

CYB650

Introduction to Cryptography

Cryptography Algorithms



Prepared by Dr. Esraa Saleh Alomari
Lecture #8



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

The Part 2 of the module 4 covers the Cryptography Algorithms are as follows:

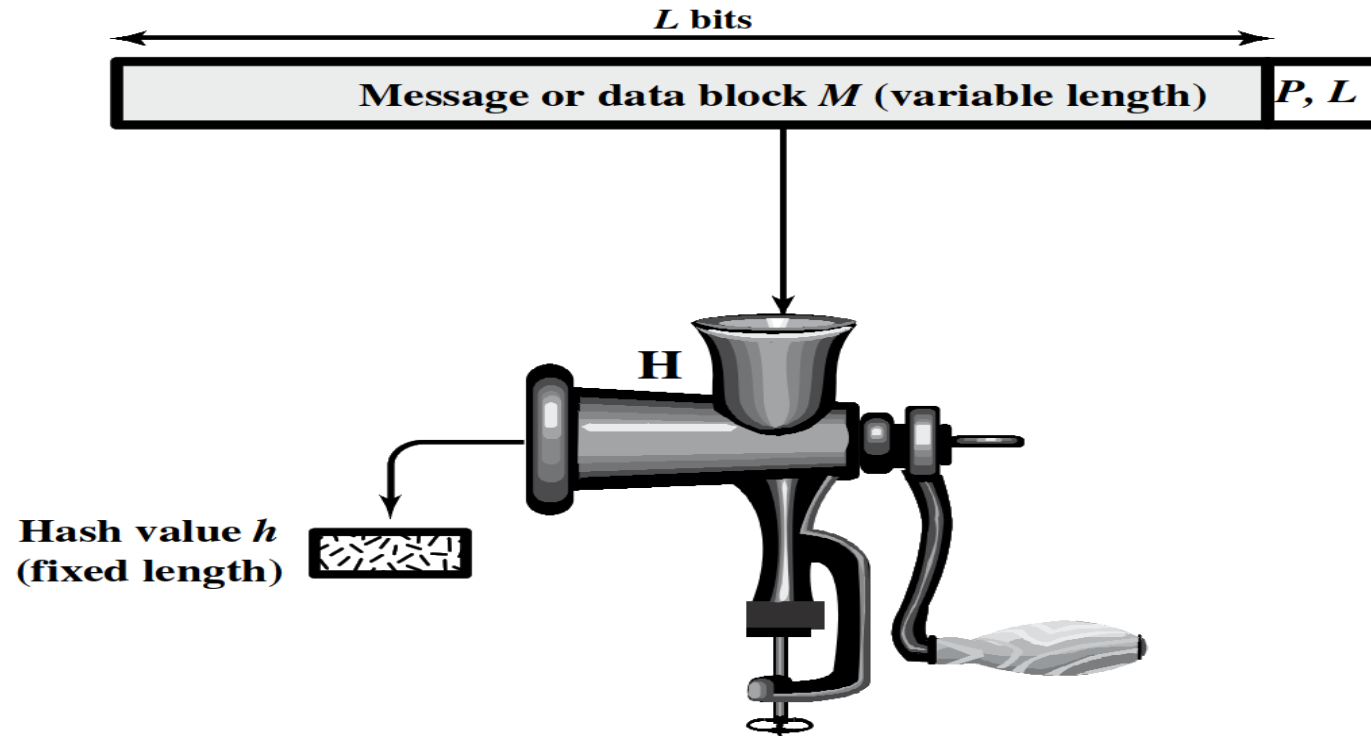
➤ *Cryptographic Hash Function*

Introduction

- ❑ A hash function H accepts a variable-length block of data M as input and produces a fixed-size hash value $h = H(M)$ “good” hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random. In general terms, the principal object of a hash function is data integrity. A change to any bit or bits in M results, with high probability, in a change to the hash code.

- ❑ The kind of hash function needed for security applications is referred to as a cryptographic hash function. A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either (a) a data object that maps to a pre-specified hash result (the one-way property) or (b) two data objects that map to the same hash result (the collision-free property). Because of these characteristics, hash functions are often used to determine whether or not data has changed.

Hash Function



P, L = padding plus length field

Figure 11.1 Cryptographic Hash Function; $h = H(M)$

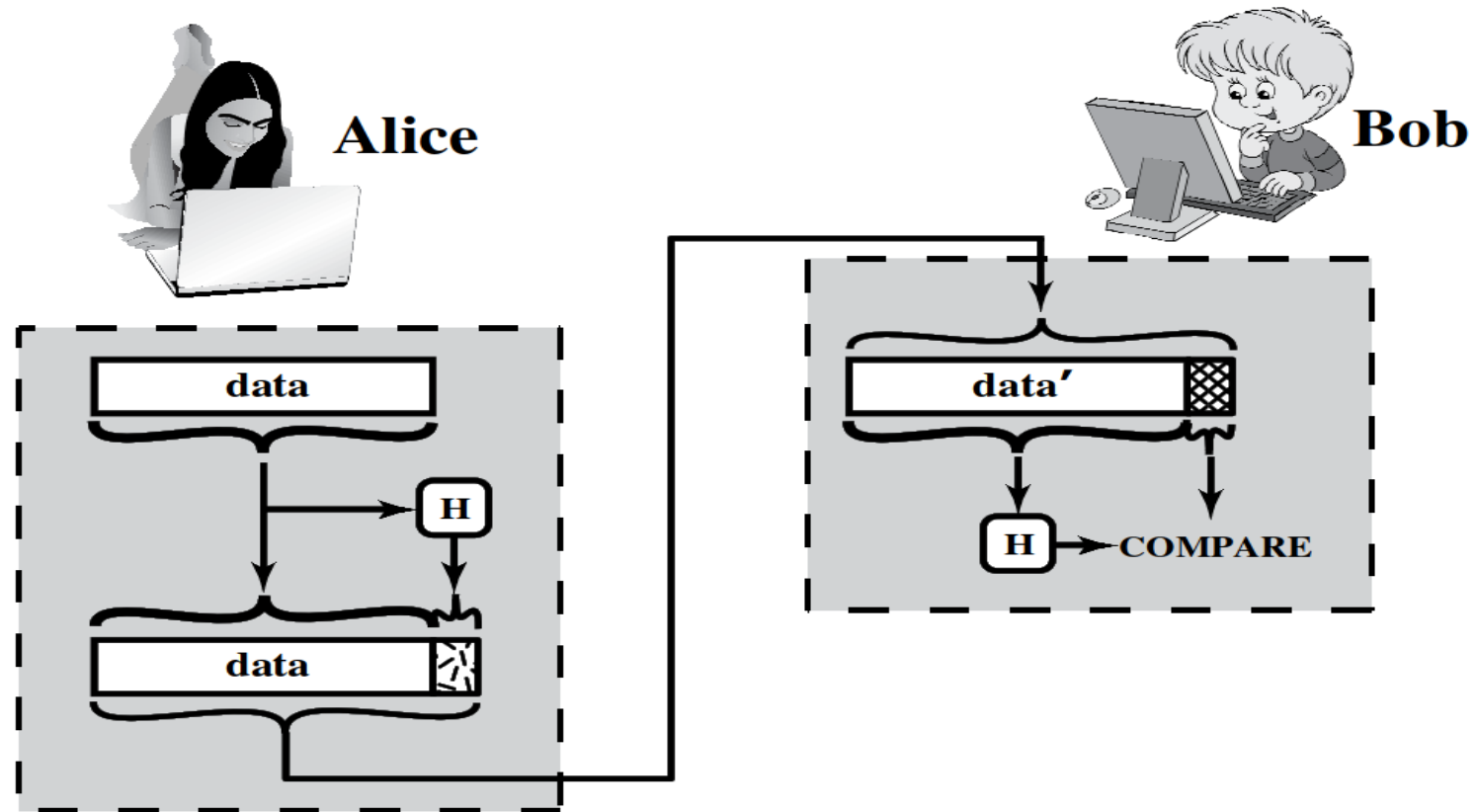
❑ Figure 11.1 depicts the general operation of a cryptographic hash function.

Typically, the input is padded out to an integer multiple of some fixed length (e.g., 1024 bits), and the padding includes the value of the length of the original message in bits. The length field is a security measure to increase the difficulty for an attacker to produce an alternative message with the same hash value.

- ❑ Perhaps the most versatile cryptographic algorithm is the cryptographic hash function. It is used in a wide variety of security applications and Internet protocols. To better understand some of the requirements and security implications for cryptographic hash functions, it is useful to look at the range of applications in which it is employed.

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

- ❑ The essence of the use of a hash function for message authentication is as follows. The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message. The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value. If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered (Figure 11.2a).



(a) Use of hash function to check data integrity

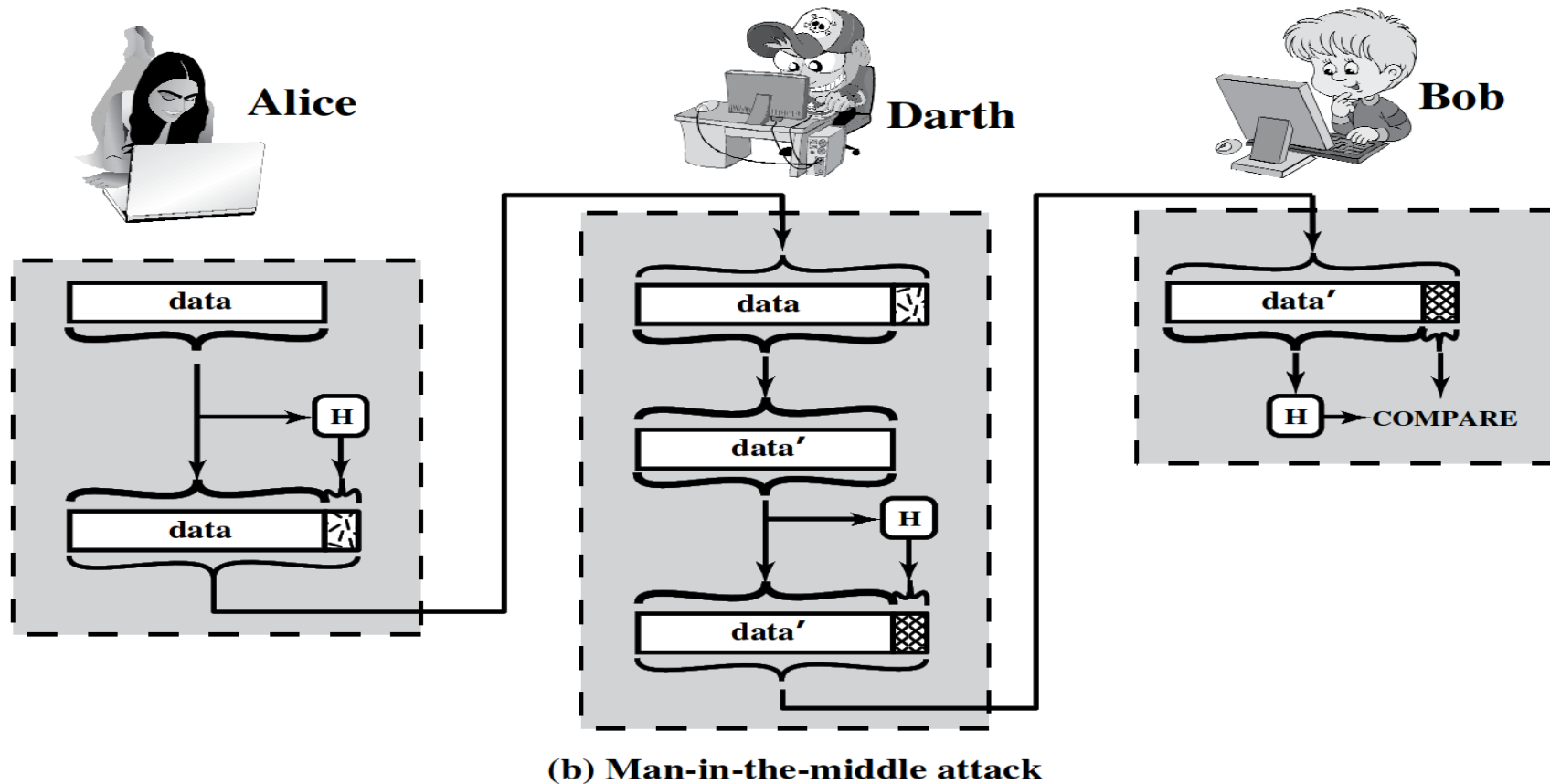


Figure 11.2 Attack Against Hash Function

- ❑ The hash function must be transmitted in a secure fashion. That is, the hash function must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver. This type of attack is shown in Figure 11.2b. In this example, Alice transmits a data block and attaches a hash value.

- ❑ Darth intercepts the message, alters or replaces the data block, and calculates and attaches a new hash value. Bob receives the altered data with the new hash value and does not detect the change. To prevent this attack, the hash value generated by Alice must be protected.

Message Authentication

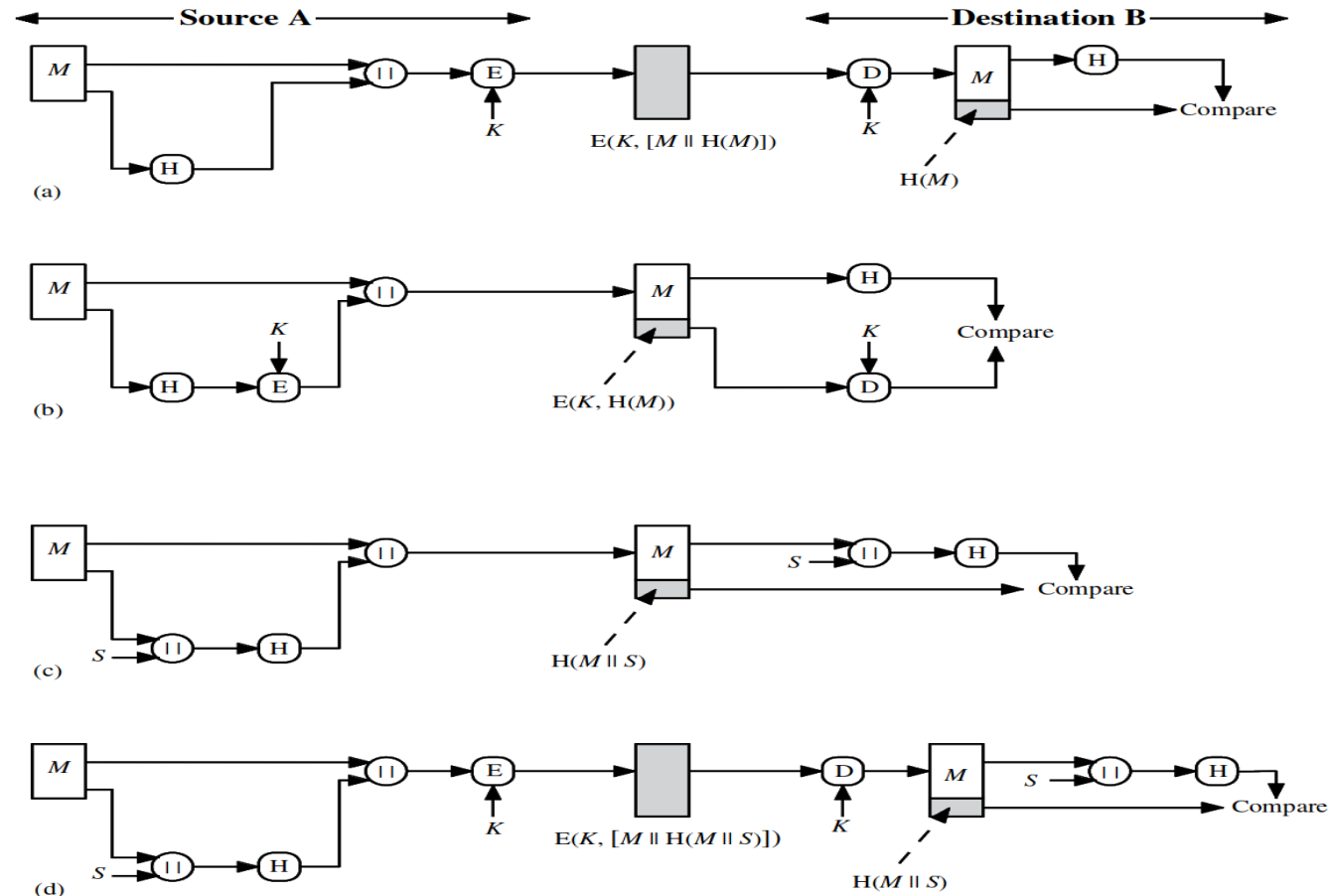


Figure 11.3 Simplified Examples of the Use of a Hash Function for Message Authentication

- ❑ The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.
- ❑ Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

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

- ❑ it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.
- ❑ Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.

- ❑ When confidentiality is not required, method (b) has an advantage over methods (a) and (d), which encrypts the entire message, in that less computation is required. Nevertheless, there has been growing interest in techniques that avoid encryption (Figure 11.3c).

- ❑ Encryption software is relatively slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- ❑ Encryption hardware costs are not negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
- ❑ Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- ❑ Encryption algorithms may be covered by patents, and there is a cost associated with licensing their use.

- ❑ More commonly, message authentication is achieved using a message authentication code (MAC), also known as a keyed hash function. Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties. A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC, which is associated with the protected message.

- ❑ If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value. An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key. Note that the verifying party also knows who the sending party is because no one else knows the secret key.

- ❑ Another important application, which is similar to the message authentication application, is the digital signature. The operation of the digital signature is similar to that of the MAC. In the case of the digital signature, the hash value of a message is encrypted with a user's private key.

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

- ❑ Figure 11.4 illustrates, in a simplified fashion, how a hash code is used to provide a digital signature.
- ❑ The hash code is encrypted, using public-key encryption with the sender's private key. As with Figure 11.3b, this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

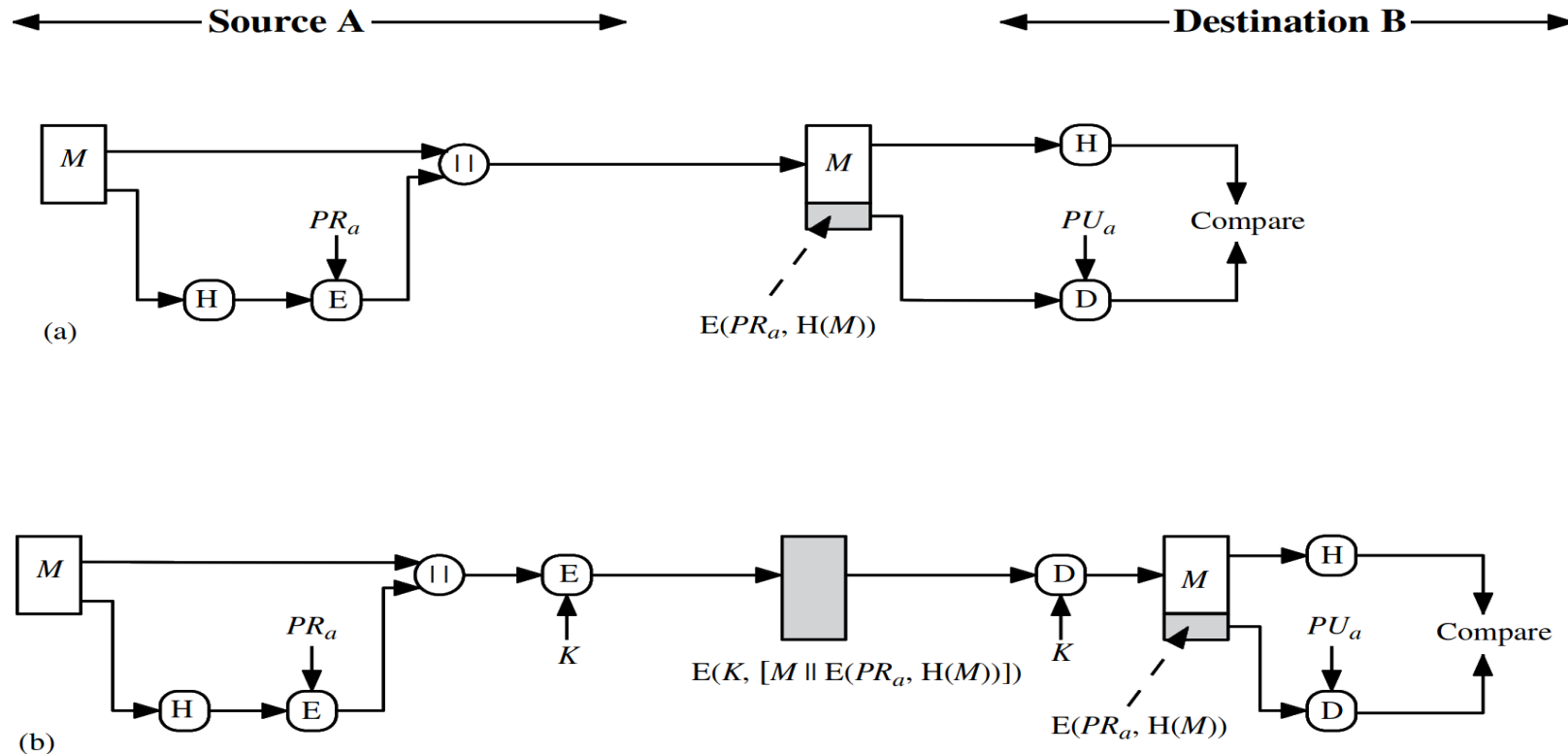


Figure 11.4 Simplified Examples of Digital Signatures

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

- ❑ To get some feel for the security considerations involved in cryptographic hash functions, we present two simple, insecure hash functions in this section. All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of n -bit blocks. The input is processed one block at a time in an iterative fashion to produce an n -bit hash



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

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

where

C_i = i th bit of the hash code, $1 \leq i \leq n$

m = number of n -bit blocks in the input

b_{ij} = i th bit in j th block

$\oplus$ = XOR operation

Two Simple Hash Functions

- ❑ This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each n -bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2^{-n} . With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2^{-128} the hash function on this type of data has an effectiveness of 2^{-112} .

□ A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows:

- 1- Initially set the n-bit hash value to zero.
- 2- Process each successive n-bit block of data as follows:
 - a. Rotate the current hash value to the left by one bit.
 - b. XOR the block into the hash value.

Table 11.1 Requirements for a Cryptographic Hash Function H

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.

Table 11.2 Hash Function Resistance Properties Required for Various Data Integrity Applications

	Preimage Resistant	Second Preimage Resistant	Collision Resistant
Hash + digital signature	yes	yes	yes*
Intrusion detection and virus detection		yes	
Hash + symmetric encryption			
One-way password file	yes		
MAC	yes	yes	yes*

*Resistance required if attacker is able to mount a chosen message attack

- ❑ As with encryption algorithms, there are two categories of attacks on hash functions: brute-force attacks and cryptanalysis. A brute-force attack does not depend on the specific algorithm but depends only on bit length. In the case of a hash function, a brute-force attack depends only on the bit length of the hash value. A cryptanalysis, in contrast, is an attack based on weaknesses in a particular cryptographic algorithm. We look first at brute-force attacks.

PREIMAGE AND SECOND PREIMAGE ATTACKS For a preimage or second preimage attack, an adversary wishes to find a value y such that $H(y)$ is equal to a given hash value h . The brute-force method is to pick values of y at random and try each value until a collision occurs. For an m -bit hash value, the level of effort is proportional to 2^m . Specifically, the adversary would have to try, on average, 2^{m-1} values of y to find one that generates a given hash value h . This result is derived in Appendix 11A [Equation (11.1)].

COLLISION RESISTANT ATTACKS For a collision resistant attack, an adversary wishes to find two messages or data blocks, x and y , that yield the same hash function: $H(x) = H(y)$. This turns out to require considerably less effort than a preimage or second preimage attack. The effort required is explained by a mathematical result referred to as the **birthday paradox**. In essence, if we choose random variables from a uniform distribution in the range 0 through $N - 1$, then the probability that a repeated element is encountered exceeds 0.5 after $\sqrt{N}$ choices have been made. Thus, for an m -bit hash value, if we pick data blocks at random, we can expect to find two data blocks with the same hash value within $\sqrt{2^m} = 2^{m/2}$ attempts. The mathematical derivation of this result is found in Appendix 11A.

- ❑ As with encryption algorithms, cryptanalytic attacks on hash functions seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a hash algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack. That is, an ideal hash algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.



- ❑ In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions. To understand these, we need to look at the overall structure of a typical secure hash function, indicated in Figure 11.8. This structure, referred to as an iterated hash function, and is the structure of most hash functions in use today, including SHA.

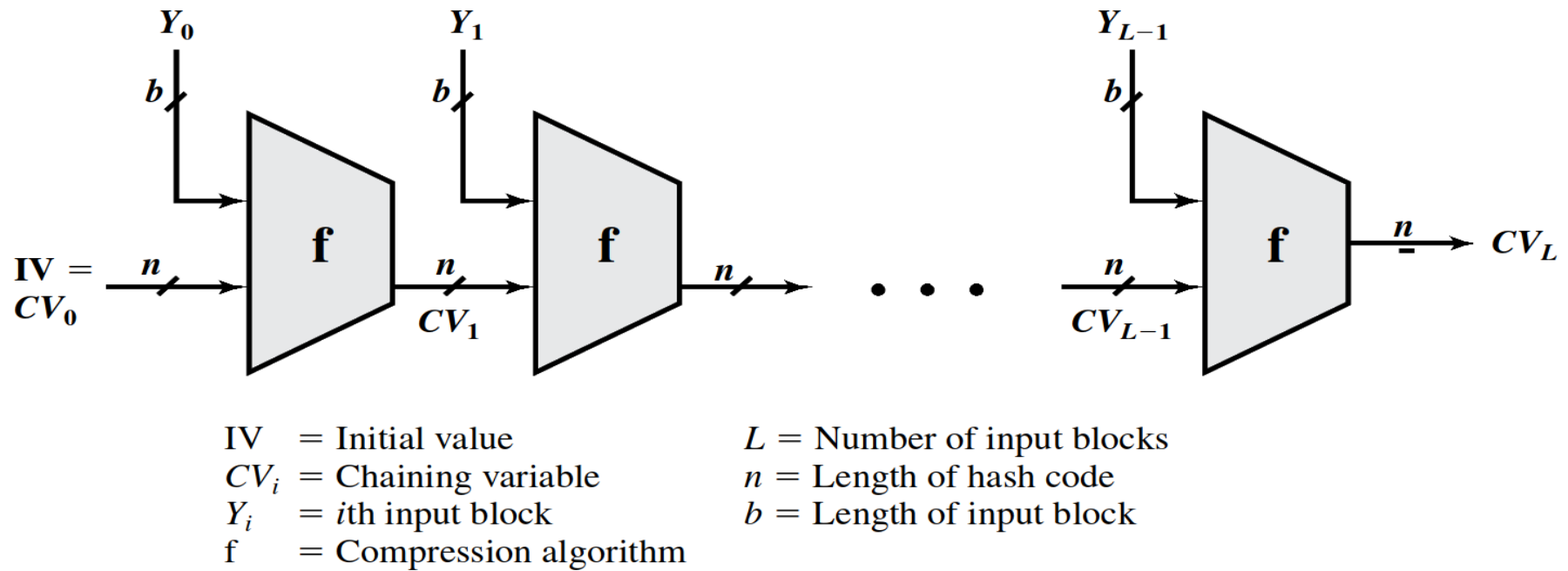


Figure 11.8 General Structure of Secure Hash Code

Thank You

ANY
QUESTIONS?

