Module 5

Data Integrity

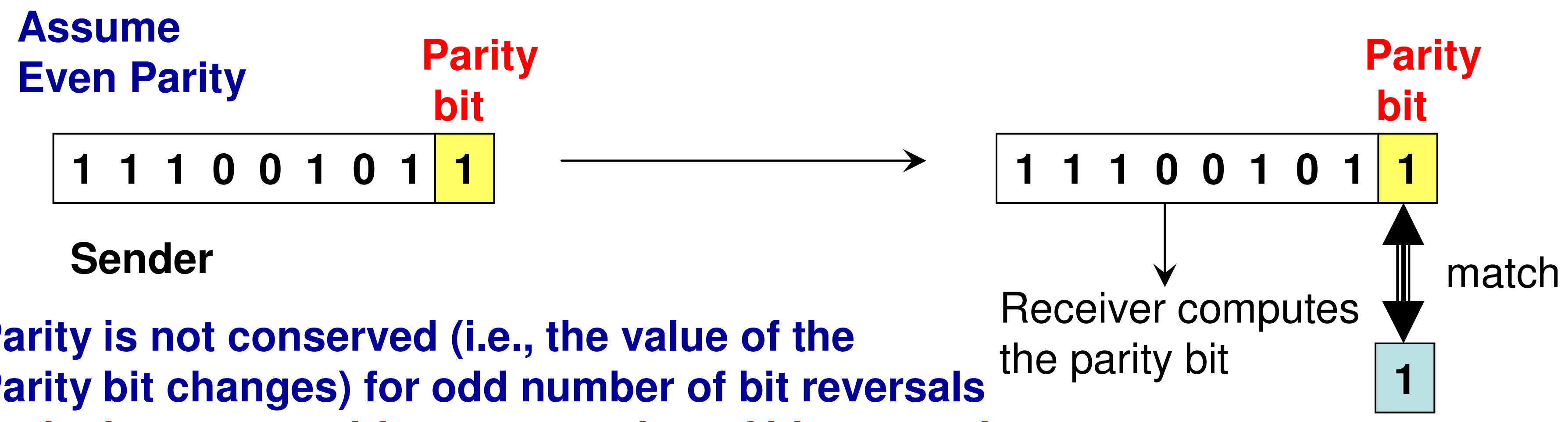
Dr. Natarajan Meghanathan
Professor of Computer Science
Jackson State University, MS
E-mail: natarajan.meghanathan@jsums.edu

Hash Functions

- A hash function H accepts a variable-length block of data M (called preimage or simply, message) as input and produces a fixed-length hash value: H(M) = h
- If even one bit of the preimage changes, it is desirable to get a different hash value (data integrity).
  - Example: Cyclic Redundancy Check, Checksum
- Cryptographic Hash Function
  - A hashing algorithm that satisfies the following two properties:
    - **One-way property: Given a hash value h, it should be computationally infeasible to find a preimage M such that H(M) = h**
    - **Collision-free property: Given a message M1 and its hash value H(M1) = h, it should be computationally infeasible to find a different message M2 that has the same hash value. i.e., H(M1) = H(M2).**
- Hash values are typically used to test for data integrity (i.e., whether the message got altered during transmission).
- Message Authentication: When a hash value is also used to assure the purported identity of the sender, it is called message authentication.

Simple Examples of Hash Functions

- **Parity-bit:** Size of the hash value is 1.
  - The sender and receiver agree on using a particular parity (odd parity or even parity)
    - Odd parity – the # of 1s in the message is odd
    - Even parity – the # of 1s in the message is even
  - Sender appends the parity bit value for a message; the receiver computes the parity bit value for the received message, and accepts the message if the parity bit values match.



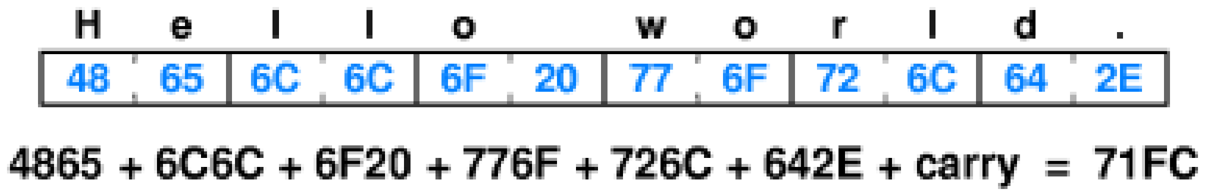
Parity is not conserved (i.e., the value of the Parity bit changes) for odd number of bit reversals

Parity is conserved for even number of bit reversals

There is a 50% chance that the parity bit value for two different messages is the same. The parity bit scheme cannot be a cryptographic hash function.

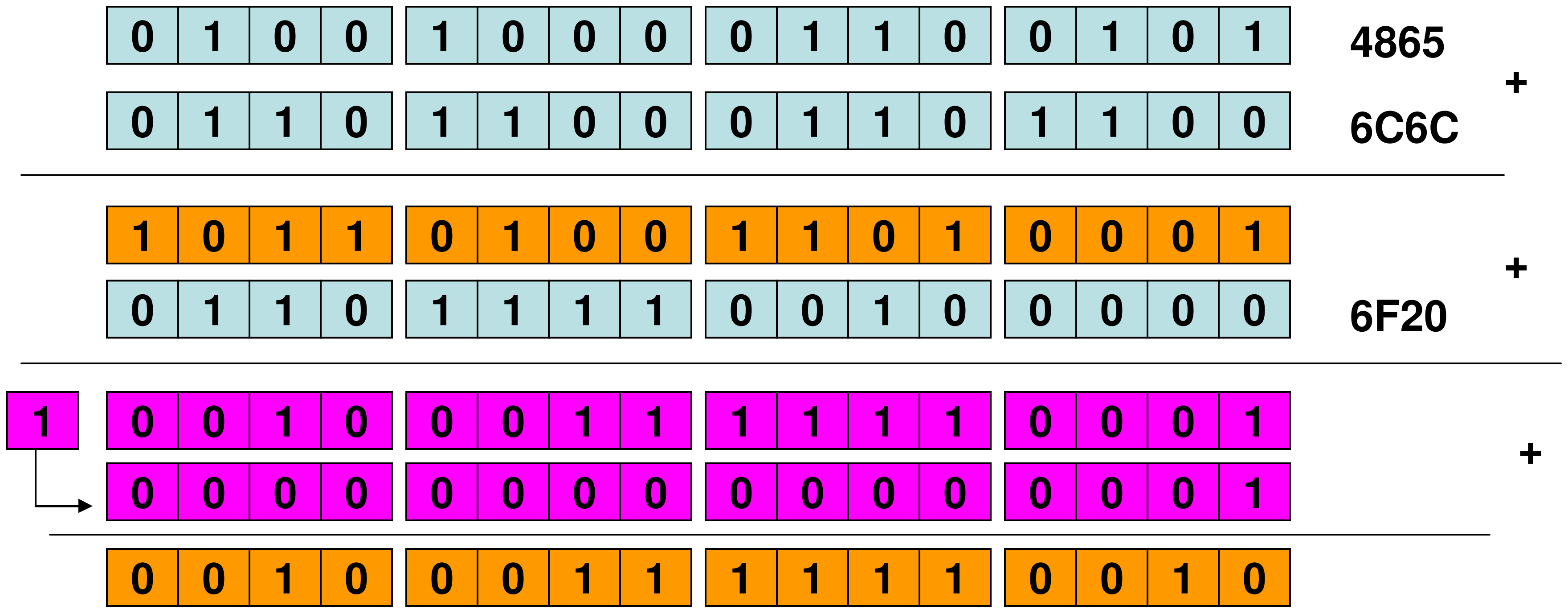
Simple Examples of Hash Functions: Checksum

- Characters are grouped into 16-bit quantities and added; the carry bits, if generated, are added to the result.



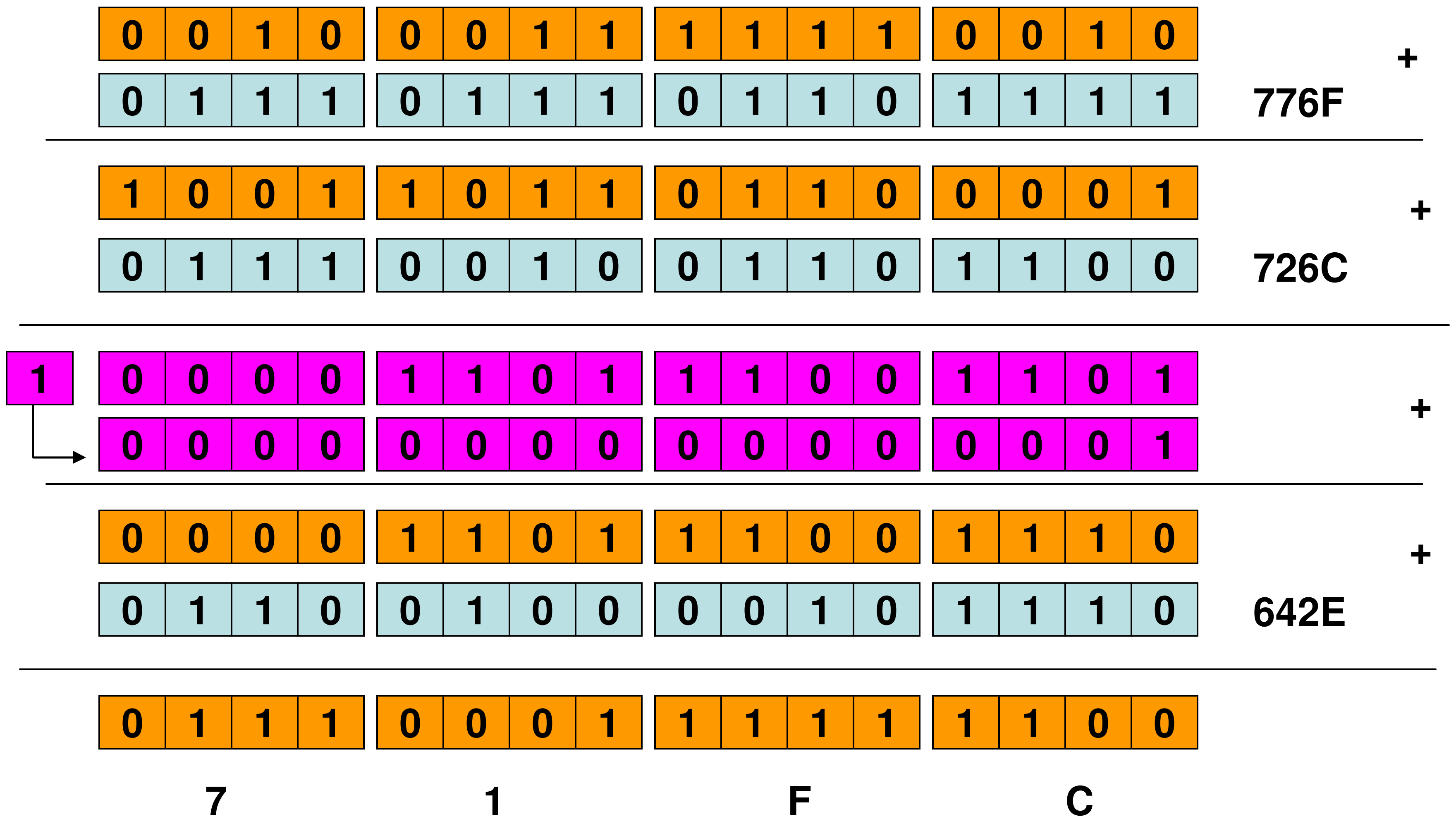
H	e	l	l	o		w	o	r	l	d	.
48	65	6C	6C	6F	20	77	6F	72	6C	64	2E

4865 + 6C6C + 6F20 + 776F + 726C + 642E + carry = 71FC



	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	1	4865
	0	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0	+ 6C6C
	1	0	1	1	0	1	0	0	1	1	0	1	0	0	0	1	
	0	1	1	0	1	1	1	1	0	0	1	0	0	0	0	0	+ 6F20
1	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	1	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	+
	0	0	1	0	0	0	1	1	1	1	1	1	0	0	1	0	

Checksum


0 0 1 0 0 0 1 1 1 1 1 1 0 0 1 0
+
0 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1
776F
1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 1
+
0 1 1 1 0 0 1 0 0 1 1 0 1 1 0 0
726C
1
0 0 0 0 1 1 0 1 1 1 0 0 1 1 0 1
+
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 1 1 0 1 1 1 0 0 1 1 1 0
+
0 1 1 0 0 1 0 0 0 0 1 0 1 1 1 0
642E
0 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0
7
1
F
C

On Hash Size

- Assume the hash value h generated for a message M is purely random.
- If the hash size is *n*-bits long, there are 2^n unique possible hash values.
  - For a particular message, the probability that the hash value is one among the 2^n values is $1/2^n$.
  - For two different messages M1 and M2, the probability that they both have the same has value is $2^n / 2^{2n} = 1/2^n$.

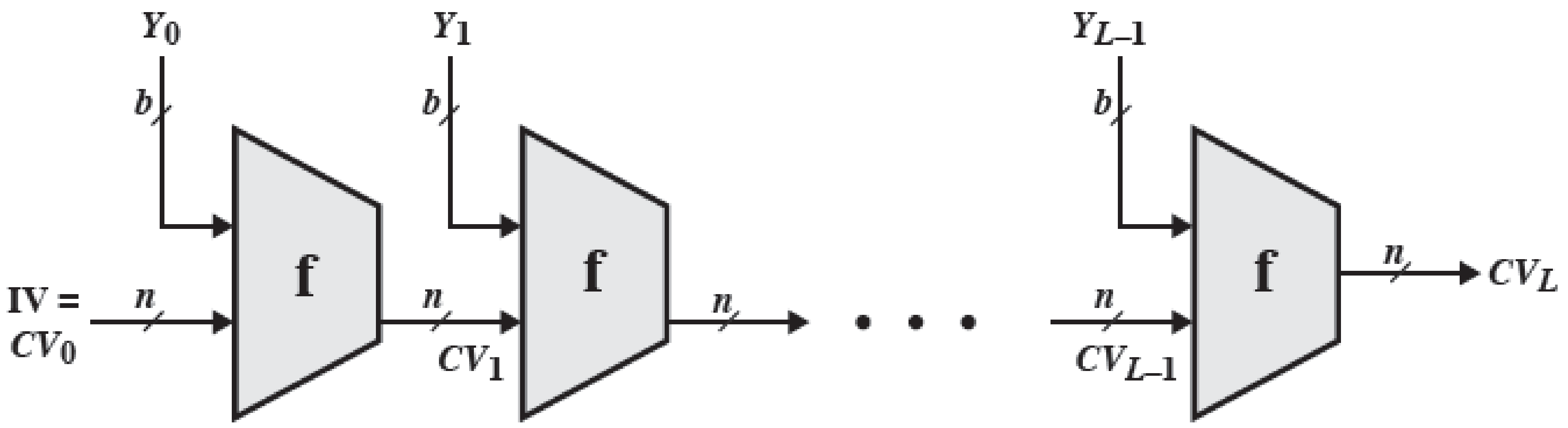
Possible Hash Values for M1 h1	Possible Hash Values for M2 h2
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 0
1 1 1	1 1 1

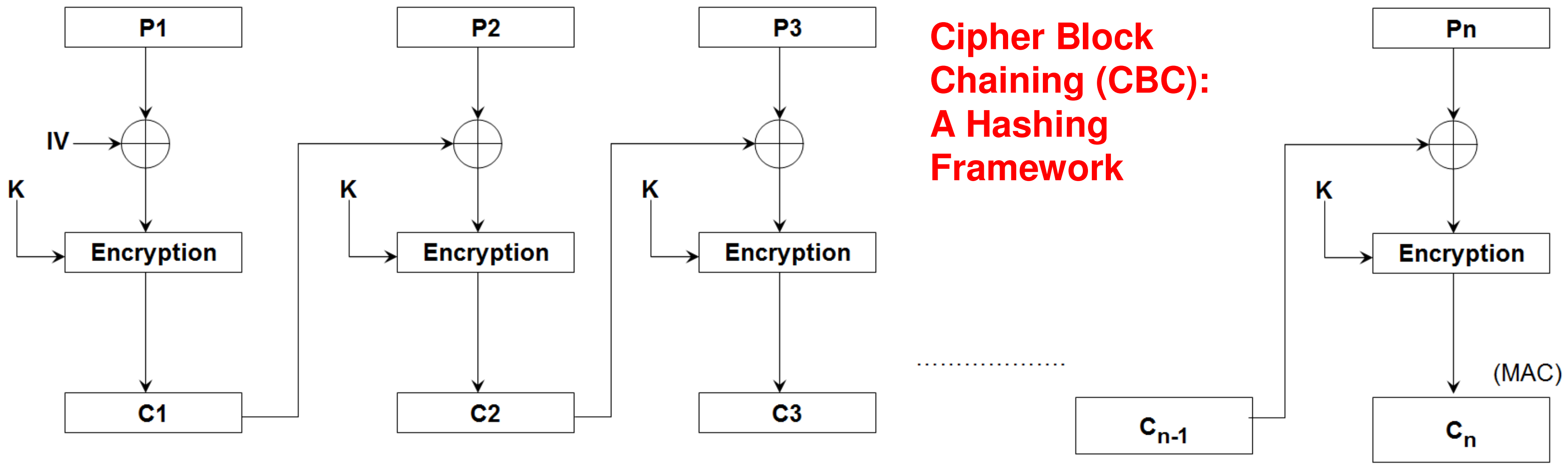
There are 64 combinations of h1 and h2 and there are only 8 of these combinations wherein h1 and h2 are the same. Hence, the probability that two different messages have the same hash value is $8/64 = 1/8 = 1/2^3$ where 3 is the size of the hash value.

General Structure of Secure Hash Functions

IV	=	Initial value
CV_i	=	chaining variable
Y_i	=	ith input block
f	=	compression algorithm

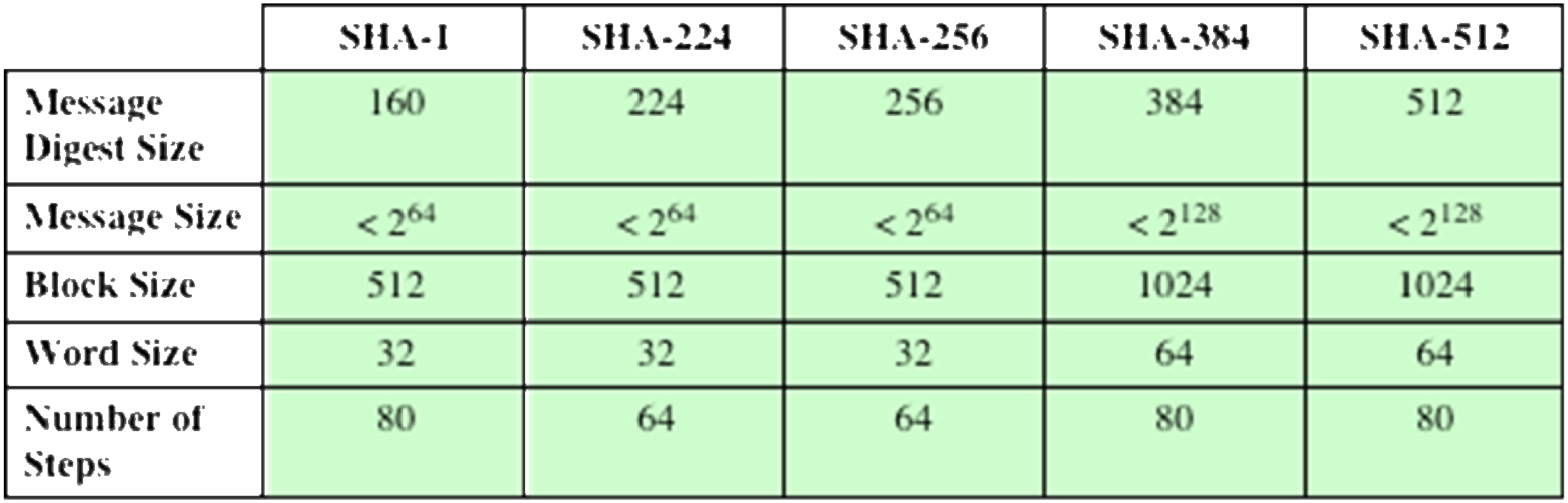
L	=	number of input blocks
n	=	length of hash code
b	=	length of input block





Cipher Block Chaining (CBC): A Hashing Framework

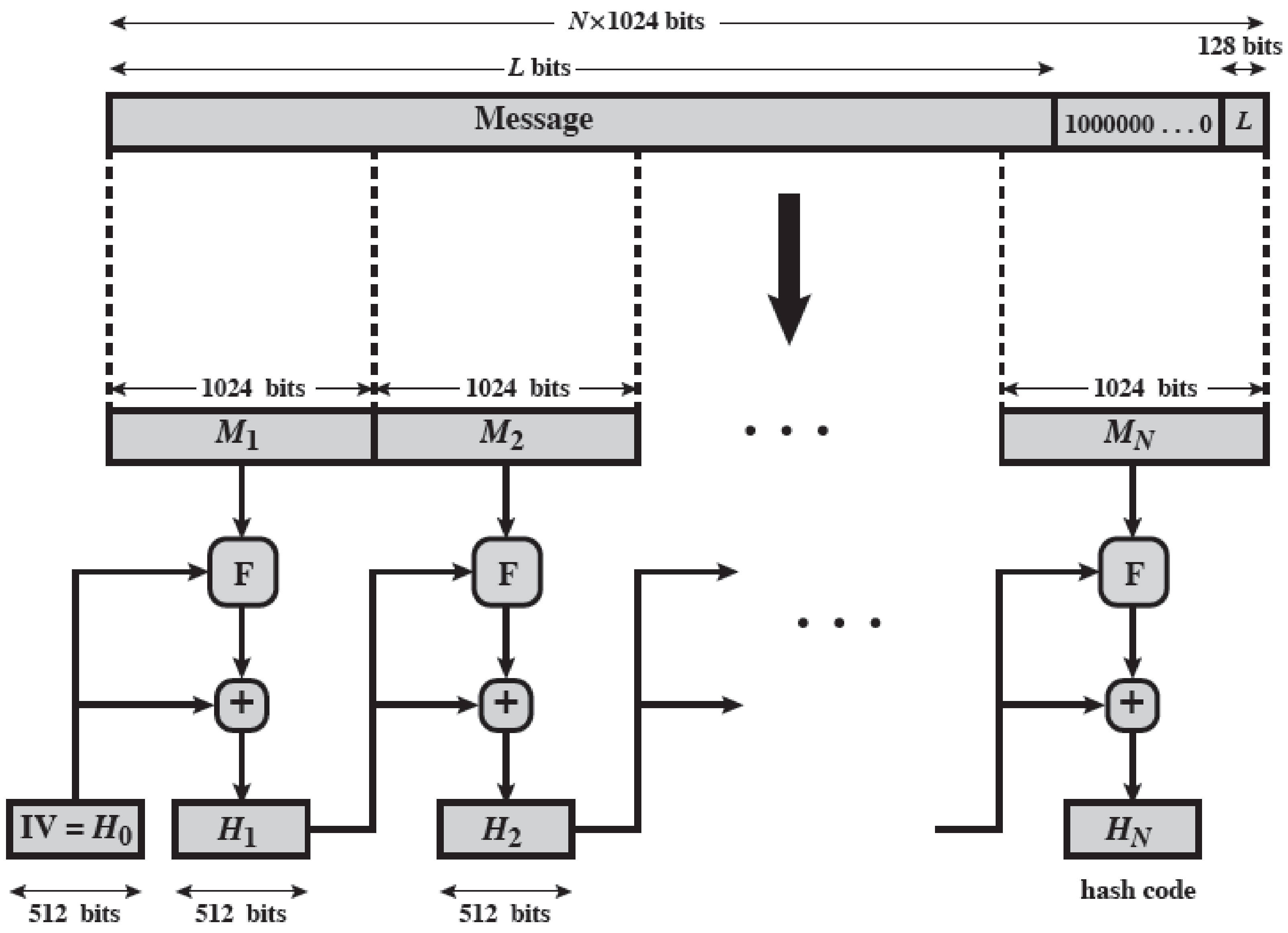
Comparison of SHA Parameters



	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message 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
Number of Steps	80	64	64	80	80

All sizes are measured in bits.

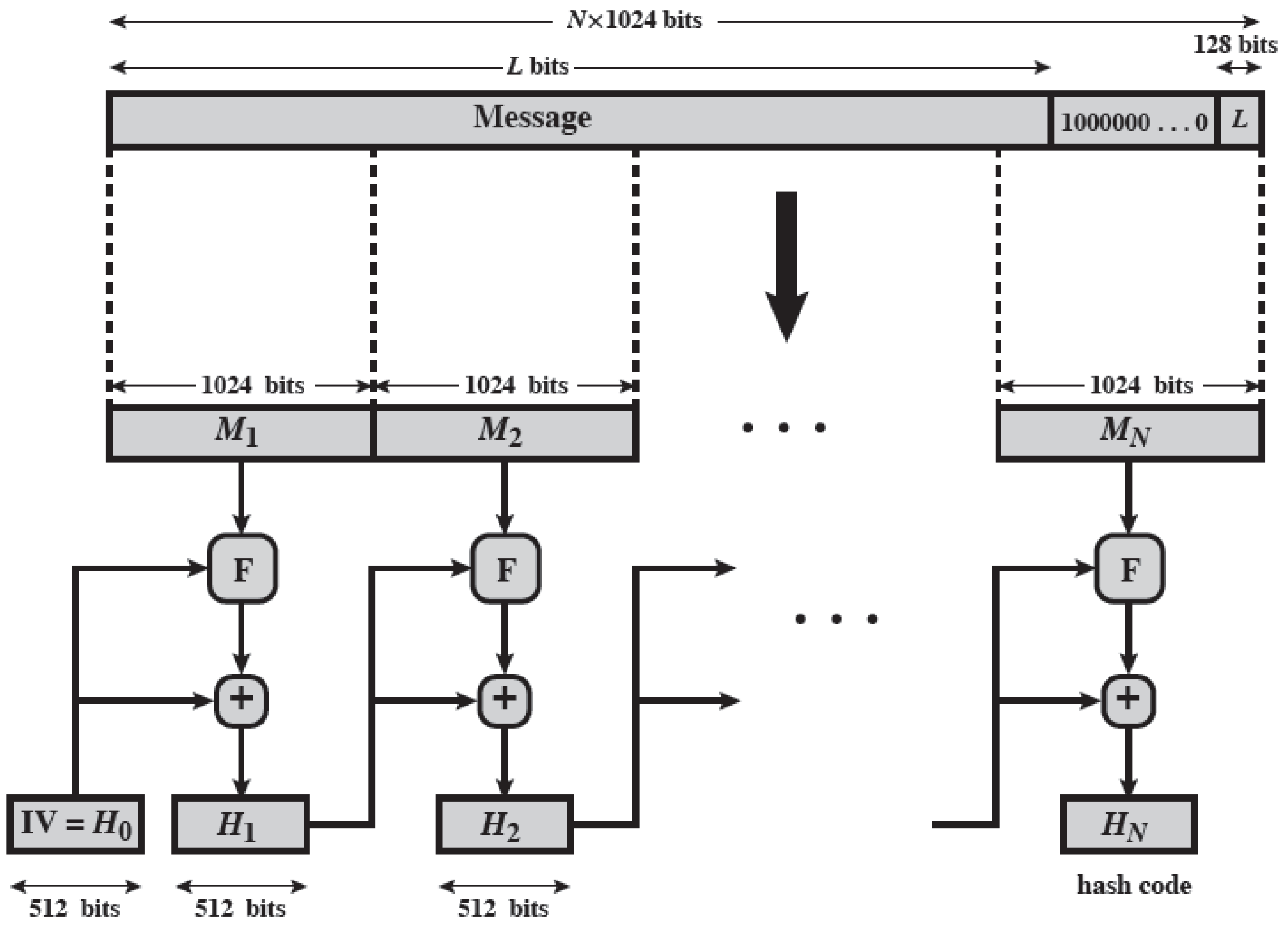
SHA-512: Message Digest Generation



+ Word-by-Word Addition mod 2^{64}

Question on SHA-512 (1)

- Consider a message of size 10,456 bytes and its hash value is computed using SHA-512. Determine the number of blocks, the number of bits of the message in the last block as well as the size of the padding 100000…0 needed for the last block.



L = 10,456 bytes
= 83,648 bits

The size of L is expressed in 128 bits.

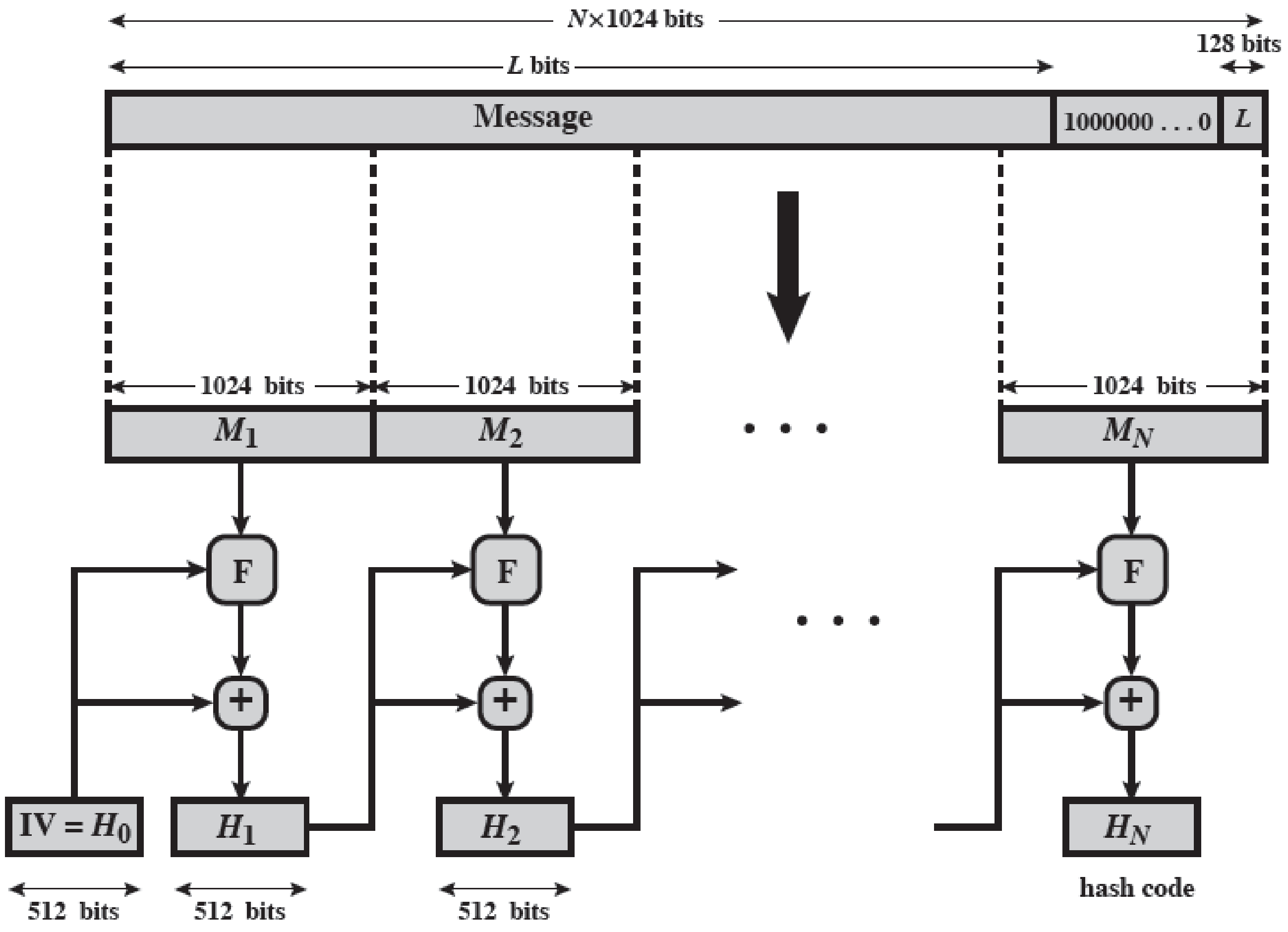
Let P be the size of the Padding 100000…0

$$\# \text{ blocks} = \frac{(L = 83{,}648 + P + 128)}{1024}$$

# blocks must be an Integer. The fraction of the L = 83,648 bits that is contained in the last block is 83,648 mod 1024 = 704 bits

Question on SHA-512 (1)

- Consider a message of size 10,456 bytes and its hash value is computed using SHA-512. Determine the number of blocks, the number of bits of the message in the last block as well as the size of the padding 100000…0 needed for the last block.



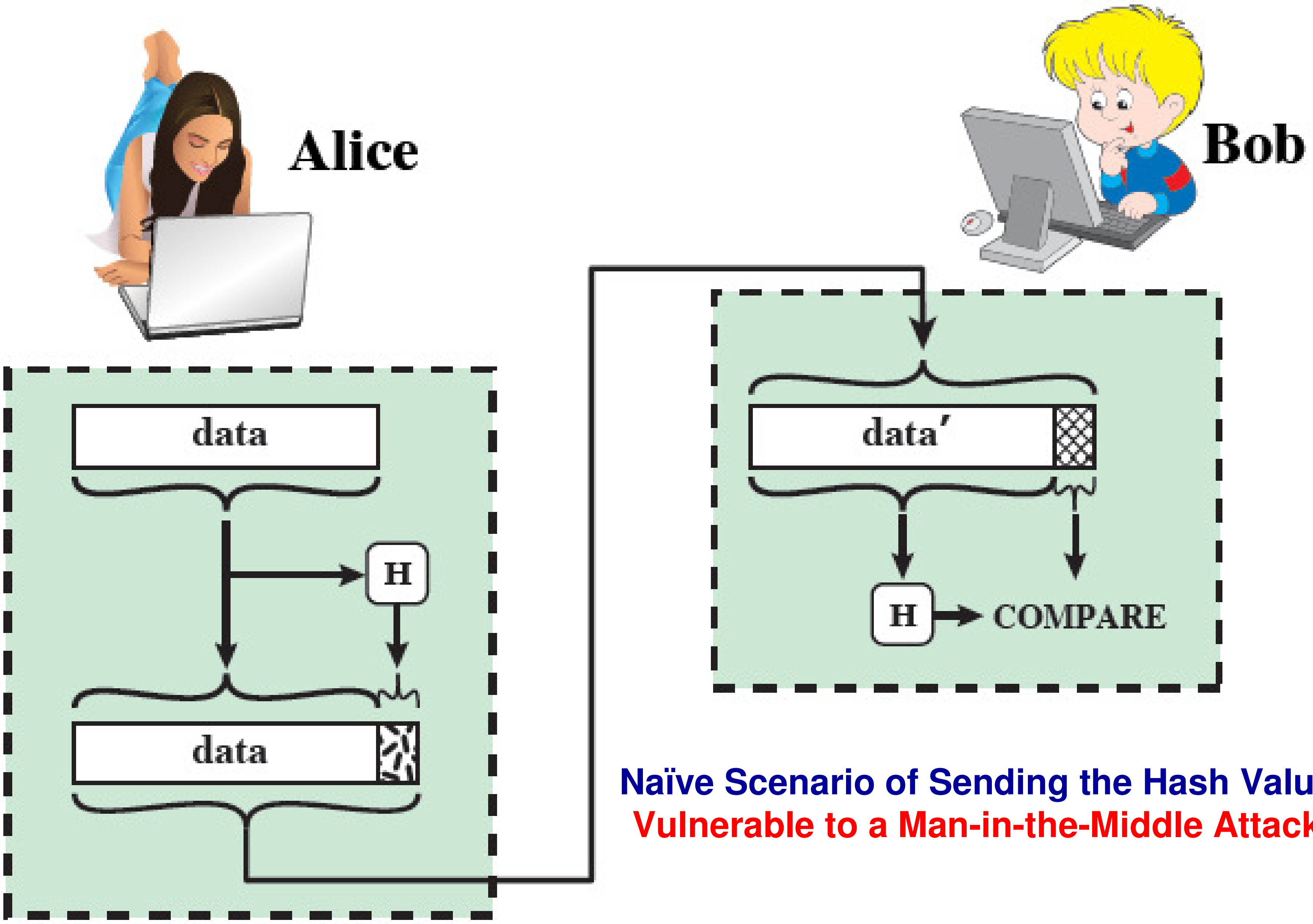
The block size = 1,024 bits

(704 + P + 128) bits = 1024

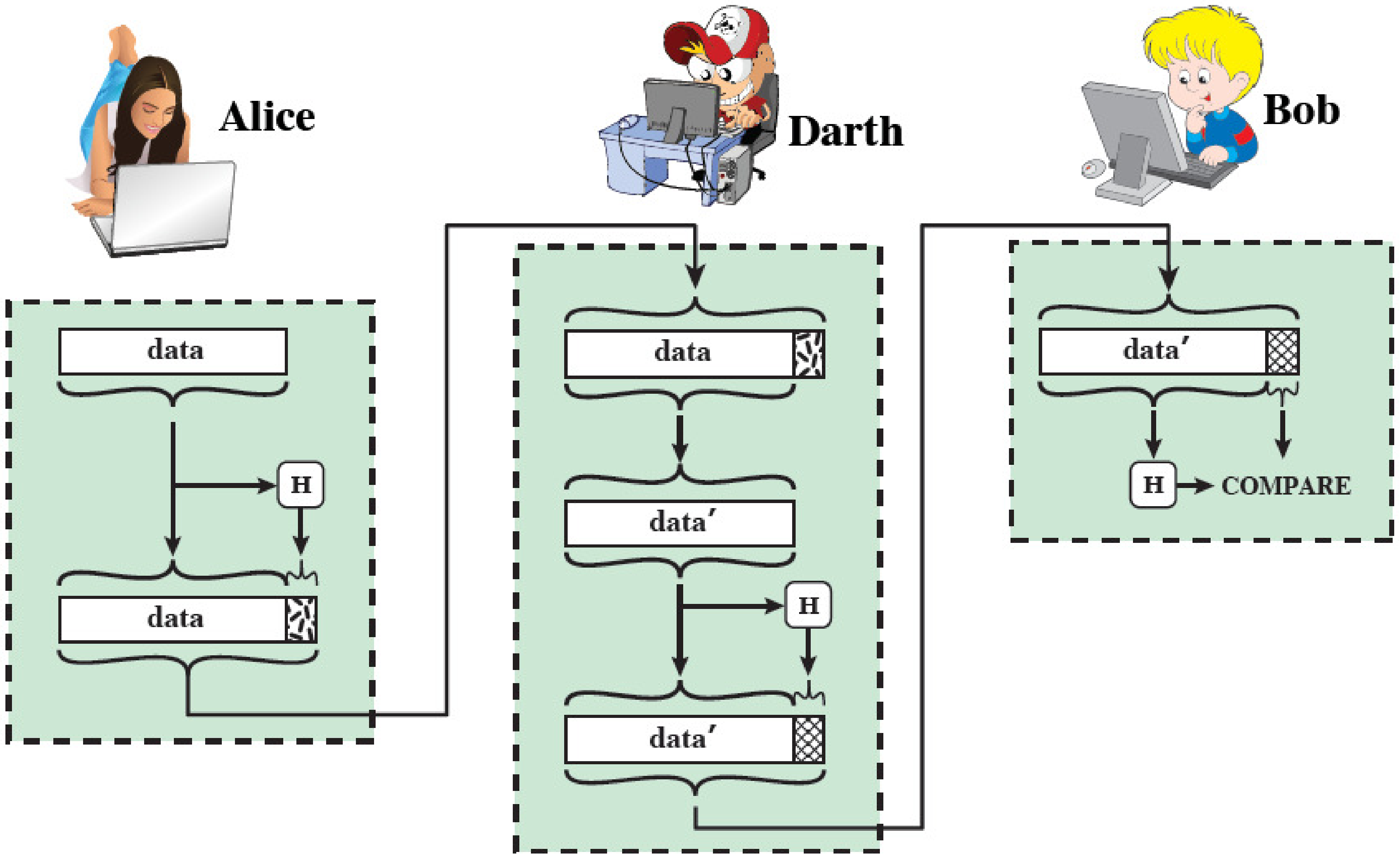
Hence, P = 192 bits

# blocks = 1 + $\lfloor 83{,}648 / 1{,}024 \rfloor$

$\boxed{\text{\# blocks} = 1 + 81 = 82}$

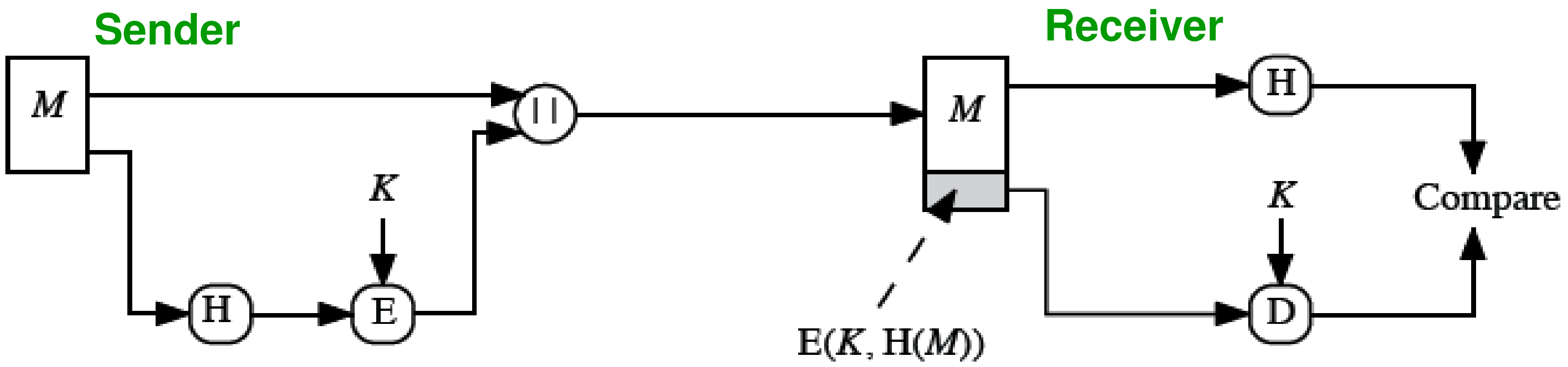


Naïve Scenario of Sending the Hash Value
Vulnerable to a Man-in-the-Middle Attack

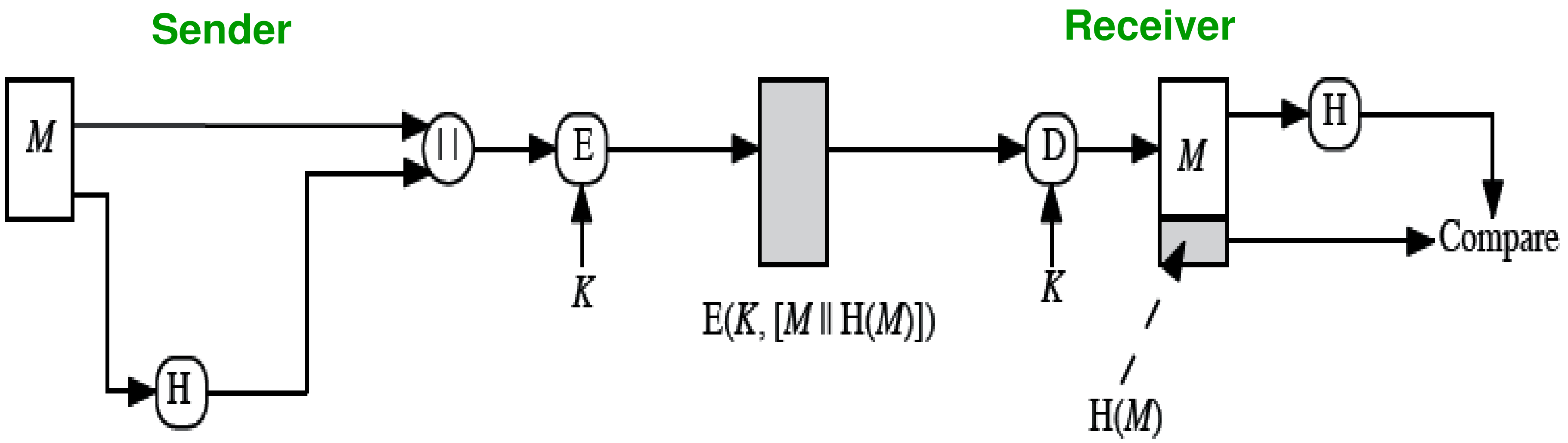


Man-in-the-Middle Attack on the Naïve Scenario of Sending Hash Values

Using Hash Functions (1)



The above scheme provides data integrity and message authentication; no confidentiality



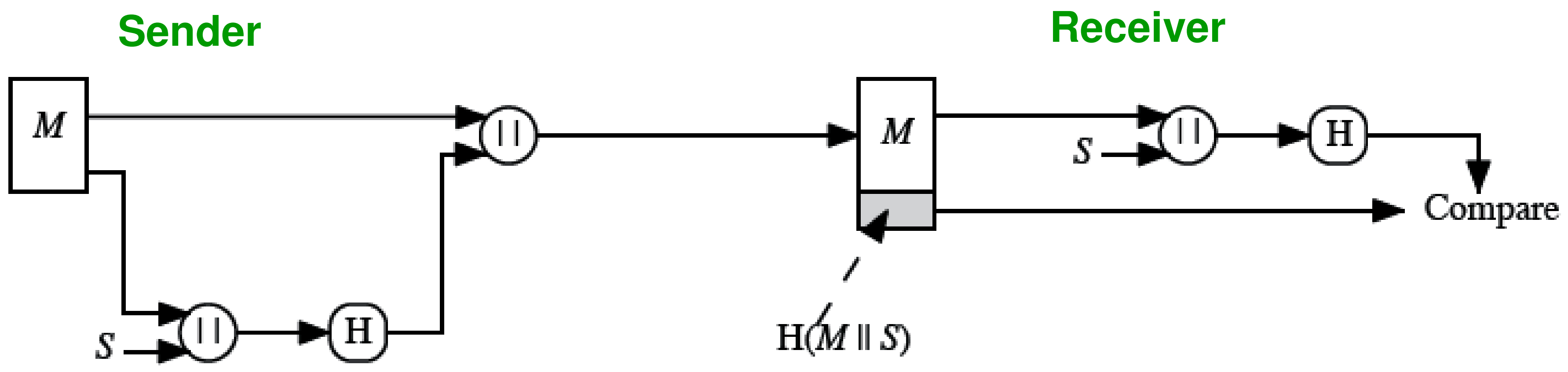
The above scheme provides data integrity, message authentication and confidentiality

Keyed Hash Functions (Message Authentication Code)

- A keyed hash function takes in a secret key (known only to the sender and receiver) and a block of data as input and produces a hash value (called the message authentication code, MAC). That is, H(K, M) = h
- If two different keys K1 and K2 are used for the same message M, then still H(K1, M) ≠ H(K2, M).
- The MAC authenticates the sender as well as validates the message for integrity.

- Note: Both H(K, M) and E(K, H(M)) serve the same purpose (message authentication and integrity).
- However, there is no encryption algorithm involved in keyed hash functions. The key is input directly to the hashing function rather than to the encryption algorithm.

Data Integrity and Message Authentication (without Encryption)

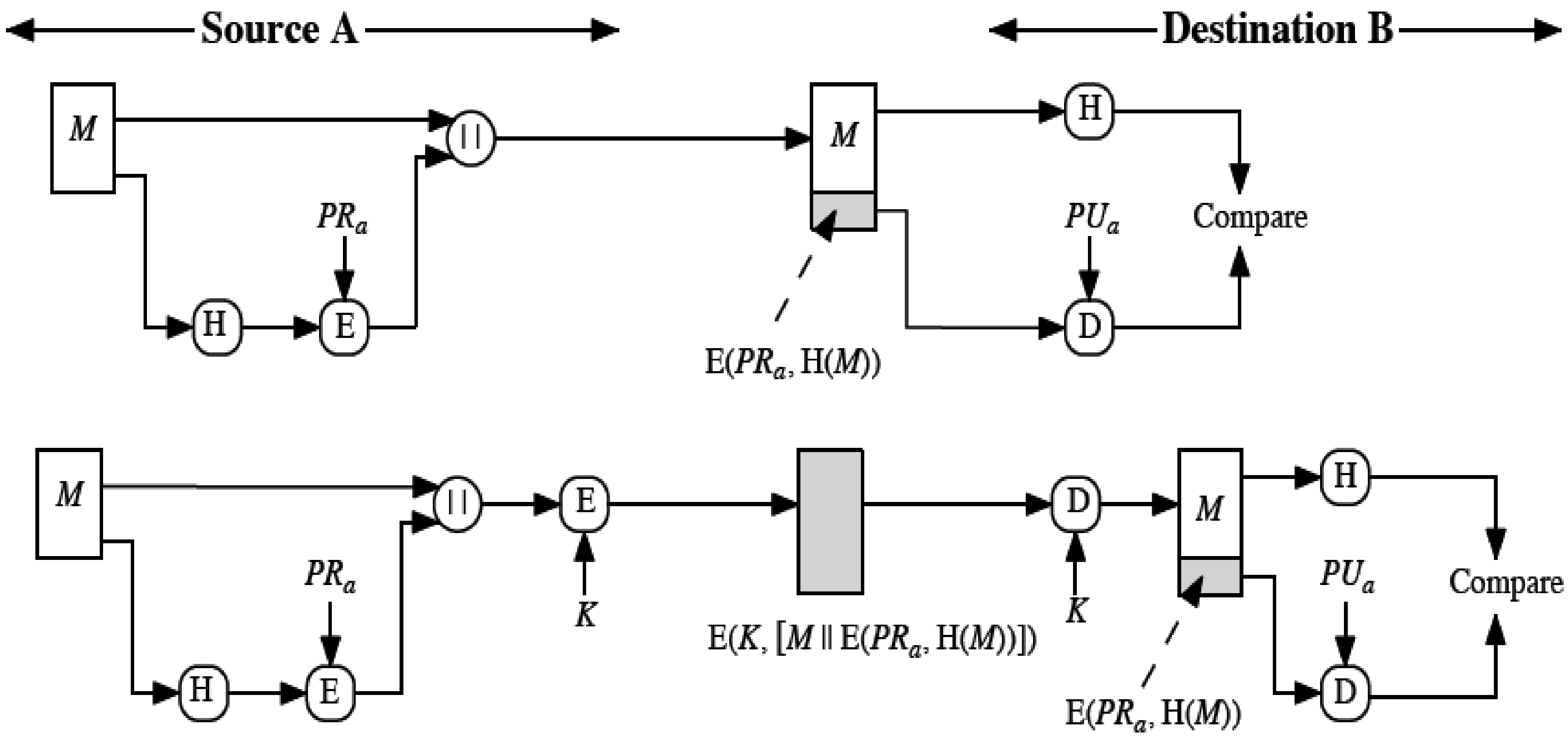


S is a secret key that is known only to the sender and receiver
The sender computes the hash value of M || S and sends along with M

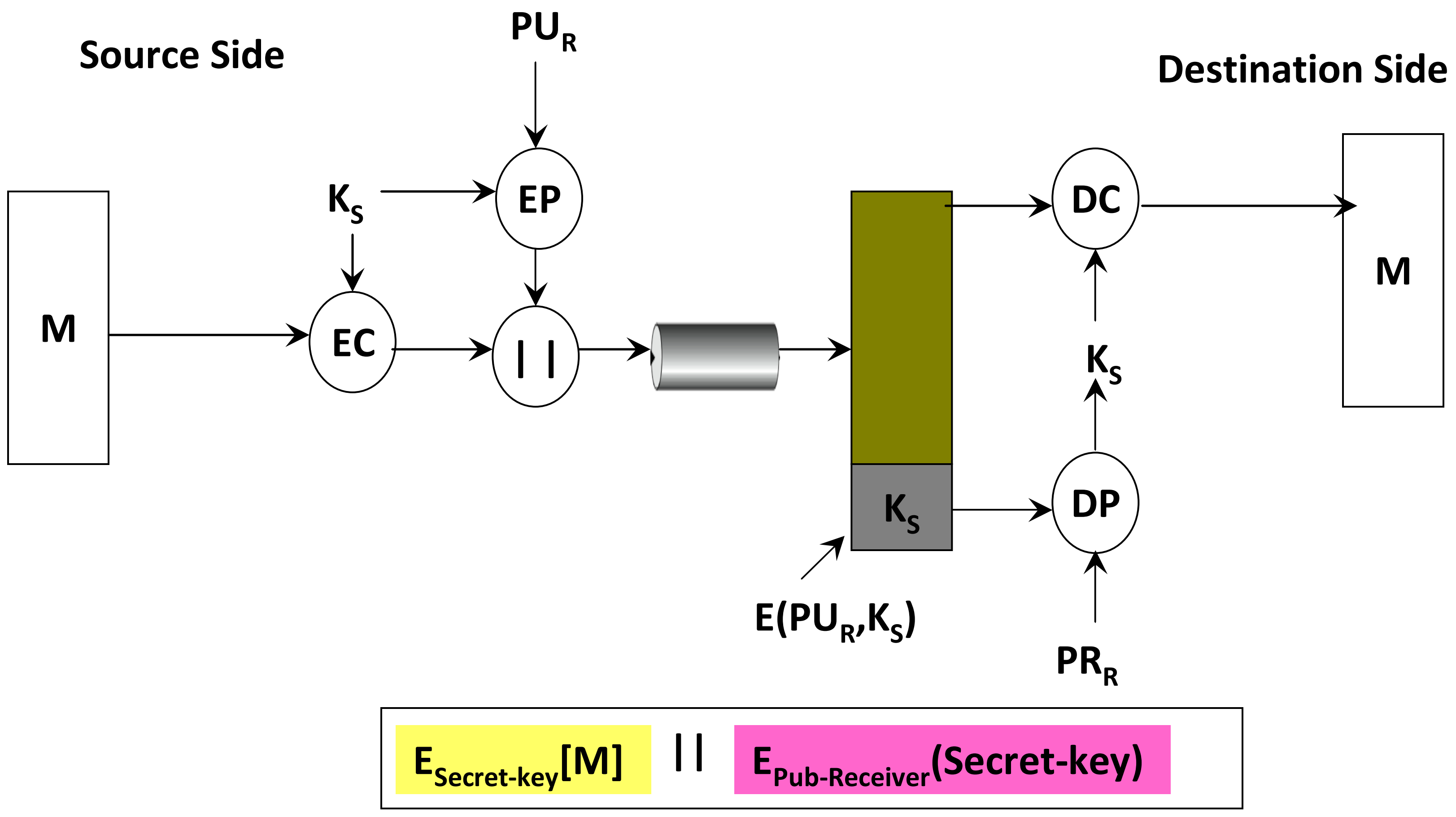
The above scheme could be considered as a simple example for Keyed Hash Functions, even though several other complicated Keyed Hash Functions exist.

Digital Signatures

- If the hash value of the message is signed with the sender's private key, it is called a digital signature.
  - Like MAC, digital signatures also facilitate to authenticate the sender as well as validate the integrity of a message.

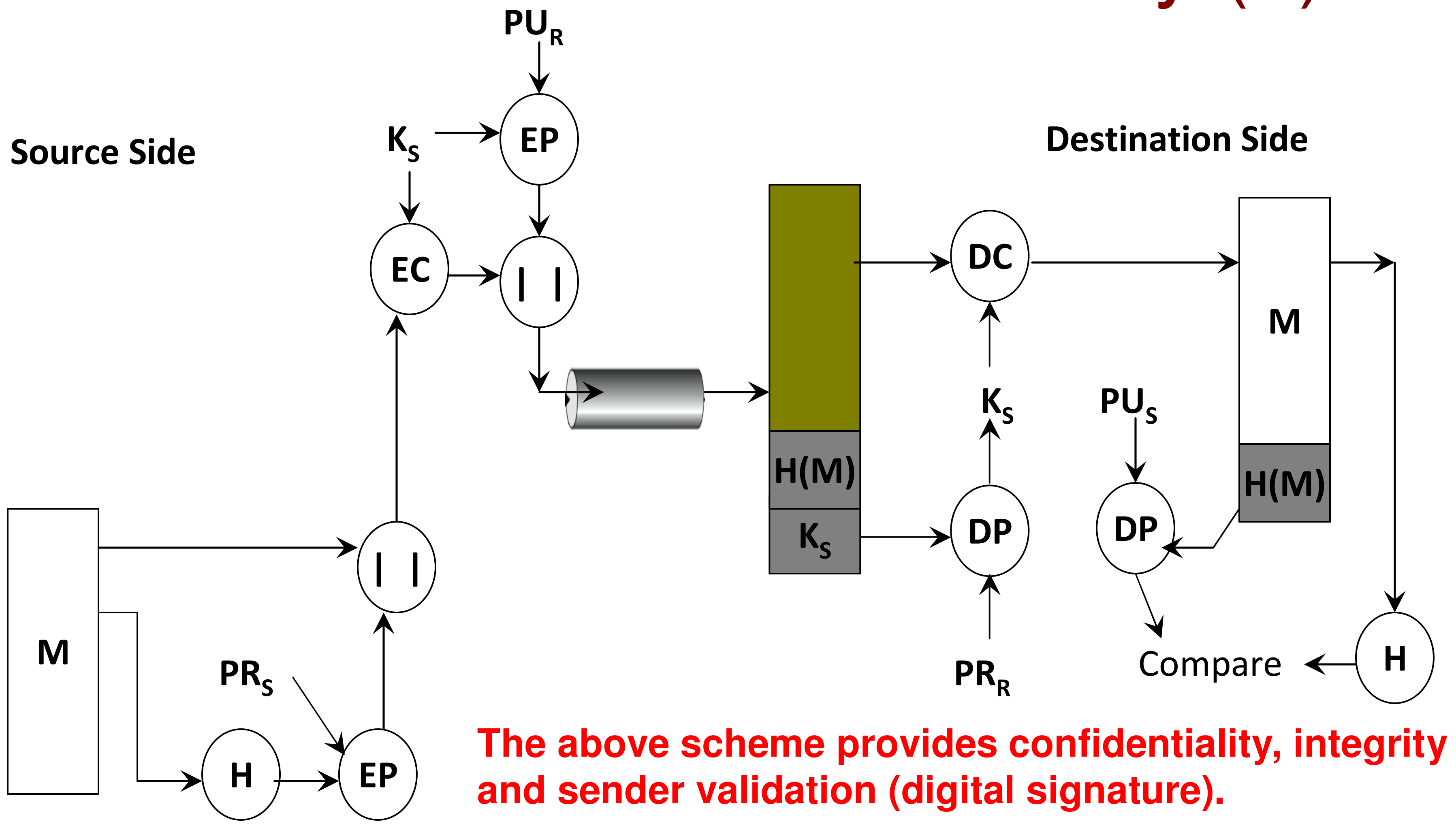


Dynamic Generation and Transmission of Secret Key (1)



The above scheme provides confidentiality only. The secret key generated at the Sender side is encrypted with the receiver's public key so that only the latter can decrypt it with its private key.

Dynamic Generation and Transmission of Secret Key (2)



The above scheme provides confidentiality, integrity and sender validation (digital signature).

$E_{Secret\text{-}key}[M \,||\, E_{Pri\text{-}Sender}(H(M))\,] \;||\; E_{Pub\text{-}Receiver}(Secret\text{-}key)$

Other Uses of Hash Functions

- To create a one-way password file
  - Operating systems store the hash value of user passwords and not the actual passwords.
  - The hash values are generated using cryptographic hash functions.
    - Due to the one-way property, one would not be able to retrieve the actual password using its hash value.
    - Due to the collision-free property, one would not be able to find an alternate password for a user password such that both these passwords have the same hash value.
- For intrusion detection and virus scanning
  - During the regular scan of the file systems, the anti-virus scanner computes and stores the hash values of the files in a secure format and location such that it cannot be tampered.
  - In order to avoid from being detected, an attacker or a virus has to be able to modify a file in such a way that its hash value would not change

Proactive Password Cracking using Bloom Filter

- Proactive Password Cracking: Store a list of bad passwords; When a user (re)sets his password, check if it is in the bad list. If so, reject the password; otherwise, accept.
- Bloom Filter: Data structure to capture the list of bad passwords.
  - A Bloom Filter of order *k* consists of a set of *k* independent hash functions $H_1(x)$, $H_2(x)$, …, $H_k(x)$, where each hash function maps a password *x* into a value in the range 0 to N-1.

$$H_i(X_j) = y \qquad 1 \le i \le k; \qquad 1 \le j \le D; \qquad 0 \le y \le N-1$$

where

X_j = *j*th word in password dictionary

D = number of words in password dictionary

Bloom Filter: Procedure and Analysis

- The Bloom Filter is a hash table.
- The hash table is of size *N* bits, with all the bits initially set to 0.
- For each password, its *k* hash values are calculated, and the corresponding bits in the hash table are set to 1. If the bit already has the value 1, it remains at 1.
- When a new password is presented to the checker, its *k* hash values are calculated. If all the corresponding bits of the hash table are equal to 1, then the password is rejected (considered to be in the list of bad passwords).
- Note that there cannot be false negatives (i.e., a user entered password that is in the bad list has to have all its *k* hash values index in the Bloom Filter to bit positions that are set to 1).
- However, there can be false positives (i.e., a user entered password that is not in the bad list could still have its *k* hash values that index to the Bloom Filter to bit positions that are set to 1).

$$H_1(\text{undertaker}) = 25 \qquad H_1(\text{hulkhogan}) = 83 \qquad H_1(\text{xG\%\#jj98}) = 665$$

$$H_2(\text{undertaker}) = 998 \qquad H_2(\text{hulkhogan}) = 665 \qquad H_2(\text{xG\%\#jj98}) = 998$$

Test password

Bloom Filter Analysis

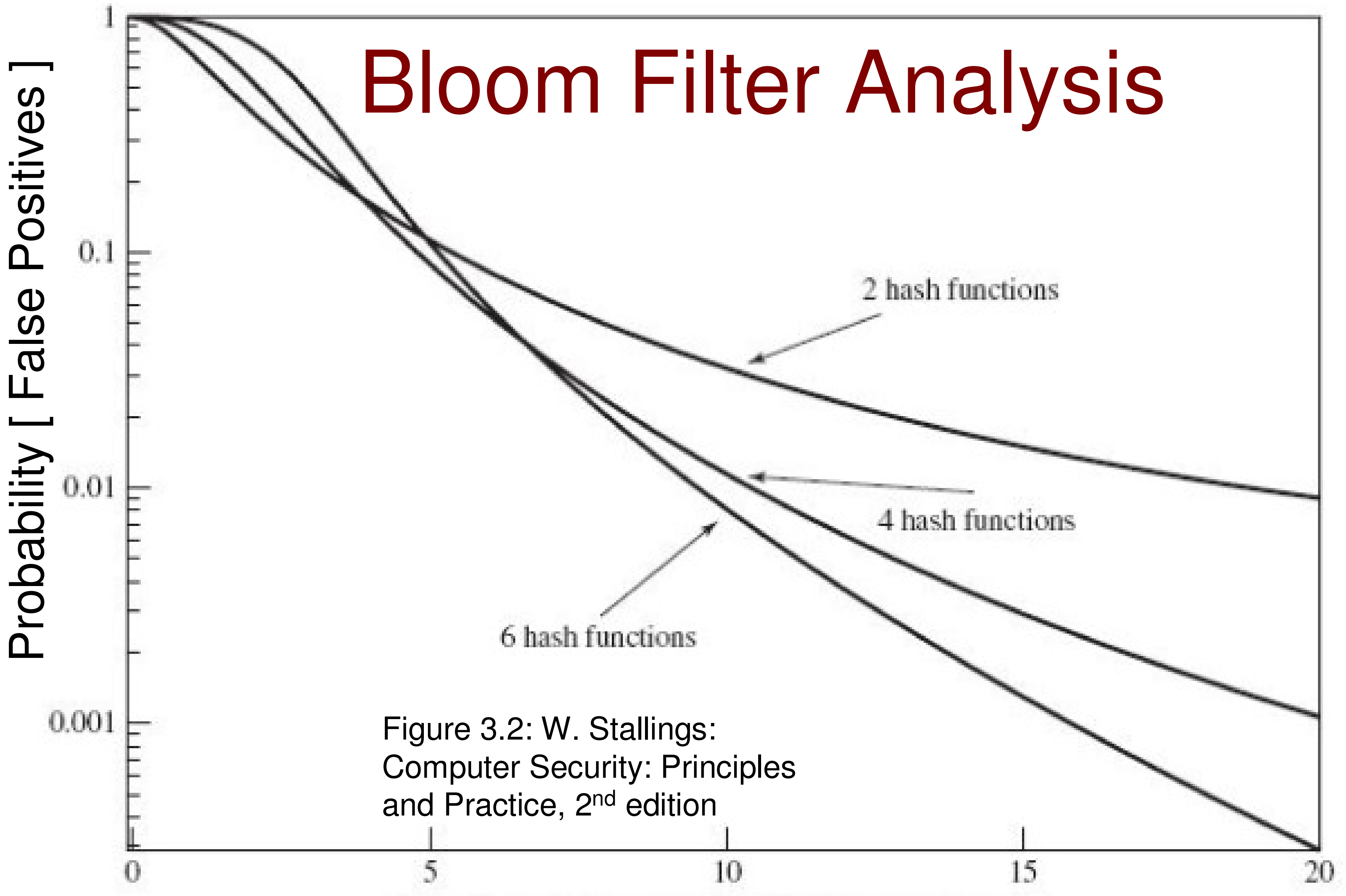


Figure 3.2: W. Stallings: Computer Security: Principles and Practice, 2nd edition

Ratio R = Max. Value in the hash table / # words in the dictionary

Bloom Filter Problem

- Consider a bloom filter hash table of size 10 bits, filled as shown below.

0	1	2	3	4	5	6	7	8	9
0	1	0	0	1	0	1	0	1	1

- Find the probability of obtaining a false positive if the number of hash functions used is varied from 1 to 6. Assume each hash function gives a distinct hash value (in the range 0…9) for a particular word (i.e., the hash value generated by two different hash functions for the same word is different).

Solution

# Hash functions: 1

For a test word, its hash value could correspond to any of the 10 bit positions and 5 of these bit positions are 1s. Hence, the chances of the hash value corresponding to one of these 1s and incurring a false positive is 5/10 = 0.5

# Hash functions: 2

There are a total of C(10, 2) combinations of bit positions that could correspond to the Hash values generated for the two hash functions. As there five 1s in the above table. So, there are C(5, 2) combinations of bit positions that could correspond to two 1s. The probability of obtaining hash values that correspond to the two 1s and incurring a false positive is C(5, 2) / C(10, 2).

Bloom Filter Problem (1)

- Consider a bloom filter hash table of size 10 bits, filled as shown below.

0	1	2	3	4	5	6	7	8	9
0	1	0	0	1	0	1	0	1	1

# Hash functions: 2

C(5, 2) = 5! / {(5-2)! * 2!} = 5! / (3! * 2!) = 10

C(10, 2) = 10! / (8! * 2!) = 45

P(false positive with two hash functions) = 10/45 = 0.22

# Hash functions: 3

C(5, 3) = 5! / {(5-3)! * 3!} = 5! / (2! * 3!) = 10

C(10, 3) = 10! / (7! * 3!) = 240

P(false positive with three hash functions) = 10/240 = 0.042

# Hash functions: 4

C(5, 4) = 5! / {(5-4)! * 4!} = 5! / (1! * 4!) = 5

C(10, 4) = 10! / (6! * 4!) = 210

P(false positive with four hash functions) = 5/210 = 0.024

Bloom Filter Problem (2)

- Consider a bloom filter hash table of size 10 bits, filled as shown below.

0	1	2	3	4	5	6	7	8	9
0	1	0	0	1	0	1	0	1	1

# Hash functions: 5

C(5, 5) = 5! / {(5-5)! * 5!} = 5! / (0! * 5!) = 1

C(10, 5) = 10! / (5! * 5!) = 252

P(false positive with five hash functions) = 1/252 = 0.004

# Hash functions: 6

C(5, 6) = 0

C(10, 6) = 10! / (4! * 6!) = 210

P(false positive with six hash functions) = 0/210 = 0