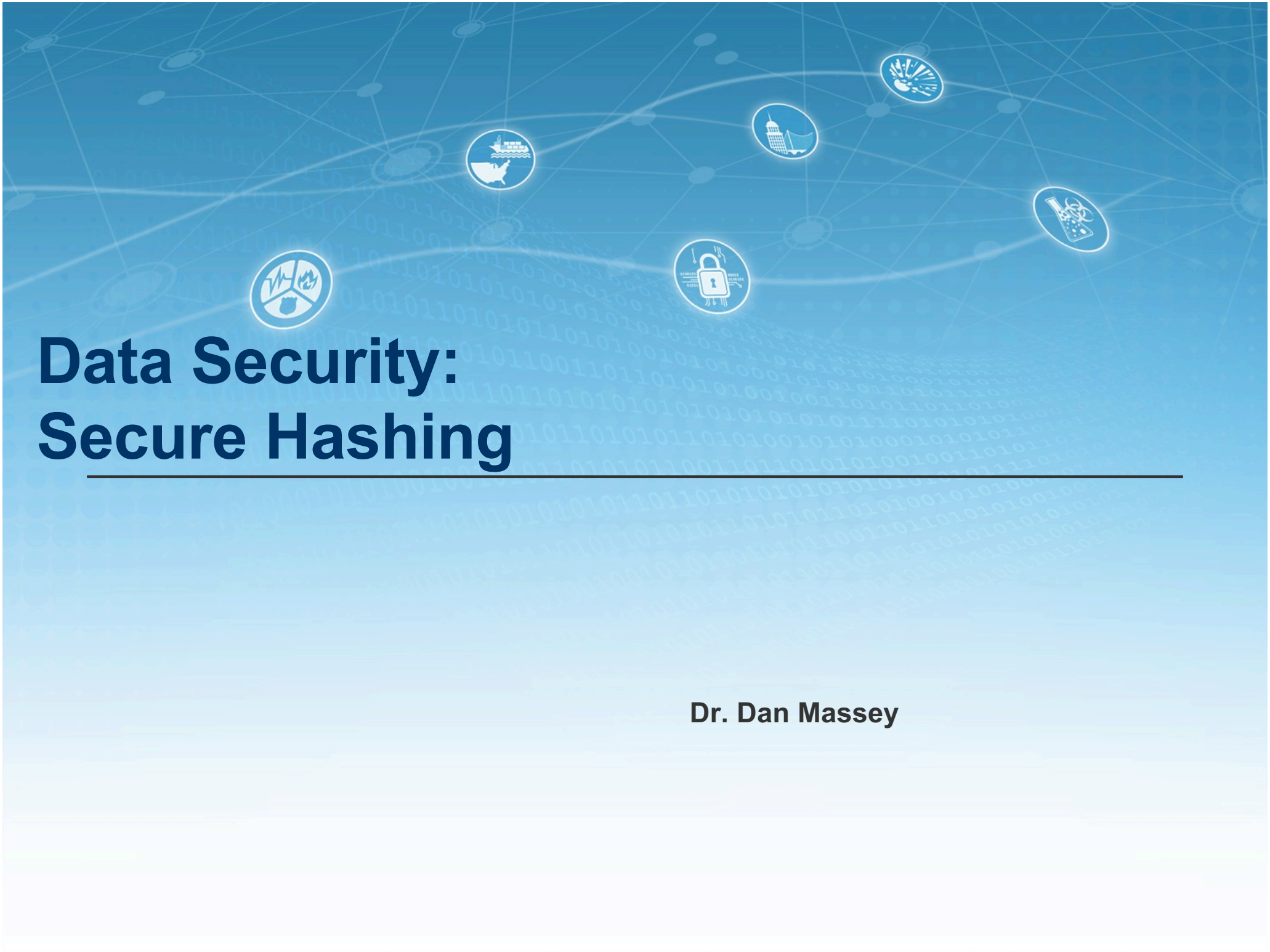


Data Security: Secure Hashing

Dr. Dan Massey

Motivating Problems:

Confidentiality: how could we encrypt a message? (email, web/http, sms, etc)

Apply AES – Select Key Size and Mode

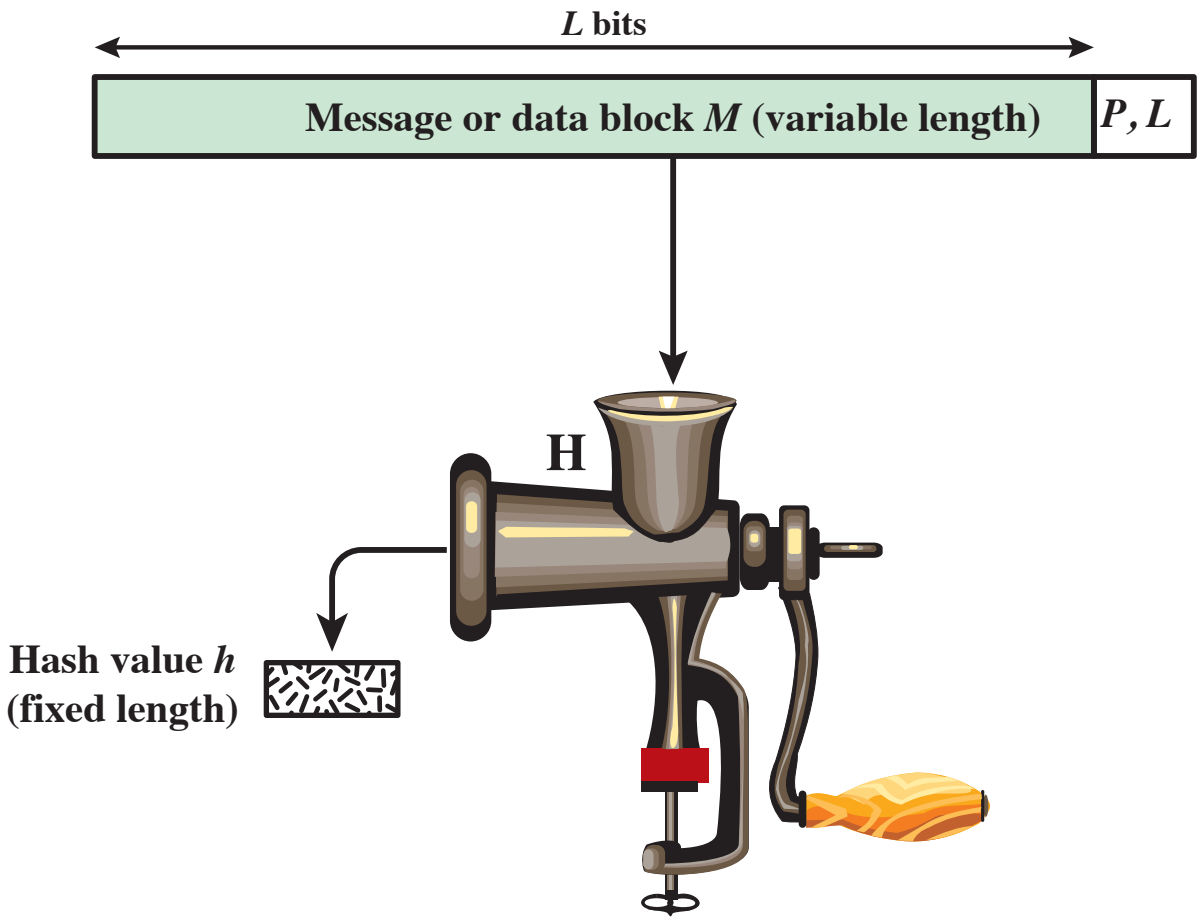
Or Encrypt Using Receiver's Public Key

Integrity: how could we authenticate a message?

Message Authentication Code with Symmetric Key

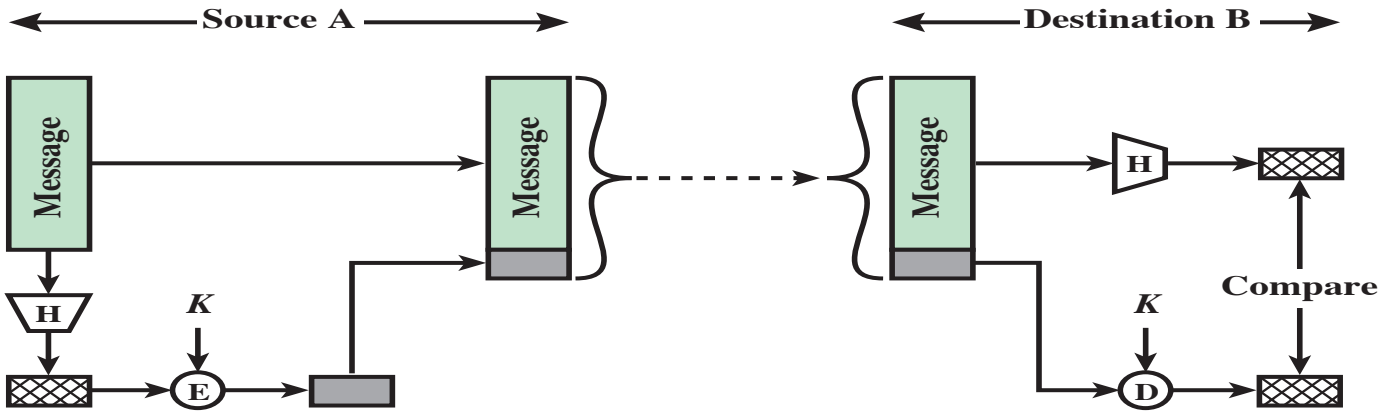
Digital Signature Using Sender's Private Key

(availability – not primary motivation now, provided the approach is feasible)

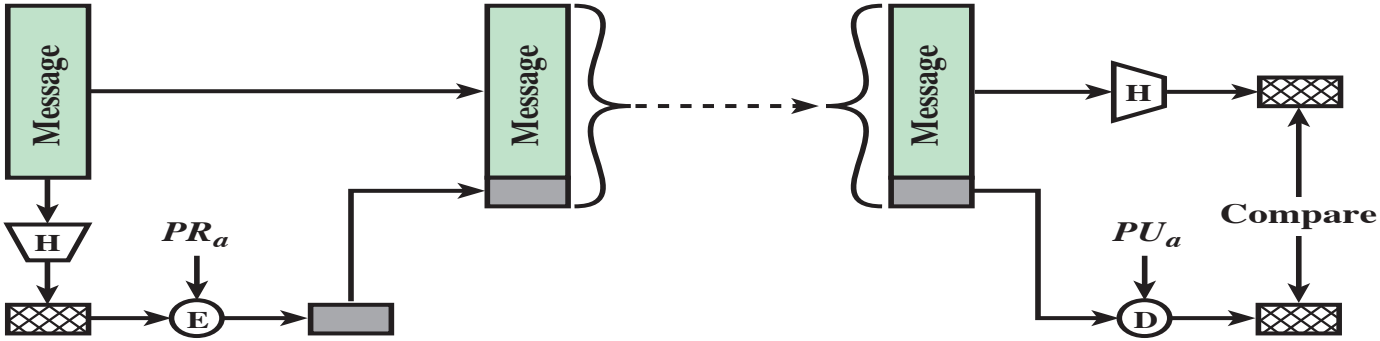


P, L = padding plus length field

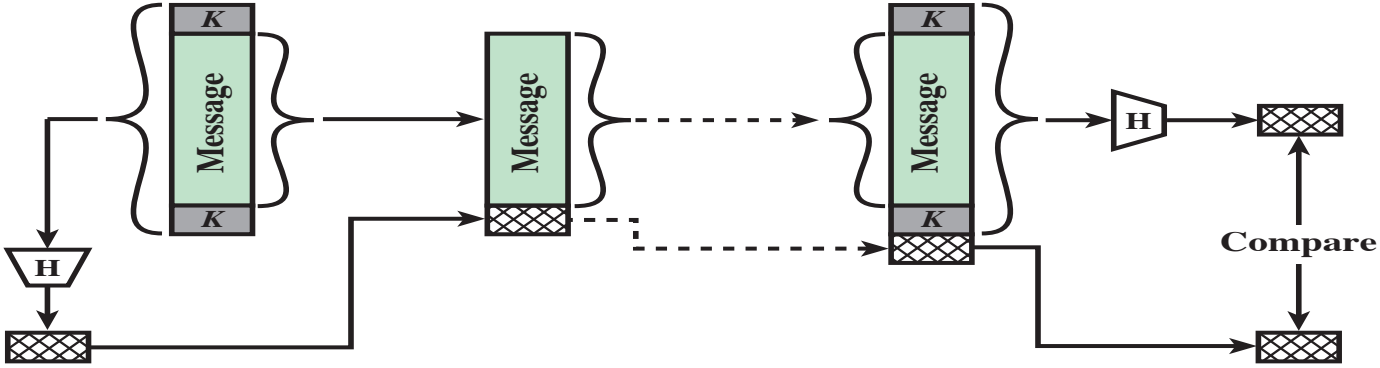
Figure 2.4 Cryptographic Hash Function; $h = H(M)$



(a) Using symmetric encryption



(b) Using public-key encryption



(c) Using secret value

Figure 2.5 Message Authentication Using a One-Way Hash Function.

To be useful for message authentication, a hash function H must have the following properties:

- Can be applied to a block of data of any size
- Produces a fixed-length output
- H(x) is relatively easy to compute for any given x
- One-way or pre-image resistant
  - Computationally infeasible to find x such that H(x) = h
- Computationally infeasible to find y ≠ x such that H(y) = H(x)
- Collision resistant or strong collision resistance
  - Computationally infeasible to find any pair (x,y) such that H(x) = H(y)

Security of Hash Functions

There are two approaches to attacking a secure hash function:

- **Cryptanalysis**
  - Exploit logical weaknesses in the algorithm
- **Brute-force attack**
  - Strength of hash function depends solely on the length of the hash code produced by the algorithm

SHA most widely used hash algorithm

Additional secure hash function applications:

- **Passwords**
  - Hash of a password is stored by an operating system
- **Intrusion detection**
  - Store H(F) for each file on a system and secure the hash values

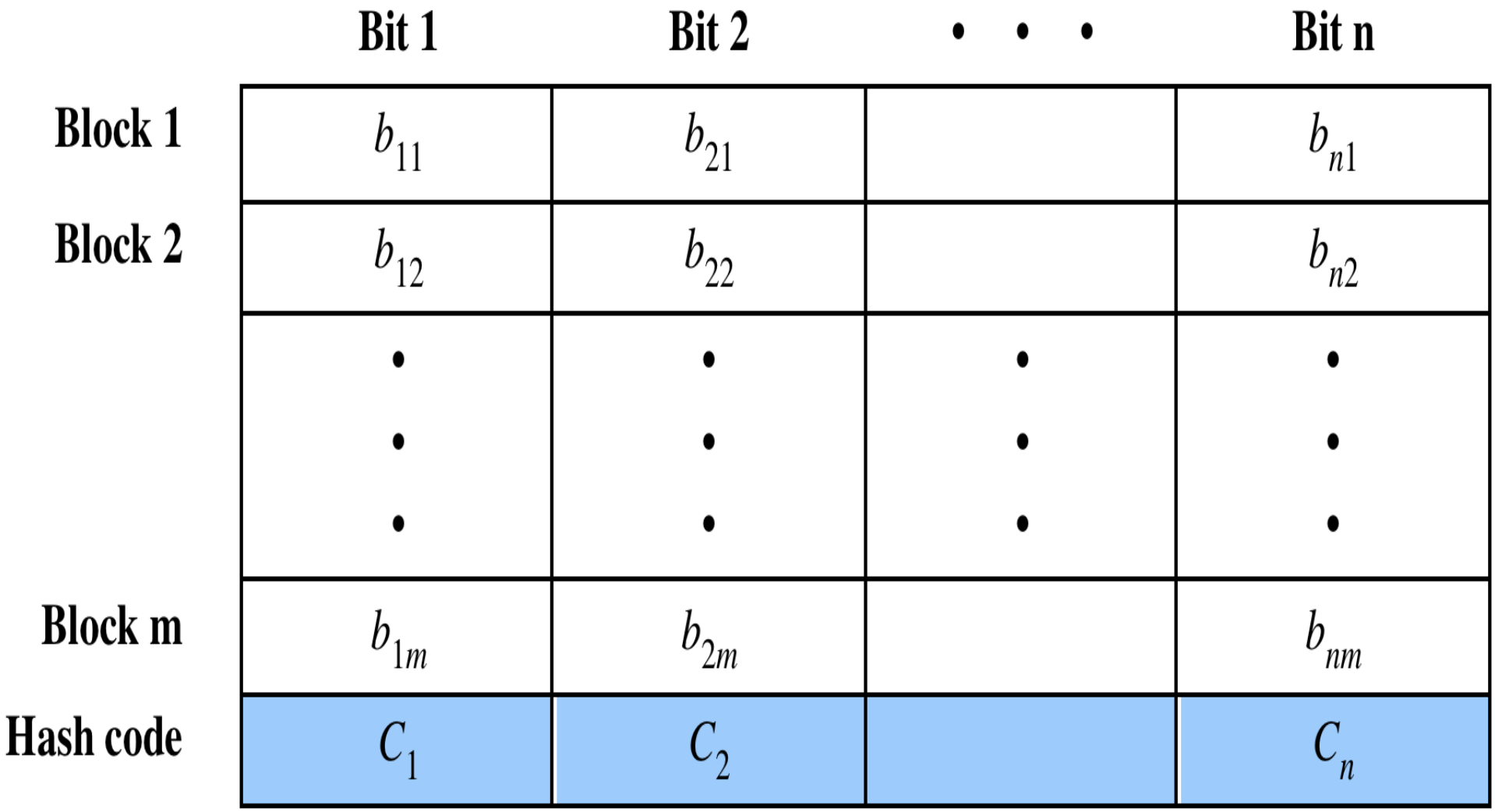


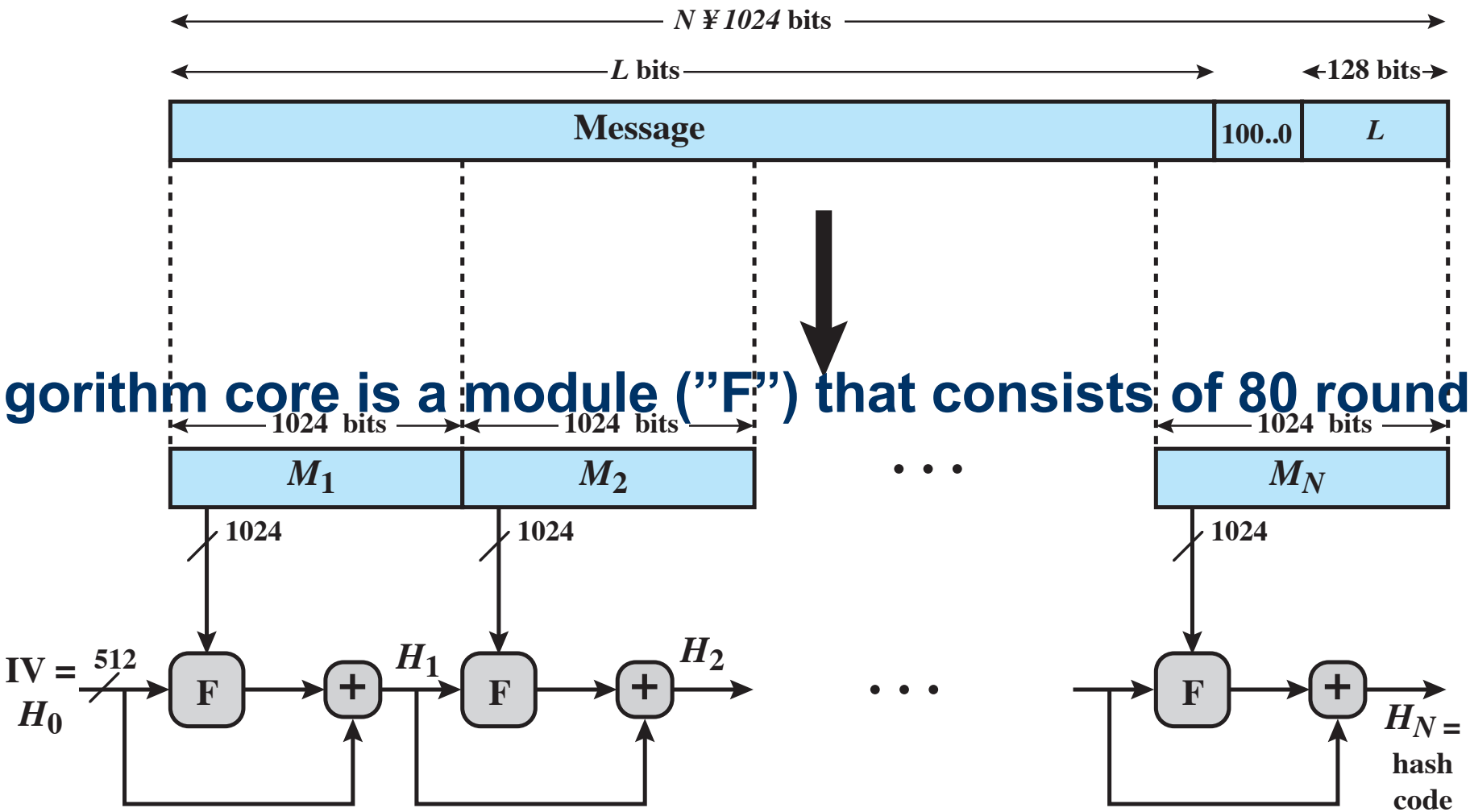
Figure 21.1 Simple Hash Function Using Bitwise XOR

Parity Bit – Effective against classes of errors, not effective against adversaries

Secure Hash Algorithm (SHA)

- SHA was originally developed by NIST
- Published as FIPS 180 in 1993
- Was revised in 1995 as SHA-1
  - Produces 160-bit hash values
- NIST issued revised FIPS 180-2 in 2002
  - Adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
  - With 256/384/512-bit hash values
  - Same basic structure as SHA-1 but greater security
- The most recent version is FIPS 180-4 which added two variants of SHA-512 with 224-bit and 256-bit hash sizes

Append padding bits and length, process 1024-bit blocks

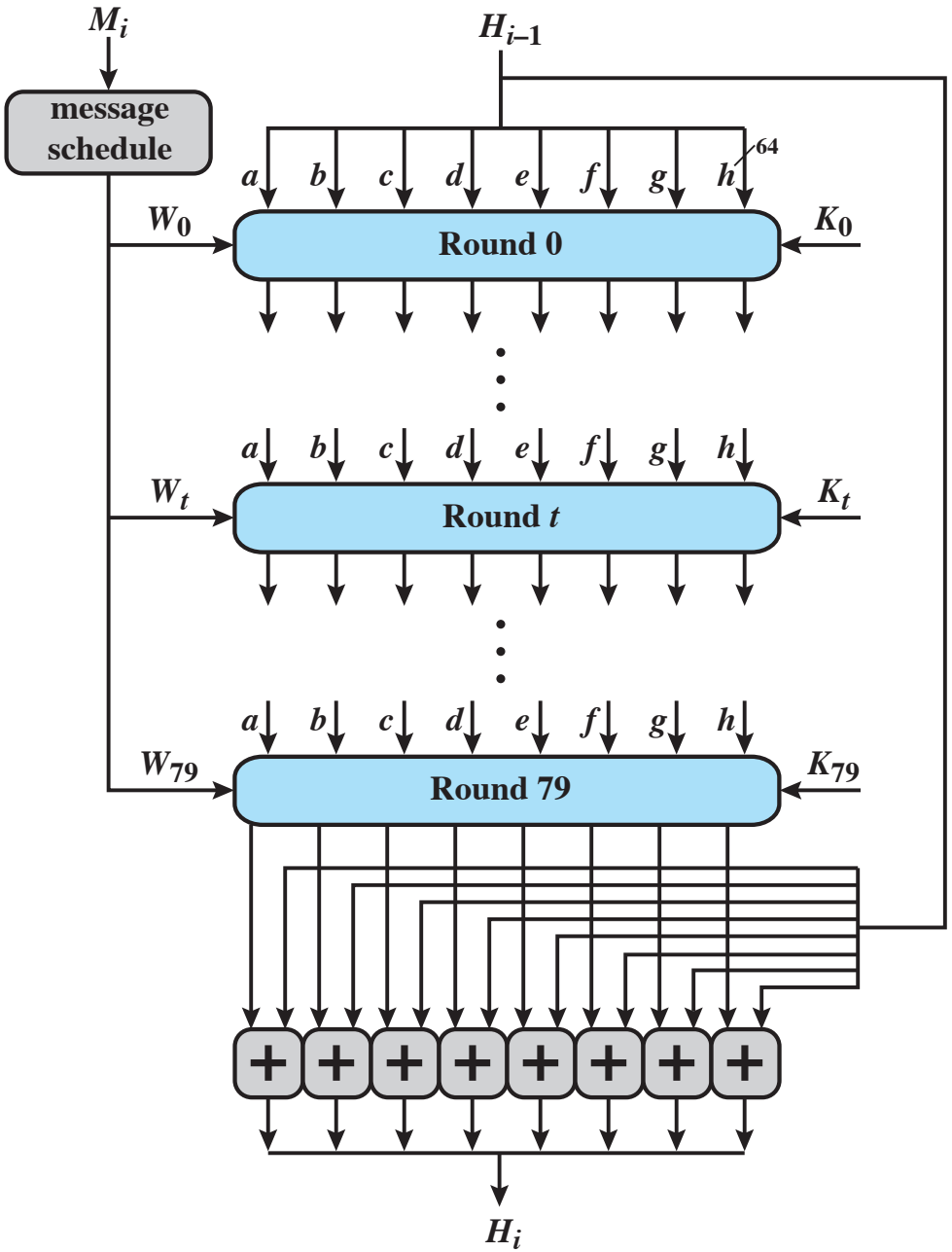


Algorithm core is a module (”F”) that consists of 80 rounds

512-bit output after all N 1024-bit blocks have been processed

+ = word-by-word addition mod 2^{64}

Figure 21.2 Message Digest Generation Using SHA-512



Basic Logic Underlying Module “F” From Previous Slide

Comparison of SHA Parameters

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512	SHA-512/224	SHA-512/256
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$	$< 2^{128}$	$< 2^{128}$
Word size	32	32	32	64	64	64	64
Block size	512	512	512	1024	1024	1024	1024
Message digest size	160	224	256	384	512	224	256
Number of steps	80	64	64	80	80	80	80
Security	80	112	128	192	256	112	128

SHA-3

- SHA-2 shares same structure and mathematical operations as its predecessors and causes concern
- Due to time required to replace SHA-2 should it become vulnerable, NIST announced in 2007 a competition to produce SHA-3

Requirements:
• Must support hash value lengths of 224, 256,384, and 512 bits • Algorithm must process small blocks at a time instead of requiring the entire message to be buffered in memory before processing it

HMAC

- Interest in developing a MAC derived from a cryptographic hash code
  - Cryptographic hash functions generally execute faster
  - Library code is widely available
  - SHA-1 was not deigned for use as a MAC because it does not rely on a secret key
- Issued as RFC2014
- Has been chosen as the mandatory-to-implement MAC for IP security
  - Used in other Internet protocols such as Transport Layer Security (TLS) and Secure Electronic Transaction (SET)

HMAC Design Objectives

To use, without modifications, available hash functions

To preserve the original performance of the hash function without incurring a significant degradation

To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required

To use and handle keys in a simple way

To have a well-understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the embedded hash function

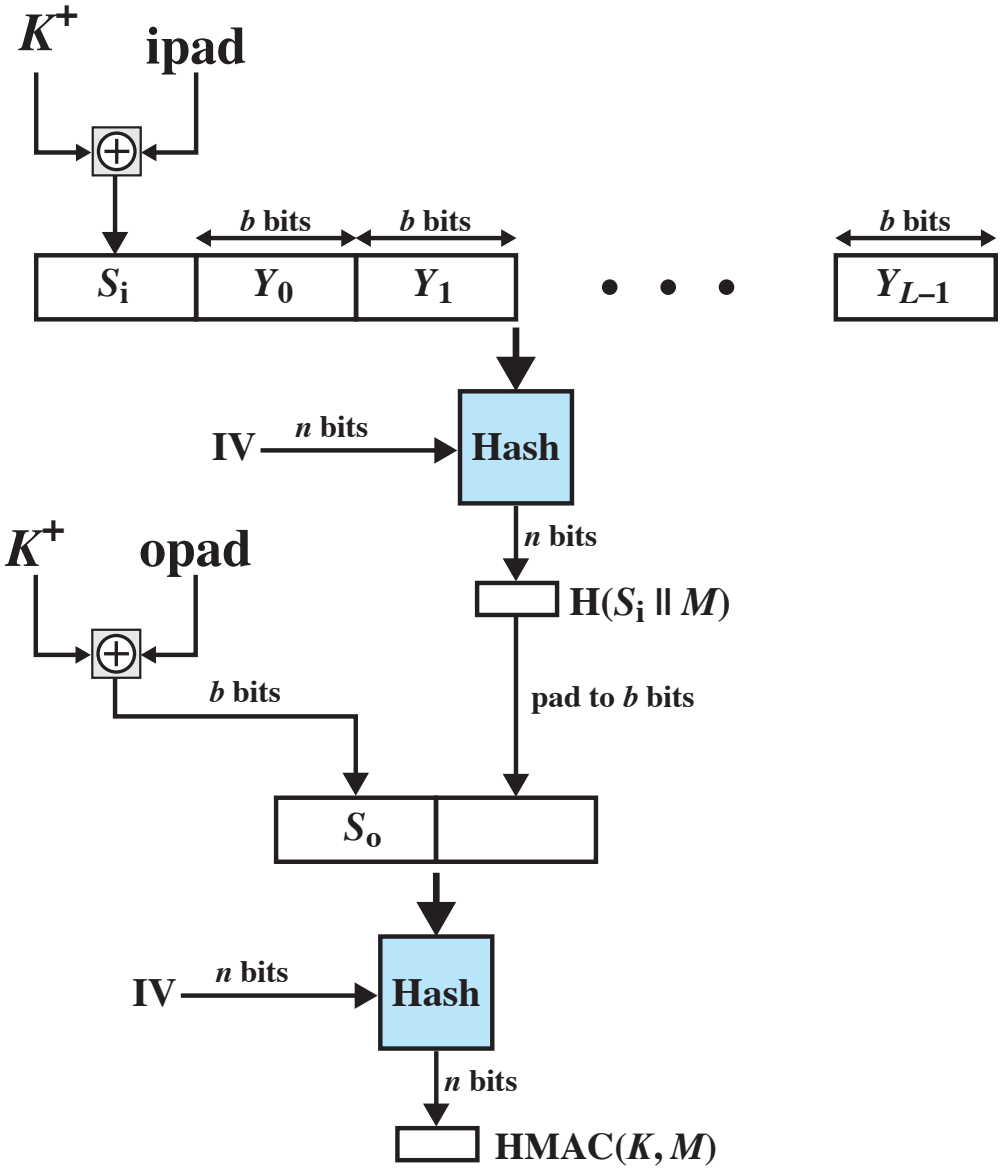


Figure 21.4 HMAC Structure

Security of HMAC

- Security depends on the cryptographic strength of the underlying hash function
- The appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC
- For a given level of effort on messages generated by a legitimate user and seen by the attacker, the probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded hash function:
  - The attacker is able to compute an output of the compression function even with an IV that is random, secret, and unknown to the attacker
  - The attacker finds collisions in the hash function even when the IV is random and secret

Random Numbers

Uses include generation of:

- Keys for public-key algorithms
- Stream key for symmetric stream cipher
- Symmetric key for use as a temporary session key or in creating a digital envelope
- Handshaking to prevent replay attacks
- Session key

Random Number Requirements

Randomness

- Criteria:
  - Uniform distribution
    - Frequency of occurrence of each of the numbers should be approximately the same
  - Independence
    - No one value in the sequence can be inferred from the others

Unpredictability

- Each number is statistically independent of other numbers in the sequence
- Opponent should not be able to predict future elements of the sequence on the basis of earlier elements

Random versus Pseudorandom

Cryptographic applications typically make use of algorithmic techniques for random number generation

- Algorithms are deterministic and therefore produce sequences of numbers that are not statistically random

Pseudorandom numbers are:

- Sequences produced that satisfy statistical randomness tests
- Likely to be predictable

True random number generator (TRNG):

- Uses a nondeterministic source to produce randomness
- Most operate by measuring unpredictable natural processes
  - e.g. radiation, gas discharge, leaky capacitors
- Increasingly provided on modern processors

Practical Application: Encryption of Stored Data

Common to encrypt transmitted data

Much less common for stored data

- There is often little protection beyond domain authentication and operating system access controls
- Data are archived for indefinite periods
- Even though erased, until disk sectors are reused data are recoverable

Approaches to encrypt stored data:

- Use a commercially available encryption package
- Back-end appliance
- Library based tape encryption
- Background laptop/PC data encryption

Timing Attacks and Side Channels

- Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages
- Timing attacks are applicable not just to RSA, but also to other public-key cryptography systems
- This attack is alarming for two reasons:
  - It comes from a completely unexpected direction
  - It is a ciphertext-only attack

Motivating Problem Solved

Confidentiality: how could we encrypt a message? (email, web/http, sms, etc)

Apply AES – Select Key Size and Mode

Or Encrypt Using Receiver's Public Key

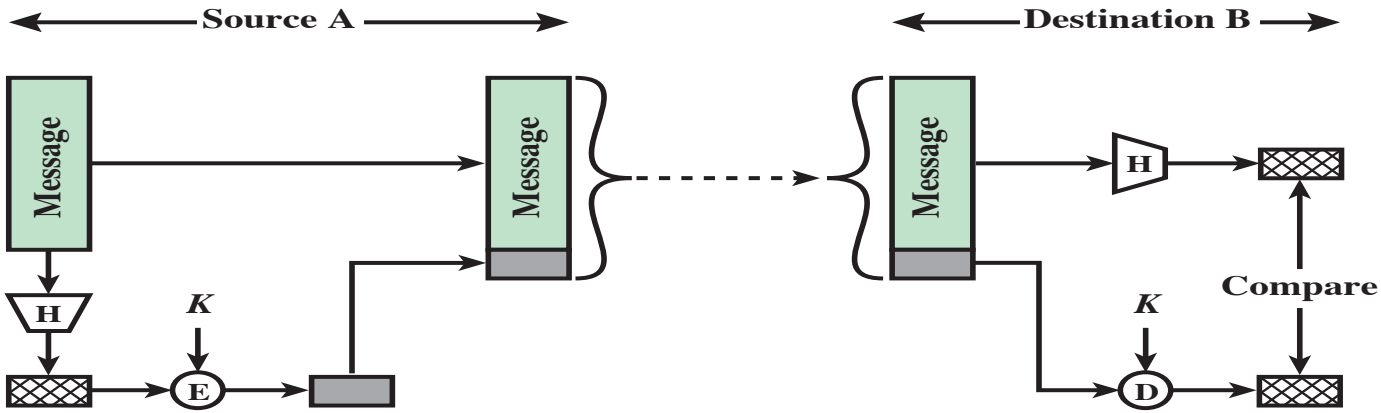
Integrity: how could we authenticate a message?

Message Authentication Code with Symmetric Key

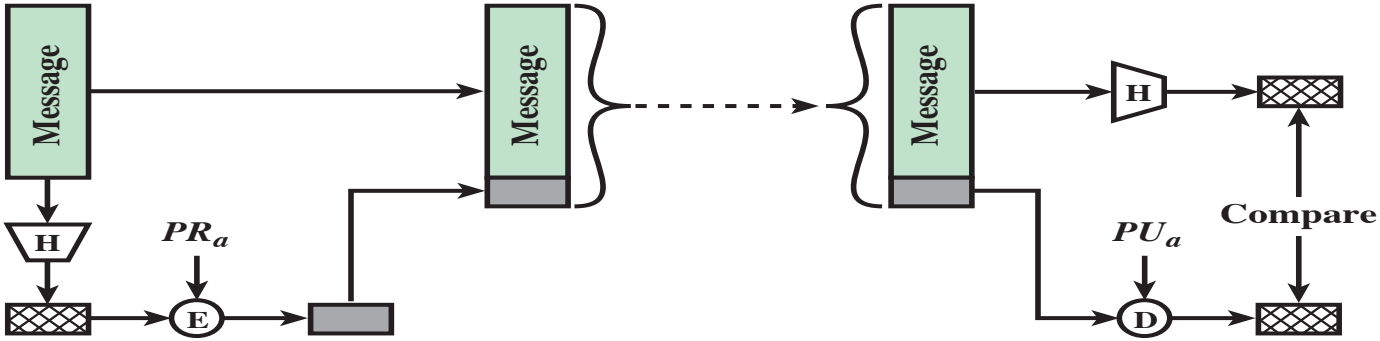
Or Digital Signature Using Sender's Private Key

Or Secret Value Appended Prior to Secure Hash

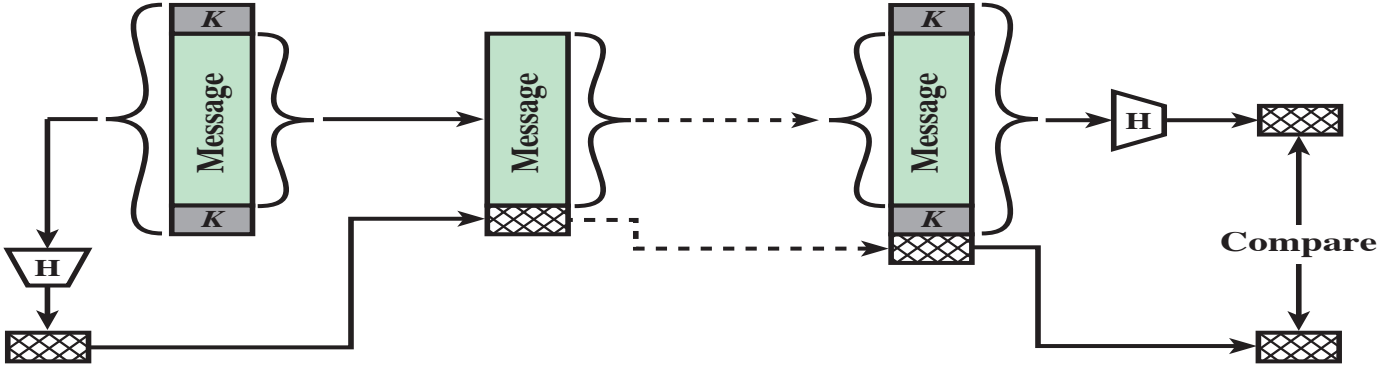
(availability – not primary motivation now,
provided the approach is feasible)



(a) Using symmetric encryption



(b) Using public-key encryption



(c) Using secret value

Figure 2.5 Message Authentication Using a One-Way Hash Function.

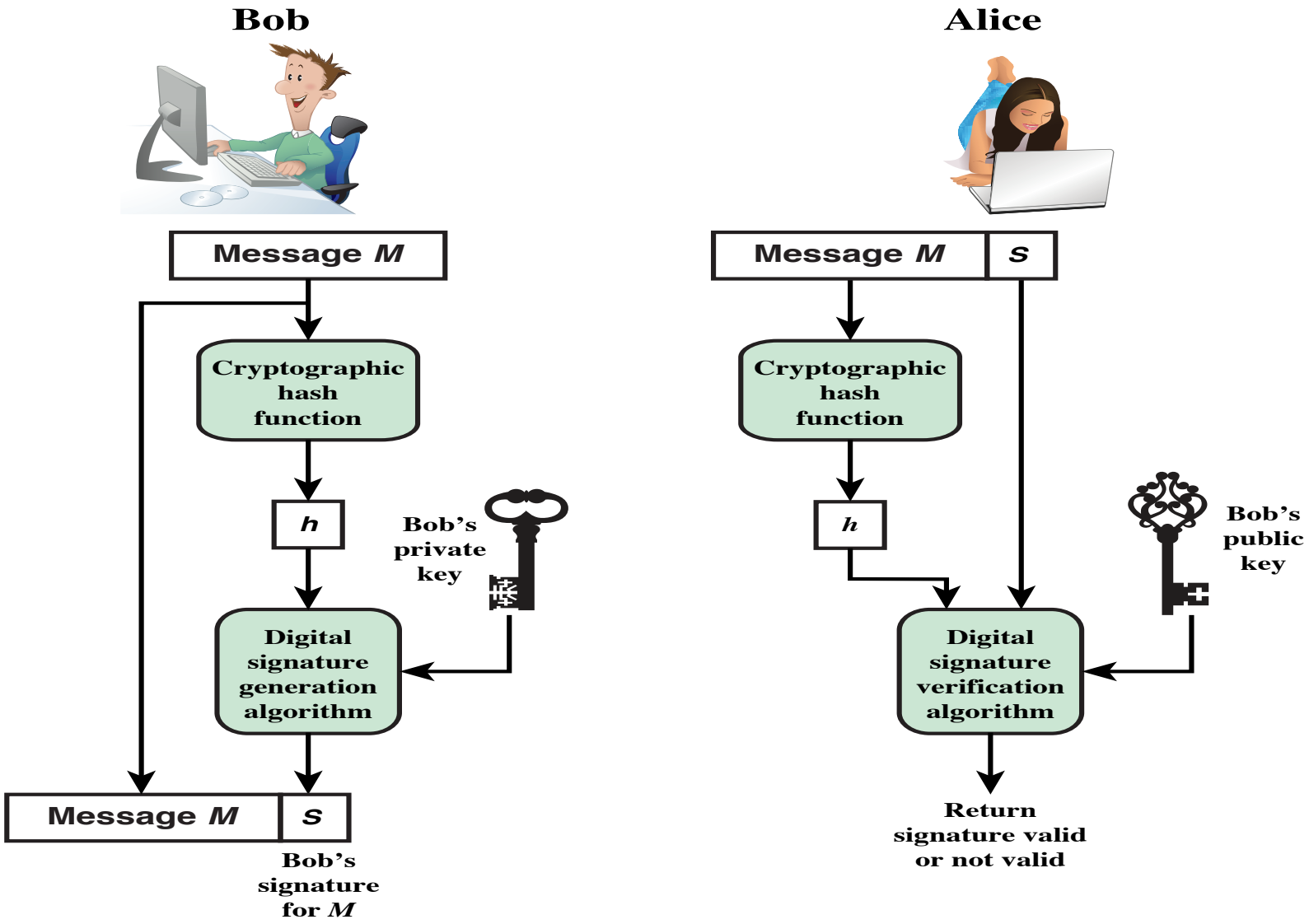
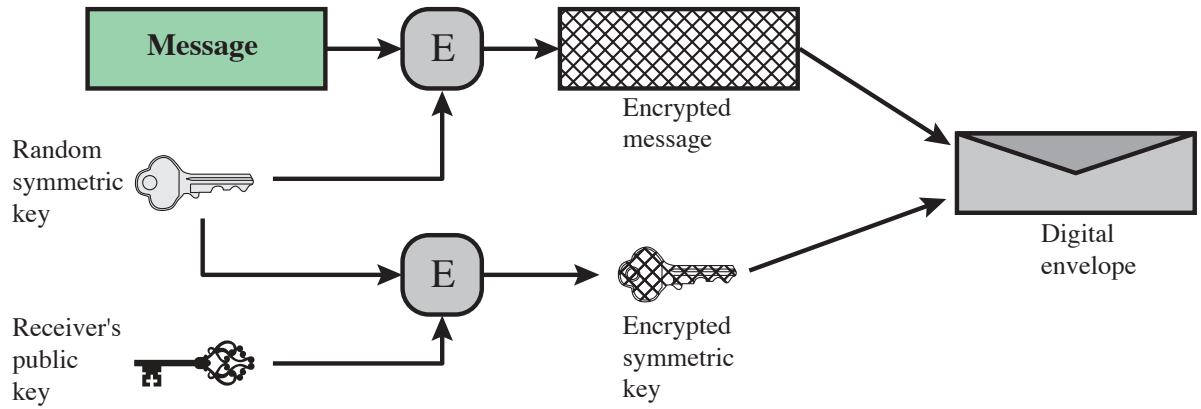
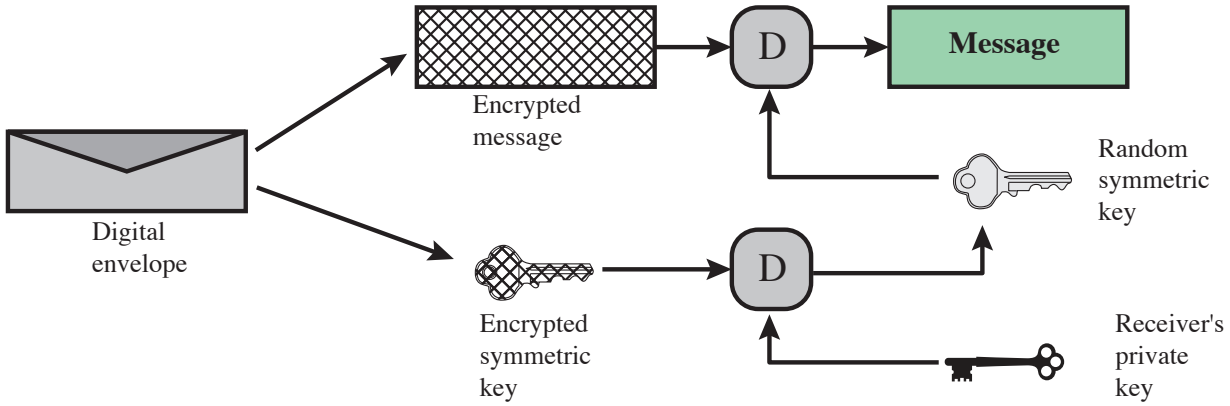


Figure 2.7 Simplified Depiction of Essential Elements of Digital Signature Process



(a) Creation of a digital envelope



(b) Opening a digital envelope

Figure 2.9 Digital Envelopes

Chapter 2 (+ 20/21) Summary

- Confidentiality with symmetric encryption
  - Symmetric encryption
  - Symmetric block encryption algorithms
  - Stream ciphers
- Message authentication and hash functions
  - Authentication using symmetric encryption
  - Message authentication without message encryption
  - Secure hash functions
  - Other applications of hash functions
- Random and pseudorandom numbers
  - The use of random numbers
  - Random versus pseudorandom
- Public-key encryption
  - Structure
  - Applications for public-key cryptosystems
  - Requirements for public-key cryptography
  - Asymmetric encryption algorithms
- Digital signatures and key management
  - Digital signature
  - Public-key certificates
  - Symmetric key exchange using public-key encryption
  - Digital envelopes

Heilmeier Questions

- What are you trying to do? Articulate your objectives using absolutely no jargon.
- How is it done today, and what are the limits of current practice?
- What is new in your approach and why do you think it will be successful?
- Who cares? If you succeed, what difference will it make?
- What are the risks?
- How much will it cost?
- How long will it take?
- What are the mid-term and final “exams” to check for success?