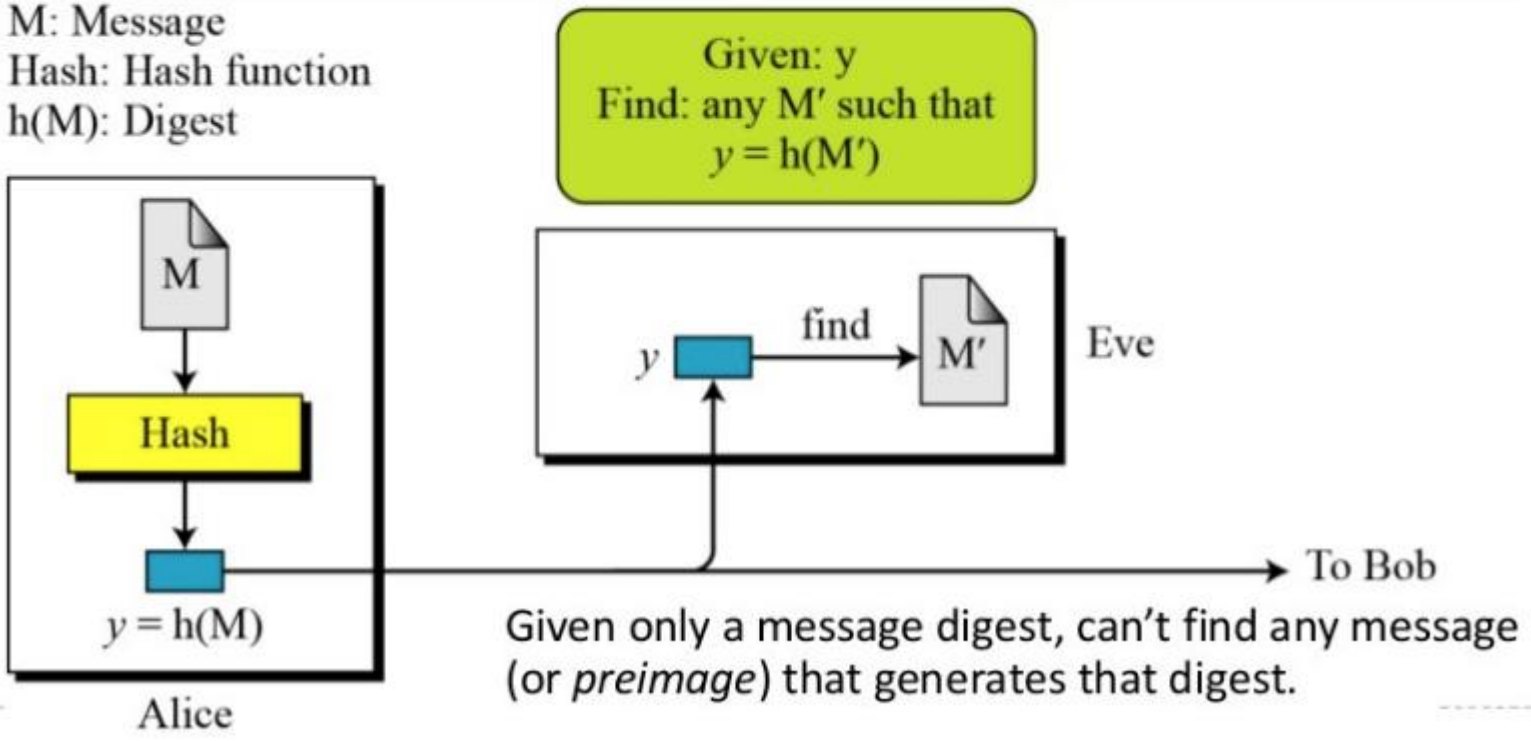
Cryptographic Hash Function Requirements and Security

Pre-Image Resistance

- This measures how difficult to devise a message which hashes to the known digest
- Roughly speaking, the hash function must be one-way.

Preimage Attack

Given: y = h(M) **Find: M′ such that y = h(M′)**



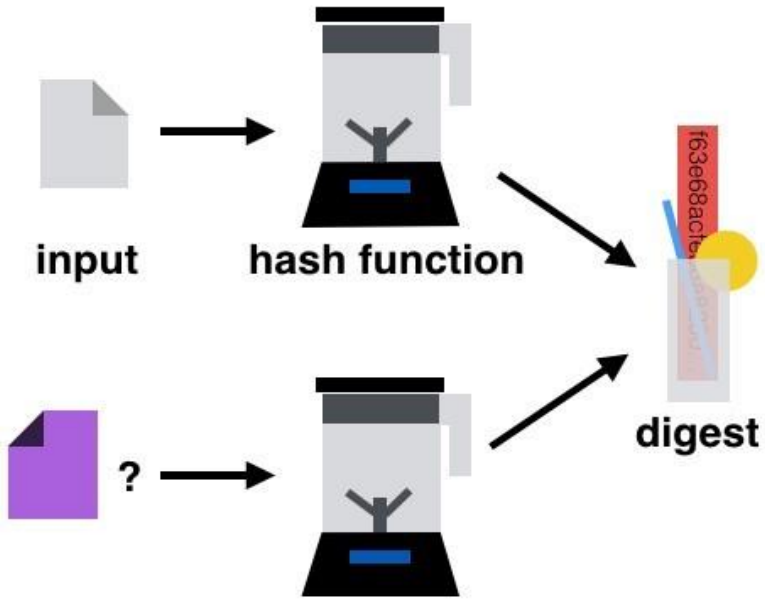
Given only a message digest, can't find any message (or *preimage*) that generates that digest.

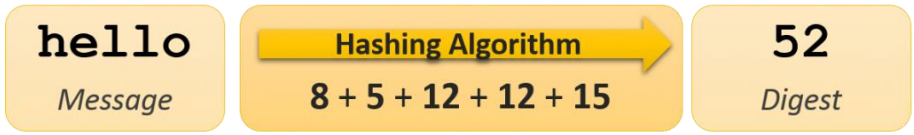
Cryptographic Hash Function Requirements and Security

Second Pre-Image Resistance

The property says the following: if I give you an input and the digest it hashes to, you should be unable to find a **different input** that hashes to the same digest.

Given message m1, it is difficult to produce another message m2 such that , H(m1) = H(m2).





Cryptographic Hash Function Requirements and Security

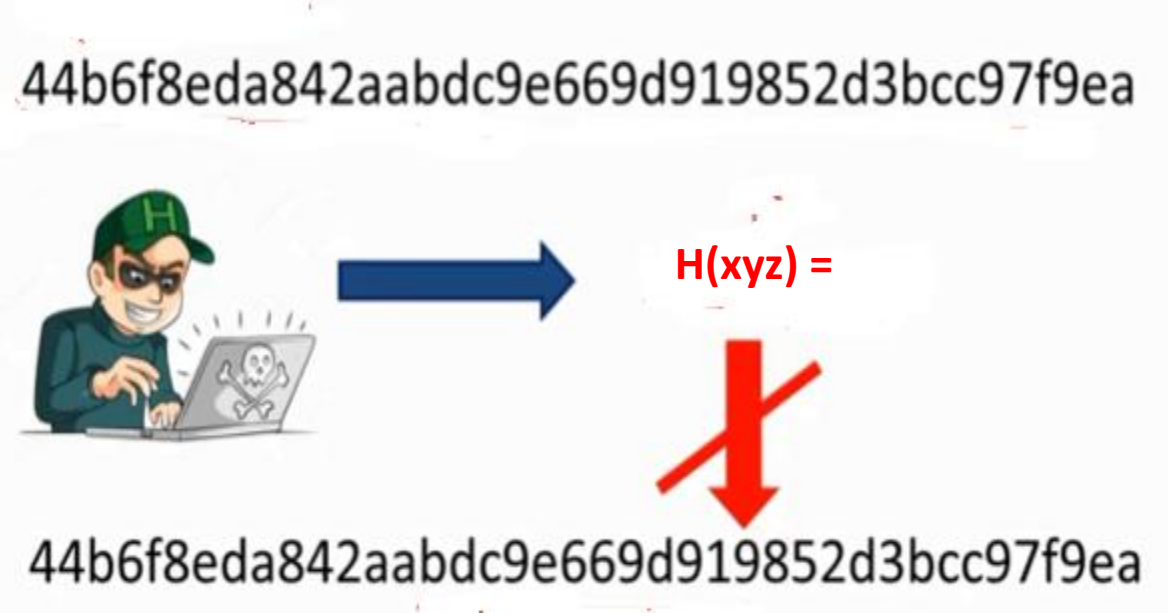
Second Pre-Image Resistance

- Given an input and its hash, it should be hard to find a different input with the same hash.
- In other words, if a hash function h for an input x produces hash value h(x), then it should be difficult to find any other input value y such that h(y) = h(x).
- This property of hash function protects against an attacker who has an input value and its hash, and wants to substitute different value as legitimate value in place of original input value.

Cryptographic Hash Function Requirements and Security

Second Pre-Image Resistance

H(abc) =



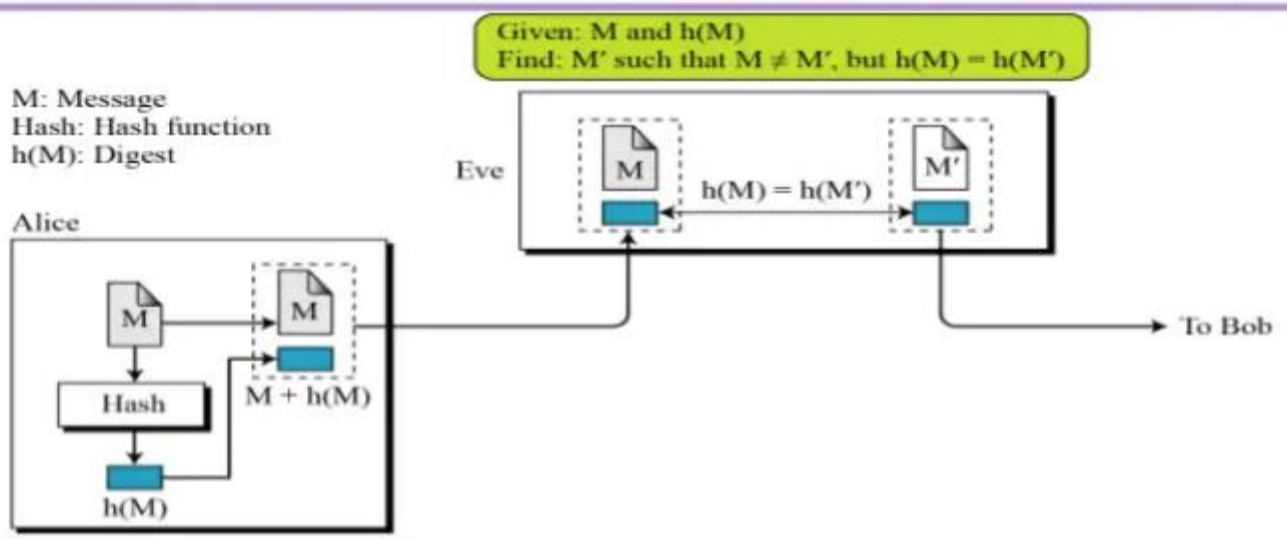
Cryptographic Hash Function Requirements and Security

Second Pre-Image Resistance

- This measures how difficult to devise a message which hashes to the known digest and its message

Second Preimage Attack

Given: M and h(M) **Find: $M' \neq M$ such that $h(M) = h(M')$**

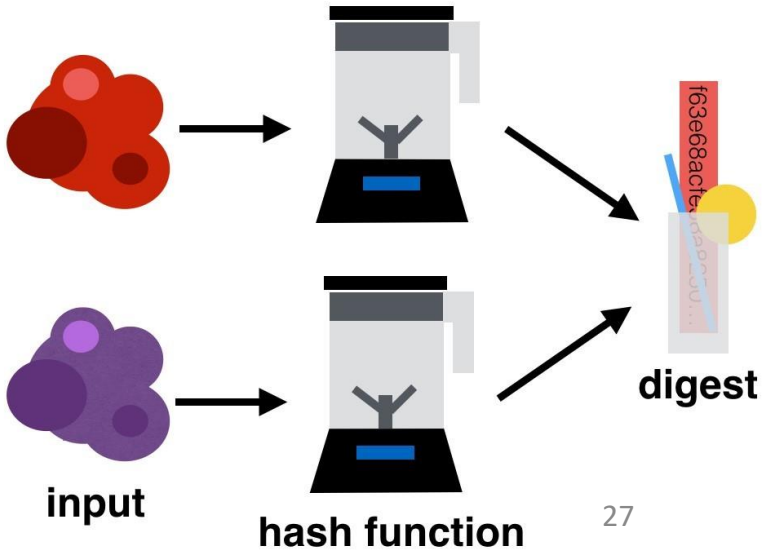


- Given one message, can't find another message that has the same message digest. An attack that finds a second message with the same message digest is a *second pre-image* attack.
  - It would be easy to forge new digital signatures from old signatures if the hash function used weren't second preimage resistant

Cryptographic Hash Function Requirements and Security

Collision Resistance

- It guarantees that no one can produce two different inputs that hash to the same output.
- Difficult to find any two different messages , m1 and m2 that have same hash value; H(m1) = H(m2)



Cryptographic Hash Function Requirements and Security

Collision Resistance

- It is hard to find two inputs that hash to the same output; that is, two inputs *a* and *b* where $a \neq b$ but $H(a) = H(b)$.
- This property makes it very difficult for an attacker to find two input values with the same hash.
- Also, if a hash function is collision-resistant **then it is second pre-image resistant.**

Cryptographic Hash Function Requirements and Security

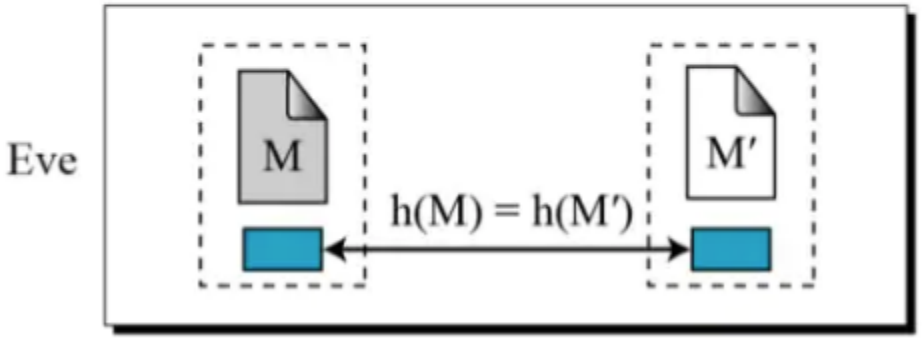
Collision Resistance

Collision Attack

Given: none **Find: $M' \neq M$ such that $h(M) = h(M')$**

M: Message
Hash: Hash function
h(M): Digest

Find: M and M′ such that $M \neq M'$, but $h(M) = h(M')$



- Can't find any two different messages with the same message digest

Security Requirements of Cryptographic Hash Functions

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

Simple Hash Function

- Here, there are two simple hash function, all hash functions are operating using same principle.
  1. The message file is like a simple input it open a sequence on n-bit blocks.
  2. When input is processed only one block at the given time in iterative fashion to generate an n-bit hash function.

- The simple hash function is the bit-by-bit XORing done of every block.

- This can be shows the following ways:

$$C_i = B_{i1} \oplus B_{i2} \oplus .... \oplus B_{im}$$

where, $C_i = C_i$ is i^{th} bit if hash code, $1 <= i <= n$

$m = m$ is the number of block in the input

$B_{ij} = i^{th}$ bit in j^{th} block

$\oplus$ = XORing 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

Hash with Authentication and Confidentiality

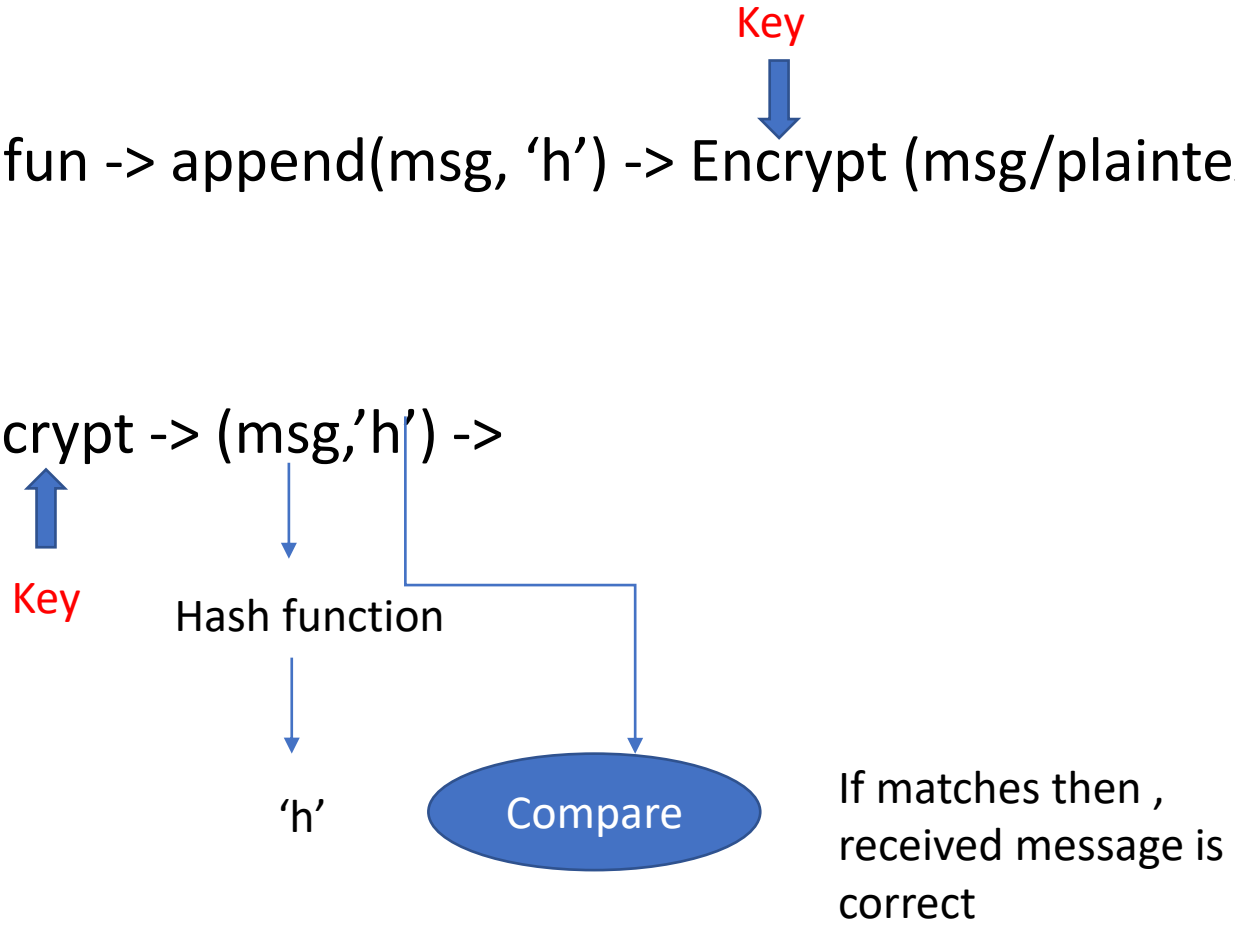
Method 1 - Message and hash code encrypted

Sender

Message -> Hash fun -> append(msg, ‘h’) -> Encrypt (msg/plaintext) -> (Cipher, ‘h’)

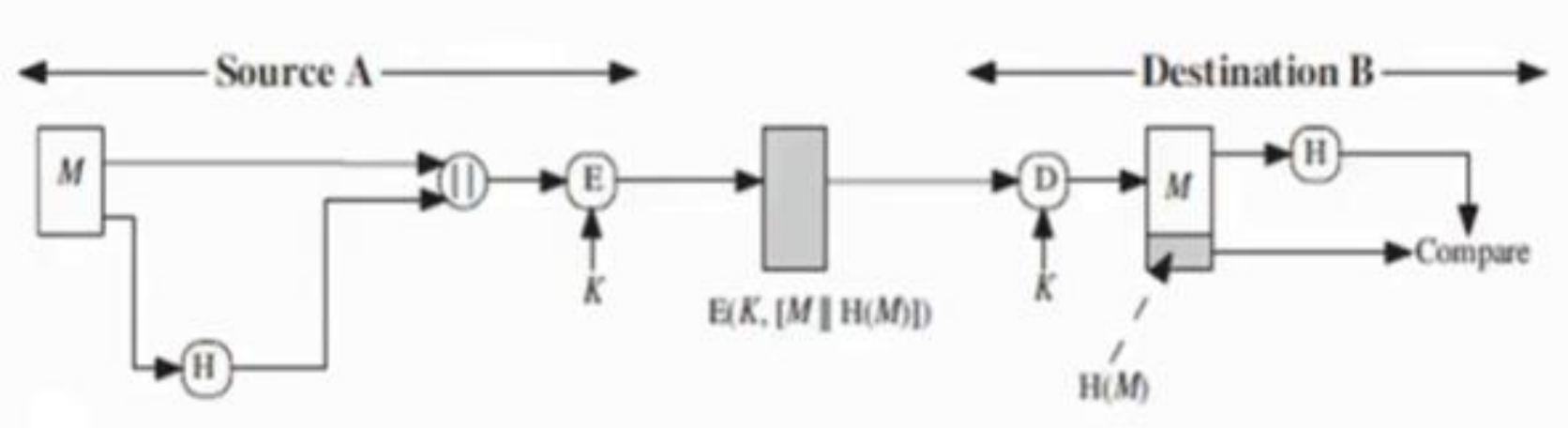
Receiver

(Cipher, ‘h’) -> Decrypt -> (msg,’h’) ->



Hash with Authentication and Confidentiality

Standard diagram – Message and hash code encrypted



Hash with Authentication and No Confidentiality

Method 2 – only hash code encrypted

Symmetric Encryption

Sender

Message -> Hash fun -> ‘h’ -> Encrypt (hash code) -> append(msg, E(‘h’))

Key

Receiver

Decrypt(‘h’) and msg passed to hash function ->

Key

‘h’

‘h’

Compare

If matches then , received message is correct

Hash with Authentication and No Confidentiality

Asymmetric Cryptography

Sender

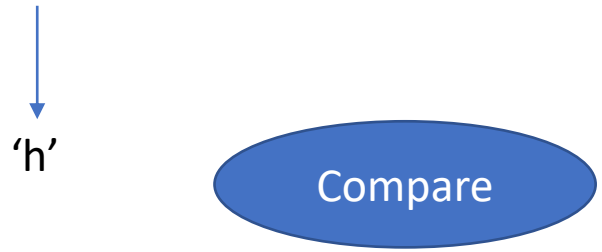
Private Key of A

Message -> Hash fun -> ‘h’ -> Encrypt (hash code) -> append(msg, E(‘h’))

Receiver

Public Key of A

Decrypt(‘h’) and msg passed to hash function ->



If matches then , received message is correct

Hash with Authentication and No Confidentiality

Asymmetric Cryptography

Sender

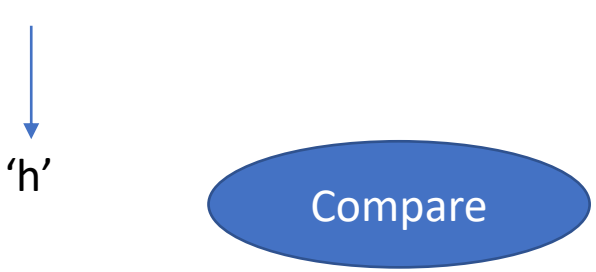
Private Key of A

Message -> Hash fun -> ‘h’ -> Encrypt (hash code) -> append(msg, E(‘h’))

Receiver

Public Key of A

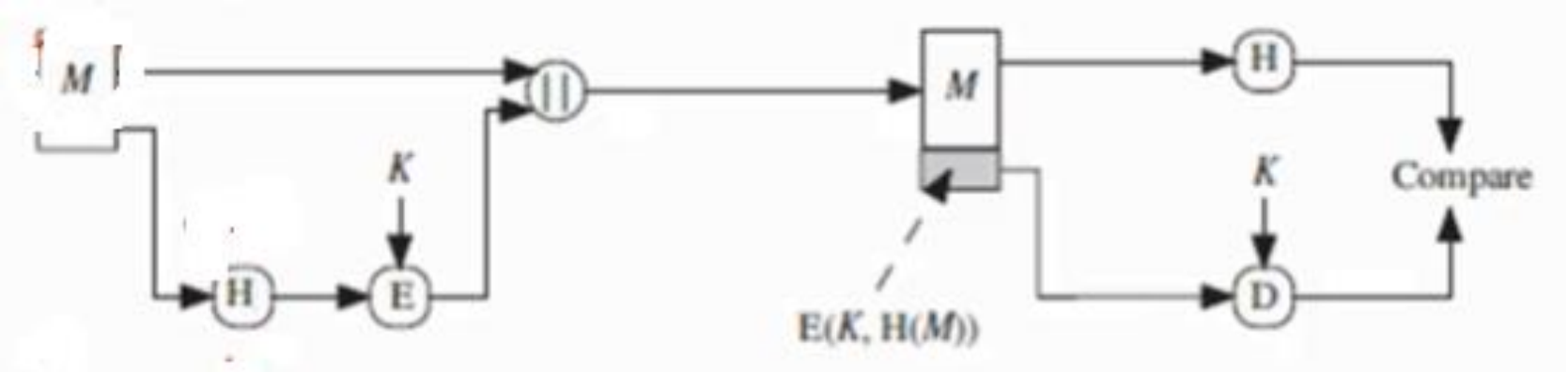
Decrypt(‘h’) and msg passed to hash function ->



If matches then , received message is correct

Hash with Authentication and No Confidentiality

Standard diagram - Only Hash code is encrypted



Hash with Authentication and No Confidentiality

Method 3 - With secret code

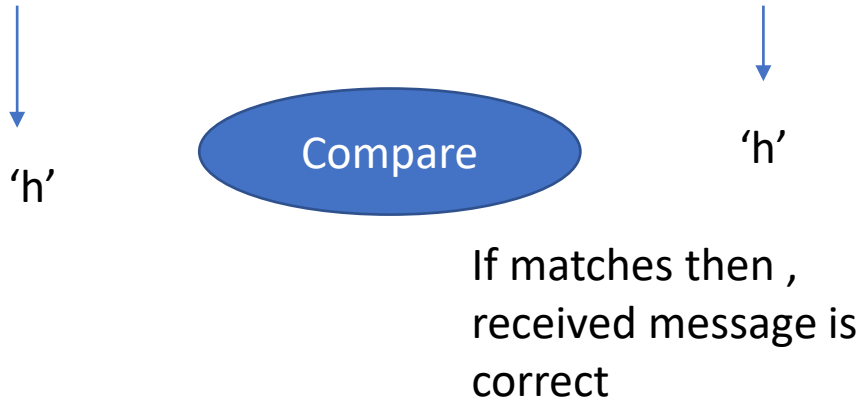
Sender

Message -> apply secret code -> pass to hash fun-> ‘h’ ->

append(msg, ‘h’)

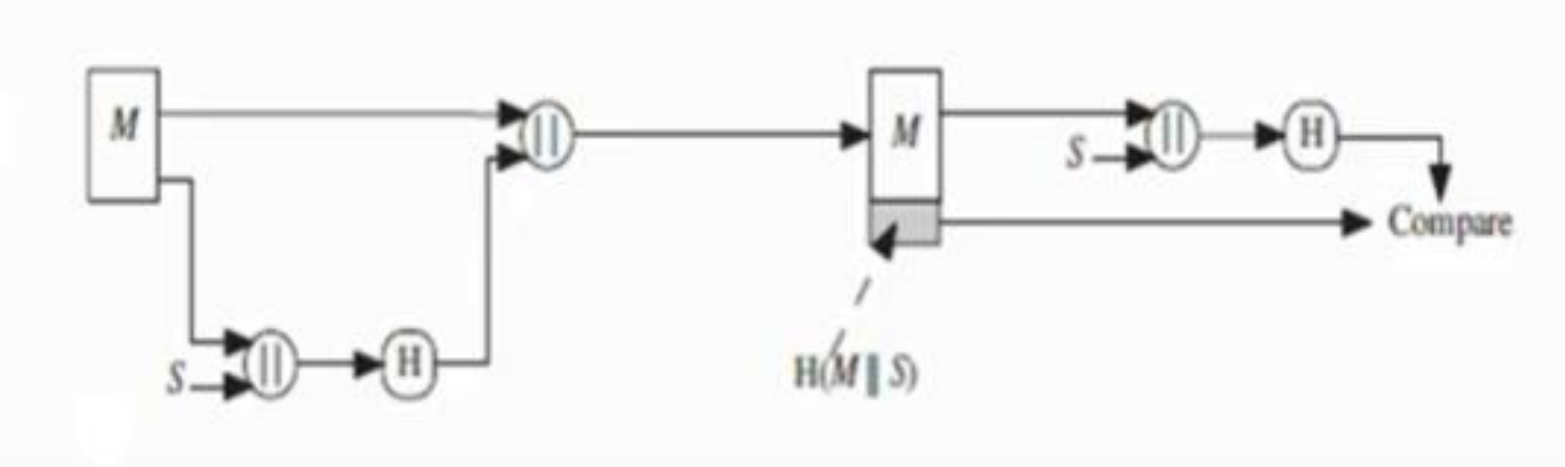
Receiver

Decrypt (msg, ‘h’)-> apply secret code to msg and send to hash fun



Hash with Authentication and No Confidentiality

With secret code/value – standard diagram



Hash with Authentication and Confidentiality

With secret code – message encrypted

Sender

Message -> apply secret code -> pass to hash fun-> ‘h’ ->

append(msg, ‘h’) -> Encrypt -> (cipher, E(‘h’))

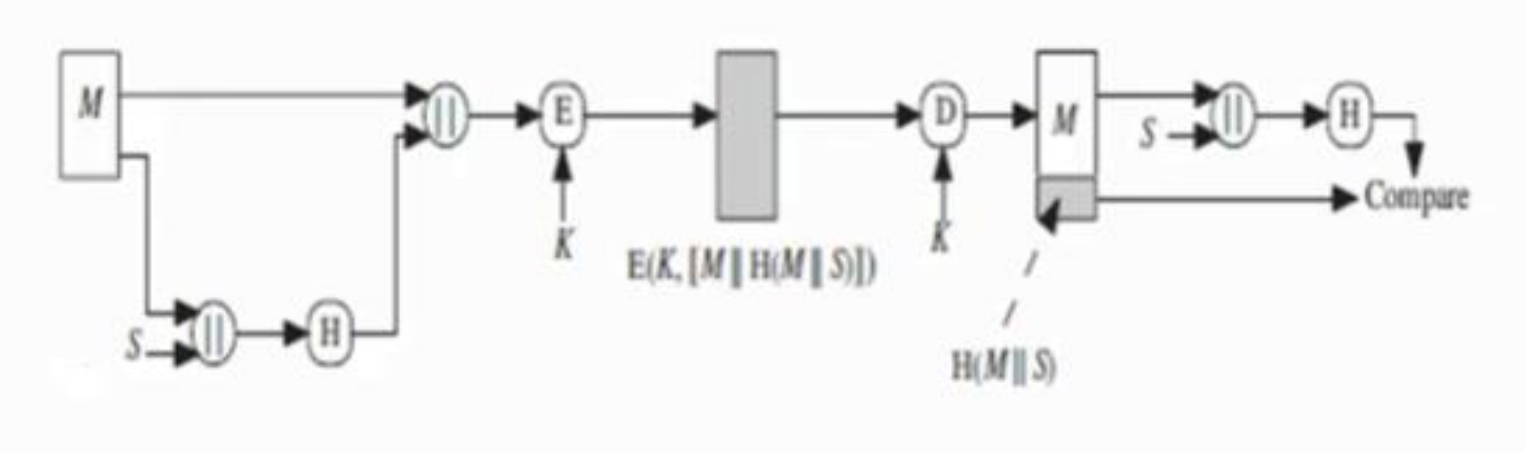
Receiver

Decrypt (cipher, E(‘h’))-> (msg,’h’) -> apply secret code to msg and send to hash fun

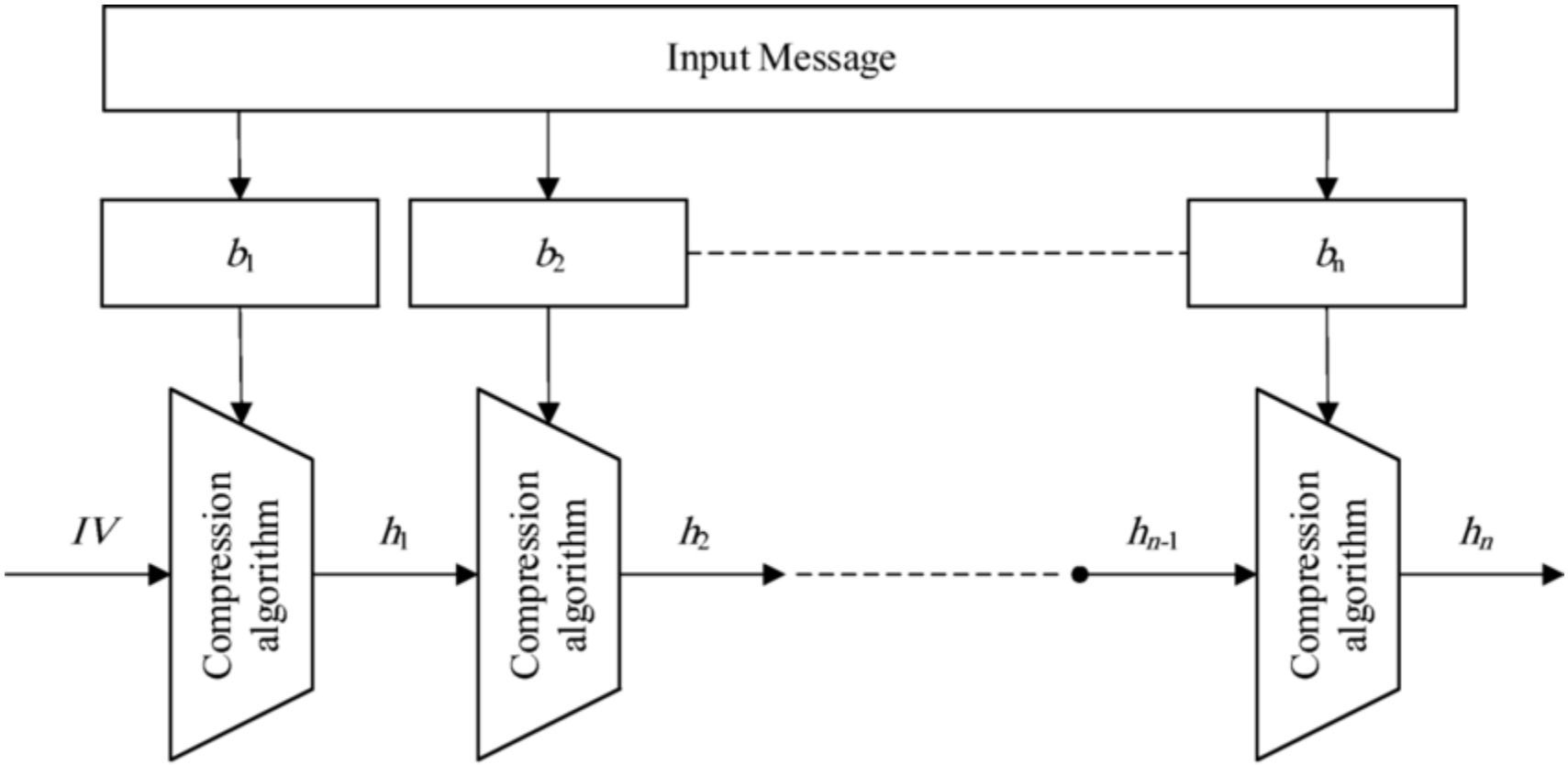
If matches then , received message is correct

Hash with Authentication and Confidentiality

With secret code (message + hash encrypted)- standard diagram



Hash Function Structure



IV = Initial value
h_i = Compression function output

b_i = ith input block
n = Number of input blocks

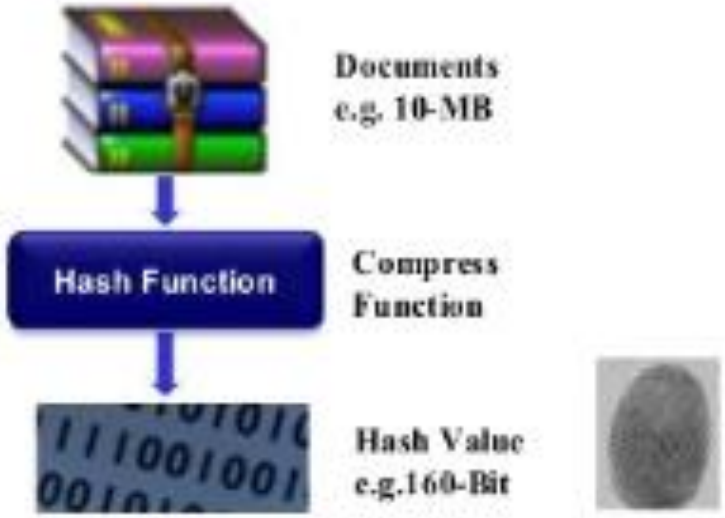
Hash Algorithm

- A hash algorithm is a one way function that converts a data string into a numeric string output of fixed length. The output string is generally much smaller than the original data. Therefore it is also called message digest or message compression algorithm.
- Hash algorithms are designed to be collision-resistant, meaning that there is a very low probability that the same string would be created for different data.
- Two of the most common hash algorithms are the MD5 (Message-Digest algorithm 5) and the SHA-1 (Secure Hash Algorithm). MD5 Message Digest checksums are commonly used to validate data integrity when digital files are transferred or stored.

One way Hash Function

A one way hash function H(M) operates on an arbitrary length pre-image message M, and return a fixed length hash value h.

h=H(M) ,where h is the length of m



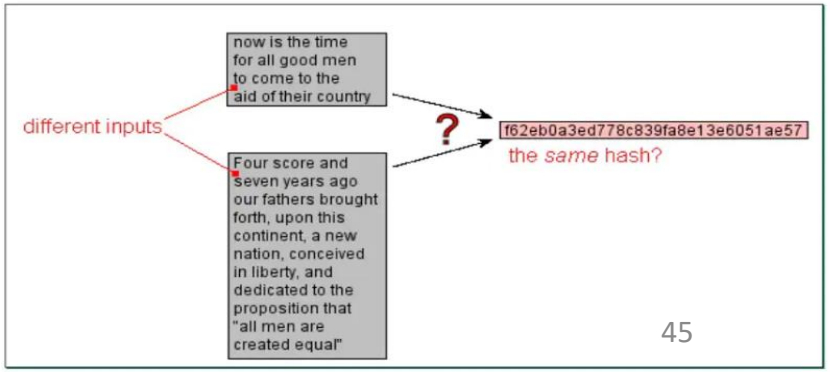
One way Hash Function

- Many functions can take an arbitrary-length input and return an output of fixed length, but one-way hash functions have additional characteristics that make them one-way:

1. It is relatively easy to compute, but significantly harder to reverse. That is, given M it is easy to compute H(M), but given H(M) it is hard to compute x.
2. Moreover it is also very hard to find another message M' such that H(M')=H(M). In other words it is collision resistant.

- In this context, "hard" is defined as something like: It would take millions of years to compute M from H(M), even if all the computers in the world were assigned to the problem.

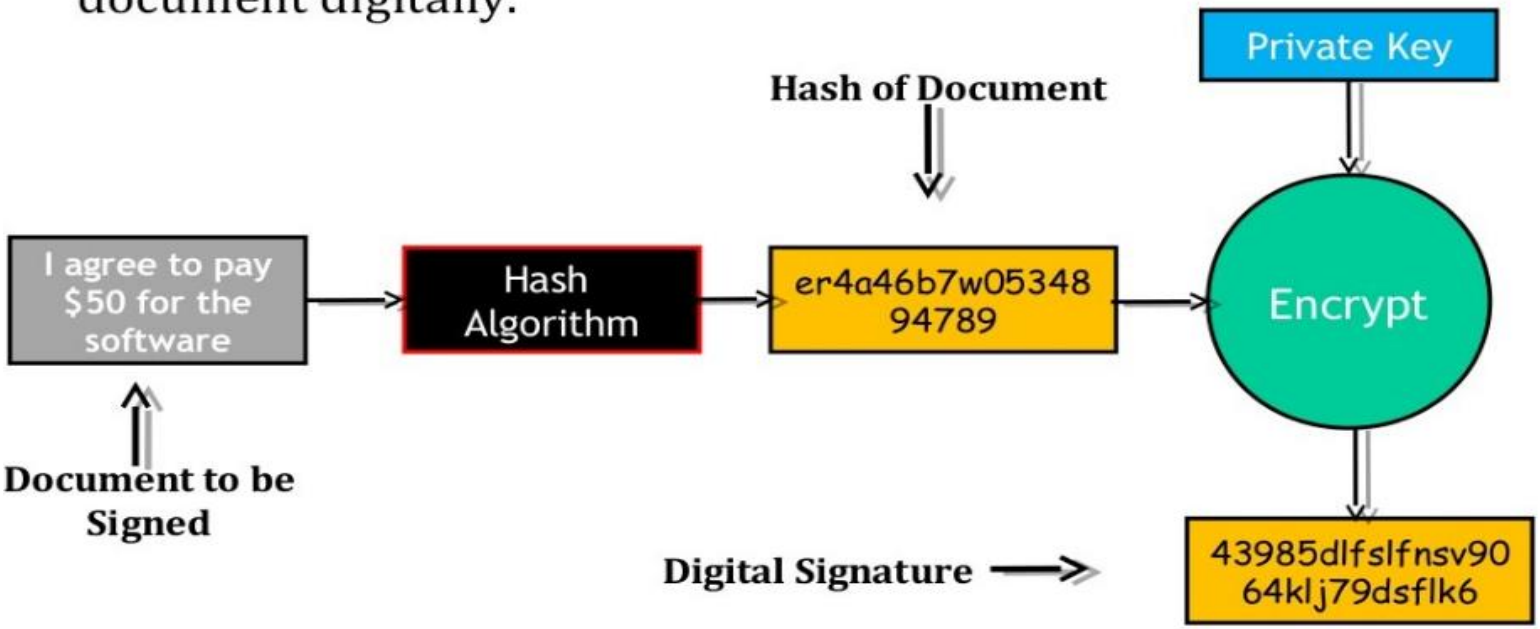
- When different input message results in the same hash value, then it is called hash collision.



One way Hash Function

- When applying digital signature to a document, we no longer need to encrypt the entire document with a sender's private key, it can be extremely slow. It is sufficient to encrypt the document's hash value instead. Therefor hash algorithm is used to digest the message before applying DSA.

- Hashing is to digest the original message while signing the document digitally.



HMAC

One way Hash Function Applications

- Digital Signatures
- Message Integrity
- Password verification
- Generation of pseudorandom bits
- Message Authentication Code (MAC)

Hash Function Family

- **MD (Message Digest)**
  - Designed by Ron Rivest
  - Family: MD2, MD4, MD5
- **SHA (Secure Hash Algorithm)**
  - Designed by NIST
  - Family: SHA-0, SHA-1, and SHA-2
    - SHA-2: SHA-224, SHA-256, SHA-384, SHA-512
    - SHA-3: New standard in competition
- **RIPEMD (Race Integrity Primitive Evaluation Message Digest)**
  - Developed by Katholieke University Leuven Team
  - Family : RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320,

MD4 family of Hash Functions

Algorithm	Output [bit]	Input [bit]	No. of rounds	Collisions found
MD5	128	512	64	yes
SHA-1	160	512	80	not yet
SHA-224	224	512	64	no
SHA-256	256	512	64	no
SHA-384	384	1024	80	no
SHA-512	512	1024	80	no

SHA Versions

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

The need of new Hash standard

- MD5 and SHA-0 already broken
- SHA-1 not yet fully "**broken**"
  - but similar to broken MD5 & SHA-0
  - so considered insecure and be fade out
- SHA-2 (esp. SHA-512) seems secure
  - shares same structure and mathematical operations as predecessors so have concern
- NIST announced in 2007 a competition for the SHA-3 next gen hash function

SHA-3 Requirements

- replace SHA-2 with SHA-3 in any use
  - so use same hash sizes
- preserve the nature of SHA-2
  - so must process small blocks (512 / 1024 bits)

SHA – Secure Hash Algorithm

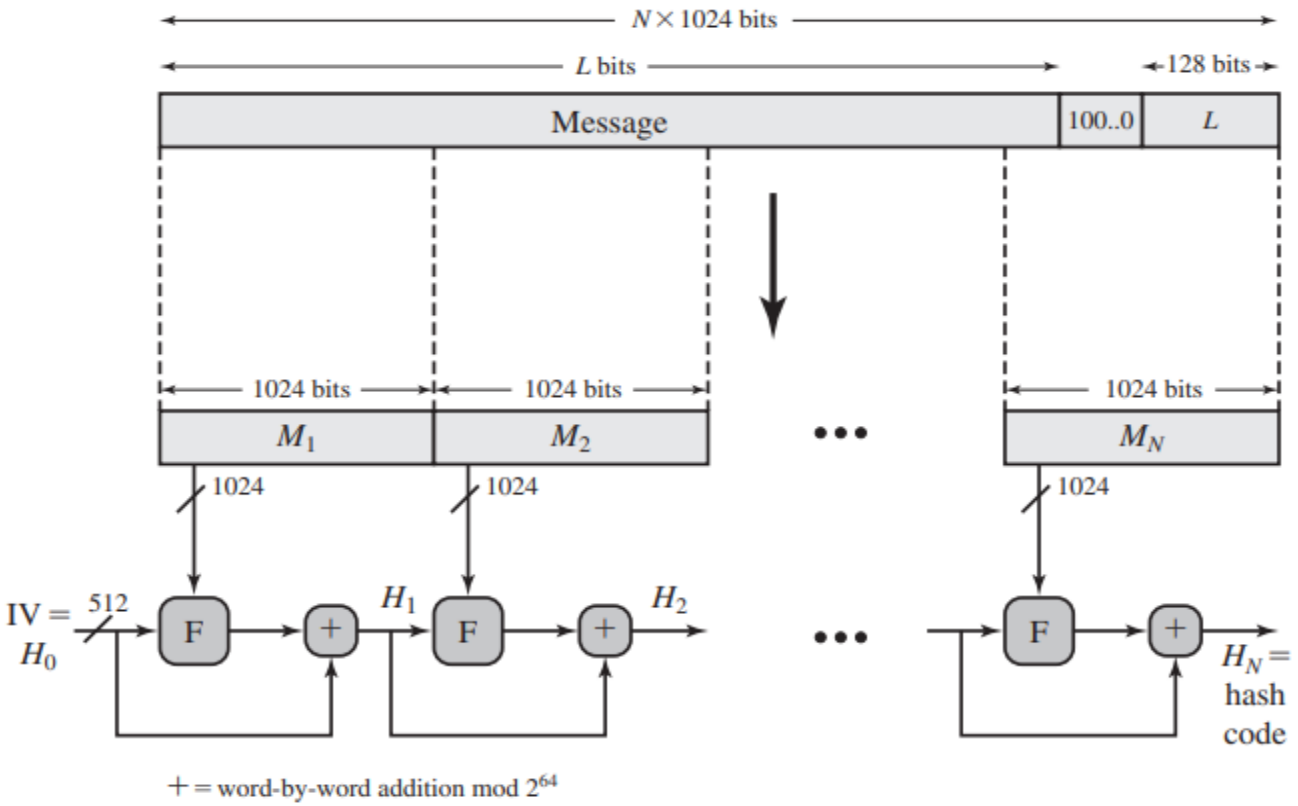
- Developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993; a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1.
- SHA-1 produces a hash value of 160 bits.
- The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest.

SHA – Secure Hash Algorithm

- In 2002, NIST produced a revision of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512.
- Collectively, these hash algorithms are known as SHA-2. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1.
- In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010.

SHA - 512

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



SHA - 512

Steps

Step 1: Append padding bits.

- The message is padded so that its length is congruent to 896 modulo 1024 [length K $\cong$ 896 (mod 1024)].
- Padding is always added, even if the message is already of the desired length.
- Thus, the number of padding bits is in the range of 1 to 1024.
- The padding consists of a single 1-bit followed by the necessary number of 0-bits

SHA - 512

SHA - 512

Step 2: Append length.

- A block of 128 bits is appended to the message. This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message (before the padding).
- The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length. In Figure, the expanded message is represented as the sequence of 1024-bit blocks M1, M2, . . . , MN, so that the total length of the expanded message is N * 1024 bits.

Padding Example

Consider Input Message – ‘abc’

Represented in binary

01100001 01100010 01100011

Message length = 24 bits

Needed,

$$Message\_length \cong 896 \bmod 1024$$

$$Message\_length \bmod 1024 \cong 896$$

$$24 + 872 \bmod 1024 \cong 896$$

Pad 872 bits to message such that

$$Message\_length \bmod 1024 \cong 896$$

872 bits to be padded – 1 bit followed by 871 zeros

Padding Example

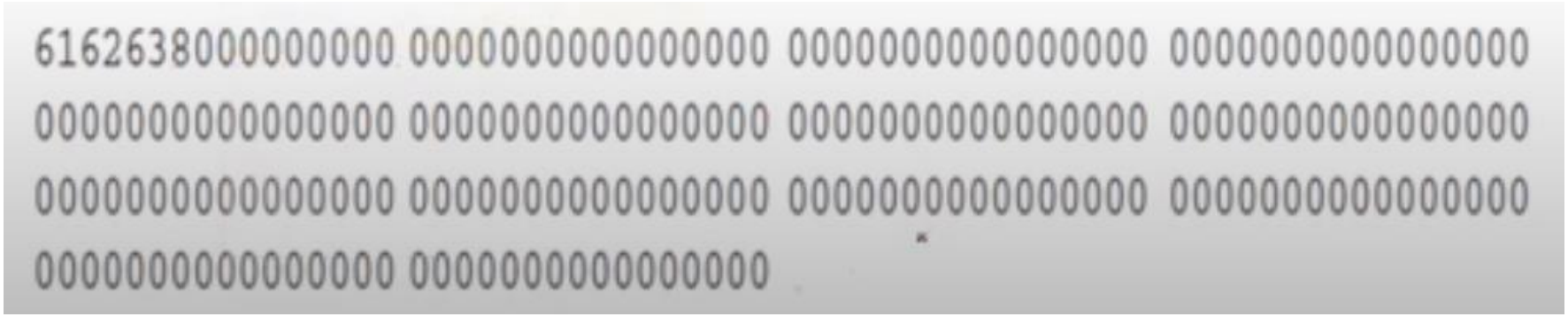
Consider Input Message – ‘abc’

Represented in binary

01100001 01100010 01100011 –

Padding(10000…….)

896 bits representation is shown below;



Padding Example

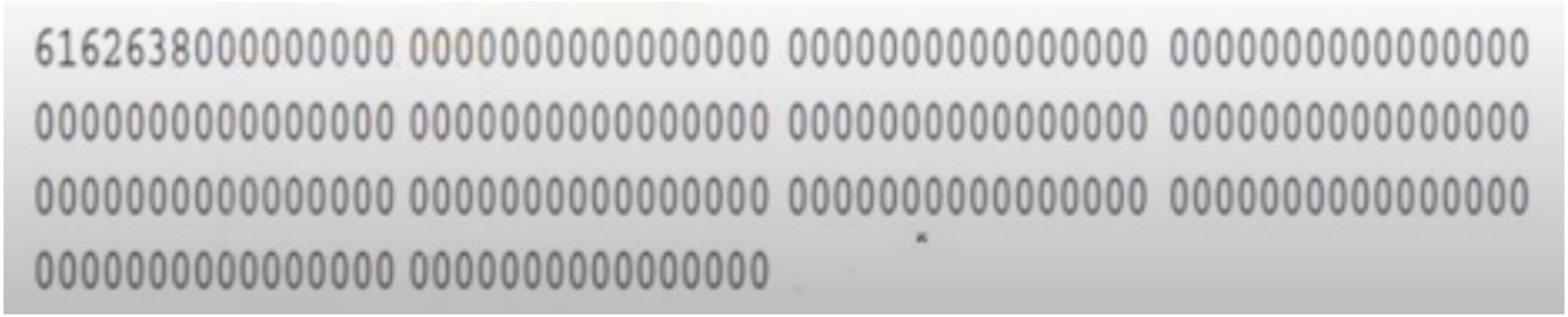
Pad the original length of the message for 128 bits at the end

Message_length = 24 bits

Convert this in hexadecimal = 18

So, represent 18 in 128 bits hexadecimal value –

0000000000000000 0000000000000018 (total 64 bits)



Message size = 896+128 =1024 bits

Exercise

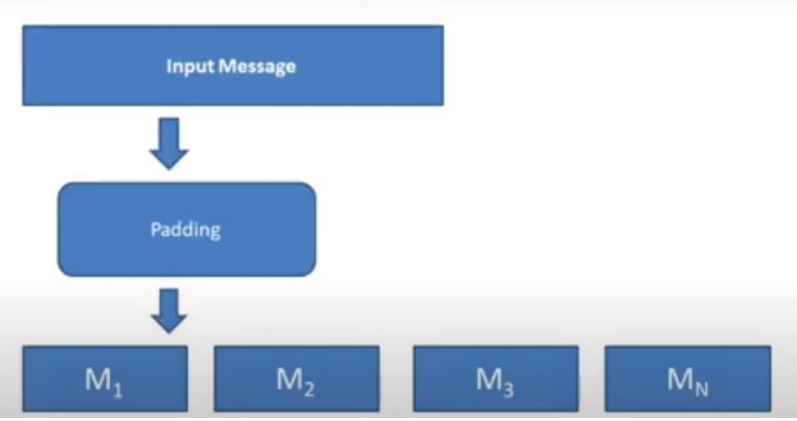
- How many bits will you pad for input message length of 2348 bits?

Exercise

- How many bits will you pad for input message length of 2348 bits?

Sol - Message_length ≅ 896 mod 1024

2348 mod 1024 = 300



Need 596 bits more

Pad 596 bits where in 1 followed by 595 zeros

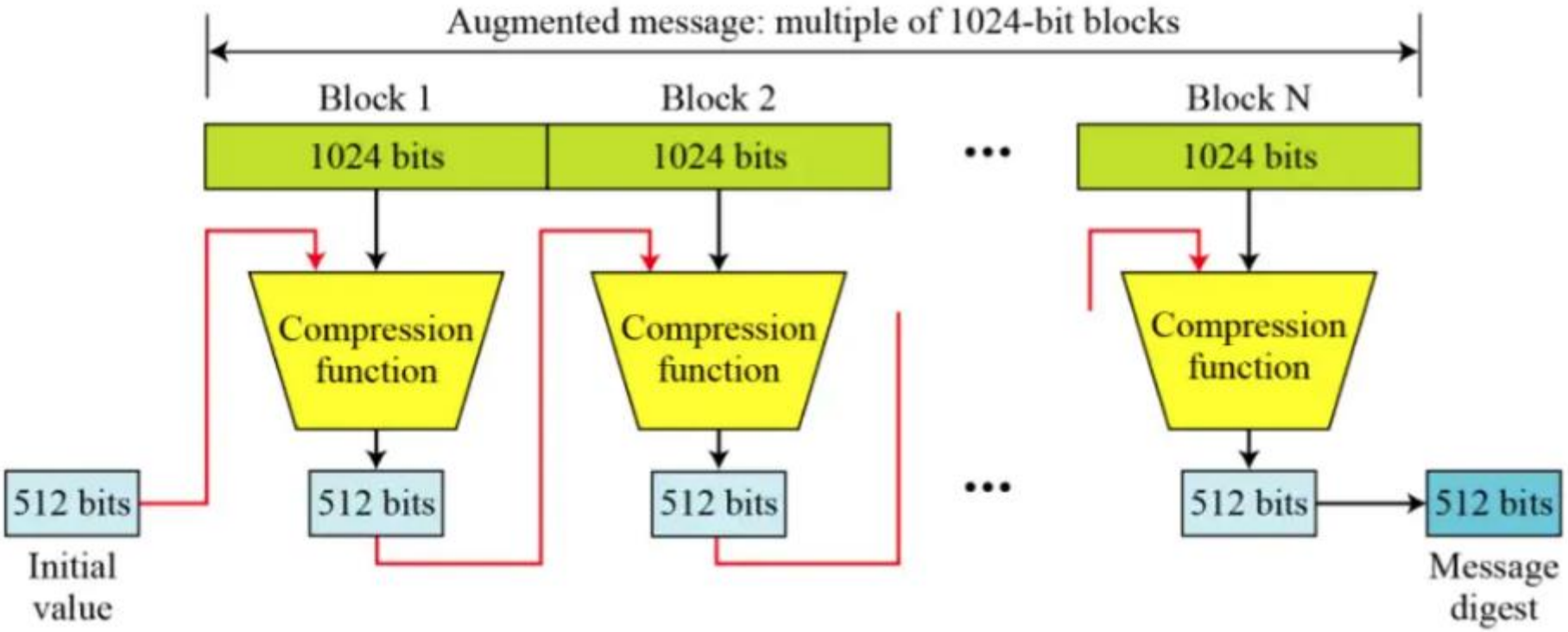
Message_length (with padding) is = 2348+596 = 2944 bits

Add actual message length 2348 as 128 bits at the end

Total bits = 2944+128 = 3072

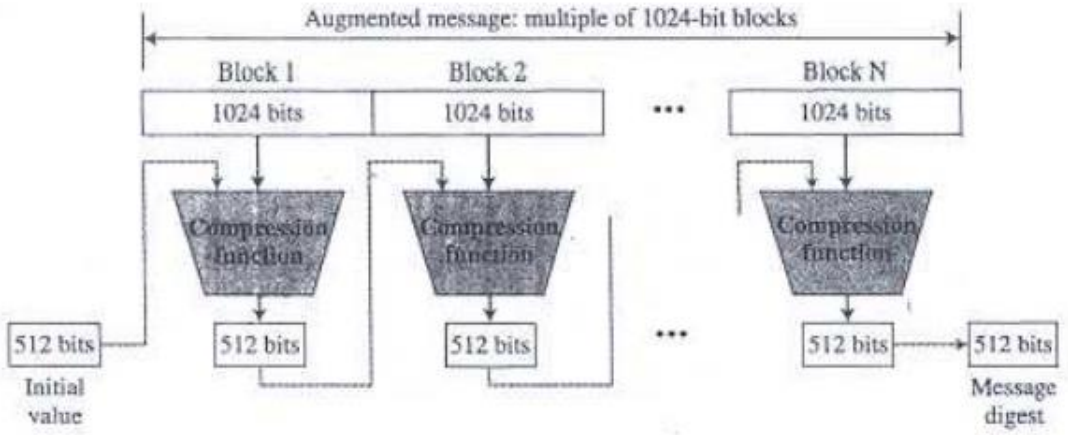
Which takes 3 M blocks of size 1024 bits each

Message Digest Creation



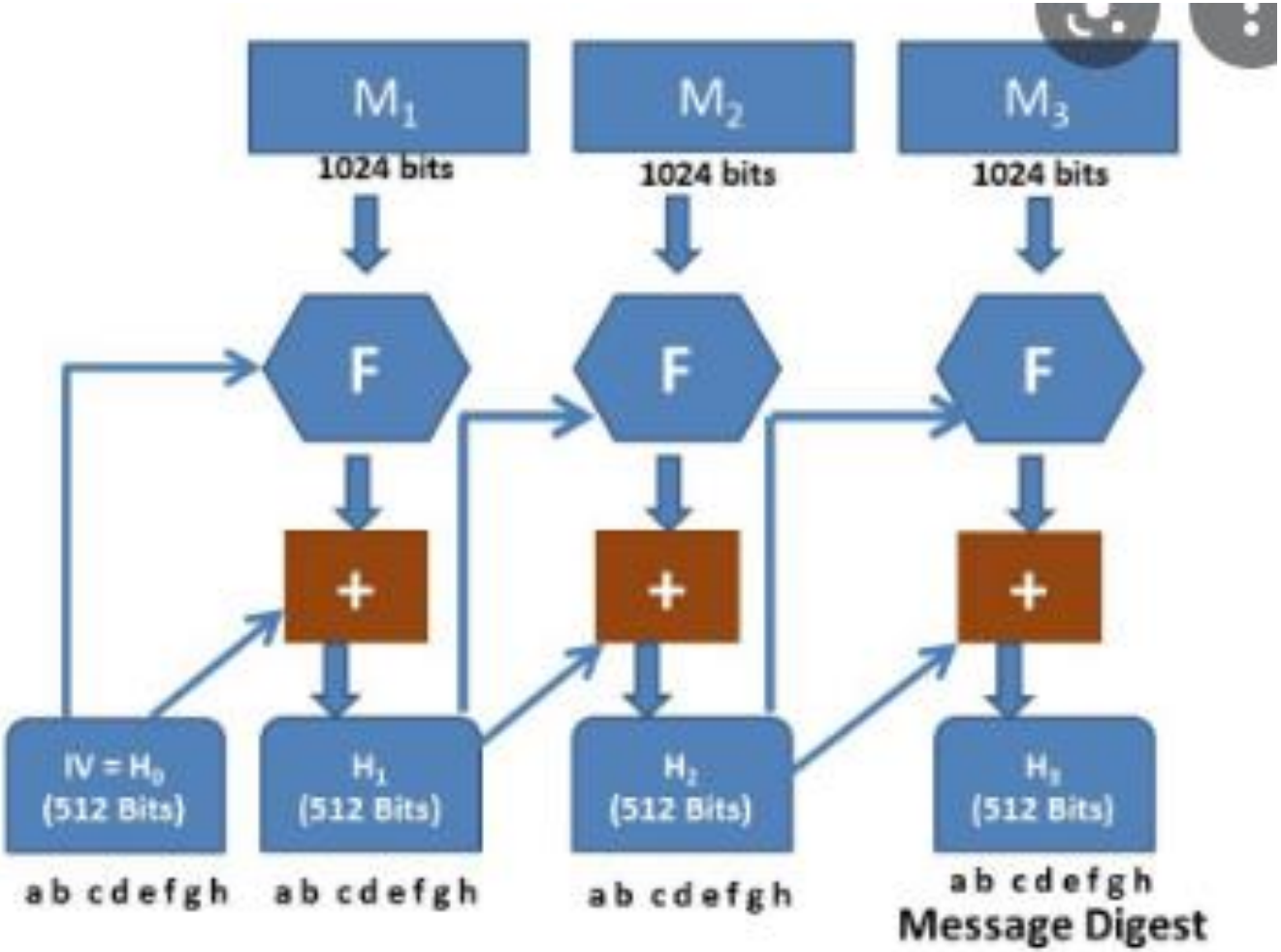
Message Digest Creation

Figure 12.6 *Message digest creation SHA-512*



The digest is initialized to a predetermined value of 512 bits. The algorithm mixes this initial value with the first block of the message to create the first intermediate message digest of 512 bits. This digest is then mixed with the second block to create the second intermediate digest. Finally, the $(N-1)$th digest is mixed with the Nth block to create the Nth digest. When the last block is processed, the resulting digest is the message digest for the entire message.

Message Digest Creation



SHA - 512

Steps

Step 3: Initialize hash buffer.

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

a = 6A09E667F3BCC908 b = BB67AE8584CAA73B

c = 3C6EF372FE94F82B d = A54FF53A5F1D36F1

e = 510E527FADE682D1 f = 9B05688C2B3E6C1F

g = 1F83D9ABFB41BD6B h = 5BE0CD19137E2179

SHA - 512

Steps

Step 3: Initialize hash buffer.

h = 5BE0CD19137E2179

The values are calculated from first eight prime numbers (2,3,5,7,11,13,17,19)

the square root $(19)^{1/2} = 4.35889894354$. Converting the number to binary with only 64 bits in the fraction part, we get

$$(100.0101\ 1011\ 1110\ \ldots\ 1001)_2 \rightarrow (4.5BE0CD19137E2179)_{16}$$

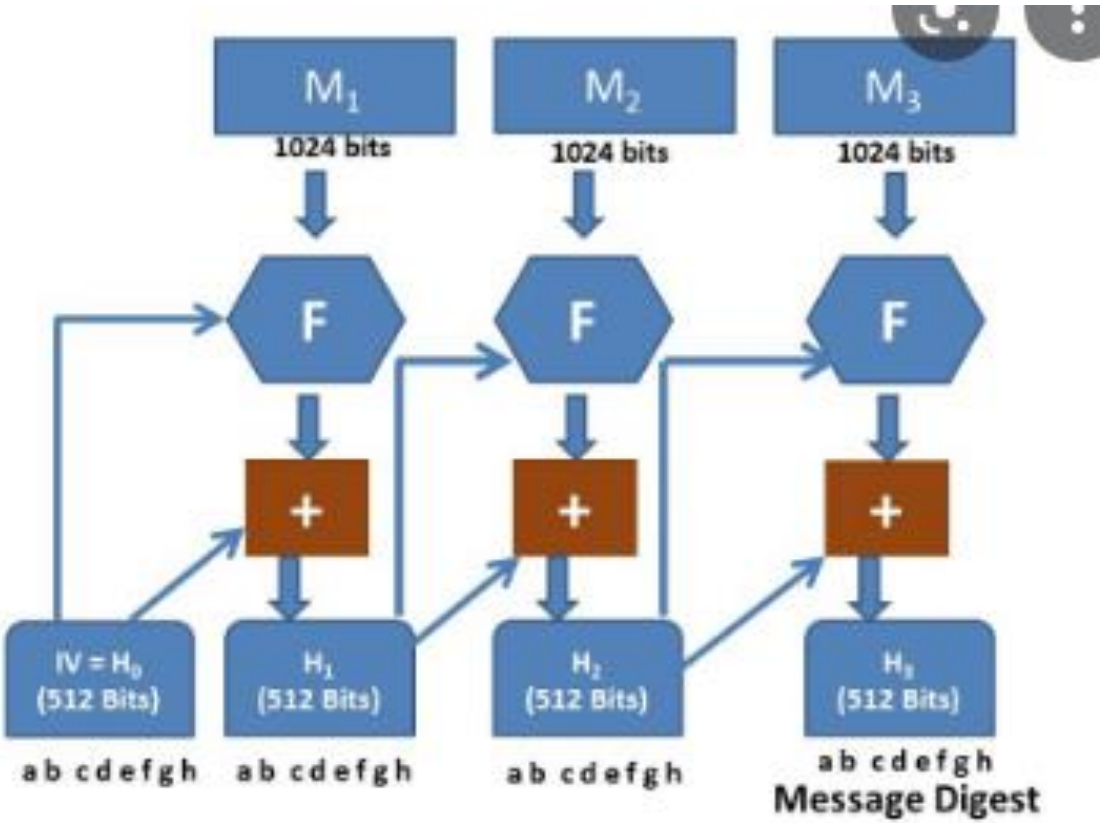
SHA-512 keeps the fraction part, $(5BE0CD19137E2179)_{16}$, as an unsigned integer.

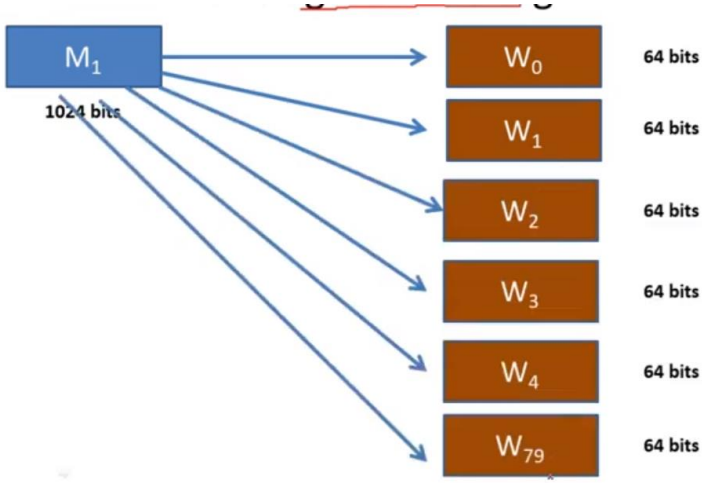
SHA - 512

Steps

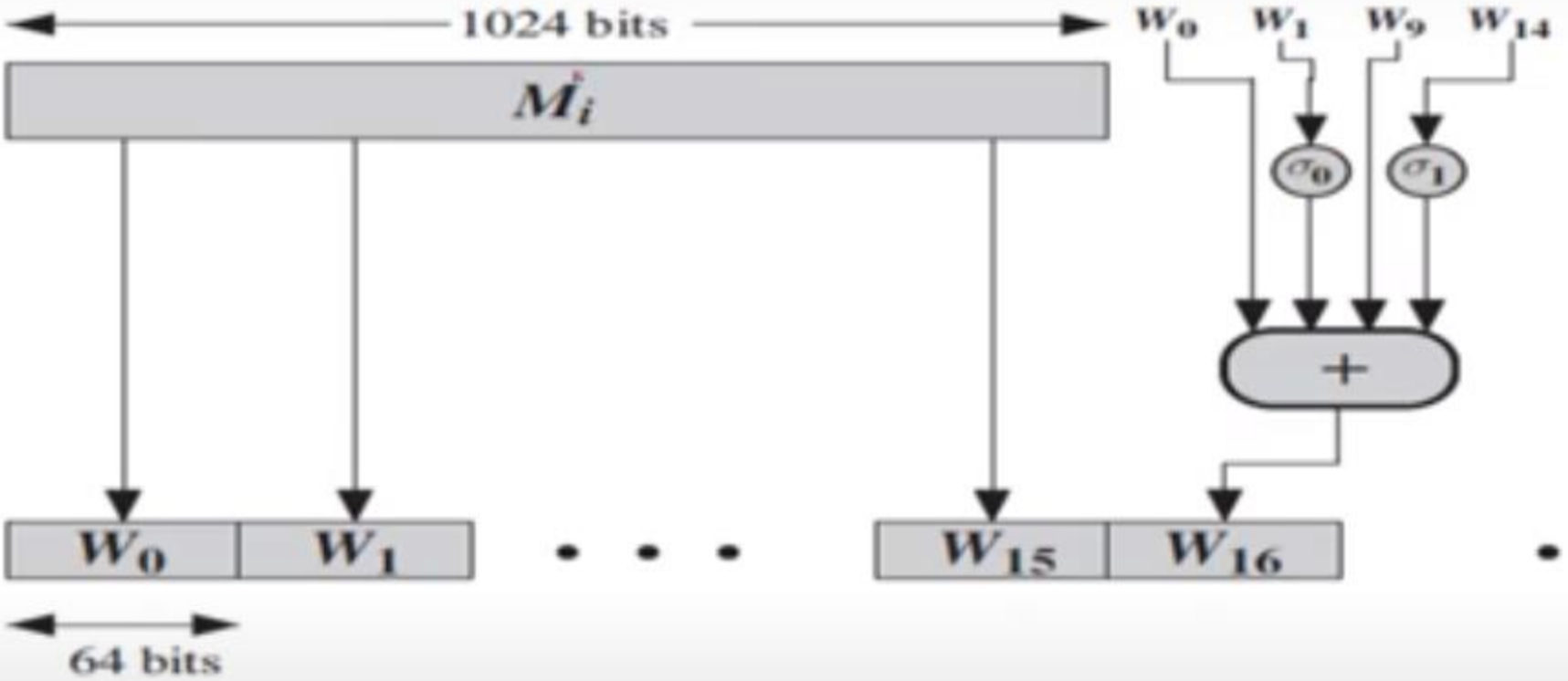
Step 4: Process message in 1024-bit blocks.

The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F. Each message block generates 80 words of 64 bits each





Word Expansion – derive 80 words from 1024 bits block



$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

where

$$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$$
$$\sigma_1^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$$

$\text{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

$\text{SHR}^n(x)$ = left shift of the 64-bit argument x by n bits with padding by zeros on the right

$+$ = addition modulo 2^{64}

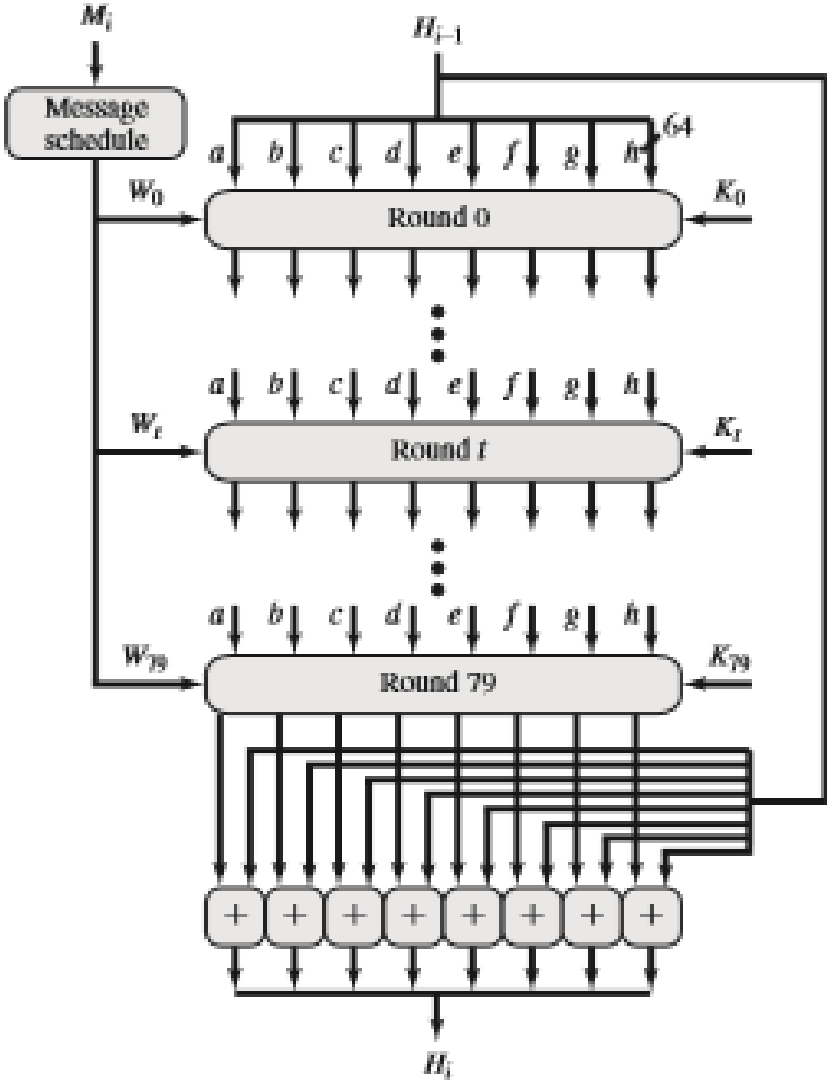
SHA - 512

Steps

Step 4:

- Each round takes as input the 512-bit buffer value abcdefgh and updates the contents of the buffer.
- At input to the first round, the buffer has the value of the intermediate hash value, Hi-1.
- Each round t makes use of a 64-bit value Wt, derived from the current 1024-bit block being processed (Mi).
- Each round also makes use of an additive constant Kt, where 0 <= t <= 79 indicates one of the 80 rounds. These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers.
- The constants provide a "randomized" set of 64-bit patterns, which should eliminate any regularities in the input data.
- The operations performed during a round consist of circular shifts, and primitive Boolean functions based on AND, OR, NOT, and XOR.
- The output of the eightieth round is added to the input to the first round (Hi-1) to produce Hi.
- The addition is done independently for each of the eight words in the buffer, with each of the corresponding words in Hi-1, using addition modulo 2^{64}

Functioning of Module F



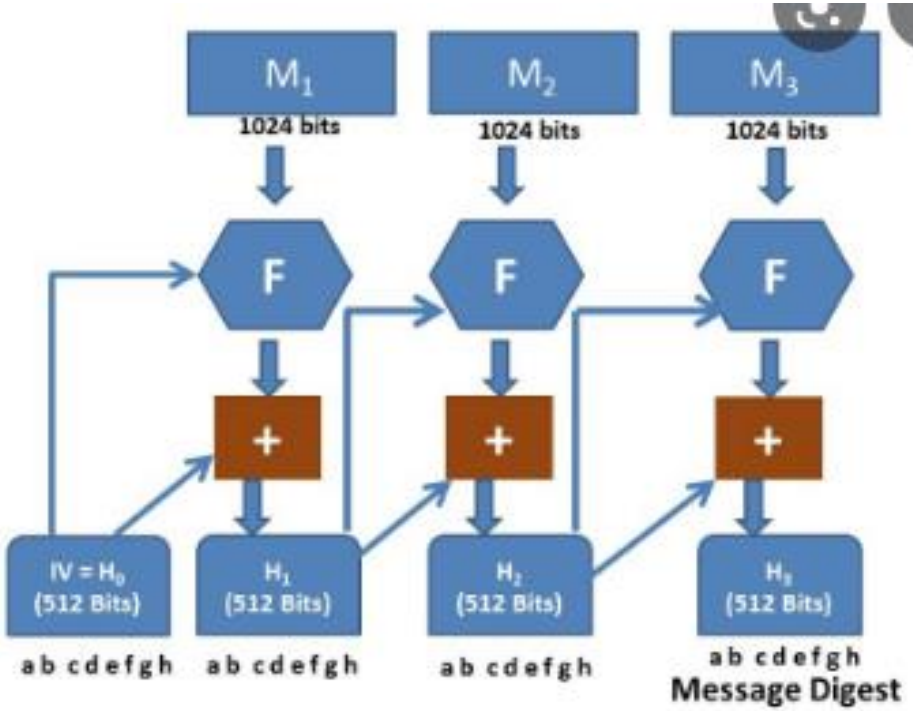
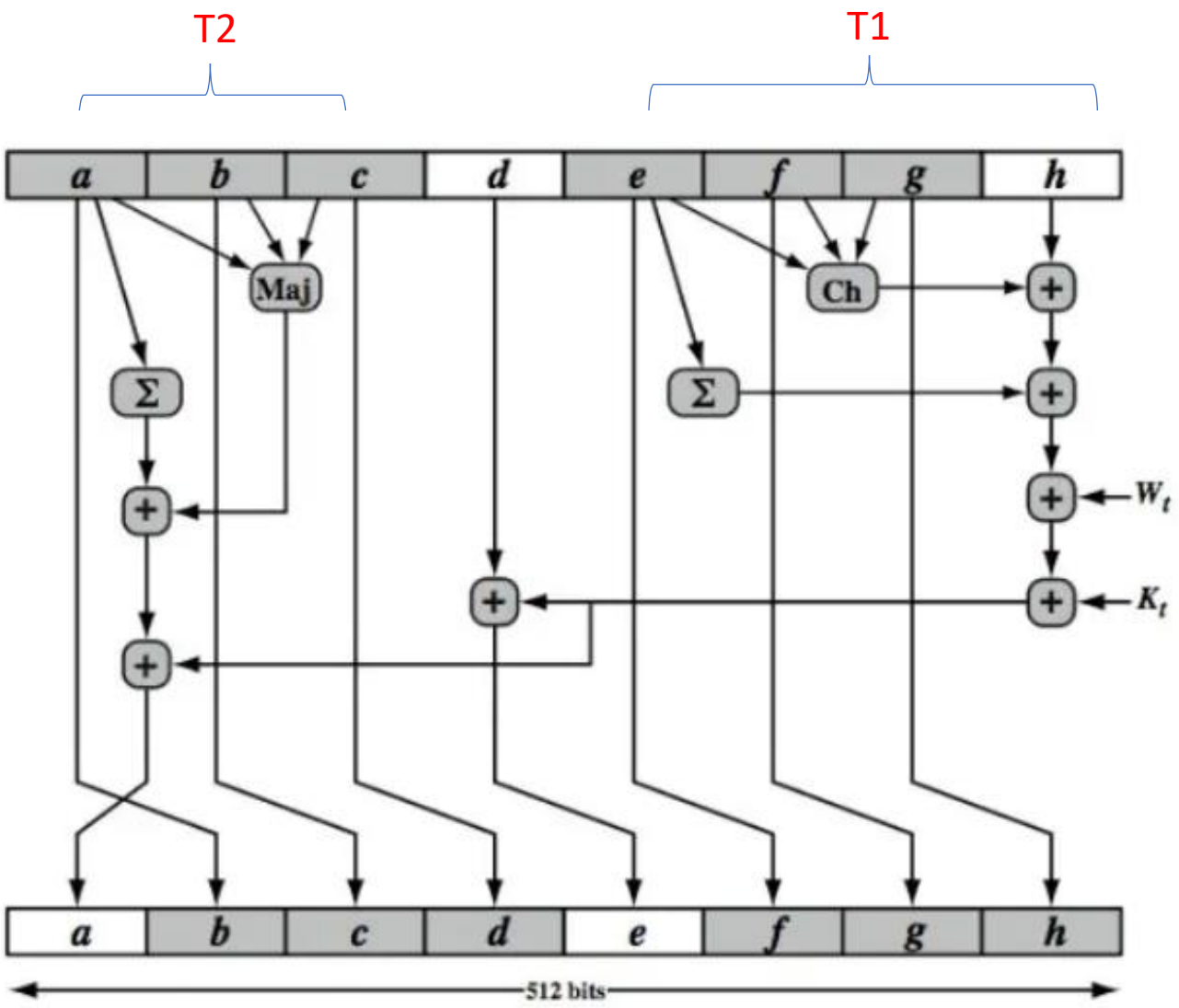


Figure 21.3 **SHA-512 Processing of a Single 1024-Bit Block**

What happens in a Round?

- **Round Function**



What happens in a Round? (cont.)

- **Computing T1 –**

$$T_1 = h + \text{Ch}(e, f, g) + \left(\sum_1^{512} e\right) + W_t + K_t$$

- Ch – conditional function

$$\text{Ch}(e, f, g) = (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$$

$$\left(\sum_1^{512} e\right) = \text{ROTR}^{14}(e) \oplus \text{ROTR}^{18}(e) \oplus \text{ROTR}^{41}(e)$$

$\text{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

What happens in a Round? (cont.)

- **Computing T1 –**

$$T_1 = h + \mathrm{Ch}(e, f, g) + \left(\sum_1^{512} e\right) + W_t + K_t$$

- Kt – 80 constants

Table 12.3 *Eighty constants used for eighty rounds in SHA-512*

428A2F98D728AE22	7137449123EF65CD	B5C0FBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	0FC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CB0A9DCBD41FBD4	76F988DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6E00BF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670A0E6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766A0ABB3C77B2A8	81C2C92E47EDAEE6	92722C8514823538
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70D0F89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2D0C8	1E376C085141AB53	2748774CDF8EEB99	34B0BCB5E19B48A8
391C0CB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1F0AB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721C0C207	EADA7DD6CDE0EB1E	F57D4F7FEE6ED178
06F067AA72176FBA	0A637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBE0A15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817

What happens in a Round? (cont.)

- **Computing T2 –**

$$T_2 = \left(\sum_0^{512} a\right) + \text{Maj}(a, b, c)$$

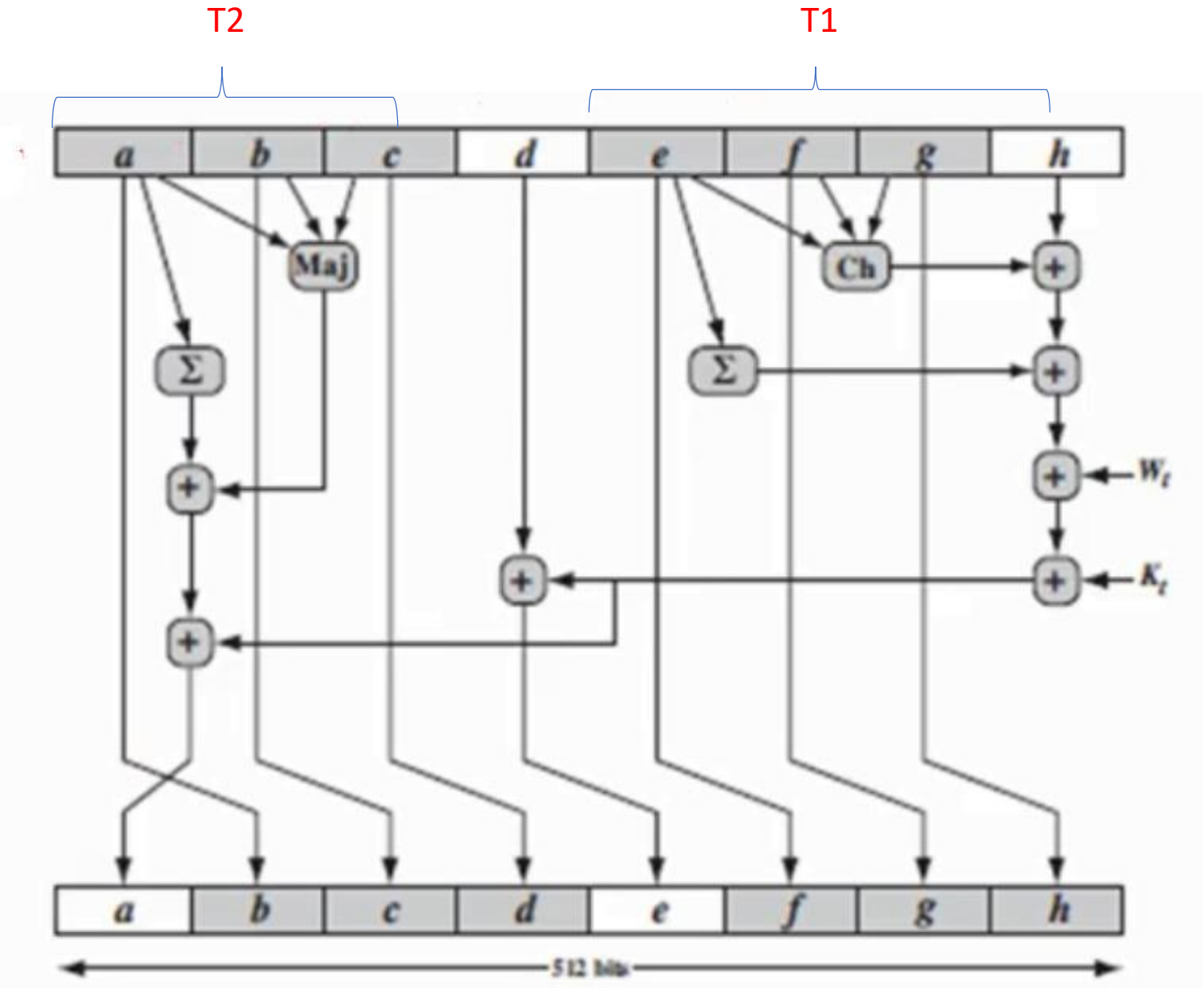
$$\text{Maj}(a, b, c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$$

$$\left(\sum_0^{512} a\right) = \text{ROTR}^{28}(a) \oplus \text{ROTR}^{34}(a) \oplus \text{ROTR}^{39}(a)$$

- Maj(a,b,c) – majority function

What happens in a Round? (cont.)

- Round



$$a = T_1 + T_2$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$

SHA - 512

Step 5:

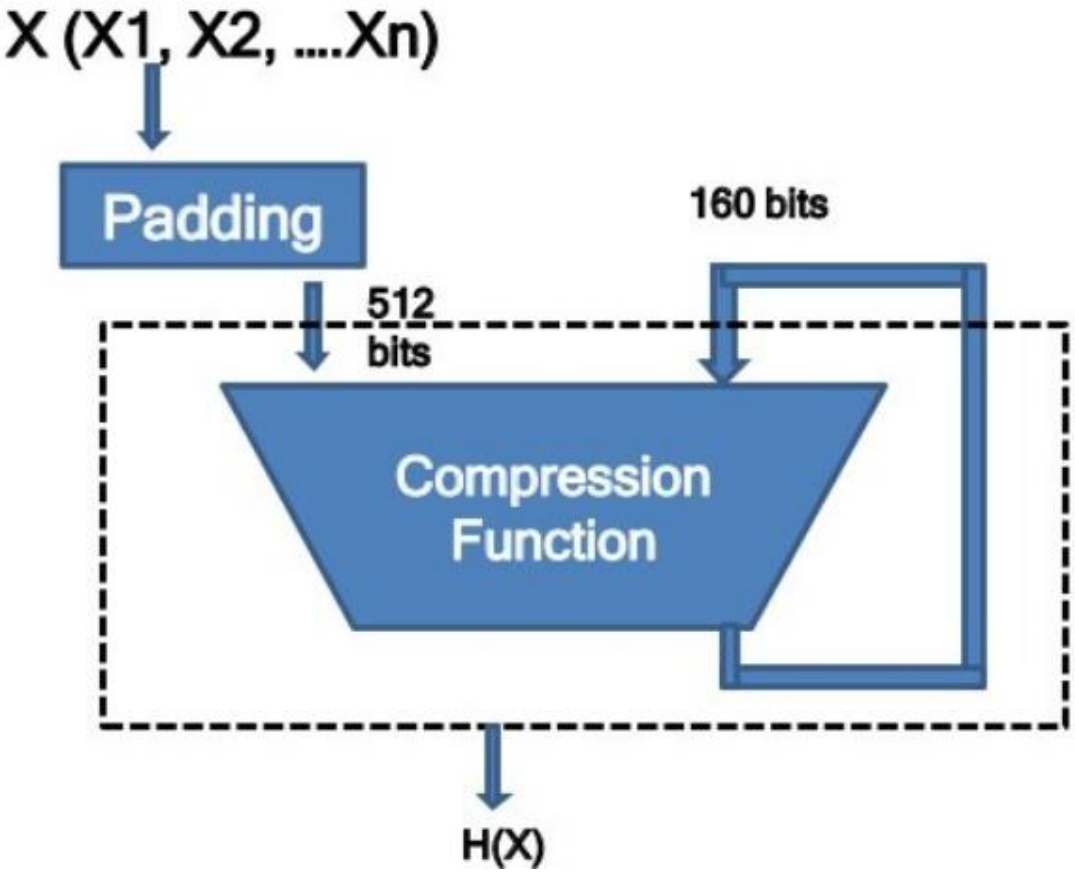
Output. After all ‘N’ 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.

Reference – Fourozan and Stalling

SHA - 1

- SHA1: Secure Hash Algorithm 1
- Designed by the United States National Security Agency
- Produces hash value known as Message Digest
- Works for any input message that is less than 2^{64} bits
- produces160 bits length message digest
- Infeasible to retain the original message from the message digest
- Same message digest to be produced from both sender and receiver
- Purpose: Authentication , not Encryption
- widely used in security applications and protocols, including TLS, SSL, PGP, SSH, IPSec and S/MIME

SHA-1 Steps



1. Append padding bits
2. Append Length
3. Initialize hash bufffer
4. SHA processing
5. Output

SHA-1 Steps

Step 1: Append Padding bits

padding bits are added to the original message to make the original message equal to a value divisible by 512.

Example –

- The massage padding is applied to the last data block such that SHA-1 can process the data of n×512 bits.
- The last two words (64 bits) of padded message are reserved of the original message length (in bits).
- Input message – ‘abcde’ – 40 bits
- 01100001 01100010 01100011 01100100 01100101.
- After ‘1” is appended, 407 ‘0’ are required to complete 448 bits. In Hex, this can be written as:

61626364 65800000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000

SHA-1 Steps

Step 2: Append length

The rest two words are preserved for the original message

length.

As per example, length of msg = 40 = “00000000 00000028” (Hexadecimal Value).

As a result, the passed massage is

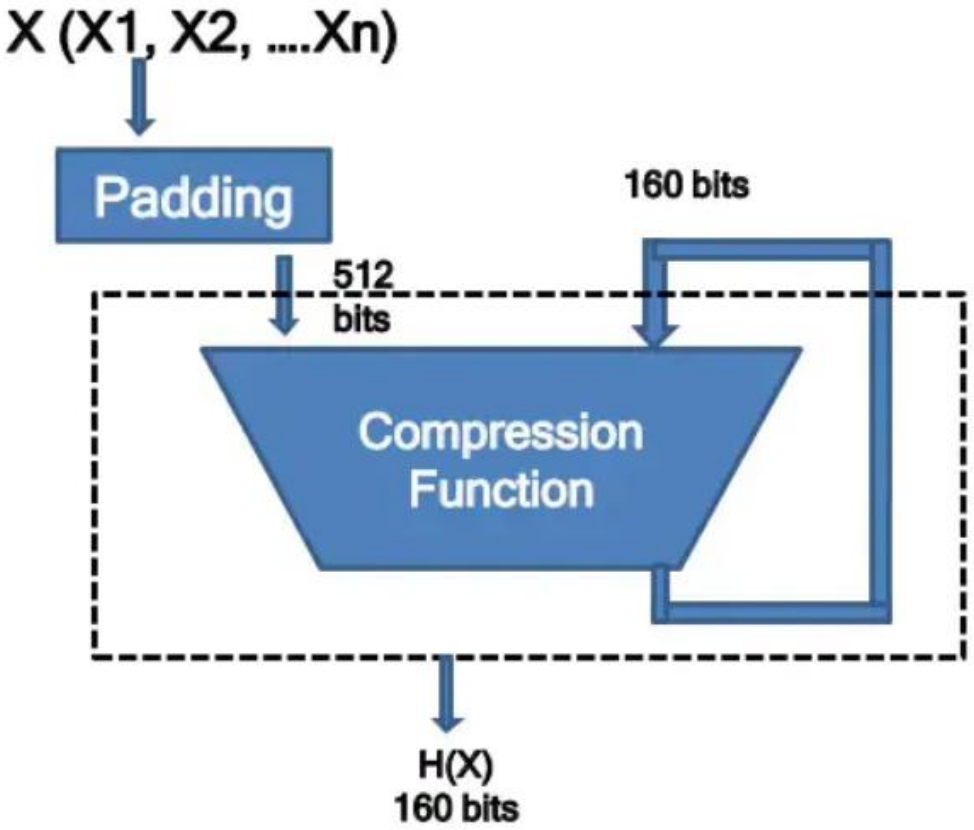
61626364 65800000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000028.

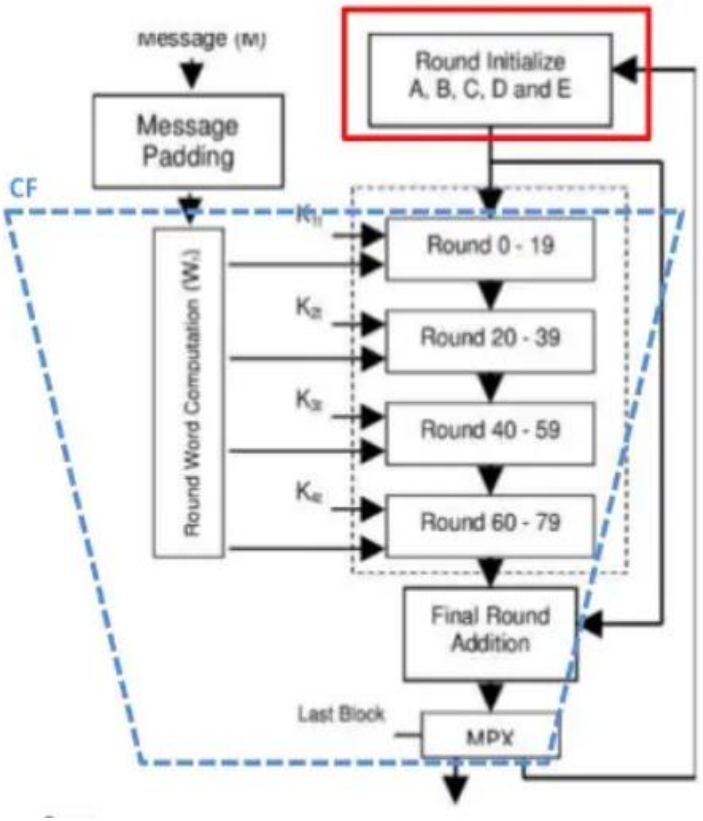
SHA-1 Steps



- This 512 bits input to the compression function
- The message divided into 16 words.
- Each word consists of 32 bits.
- 512/32 =16 words

SHA-1 Steps

Step 3: Initialize the hash buffer



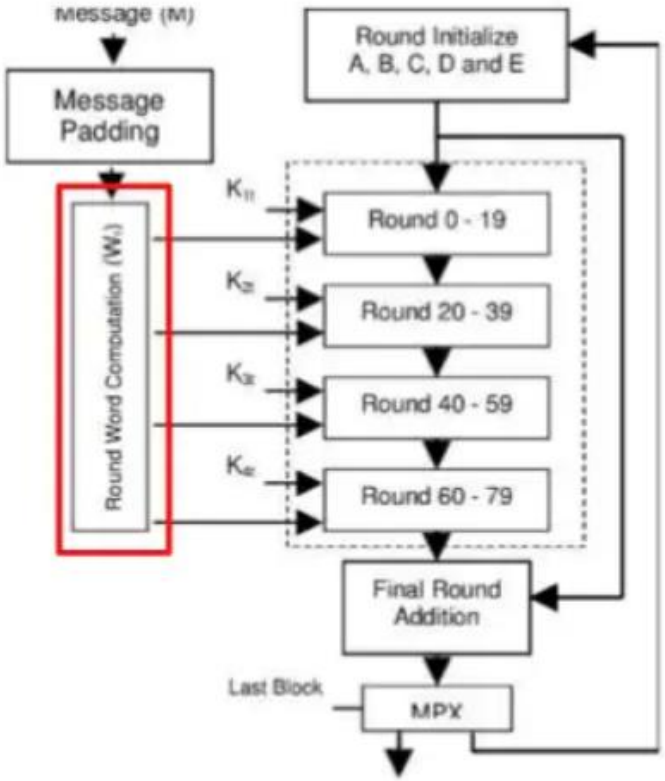
Initial values of Ho are predefined and stored in registers ABCDE

H	Hex values
Ho(A)	01234567
Ho(B)	89ABCDEF
Ho(C)	FEDCBA98
Ho(D)	76543210
Ho(E)	C3D2E1F0

These initial values are used in Round 0.

SHA-1 Steps

Step 4: SHA Processing



Word assigning to rounds:

SHA1 has 80 rounds defined.

The Message Scheduler Algorithm schedules each word to rounds as:

- W_0 → Round 0
- W_1 → Round 1

  .....
- W_{15} → Round 15
- W_{16} → Round 16

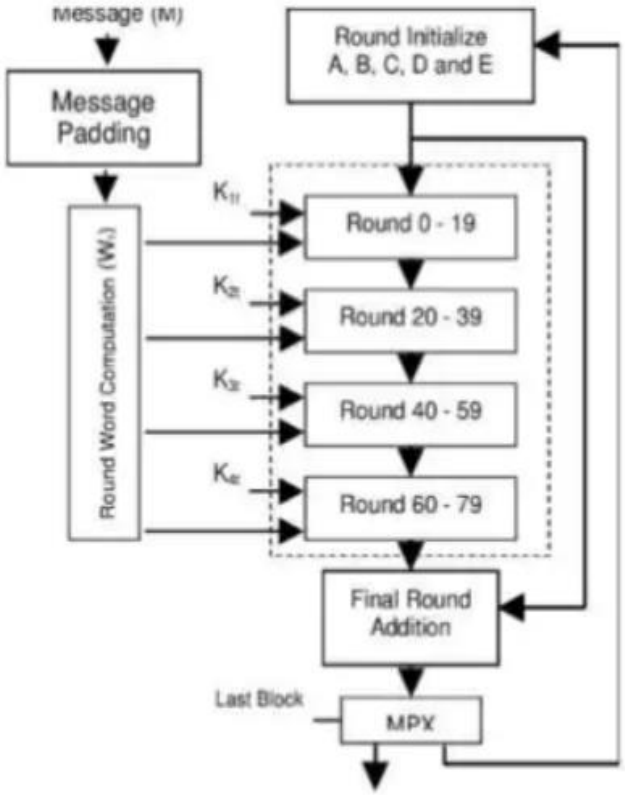
  .....
- W_{79} → Round 79

W_t is calculated

Each round = 20 iterations. Total iterations = 80

SHA-1 Steps

Step 4: SHA Processing



Word assigning to other rounds:

For others (i.e round 16- 79)

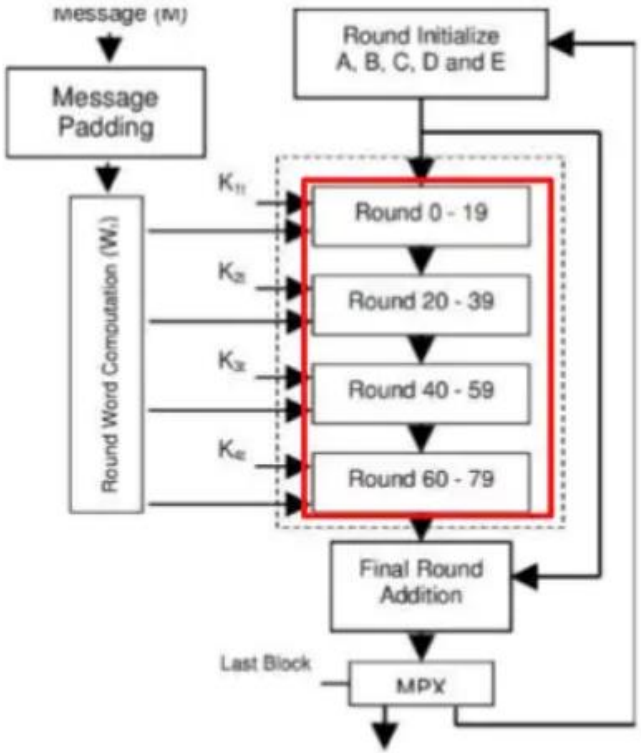
$W_{[t]} = S^1(w_{[t-16]} \text{ XOR } W_{[t-14]} \text{ XOR } W_{[t-8]} \text{ XOR } W_{[t-3]})$

For example: when round is 16,

- $W_{[16]} = S^1(w_{[16-16]} \text{ XOR } W_{[16-14]} \text{ XOR } W_{[16-8]} \text{ XOR } W_{[16-3]})$
- Here W_0, W_2, W_8 and W_{13} are XORed.
- The output is the new word for round 16.

SHA-1 Steps

Step 4: SHA Processing



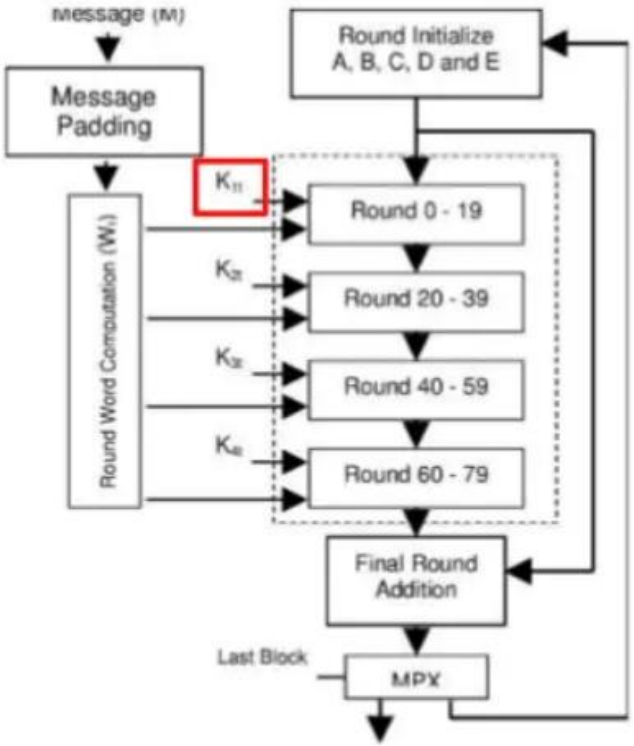
Division of stages:

Stages	Round
t=1	0 to 19
t=2	20 to 39
t=3	40 to 59
t=4	60 to 79

Each stage has 20 rounds.

SHA-1 Steps

Step 4: SHA Processing



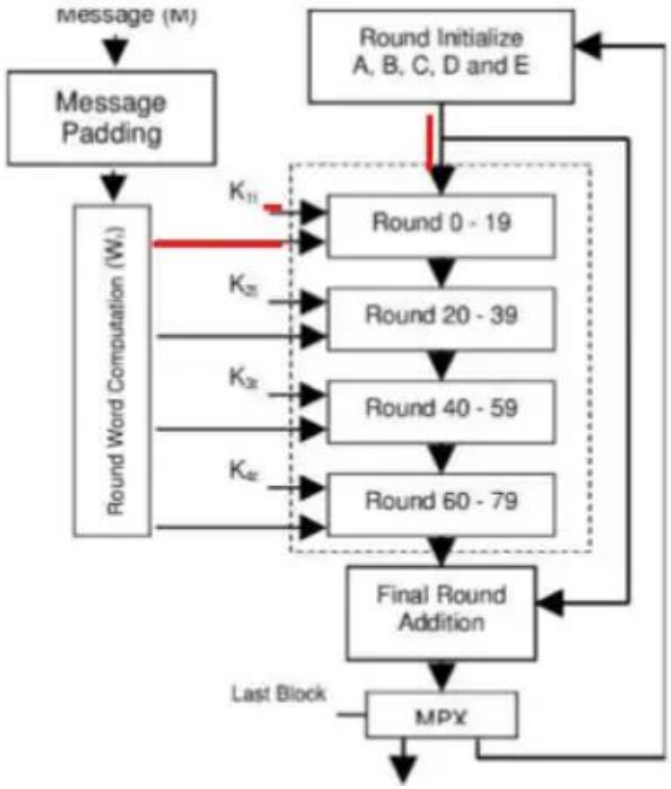
Constant values:

At each stage:

Stage	Predefined value of k
t_1	K_1=0X5A827999
t_2	K_2= 0X6ED9EBA1
t_3	K_3=0X8F1BBCDC
t_4	K_4=0XCA62C1D6

SHA-1 Steps

Step 4: SHA Processing



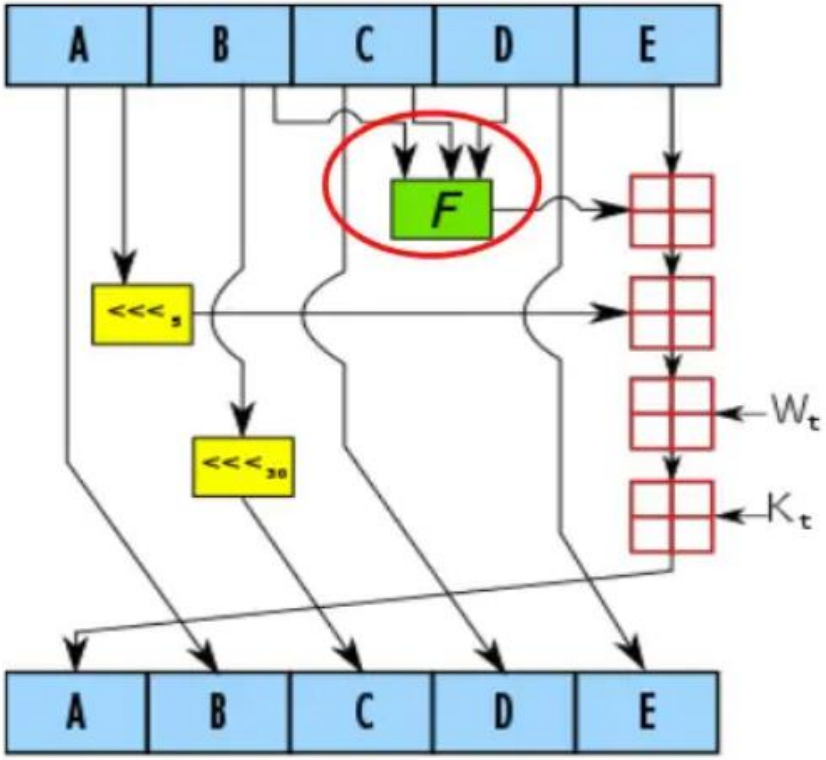
Process in each round:

Each round takes 3 inputs:

- 32 bit word form 512 bit block (i.e.W_t)
- The values from register ABCDE
- Constant K_t

SHA-1 Steps

Step 4: SHA Processing

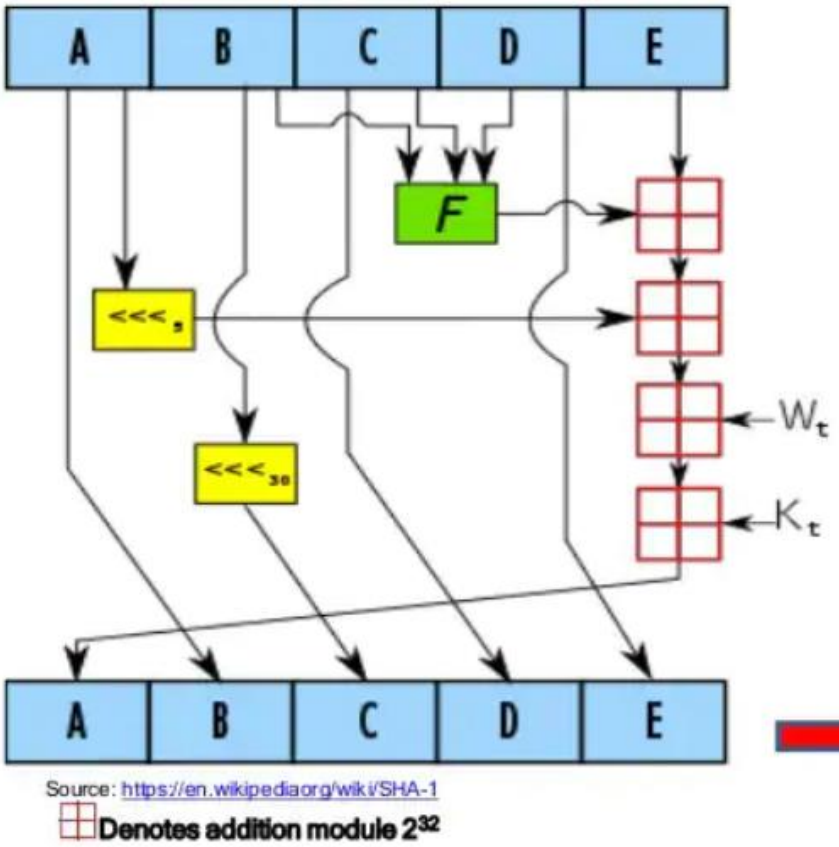


F_t at different stages:

Stage	F_t
t_1	$F_t(B.C,D)$ = (B AND C) OR ((NOT B) AND D)
t_2	$F_t(B.C,D)$ = B XOR C XOR D
t_3	$F_t(B.C,D)$ = (B AND C) OR (B AND D) OR (C AND D)
t_4	$F_t(B.C,D)$ = B XOR C XOR D

SHA-1 Steps

Step 4: SHA Processing



Source: https://en.wikipediaorg/wiki/SHA-1

Denotes addition module 2^{32}

At each Round:

- Output of F_t and E are added
- Value in register A is 5 bit circular-left shifted.
- This then added to previous sum.
- W_t is added
- K_t introduced
- B is circular-left shifted by 30 bits.

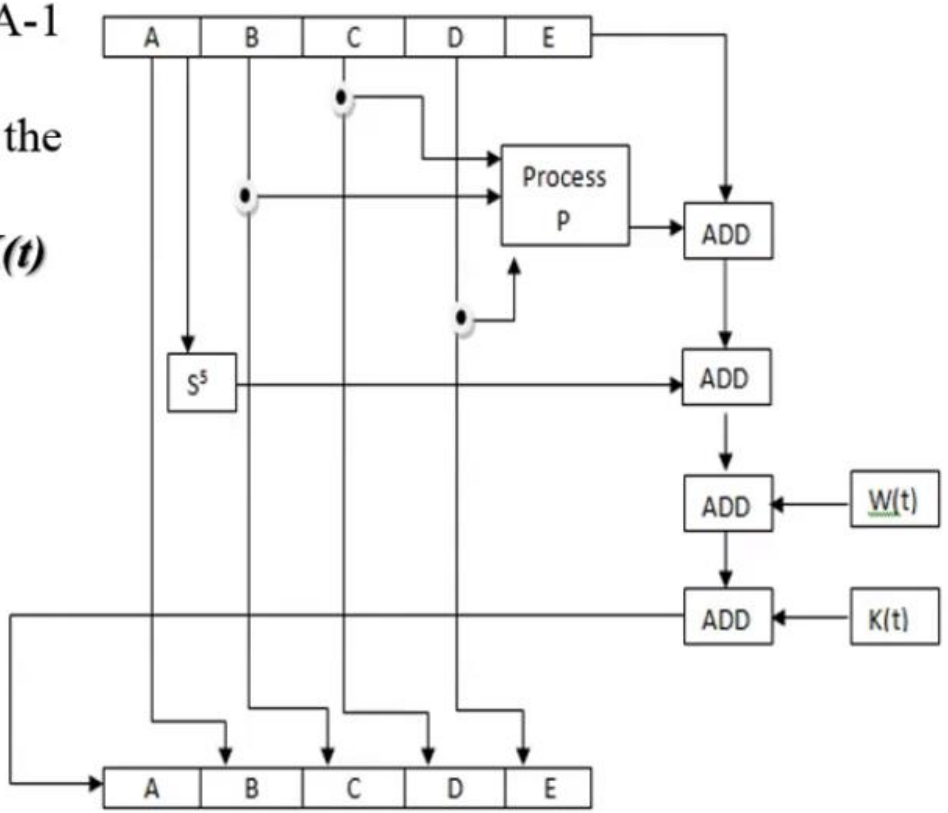
New values for next round

SHA-1 Steps

Step 4: SHA Processing

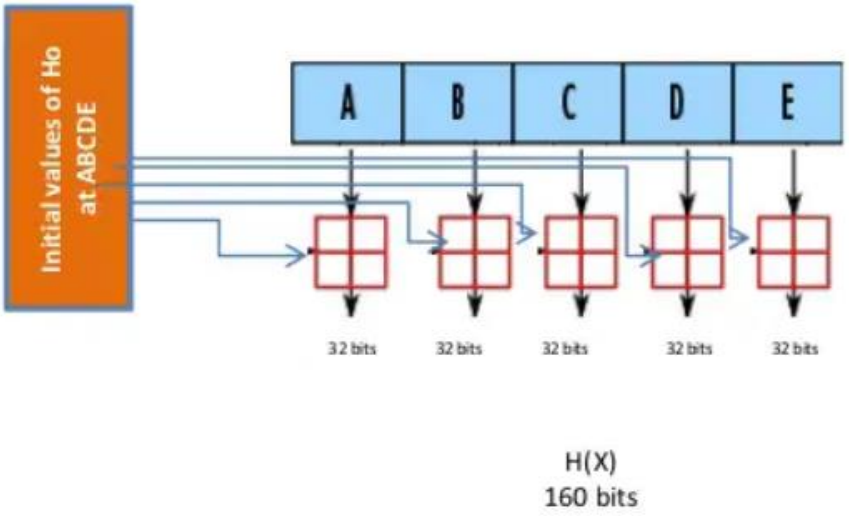
- SHA-1 has perform four rounds.
- Each round takes the current 512-bit block, the register ABCDE and constant K(t) (where t=0 to 79) as input.
- SHA consists of four rounds, each round containing 20 iteration. So total iteration is 80.
- The logical operation of a single SHA-1 iteration looks as shown in figure.
- Mathematically, an iteration consists of the following operation:

$$ABCDE = E + Process\ P + S^5(a) + W(t) + K(t)$$



SHA-1 Steps

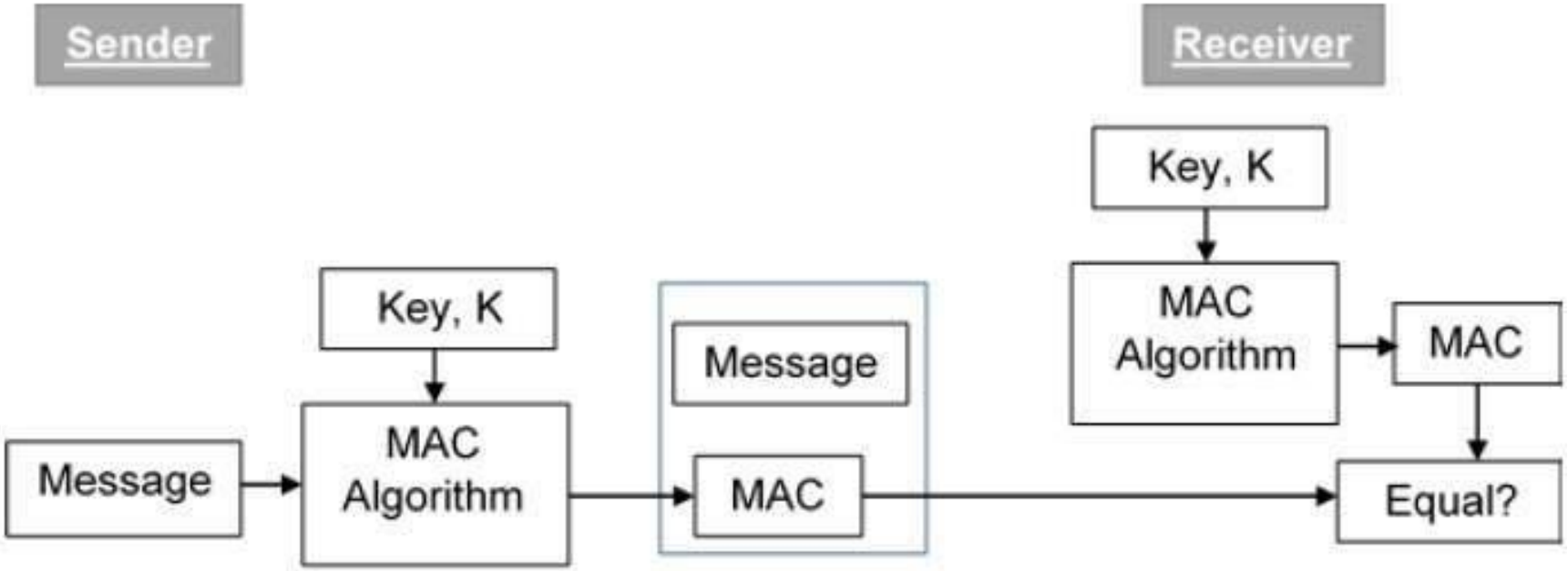
Step 4: The Output



After Final Round:

- The 160 bit output from the final round is modulo added to the initial predefined values of Ho at registers ABCDE.
- Output obtained thus is a 160 bit hash code.

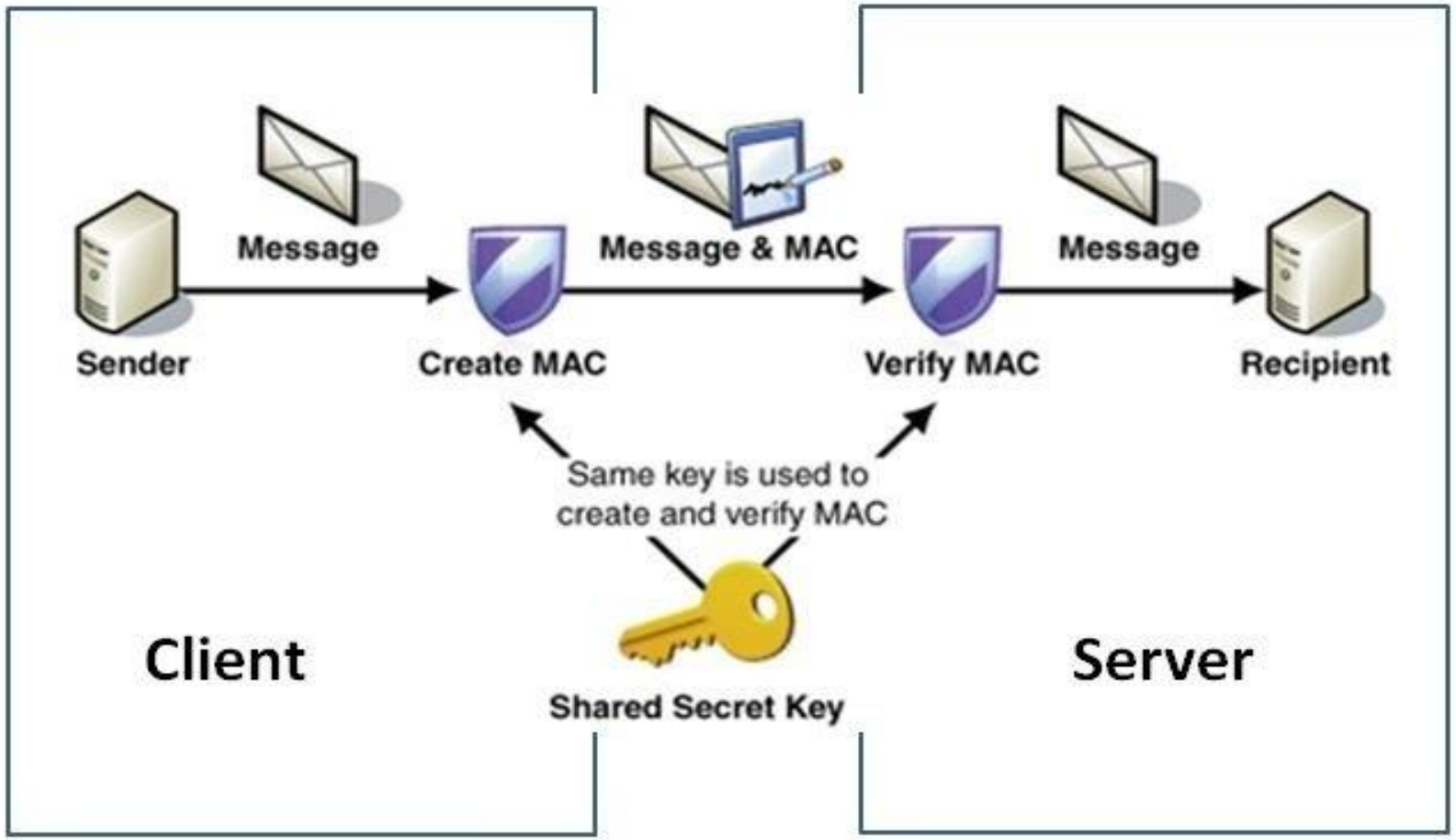
Let's recall MAC ..



Hash based Message Authentication Code (HMAC)

- Hash-based message authentication code (HMAC) is a mechanism for calculating a message authentication code involving a hash function in combination with a secret key. This can be used to verify the integrity and authenticity of a message.
- HMACs are almost similar to digital signatures. They both enforce integrity and authenticity. They both use cryptography keys. And they both employ hash functions.
- The main difference is that digital signatures use asymmetric keys, while HMACs use symmetric keys (no public key).

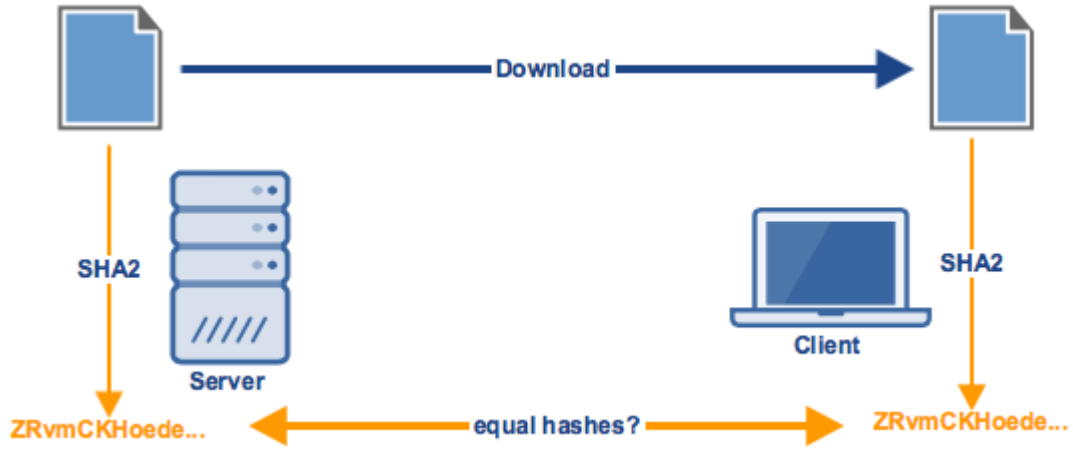
HMAC Authentication



How does HMAC Work ?

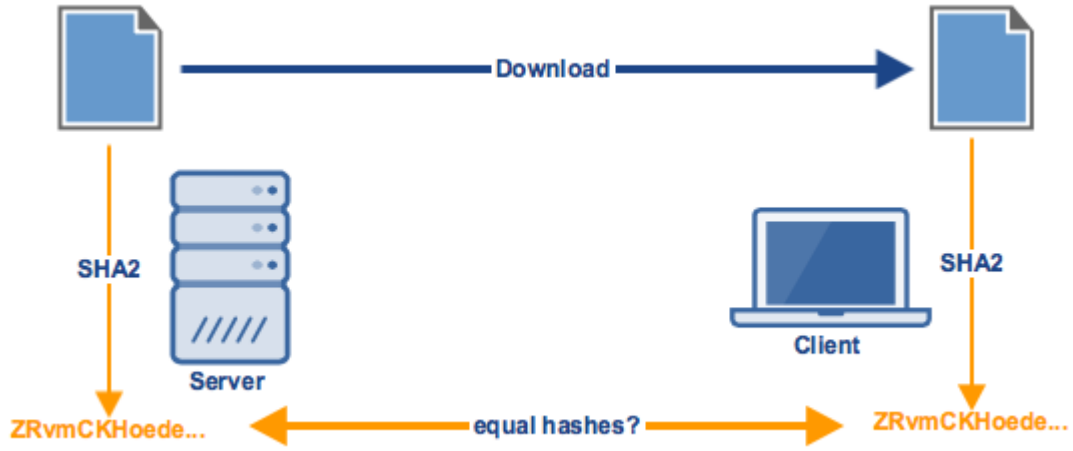
A data integrity check on a file transfer.

- Let's say a client application downloads a file from a remote server. It's assumed that the client and server have already agreed on a common hash function, for example SHA2.



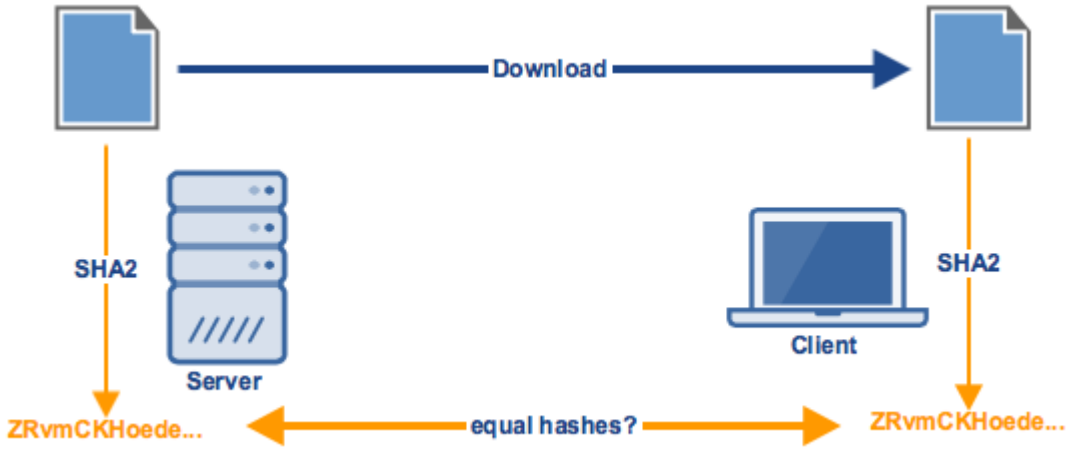
How does HMAC Work ?

- Before the server sends out the file, it first obtains a hash of that file using the SHA2 hash function. It then sends that hash (ex. a message digest) along with the file itself.
- Upon receiving the two items (ex. the downloaded file and the hash), the client obtains the SHA2 hash of the downloaded file and then compares it with the downloaded hash.
- If the two match, then that would mean the file was not tampered with.



How does HMAC Work ?

- If an attacker manages to intercept the downloaded file, alter the file's contents, and then forward the altered file to the recipient, that malicious act won't go unnoticed.
- That's because, once the client runs the tampered file through the agreed hash algorithm, the resulting hash won't match the downloaded hash.
- This will let the receiver know the file was tampered with during transmission.



How does HMAC Work ?

Authenticity Check

- An HMAC employs both a hash function and a shared secret key.
- A shared secret key provides exchanging parties a way to establish the authenticity of the message.
- That is, it provides the two parties a way of verifying whether both the message and MAC (more specifically, an HMAC) they receive really came from the party they're supposed to be transacting with.

How does HMAC Work ?

Suitable for File Transfers

Efficiency - hash functions can take a message of arbitrary length and transform it into a fixed-length digest. That means, even if you have relatively long messages, their corresponding message digests can remain short, allowing you to maximize bandwidth.

HMAC Structure

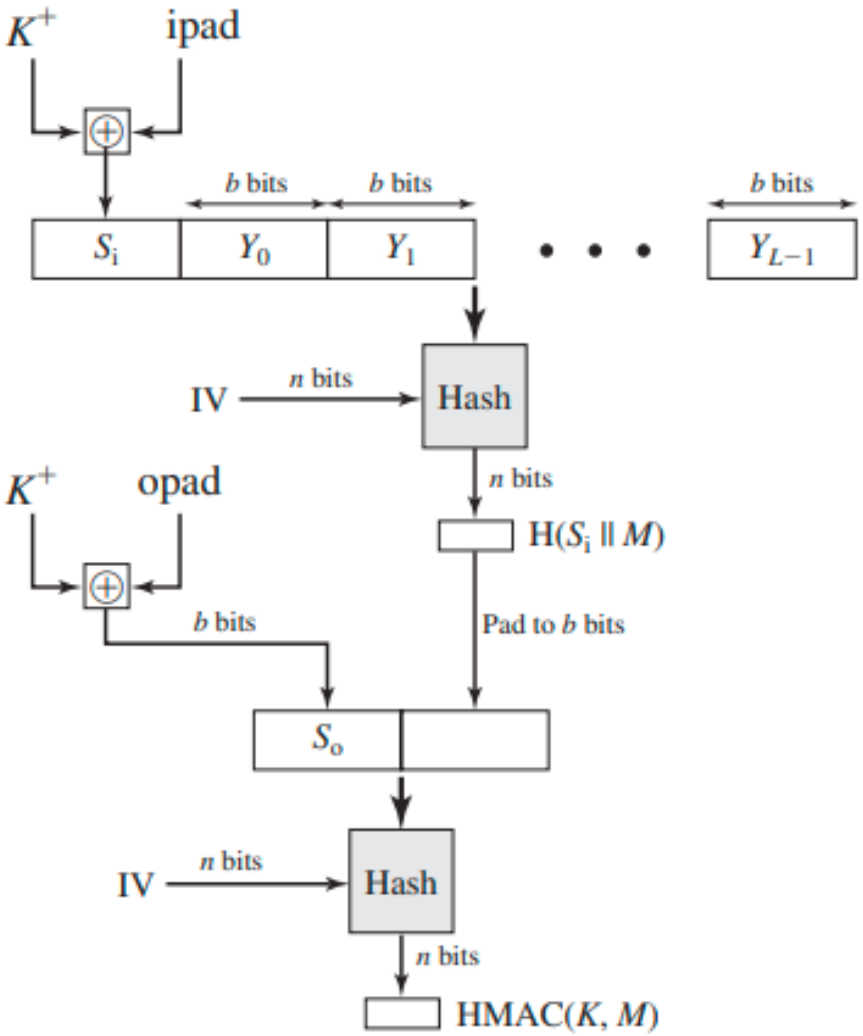


Figure 21.4 **HMAC Structure**

HMAC Structure

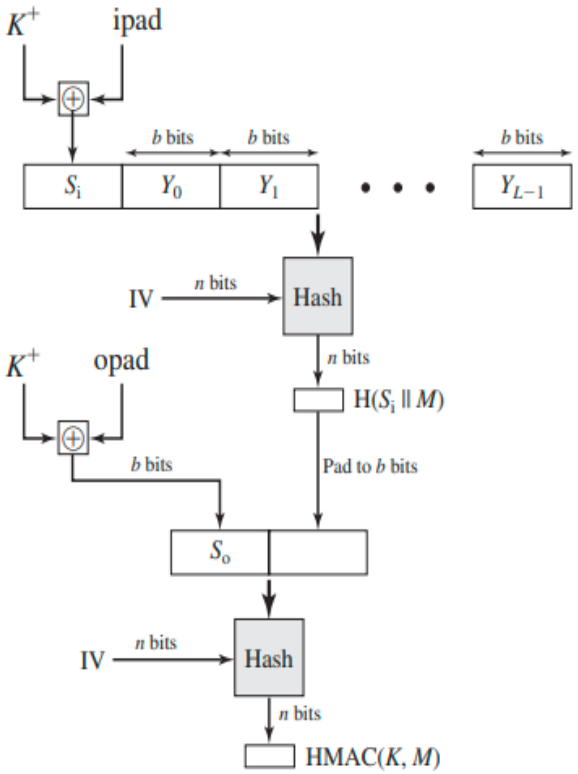


Figure 21.4 **HMAC Structure**

Figure 21.4 illustrates the overall operation of HMAC. Define the following terms:

- II = embedded hash function (e.g., SHA)
- M = message input to HMAC (including the padding specified in the embedded hash function)
- Y_i = ith block of $M, 0 \leq i \leq (L - 1)$
- L = number of blocks in M
- b = number of bits in a block
- n = length of hash code produced by embedded hash function
- K = secret key; if key length is greater than b, the key is input to the hash function to produce an n-bit key; recommended length is $\geq n$
- 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

Then HMAC can be expressed as follows:

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

In words,

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 0x00).
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.

Reference - Stallings

MAC and HMAC reference

- Stalling

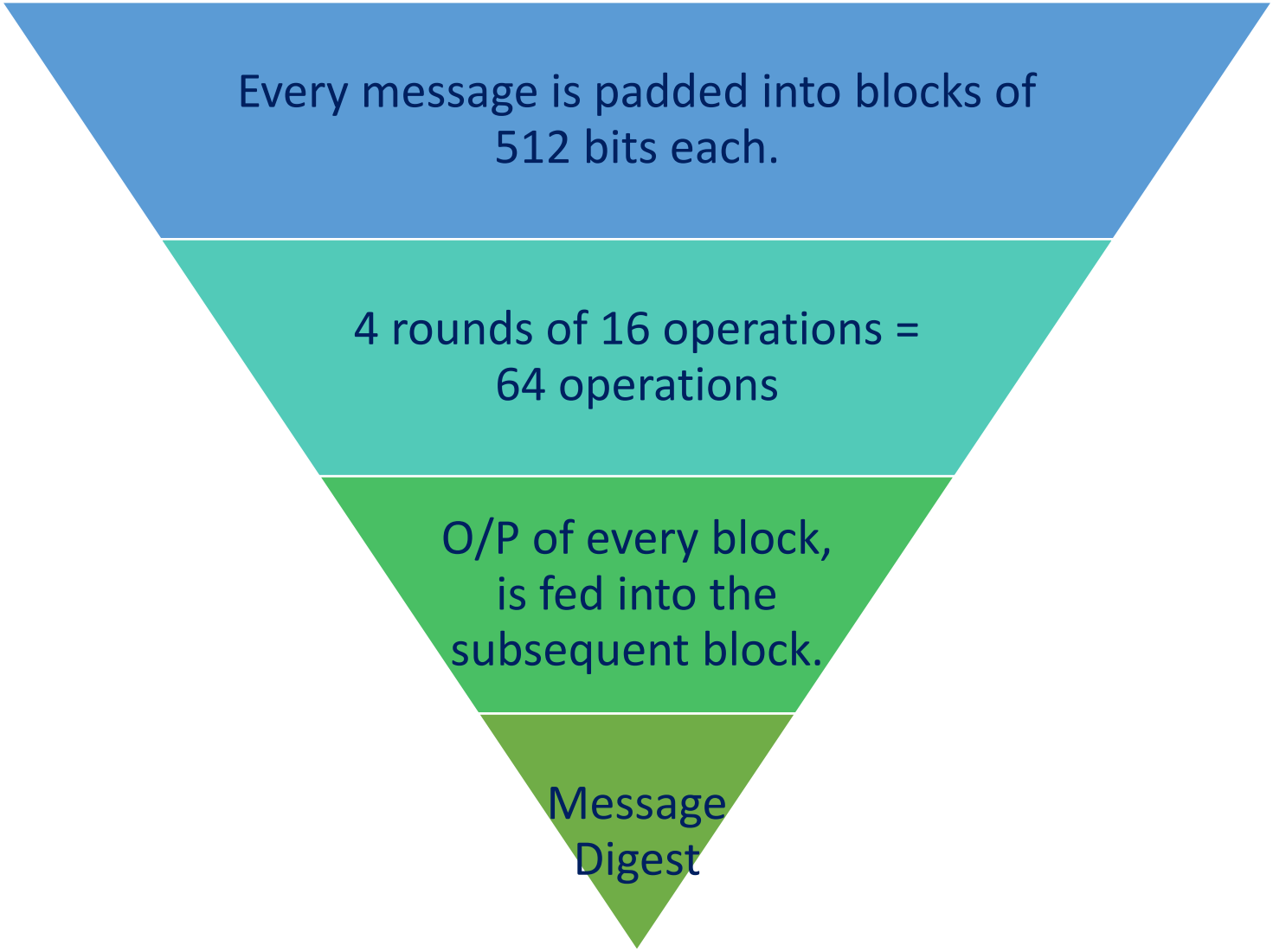
HMAC Security

- Security depends on the cryptographic strength of the underlying hash function
- It is much harder to launch successful collision attacks on HMAC because of secret key

Message Digest (MD 5)

- MD5 is the Message Digest algorithm 5, created by Ronald Rivest.
- It is the most widely used of the MD family of hash algorithms.
- MD5 creates a 128-bit hash value based on arbitrary input length.
- It verifies Integrity and authenticity of message.
- Initially designed for digital signatures.
- MD5 hashing is no longer considered reliable for use because security experts have demonstrated techniques capable of easily producing MD5 collisions on commercial off-the-shelf computers.

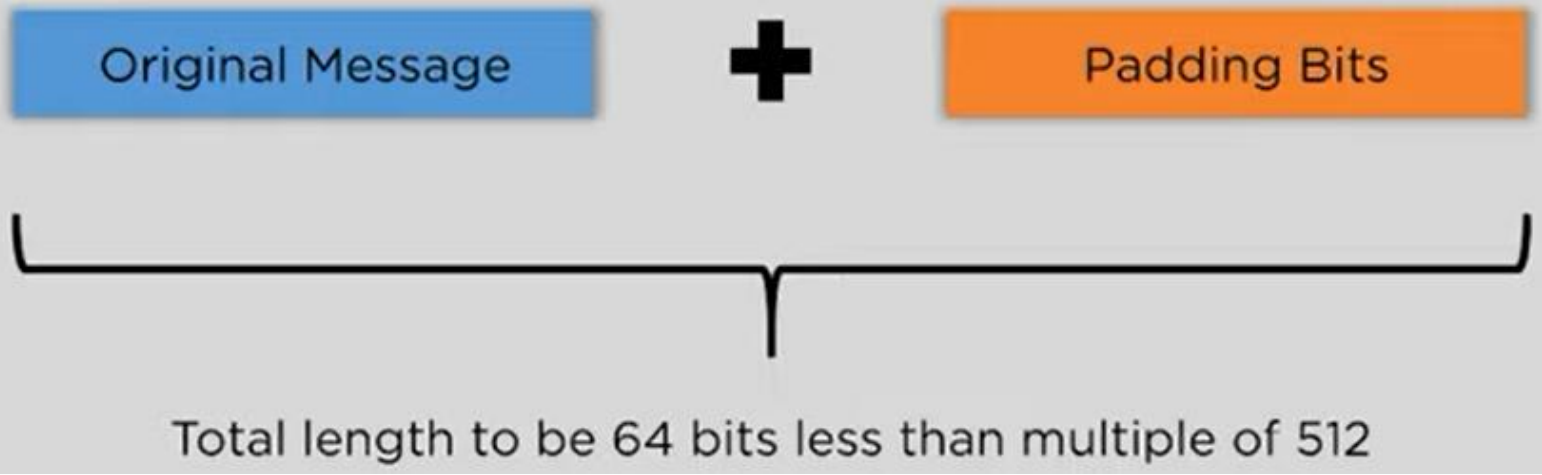
MD 5 Concept


Every message is padded into blocks of 512 bits each.
4 rounds of 16 operations = 64 operations
O/P of every block, is fed into the subsequent block.
Message Digest

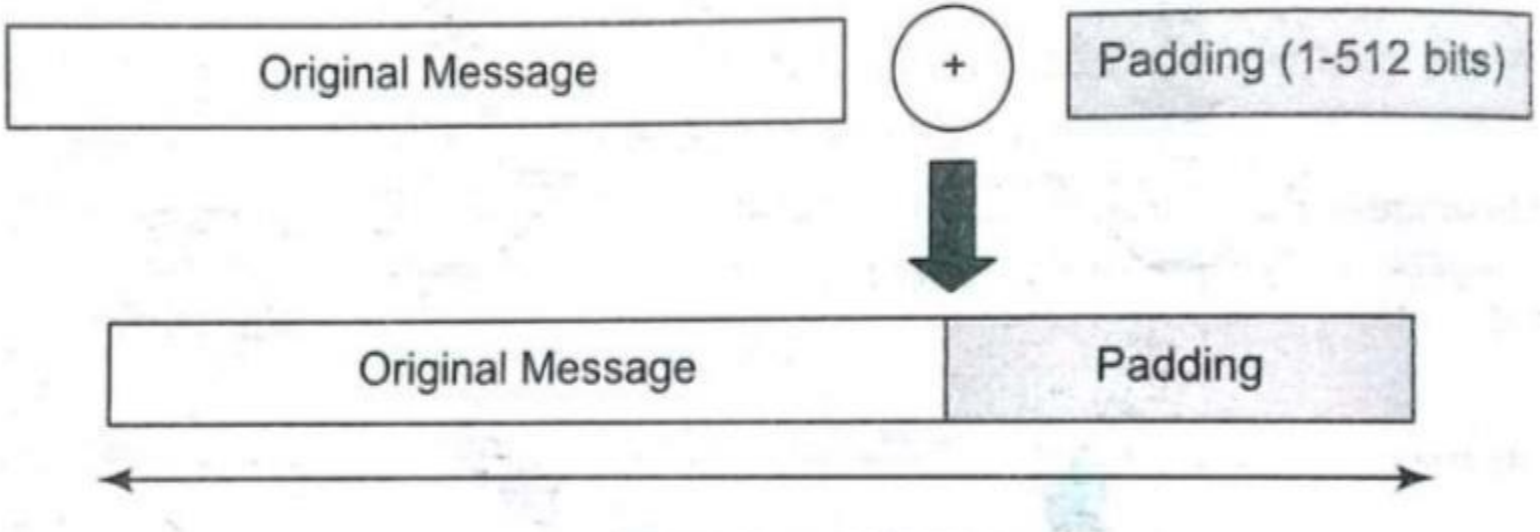
How MD5 works?

Step 1: Padding bits

- Bits are appended to the original input to make it **compatible** with the hash function.
- Total bits must always be **64 bits short** of any multiple of 512.
- The **first bit added is '1'**, and the rest are all zeroes.



Step 1: Padding bits (cont.)



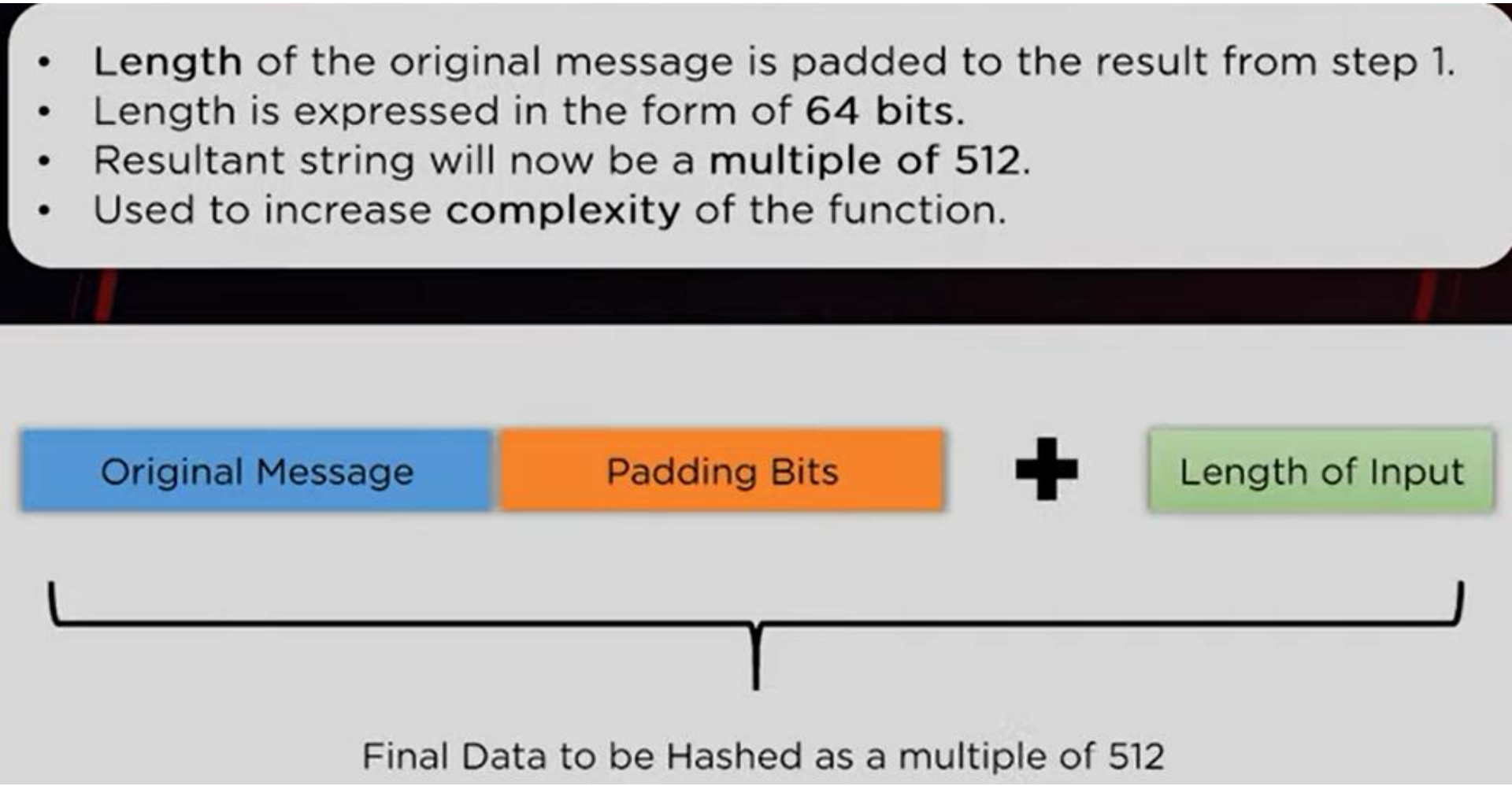
The total length of this should be 64 bits less than a multiple of 512.

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.

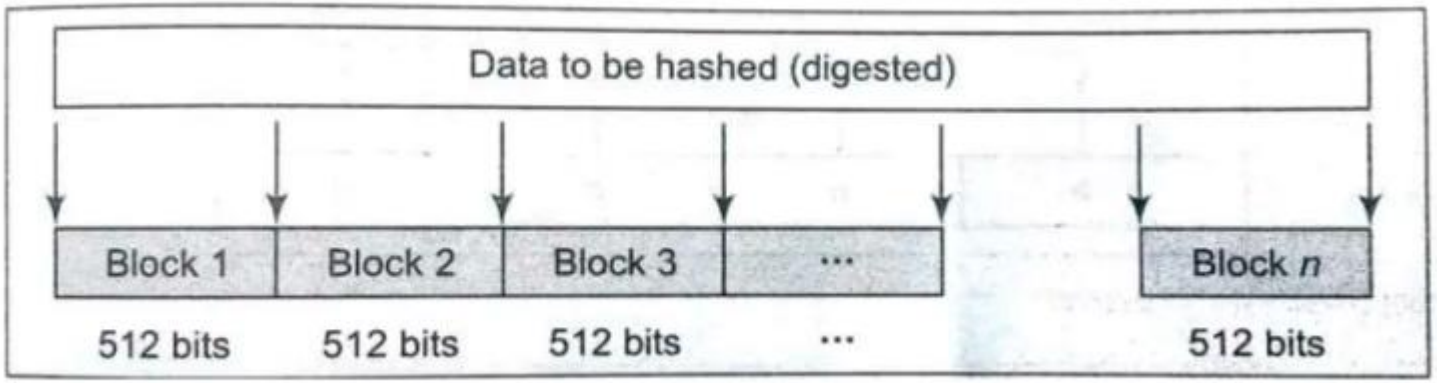
Note: Padding is always added, even if the original message is already a multiple of 512.

Step 2: Padding length

- **Length** of the original message is padded to the result from step 1.
- Length is expressed in the form of **64 bits.**
- Resultant string will now be a **multiple of 512.**
- Used to increase **complexity** of the function.



Step 3: Divide the I/P in 512 - bit block and Initialize Buffer



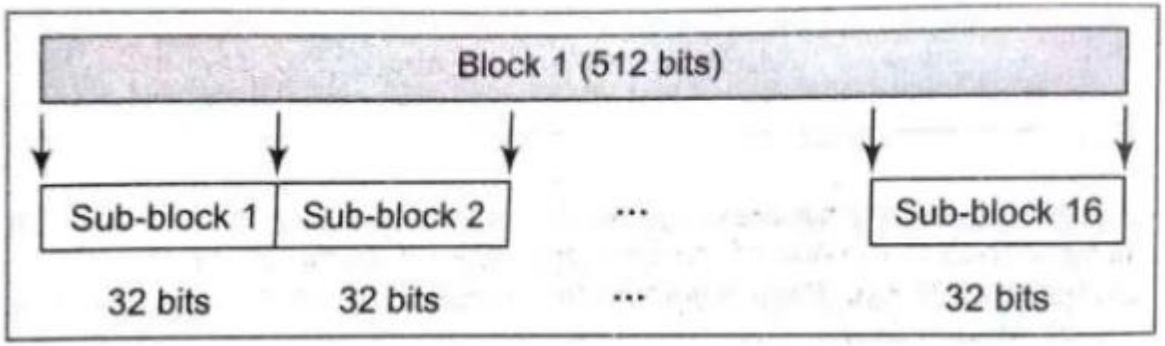
- The entire message is broken down into blocks of **512 bits** each.
- 4 buffers are used of **32 bits** each.
- They are **4 words** named A, B, C and D.
- The first iteration has **fixed** hexadecimal values.

A = 01 23 45 67 B = 89 ab cd ef

C = fe dc ba 98 D = 76 54 32 10

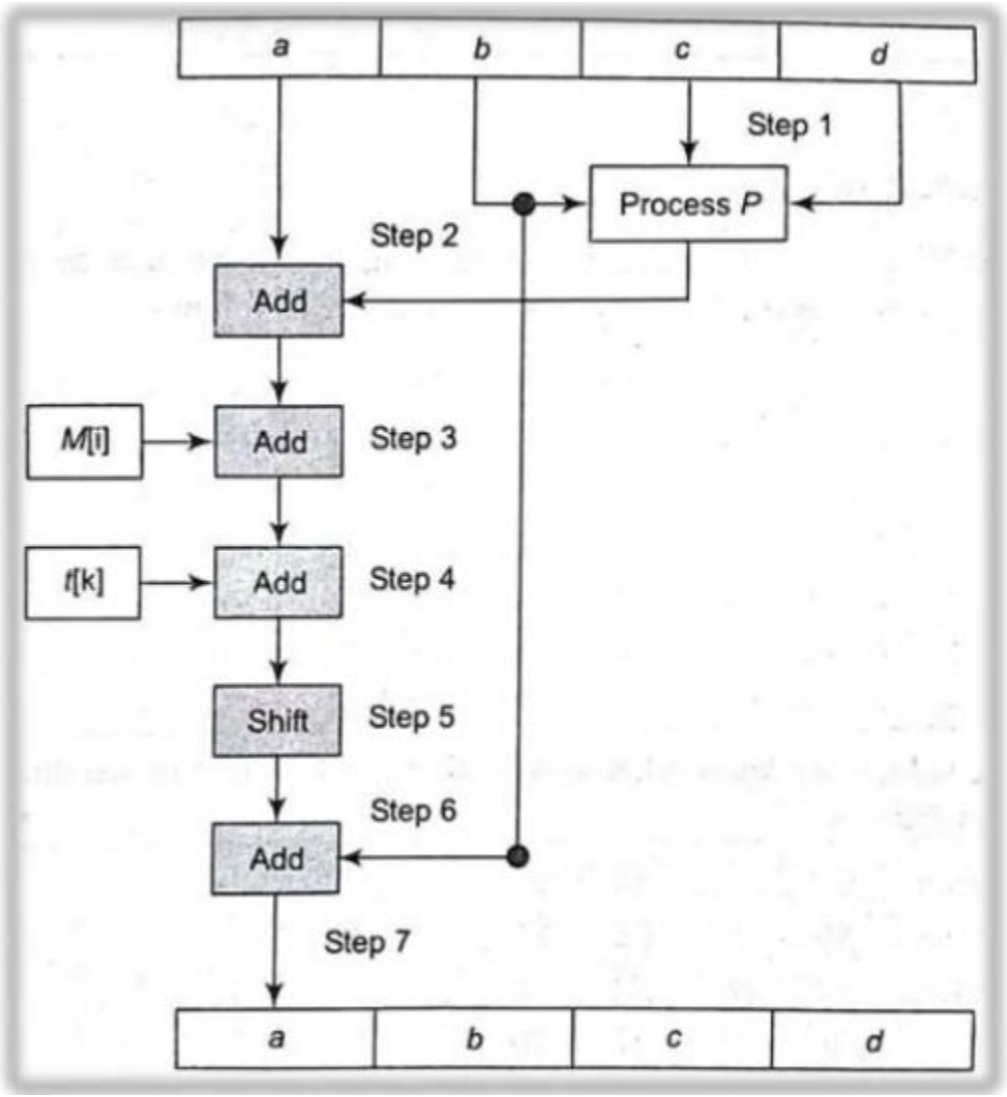
Step 4: Process each block

Divide the current 512-bit block in 16 sub blocks.



- Each block is broken down to **16 sub blocks** of 32 bit each.
- There are **4 rounds** of operations, each of them utilizing all 16 sub blocks, the 4 buffers and other constants.
- The constant value is an **array of 64 elements**, with 16 elements being used every round.
- Sub blocks : M[0], M[1], ... M[15]
- Constant array : T[1], T[2], ... T[64]

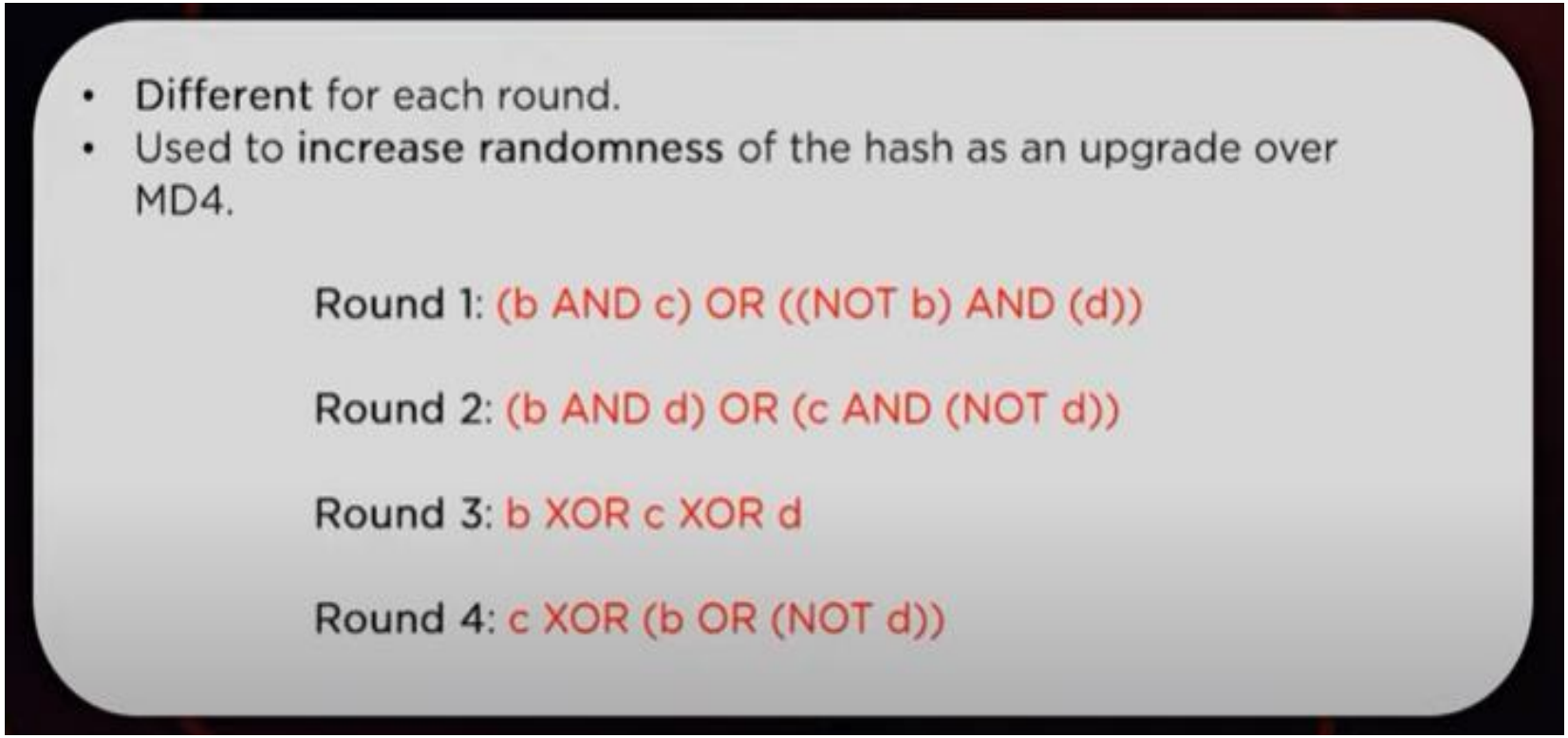
Step 4: Process each block (cont.)



We can mathematically express a single MD5 operation as follows:

$$a = b+((a + \text{Process P } (b,c,d)+M[i]+t[k])<<<s)$$

Non-linear Process Function



Compressed Function

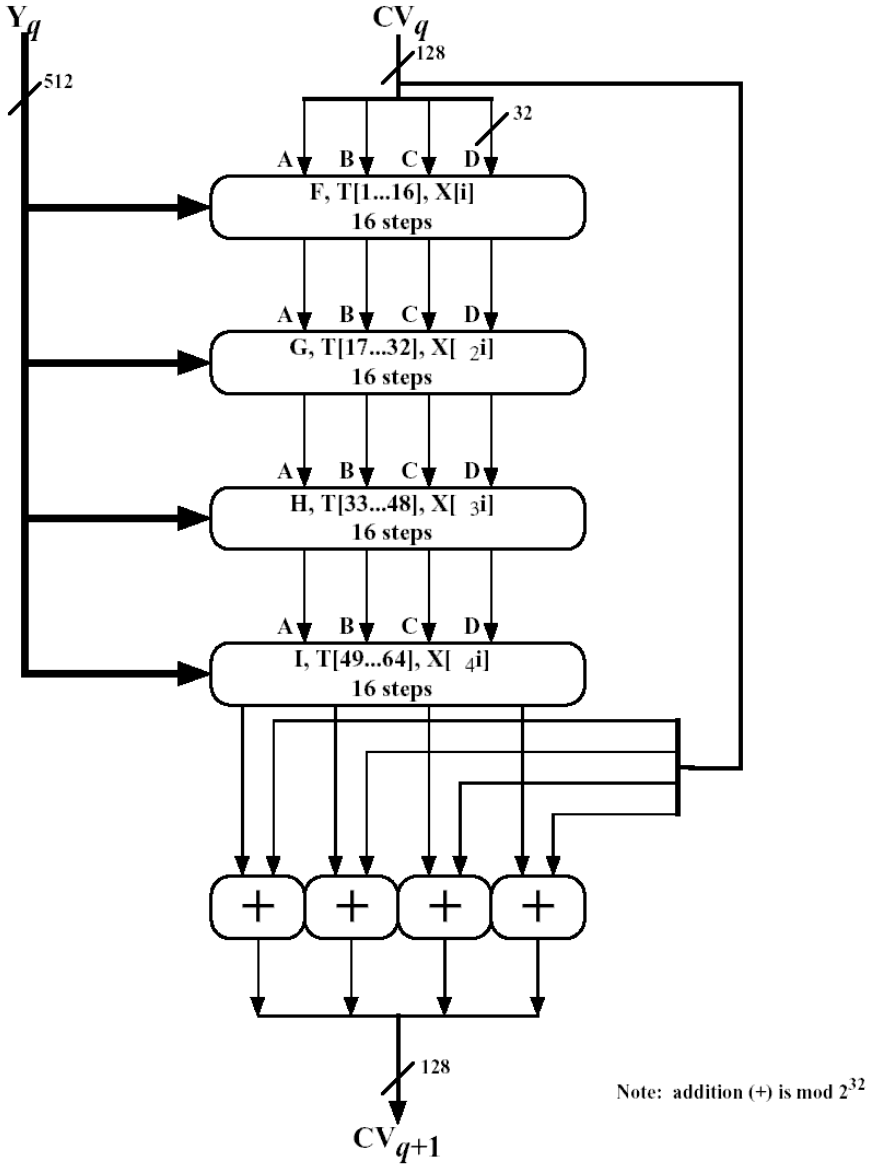
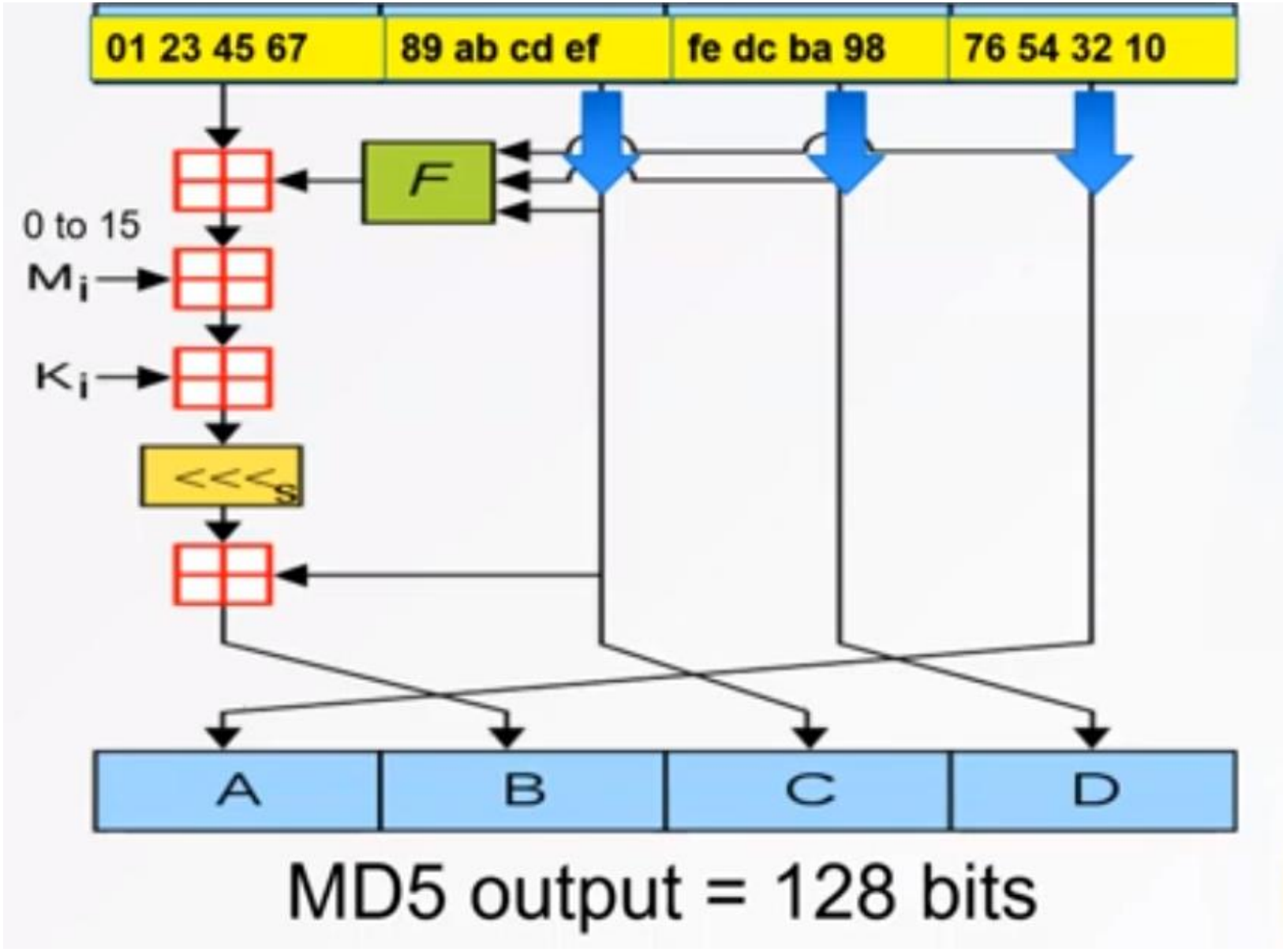


Figure 9.2 MD5 Processing of a Single 512-bit Block (MD5 Compression Function)

MD5 – new A,B,C,D


01 23 45 67
89 ab cd ef
fe dc ba 98
76 54 32 10
F
0 to 15
M_i
K_i
<<<s
A
B
C
D
MD5 output = 128 bits

Reference MD5

- Research paper's PDFs will be shared
- https://en.wikipedia.org/wiki/MD5
- https://datatracker.ietf.org/doc/html/rfc1321

Learn …

- Difference between MD5 and SHA
- Advantages and Disadvantages of
  - MD5
  - SHA
  - HMAC

MAC Implementation

- To realize and construct MAC algorithms, two different cryptographic primitives are used.
- MACs can be implemented using cryptographic hash functions or using symmetric block ciphers.
- Cryptographic hash functions - HMAC
- Symmetric block ciphers –
  - DAA,
  - CMAC
    - (1. CBC – MAC (it used AES) and
    - 2. variant of CBC-MAC is CMAC (AES + triple DES))
- *Reference – Authentication_paper.pdf and Stalling (for CMAC)*

MAC based on Block Ciphers

- Data Authentication Algorithm(DAA)
- Cipher Based Message Authentication Code(CMAC)

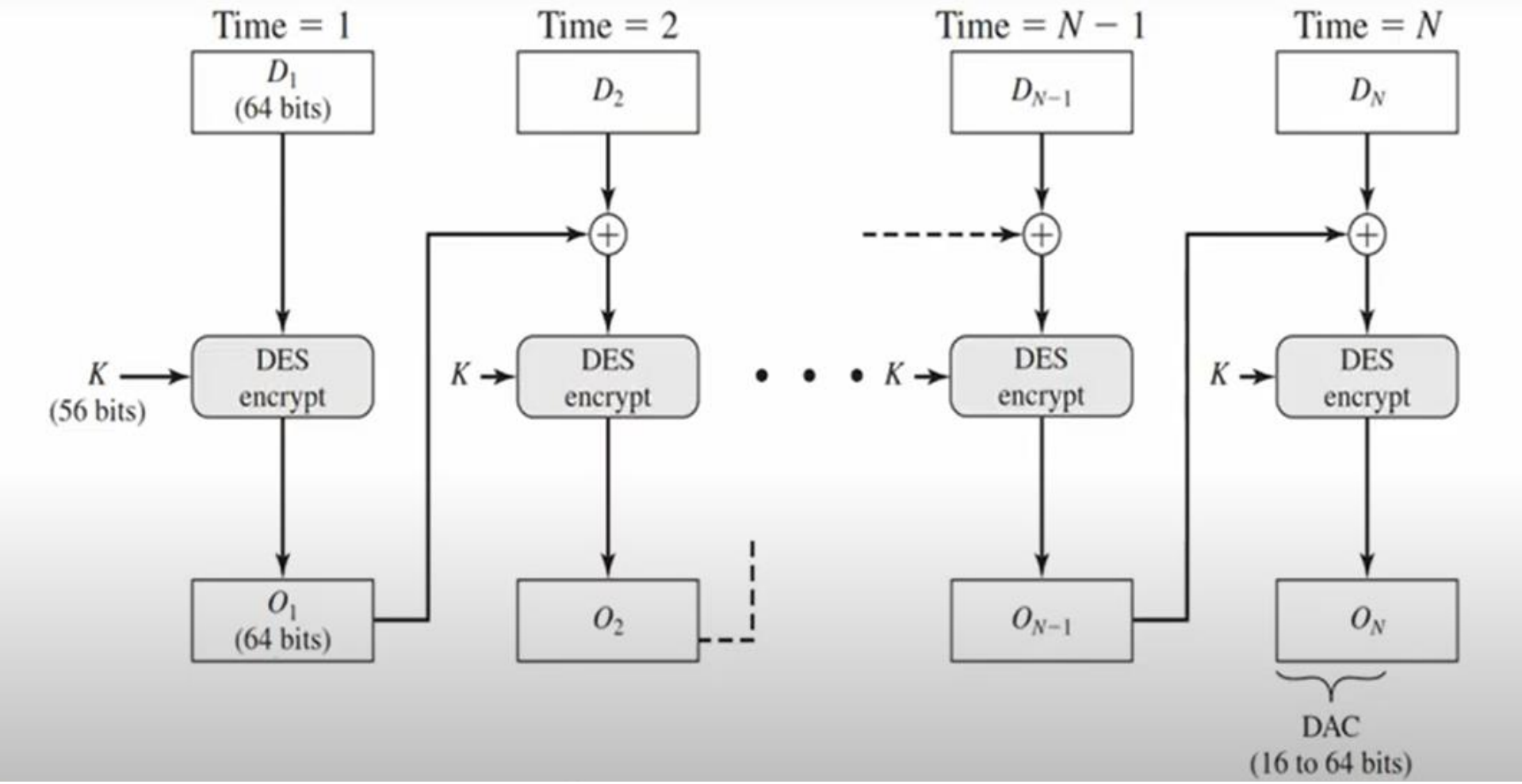
Data Authentication Algorithm(DAA)

- Data Authentication Algorithm (DAA) widely used MAC based on DES-CBC
- The message to be authenticated grouped into contiguous 64-bit blocks:

  D1, D2,…, DN.
- The final block is padded on the right with zeros to form a full 64-bit block
- Using DES encryption algorithm E and a secret key K, Data Authentication Code (DAC) is calculated.

$$
\begin{aligned}
O_1 &= \mathrm{E}(K, D) \\
O_2 &= \mathrm{E}(K, [D_2 \oplus O_1]) \\
O_3 &= \mathrm{E}(K, [D_3 \oplus O_2]) \\
&\cdot \\
&\cdot \\
&\cdot \\
O_N &= \mathrm{E}(K, [D_N \oplus O_{N-1}])
\end{aligned}
$$

Data Authentication Algorithm(DAA)



DAA

- Reference - Stallings

Understand

- Digital Signature
- E- signature
- Conventional Signature

Digital Signatures

- A **digital signature** is a mathematical scheme for verifying the authenticity of digital messages or documents. A valid digital signature, where the prerequisites are satisfied, gives a recipient very strong reason to believe that the message was created by a known sender (authentication), and that the message was not altered in transit (integrity).

Conventional –

- Traditional method of document signing (Handwritten, seal etc.)
- Physical part of document
- Verified by comparing it to authentic signatures
- Same sign on various docs – (one to many)

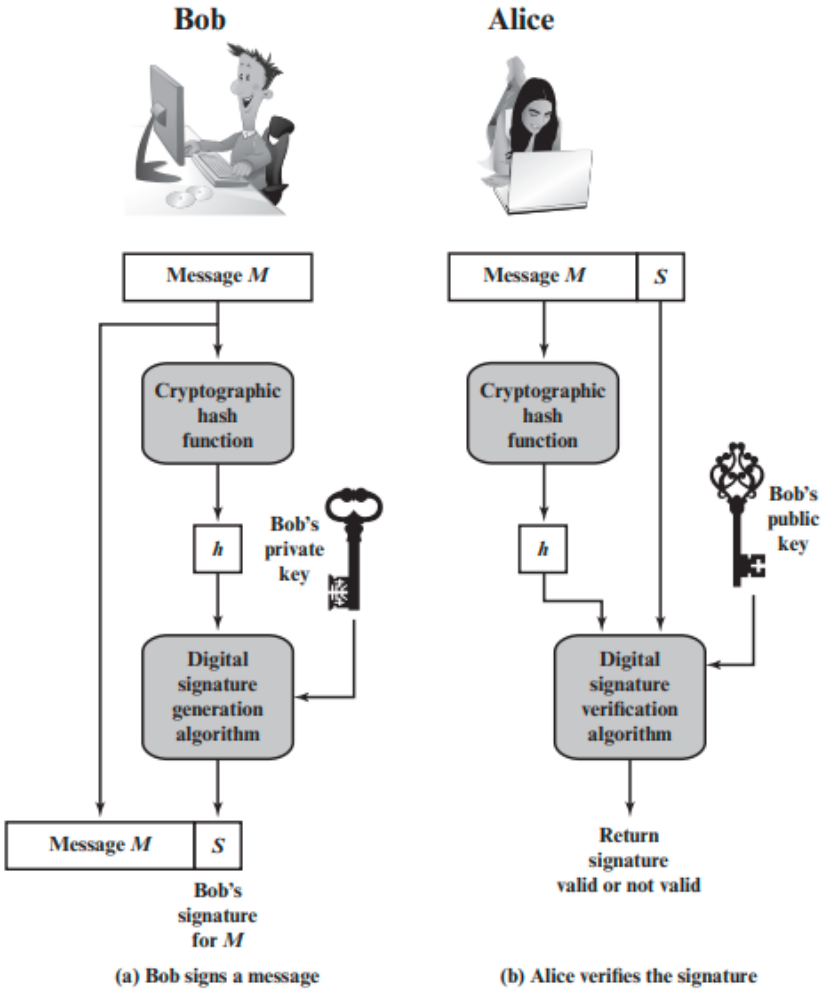
Digital –

- E-signature based on public key cryptography
- Authenticating digital documents or message
- Issued by Certificate Authority (CA)
- Schemes – RSA, Elgamal
- Verified by verification algorithm
- Different sign for different documents(one to one)

Digital vs Conventional Signature

Digital Signature Process

- Bob uses a secure hash function, such as SHA-512, to generate a hash value for the message.
- That hash value, together with Bob's private key serves as input to a digital signature generation algorithm, which produces a short block that functions as a digital signature
- Bob sends the message with the signature attached.
- When Alice receives the message plus signature, she (1) calculates a hash value for the message; (2) provides the hash value and Bob's public key as inputs to a digital signature verification algorithm.
- If the algorithm returns the result that the signature is valid, Alice is assured that the message must have been signed by Bob.



Digital Signature Properties

- Message Authentication does not protect the two parties against each other.
  - Disputes:
    - Receiver may forge the message
    - Sender denies sending the message
- The most attractive solution to this problem is the digital signature. The digital signature must have the following properties:
  - 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.

Digital Signature Services

- Authentication
  - Bob can verify the message is sent by Alice as Alice's public key is used for verification
- Integrity
  - Different signature will be produced if message is changed.
  - Hash preserves integrity
- Nonrepudiation
  - Using Trusted Party

Digital Signature Services

- Nonrepudiation (cont.)
  - Using Trusted Party

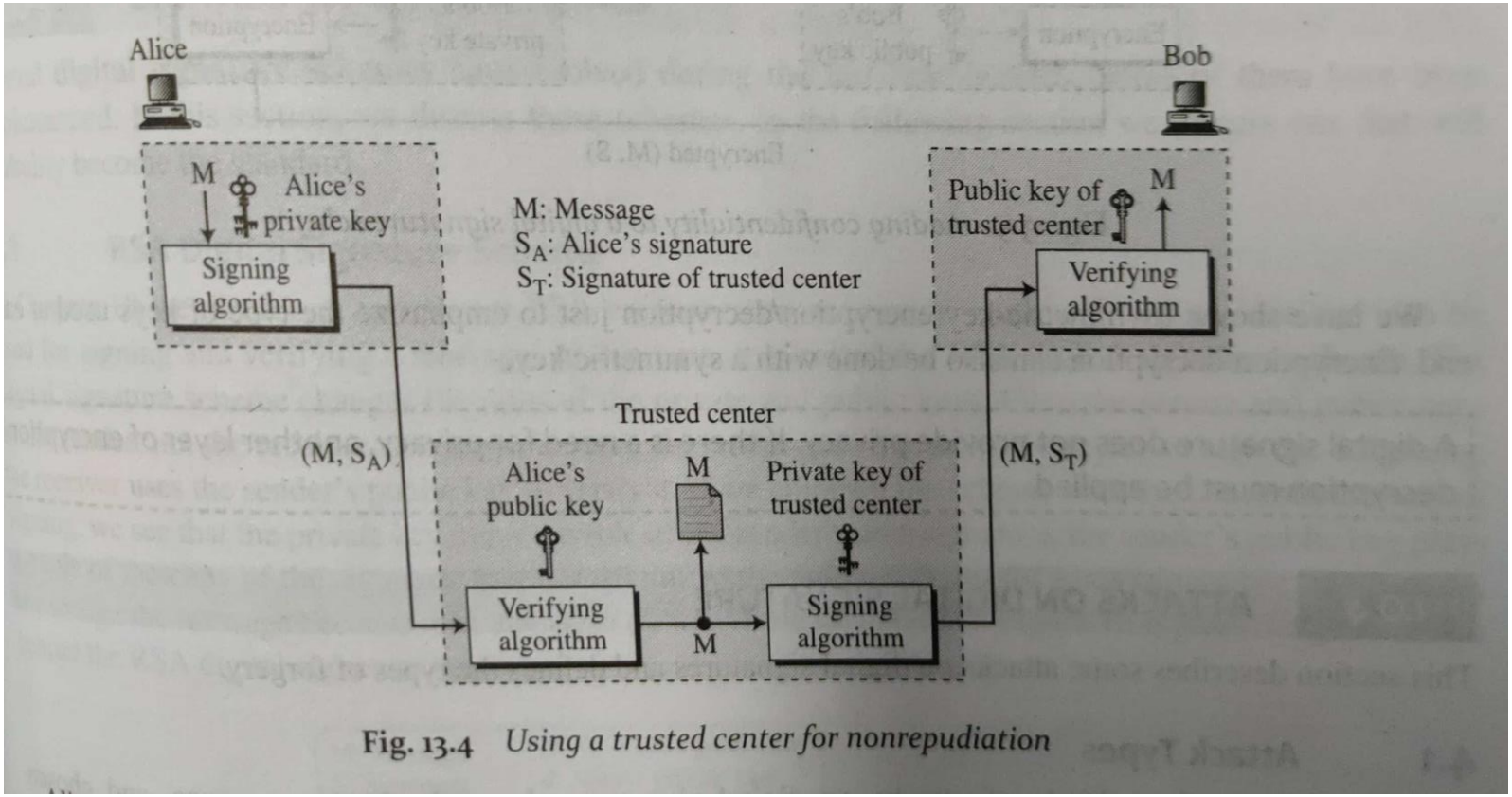


Fig. 13.4 *Using a trusted center for nonrepudiation*

Digital Signature Services

- Confidentiality
  - Not provided.
  - If required ; then message and encryption must be encrypted.

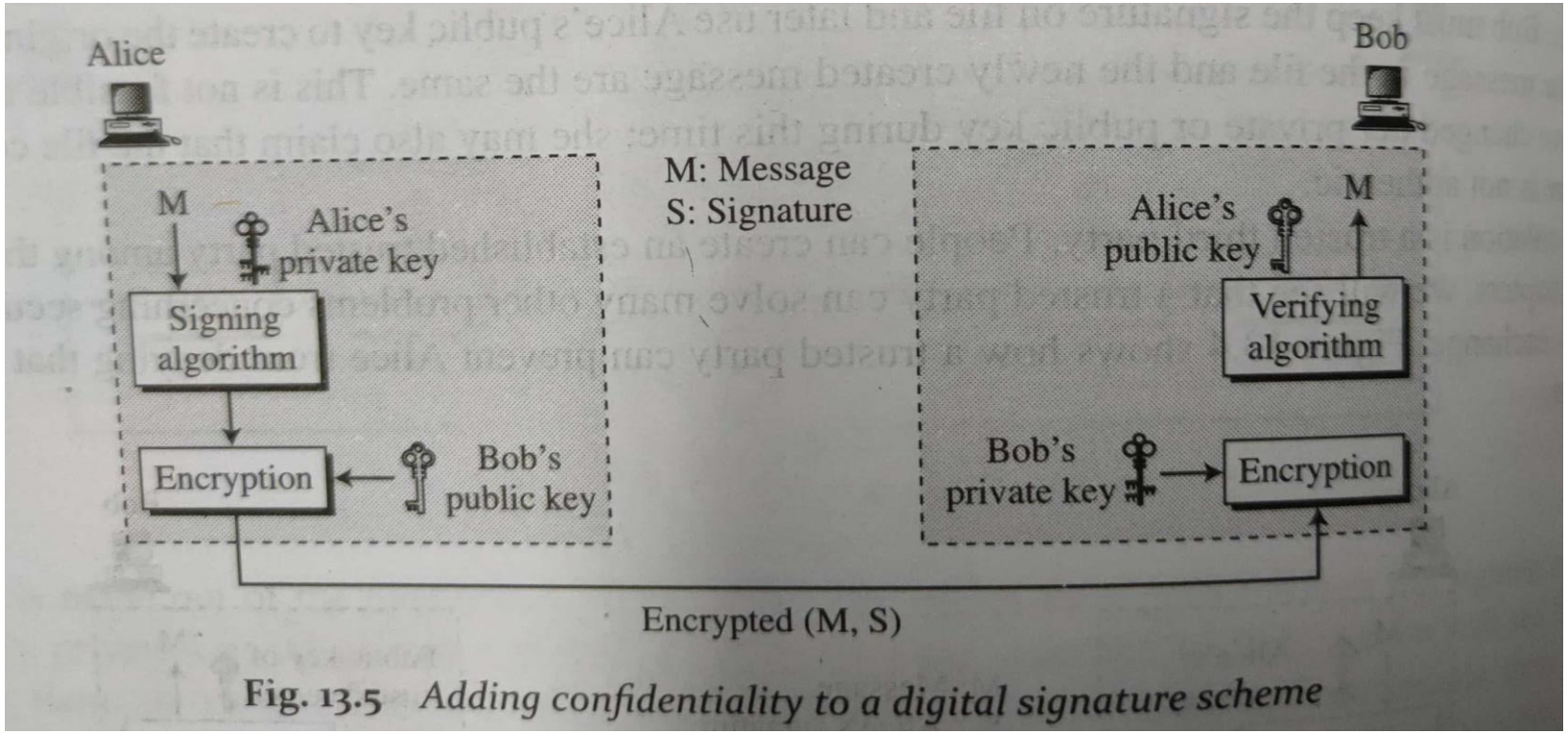


Fig. 13.5 *Adding confidentiality to a digital signature scheme*

Attacks on Digital Signature

Key-Only

Same as Ciphertext – only attack

- C – attacker, A – sender, B- receiver.
- A's public key is known to everyone.
- C recreates signature using A's public key and digitally sign the documents which A doesn't intend to do.

Known - Message

Same as Known – plaintext attack

- C knows previous message-signature pairs of A.
- C recreate signature by analyzing previous data (by using brute force)

Chosen - Message

Similar to Chosen – plaintext attack

- C makes A to sign one or more messages.
- C has message-digital pairs.

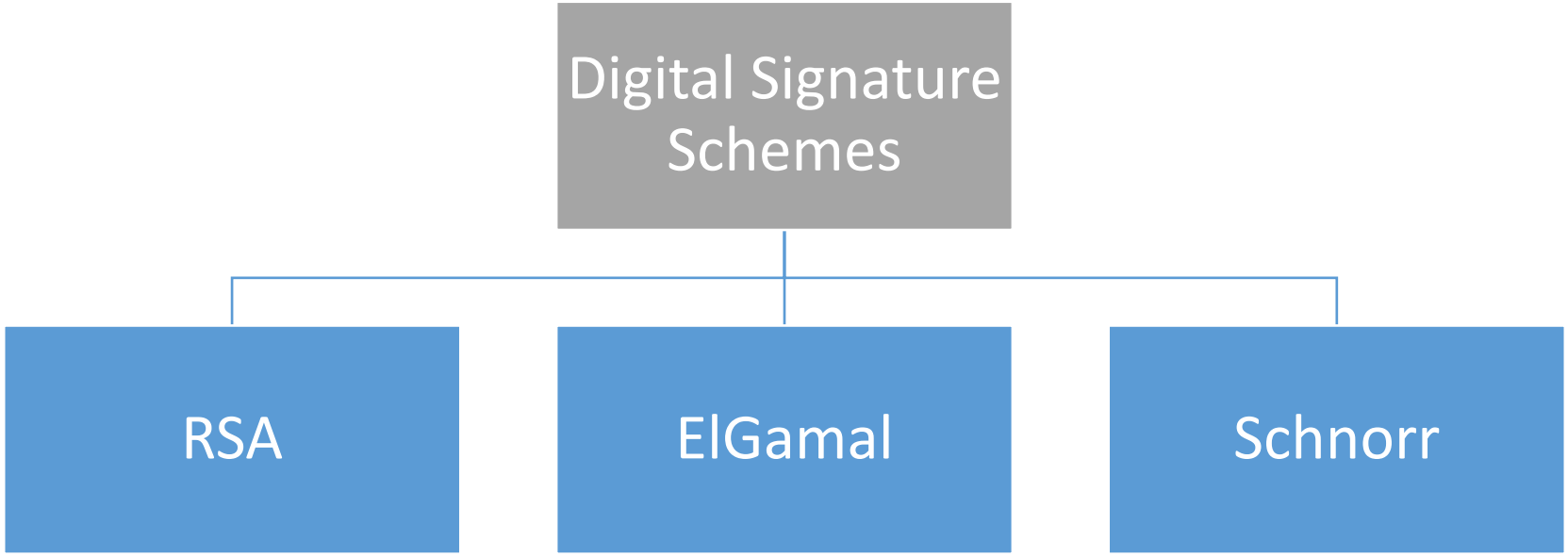
Forgery Types

Existential Forgery

- Attacker may be able to create a valid message-signature pair but not that she can really use.
- Attacker's message could be syntactically and semantically unintelligible.

Selective Forgery

- Attacker may be able to forge Sender's signature on a message with the content selectively chosen by attacker.

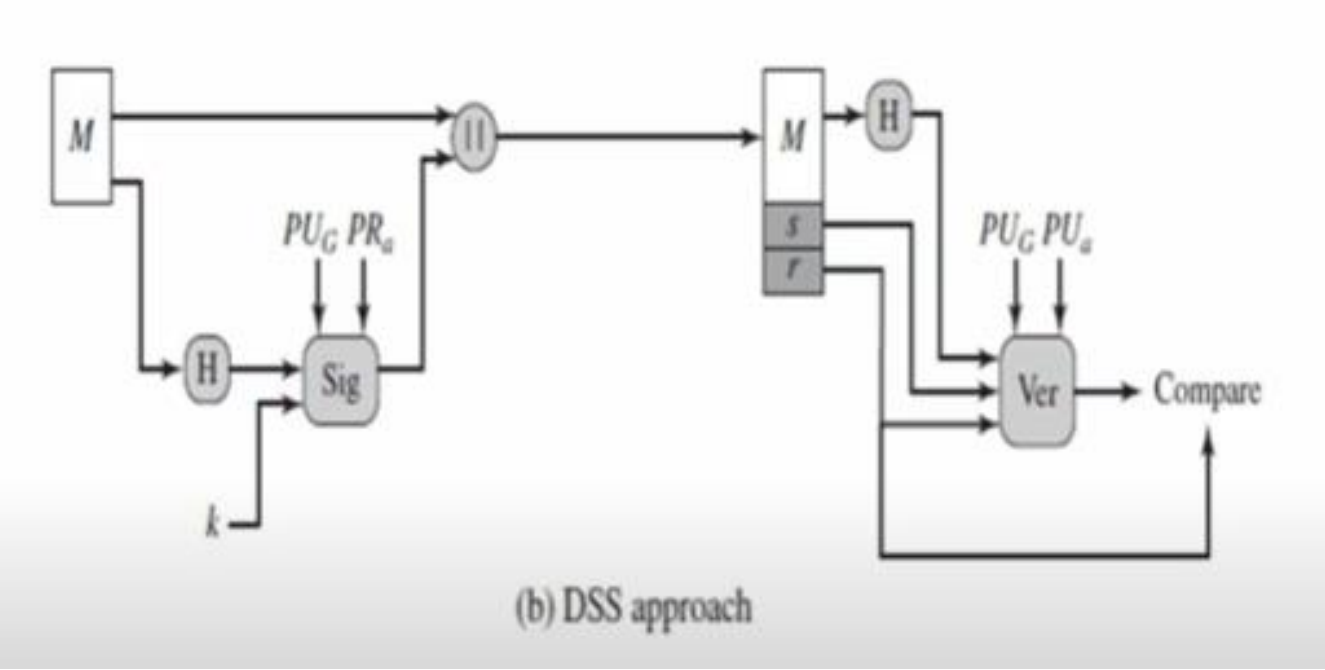

Digital Signature Schemes
RSA
ElGamal
Schnorr

Digital Signature Standard (DSS)

- NIST has published Federal Information Processing Standard FIPS 186, known as DSS.
- It makes use of the Secure Hash Algorithm (SHA)
- It was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme.

DSS Steps

- Generation of Public and Private key for User A
- Creation of Digital Signature by User A for message M
- User B verifies the Digital Signature



Generation of Global Public Key Components {p,q,g}

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 A Public Key and User A Private Key

User's Private Key

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

User's Public Key

$y = g^x \bmod p$

Generating Signature {r,s}

User's Per-Message Secret Number

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

Signing

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

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

Signature = (r, s)

Verifying Signature {r,s}

Verifying

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

$$u_1 = [H(M')w] \bmod q$$

$$u_2 = (r')w \bmod q$$

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

TEST: $v = r'$

M = message to be signed
$H(M)$ = hash of M using SHA-1
M', r', s' = received versions of M, r, s

Authentication Applications

Verifying User's Identity:

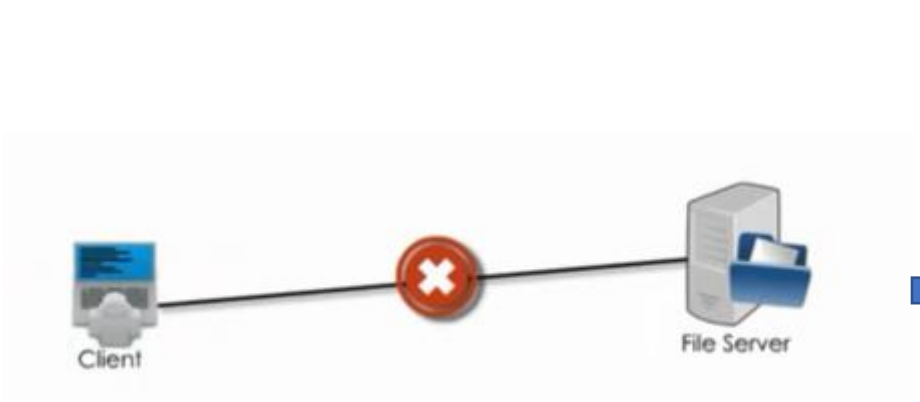
- Kerberos,
- X.509 Authentication Service

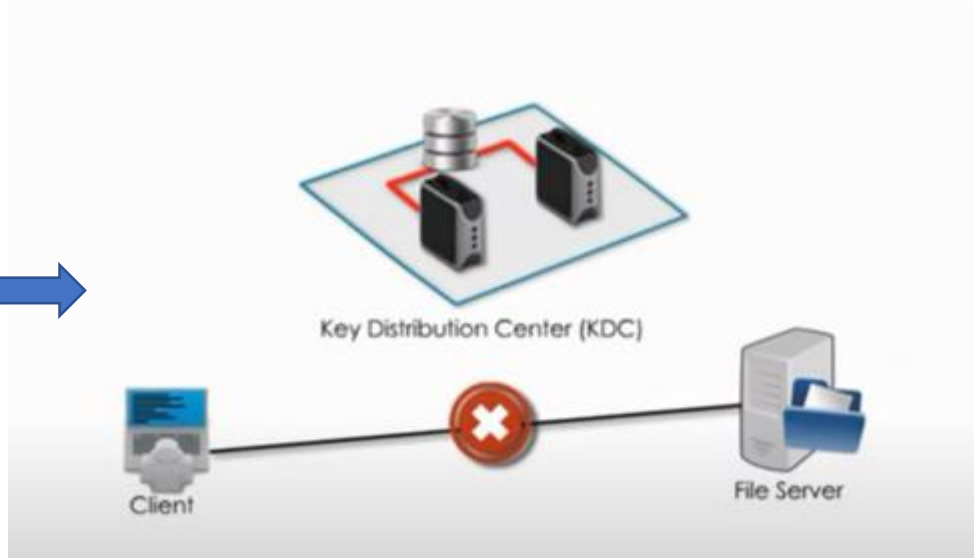
Kerberos

- In mythology, Kerberos (also known as Cerberus) is a large, three-headed dog that guards the gates to the underworld to keep souls from escaping.
- Kerberos is the computer network authentication protocol initially developed in the 1980s by Massachusetts Institute of Technology (MIT) computer scientists.
- The idea behind Kerberos is to authenticate users while preventing passwords from being sent over the internet.
- It uses secret-key cryptography and a trusted third party for authenticating client-server applications and verifying users' identities.
- But in the protocol's case, the three heads of Kerberos represent the client, the server, and the Key Distribution Center (KDC).

Kerberos Steps

- Client wants to access file on a server and with third party client must be verified through trusted –third party





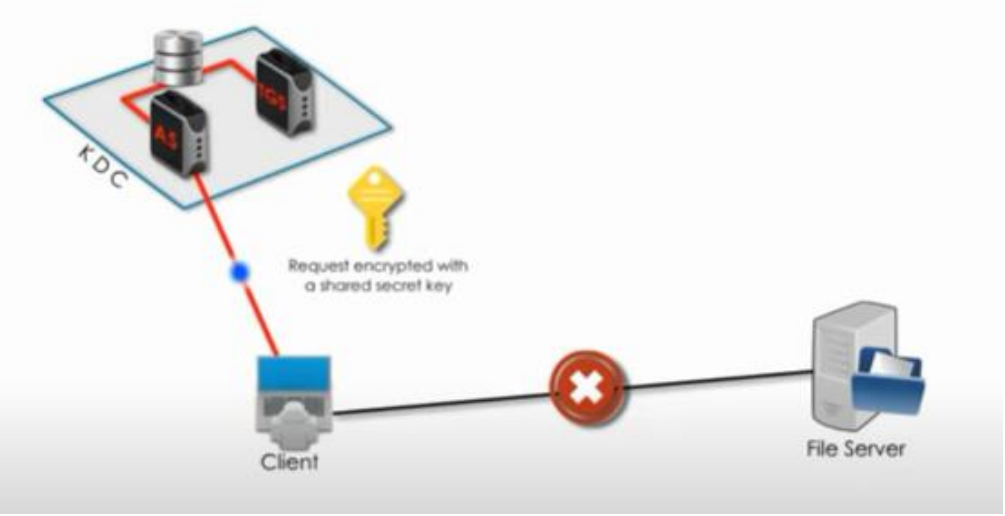
Key Distribution Center (KDC)

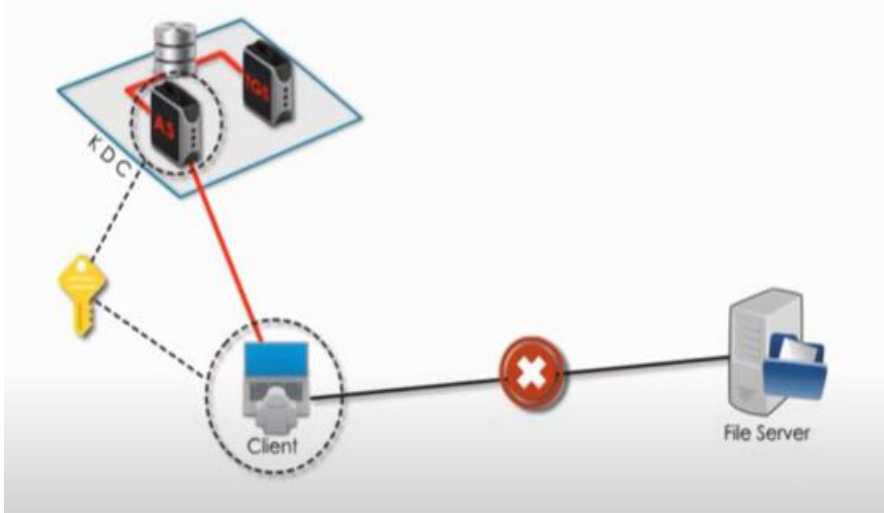
- Authentication Server(AS)
- Ticket Granting Server(TGS)

Kerberos Steps

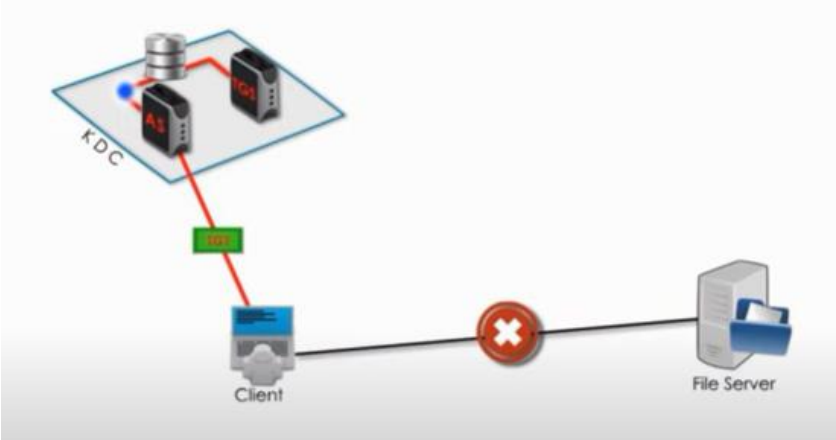
Step 1: Login.

- The user asks for a Ticket Granting Ticket (TGT) from the authentication server (AS).
- This request includes the client ID. And client's password is a shared secret key.





Kerberos Steps



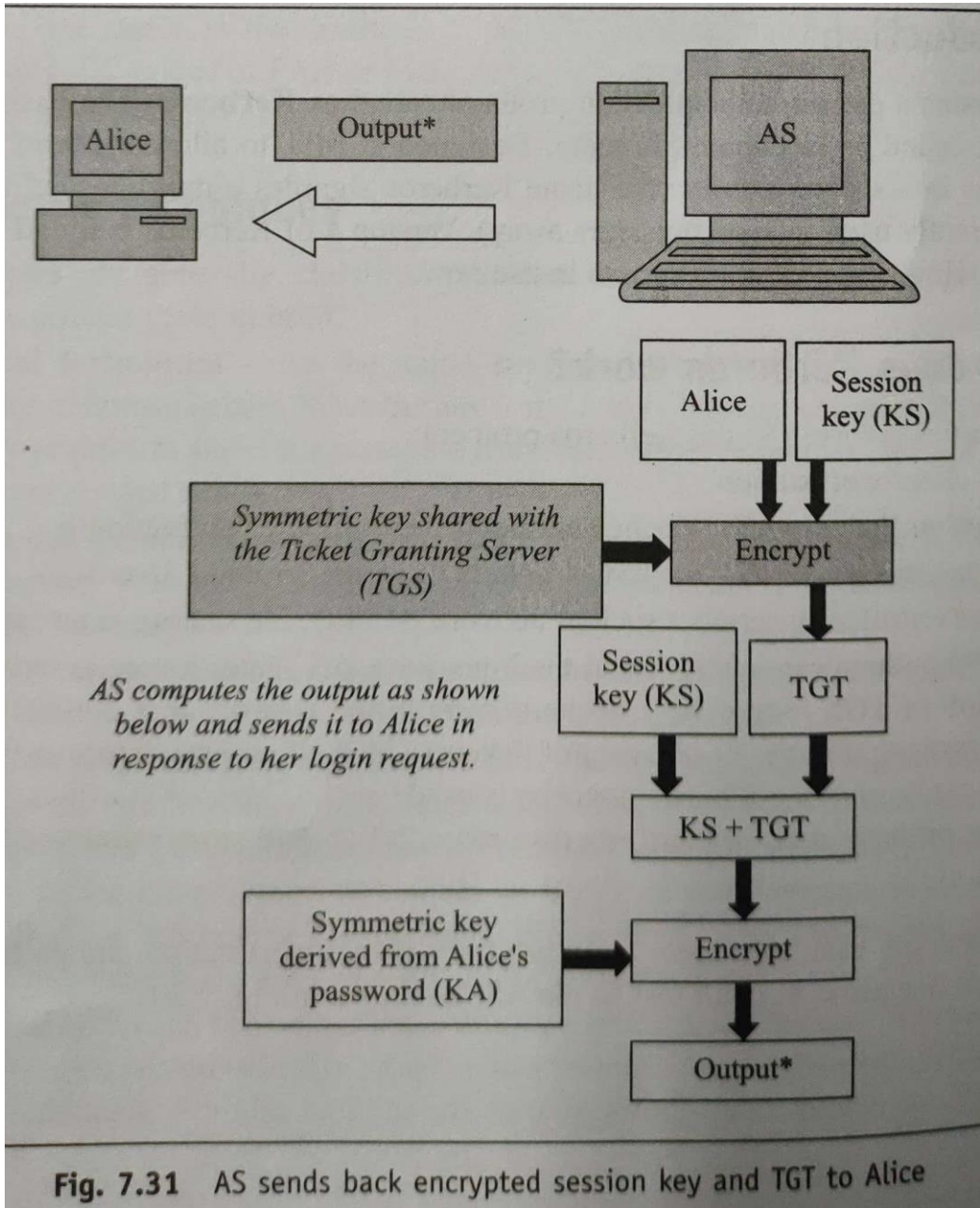


Fig. 7.31 AS sends back encrypted session key and TGT to Alice

Kerberos Steps

Step 2: Obtaining a Service Granting Ticket (SGT)

i. Request

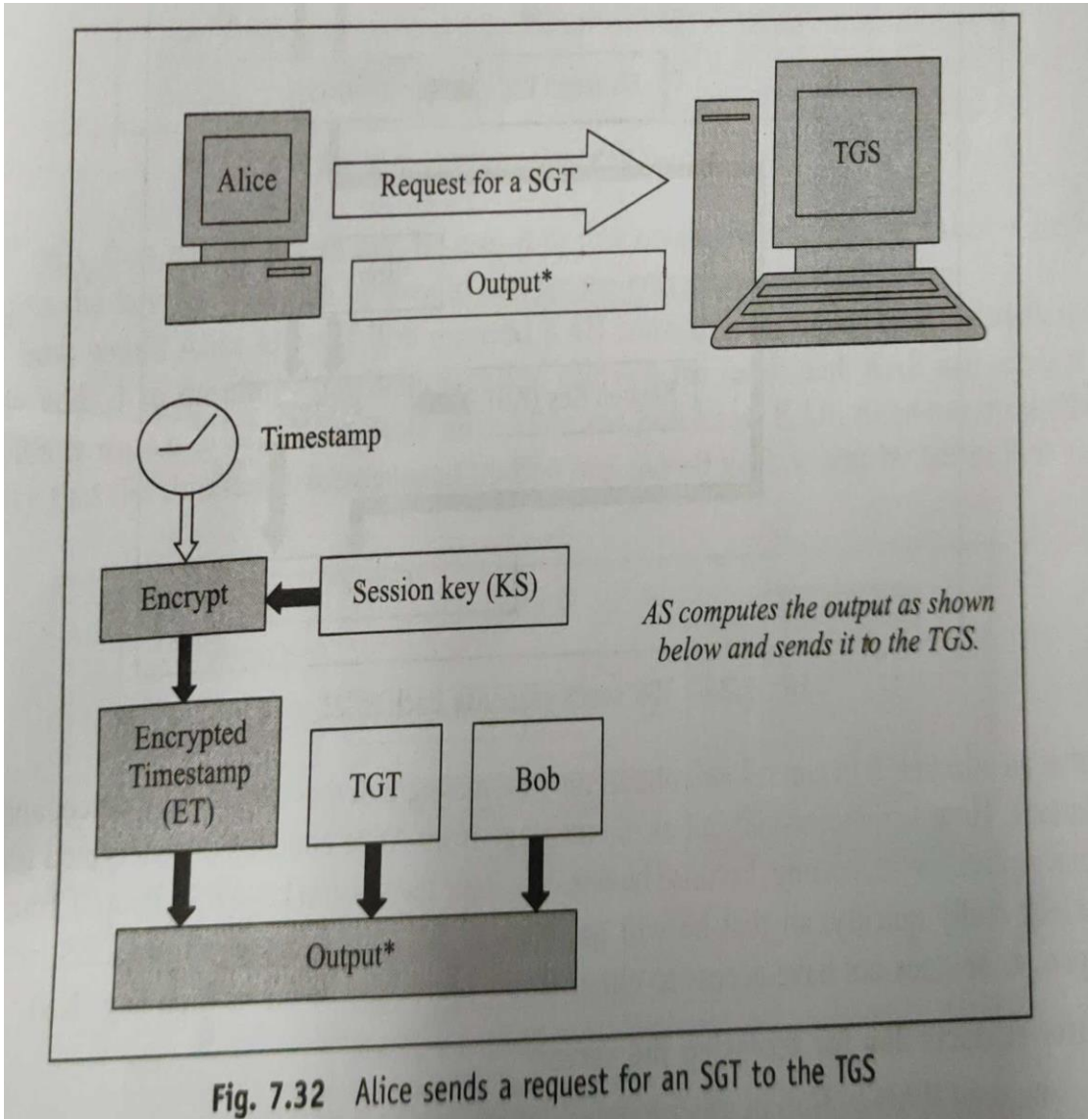


Fig. 7.32 Alice sends a request for an SGT to the TGS

Kerberos Steps

Step 2: Obtaining a Service Granting Ticket (SGT)

ii. Response from TGS

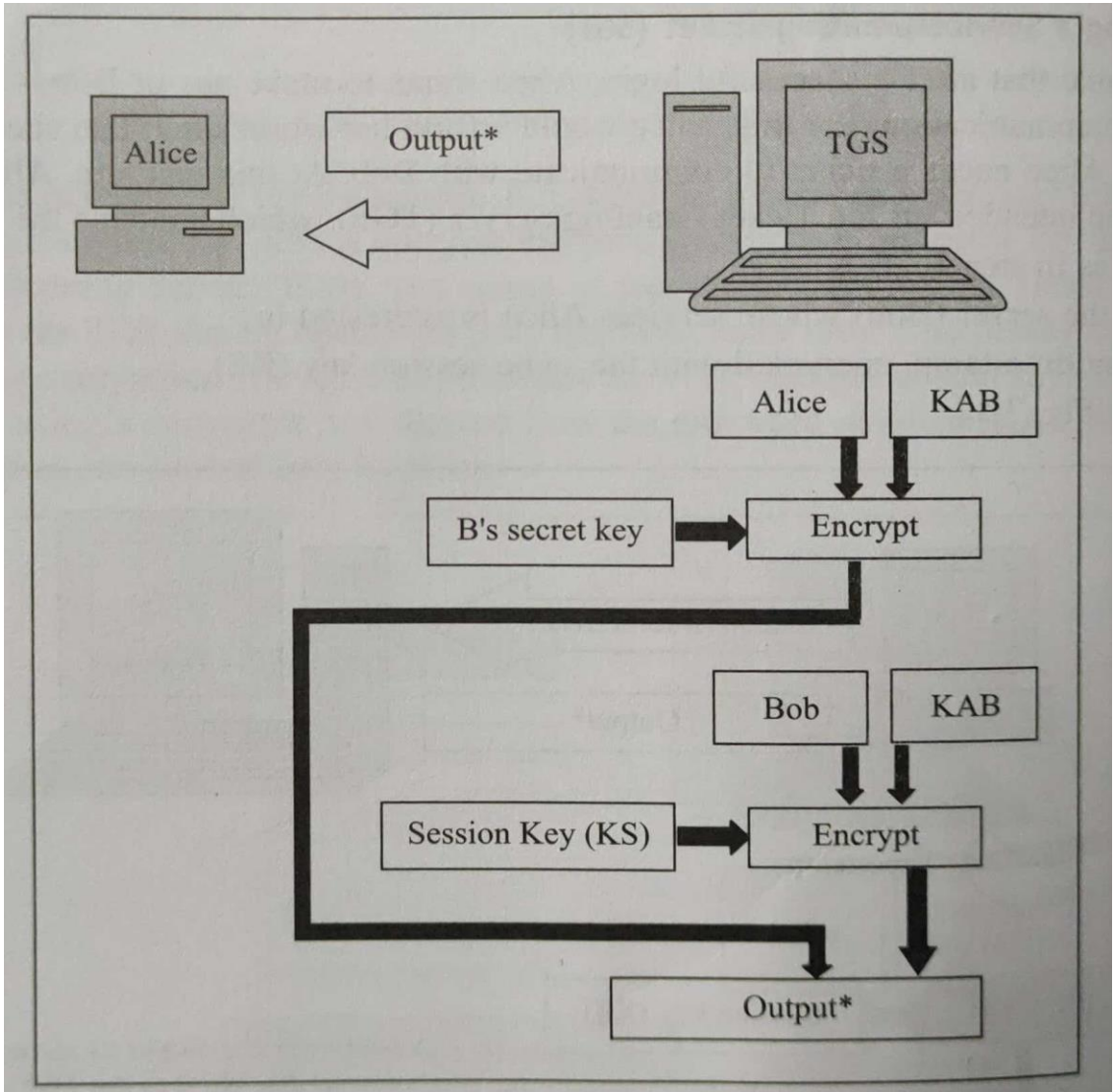


Fig. 7.33 TGS sends response back to Alice

Kerberos Steps

Step 3: User contacts Bob for accessing the server

i. Alice sends KAB to Bob

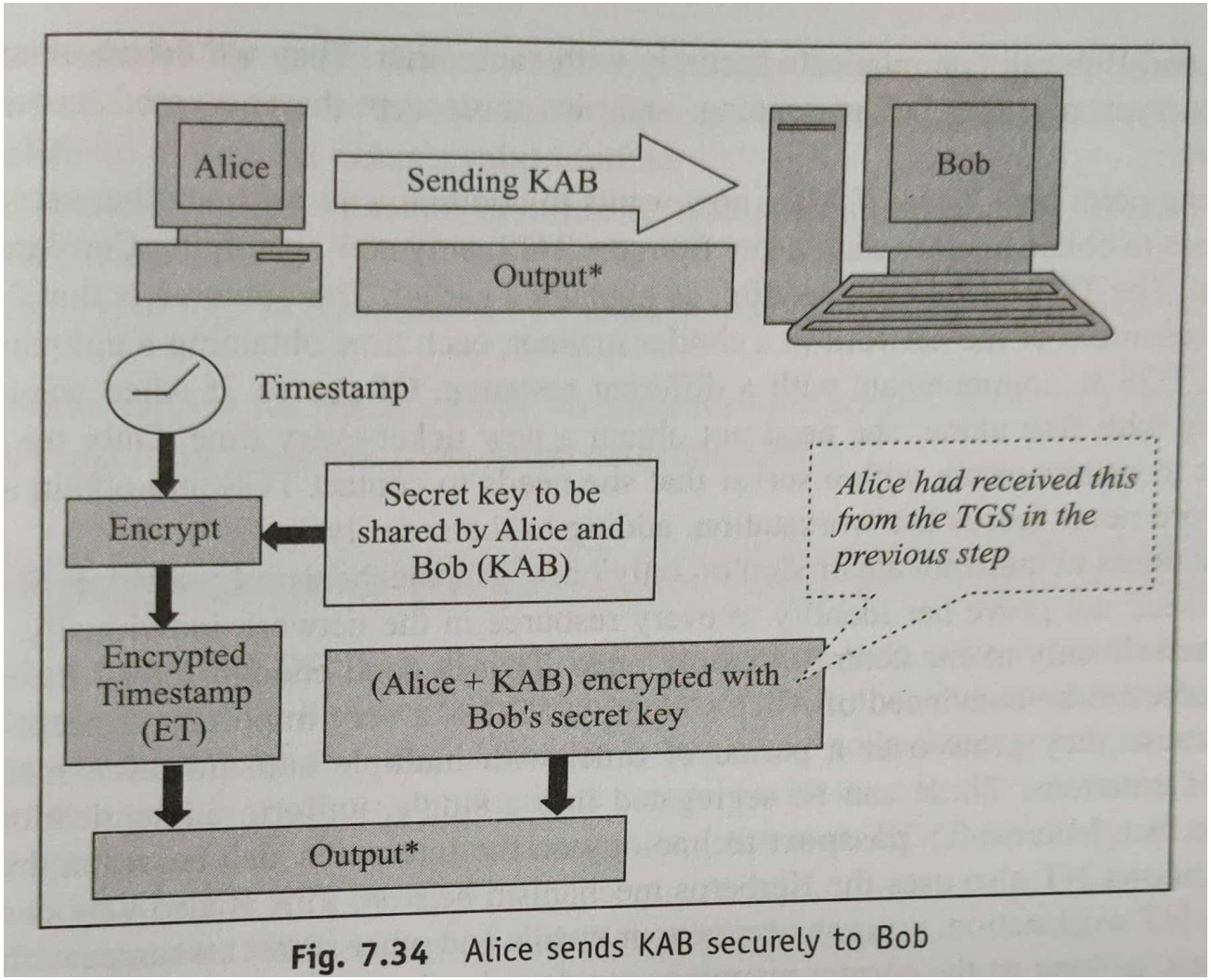


Fig. 7.34 Alice sends KAB securely to Bob

Kerberos Steps

Step 3: User contacts Bob for accessing the server

ii. Acknowledgement from Bob

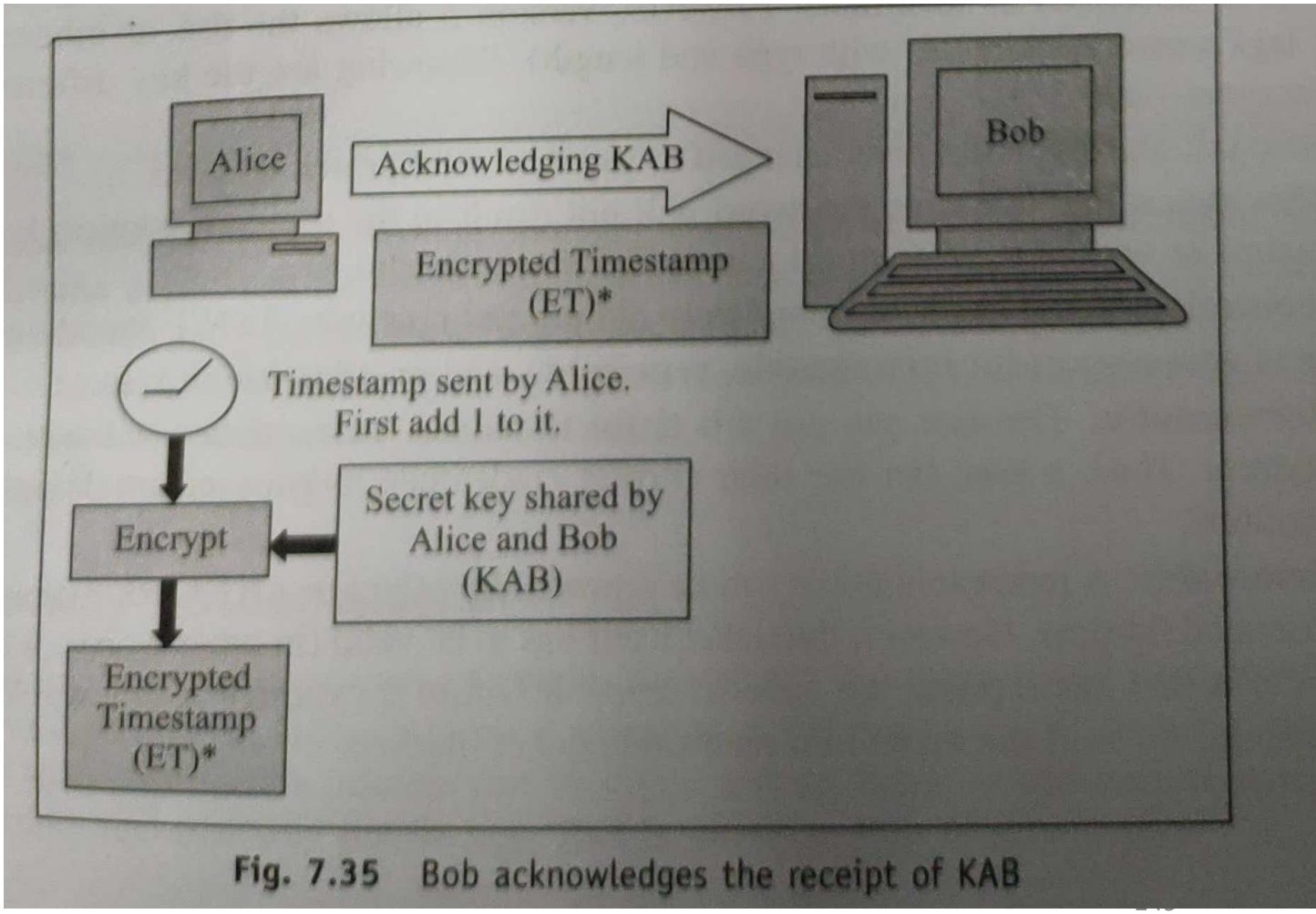


Fig. 7.35 Bob acknowledges the receipt of KAB

Enable Kerberos support in browsers

- http://woshub.com/enable-kerberos-authentication-in-browser/

Kerberos Reference

- Cryptography and Network Security by Atul Kahate

Kerberos 4 vs 5

- Home laptop

Digital Certificates

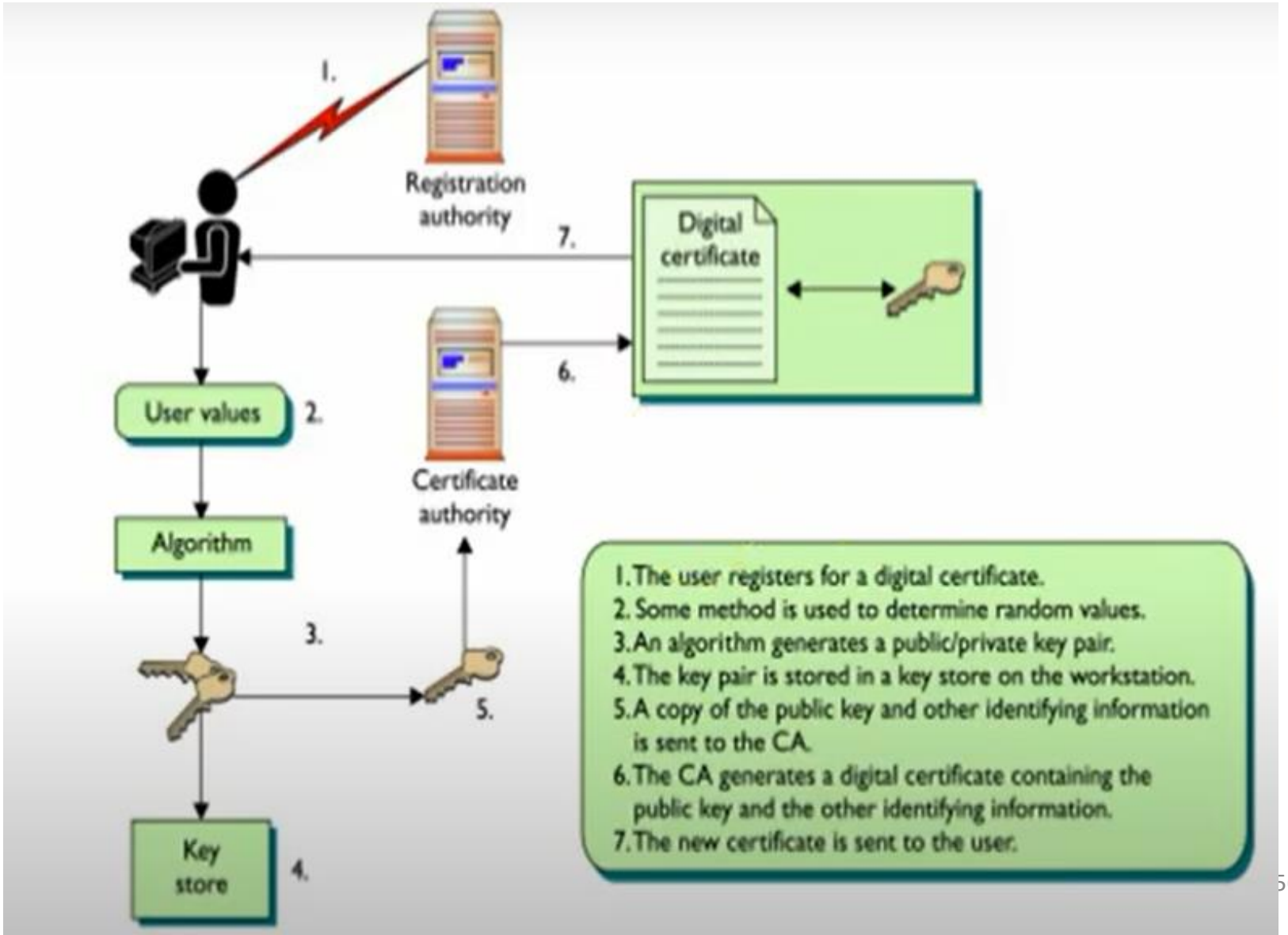
Digital Certificate

- In cryptography, a **public key certificate**, also known as a **digital certificate** or **identity certificate**, is an electronic document used to prove the ownership of a public key.
- The certificate includes
  - information about the key,
  - information about the identity of its owner (called the subject), and
  - The digital signature of an entity that has verified the certificate's contents (called the issuer).

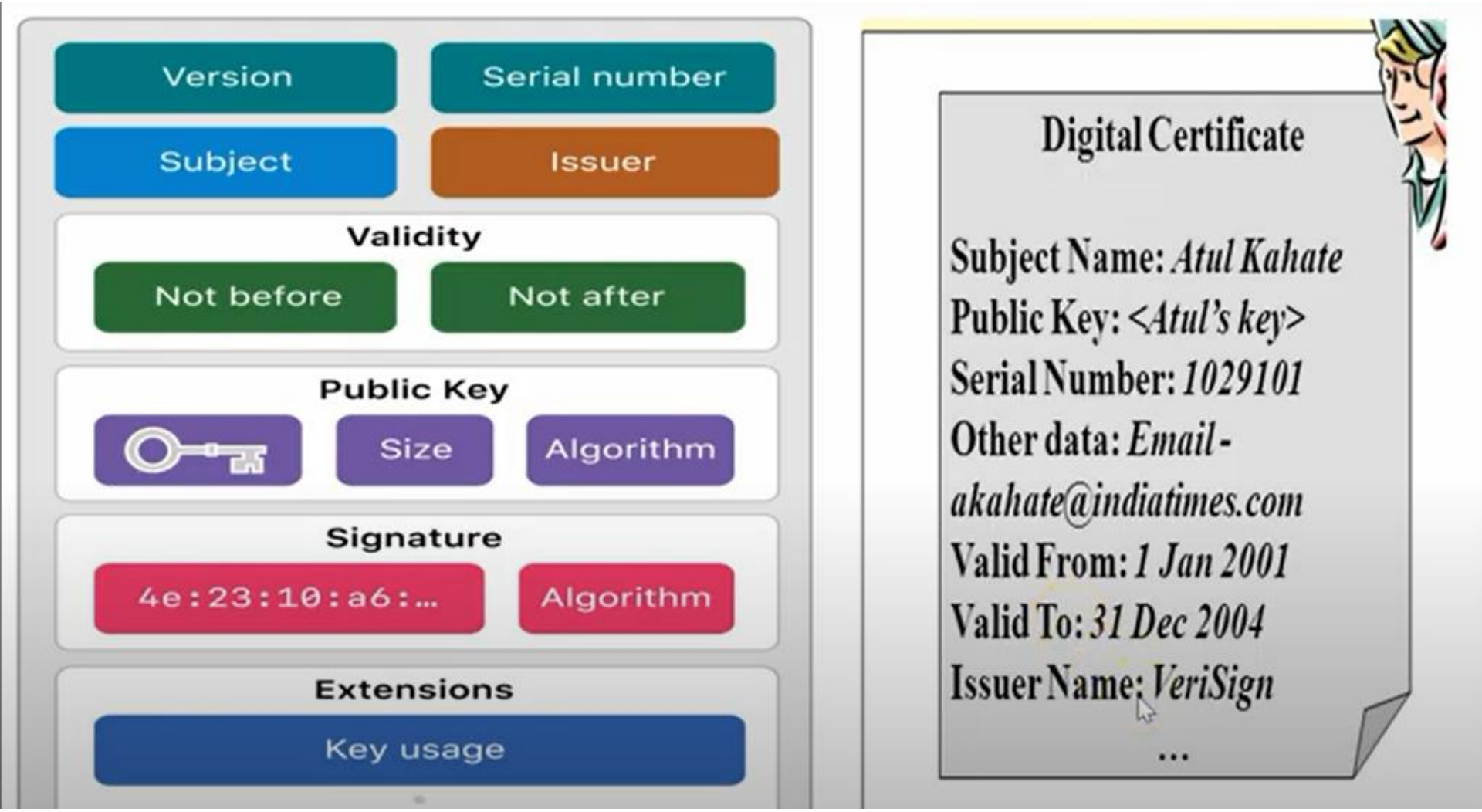
Digital Certificate (cont.)

- In a typical public-key infrastructure (PKI) scheme, the certificate issuer is a certificate authority (CA), usually a company that charges customers to issue certificates for them.
- The most common format for public key certificates is defined by X.509 defined in RFC 5280.

Steps for obtaining Digital Certificate



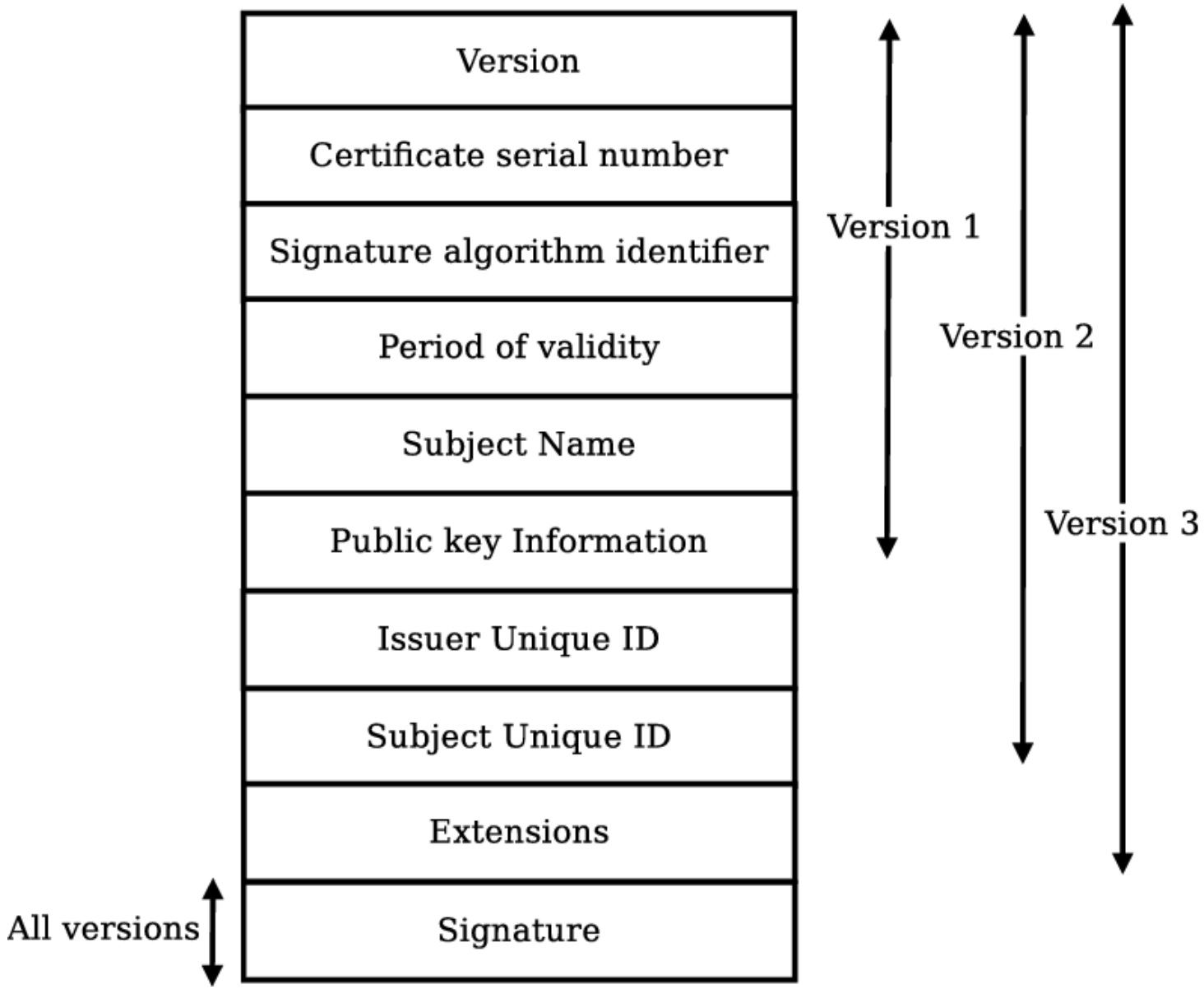
What is inside a Digital Certificate?



X.509 Certificates

- Defines the structure of a digital certificate.
- The International Telecommunication Union (ITU) released this standard 1988. It was a part of X.500.
- Since then, X.509 was revised twice. And, the current version is Version 3 – X.509V3.
- IETF published the RFC2459 for X.509 in 1999.

X.509 Certificates (cont.)



X.509 Certificates Contents

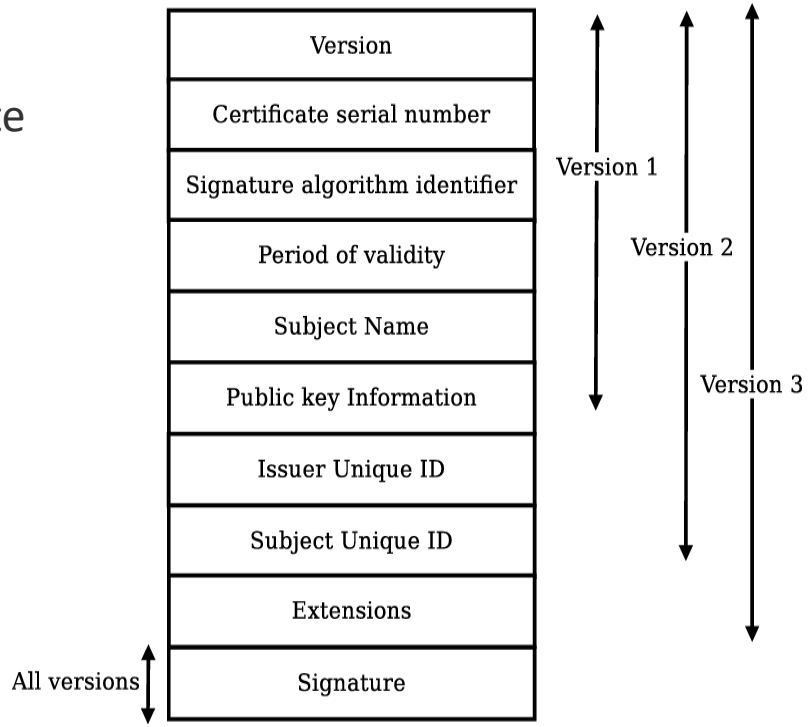
Version: which X.509 version applies to the certificate (which indicates what data the certificate must include)

Serial number: the identity creating the certificate must assign it a serial number that distinguishes it from other certificates

Algorithm information: the algorithm used by the issuer to sign the certificate

Issuer distinguished name: the name of the entity issuing the certificate (usually a certificate authority)

Validity period of the certificate: the period of time for which the certificate is valid with the start/end date.

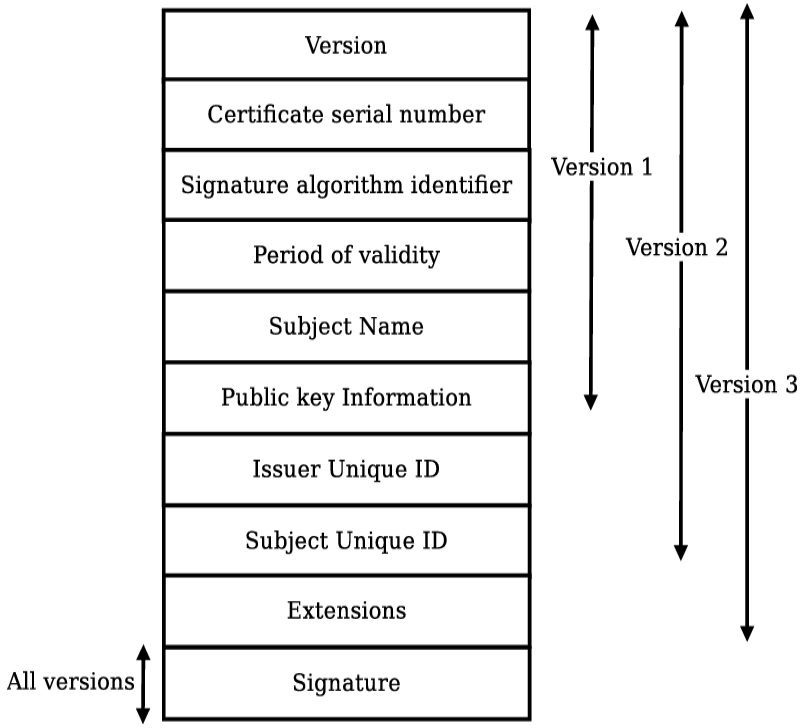


X.509 Certificates Contents

Subject distinguished name: the name of the identity the certificate is issued to

Subject public key information the public key associated with the identity

Extensions (optional)



X.509 – Version 3

Field	Description
Authority Key Identifier	Identifies the certification authority (CA) public key that corresponds to the CA private key used to sign the certificate.
Subject Key Identifier	Differentiates between multiple public keys held by the certificate subject. The extension value is typically a SHA-1 hash of the key.
Key Usage	Specifies restrictions on the operations that can be performed by the public key contained in the certificate.
Enhanced Key Usage	Specifies the manner in which the public key contained in the certificate can be used.
Private Key Usage Period	Specifies a different validity period for the private key than for the certificate with which the private key is associated.

X.509 – Version 3 (cont.)

Field	Description
Certificate Policies	Specifies the policies under which the certificate has been issued and the purposes for which it can be used.
Policy Mappings	Specifies the policies in a subordinate CA that correspond to policies in the issuing CA.
Subject Alternative Name	Specifies one or more alternative name forms for the subject of the certificate request. Example alternative forms include email addresses, DNS names, IP addresses, and URIs.
Issuer Alternative Name	Specifies one or more alternative name forms for the issuer of the certificate request.

X.509 – Version 3 (cont.)

Field	Description
Subject Directory Attributes	Conveys identification attributes such as the nationality of the certificate subject.
Basic Constraints	Specifies whether the entity can be used as a CA and, if so, the number of subordinate CAs that can exist beneath it in the certificate chain.
Name Constraints	Specifies the namespace within which all subject names in a certificate hierarchy must be located. The extension is used only in a CA certificate.

X.509 – Version 3 (cont.)

Field	Description
Name Constraints	Specifies the namespace within which all subject names in a certificate hierarchy must be located. The extension is used only in a CA certificate.

References:
https://docs.microsoft.com/en-us/windows/win32/seccertenroll/about-version-3-extensions

Book:
Cryptography and Network Security by Atul Kahate

References

- Books
  - William Stalling
  - Fourozan
  - Atul kahate