ECE 4309

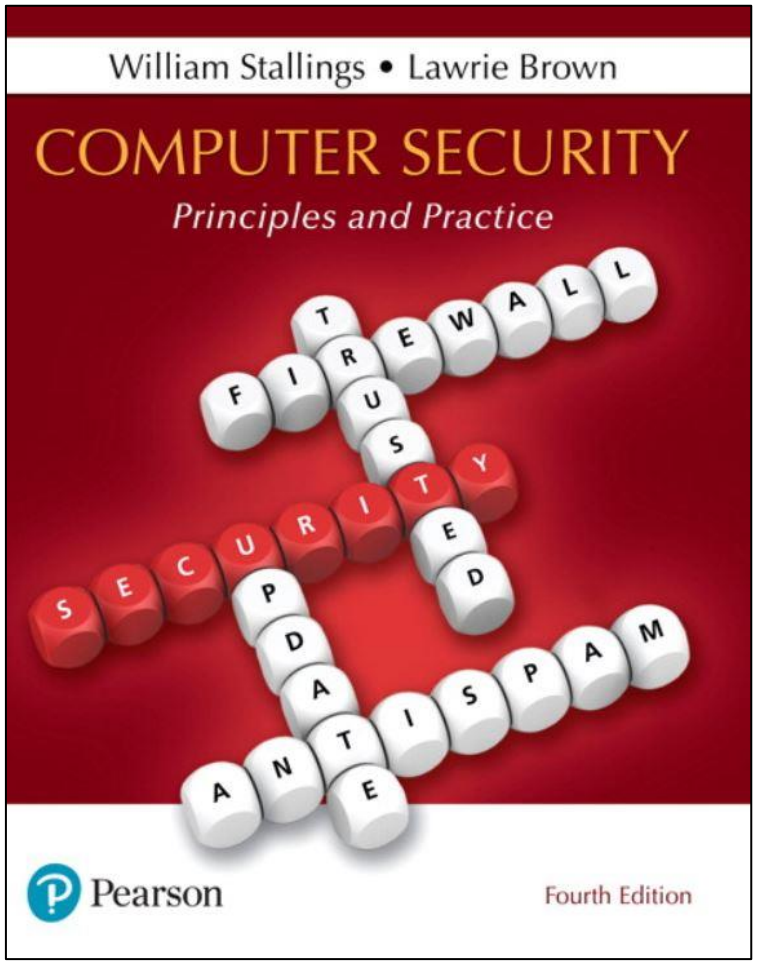
Basics of cryptography – part 2 – MAC and hash functions

Dr. Valerio Formicola

CalPolyPomona

Computer Security: Principles and Practice

Fourth Edition



Chapter 2 and 21

Cryptographic Tools

Pearson

Message authentication and hash functions

CalPolyPomona

Message encryption alone cannot provide authentication

- Merely encrypting a message content, does not guarantee that the message is generated by an authentic source.
  - It's only for **confidentiality**
- A malicious actor might intercept the sequence of blocks in ECB encrypted by legitimate user and retransmit the blocks in a different order, hence changing the meaning of the message.
  - (aka, *message reordering attack*)
  - In general, we need more protection against *active attacks*

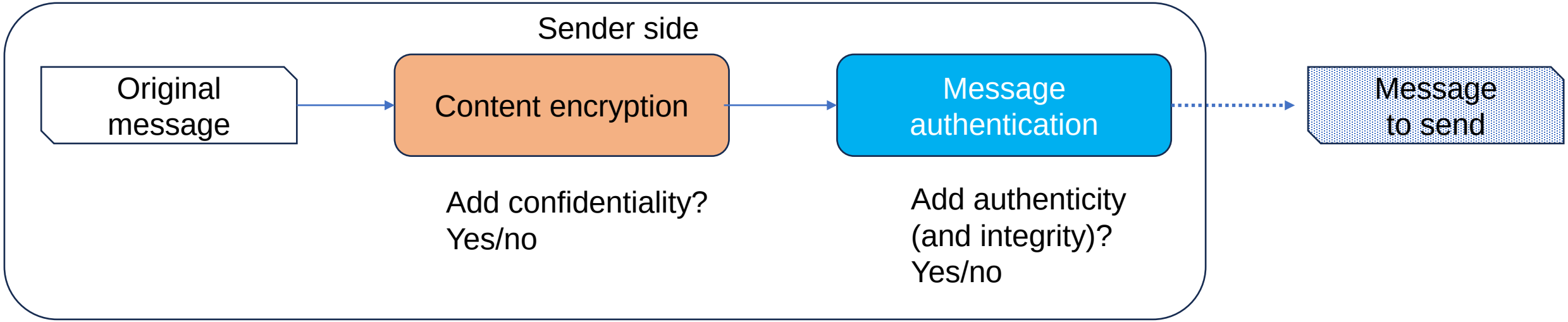
Message Authentication

- It's a mechanism to guarantee a message is integer (i.e., not altered) and authentic:
  - Authenticity: data comes from authentic source
  - Integrity: we guarantee that contents of message/data have not been altered
- As part of the Integrity, we also add information that indicates a message is:
  - Timely and in correct sequence

- How do we transmit this information? In a *Message Tag* preceding or following the original message

Observation: Combining Authentication Vs Confidentiality as a functions

- Content encryption and Message authentication/integrity can work as separate functions
- Multiple combinations are possible and for each, cryptography can be symmetric or asymmetric:
  1. Content Encryption (No) – Authentication (No)
  2. Content Encryption (Yes) – Authentication (No)
  3. Content Encryption (No) – Authentication (Yes)
  4. Content Encryption (Yes) – Authentication (Yes)



Case 1 (not used): Message authentication as effect of content symmetric encryption

- Sender and receiver are the only ones having the key and they know each other.
  - Apparently, we might achieve authenticity this way
  - However, …
- Pure encryption of messages does not guarantee:
  - Protection from ***reordering attacks***: the attacker simply stays in the middle of a communication and changes the sequence of data messages (1-2-3 -> 3-1-2)
  - Protection from ***replay/delay attacks***: the attacker replays packets even if encrypted, to generate the same data at later time

- In practice: symmetric encryption alone is not a suitable tool for data authentication

Case 2: Message authentication without content encryption (i.e., no confidentiality)

- We guarantee the message is sent from authentic source and it's not altered, but we don't encrypt the content of the message
  - Only authenticity and integrity, but not confidentiality

- Situations in which message authentication without confidentiality may be preferable include:
  - There are several applications in which the same message is broadcast to several destinations, like alarms or public messages from an authoritative source (e.g., police department)
  - An exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages
    - Only a few messages are encrypted in the content, most likely the ones that contain more critical information
  - Authentication of a computer program in plaintext is an attractive service

Case 2: approaches

1. **Approach 1:** Use of plain MAC (Message Authentication Code):
   - Compute a Tag that depends on the content of original message, plus a sequence number (*anti-reordering attack protection*)
   - Encrypt the Tag using a symmetric key (shared key)
   - Append to the original message (in clear)
   - Receiver will use secret key to decrypt the encrypted MAC tag and compare to the actual MAC of the message received
2. **Approach 2:** One-way has function:
   - As before, compute a Tag but this time use a *one-way hash function* of the original message + original length information + padding
   - Encrypt the hash:
     - (2A) Using a symmetric key (shared key)
     - (2B) Using an asymmetric key (aka, public key)
   - Append to the original message (in clear)
   - Receiver will use the key (public or shared) to decrypt the encrypted Hash tag and compare to the actual Hash of message received
3. **Approach 3:** Keyed hash MAC:
   - You evaluate the Hash of a message concatenated to a secret key (shared key): H(K || M || K)
   - Append the keyed Hash tag to the original message
   - Receiver will concatenate the received message with secret key, recalculate the keyed hash and compare with received Hash tag

Approach 1: Message Authentication Using a Message Authentication Code (MAC)

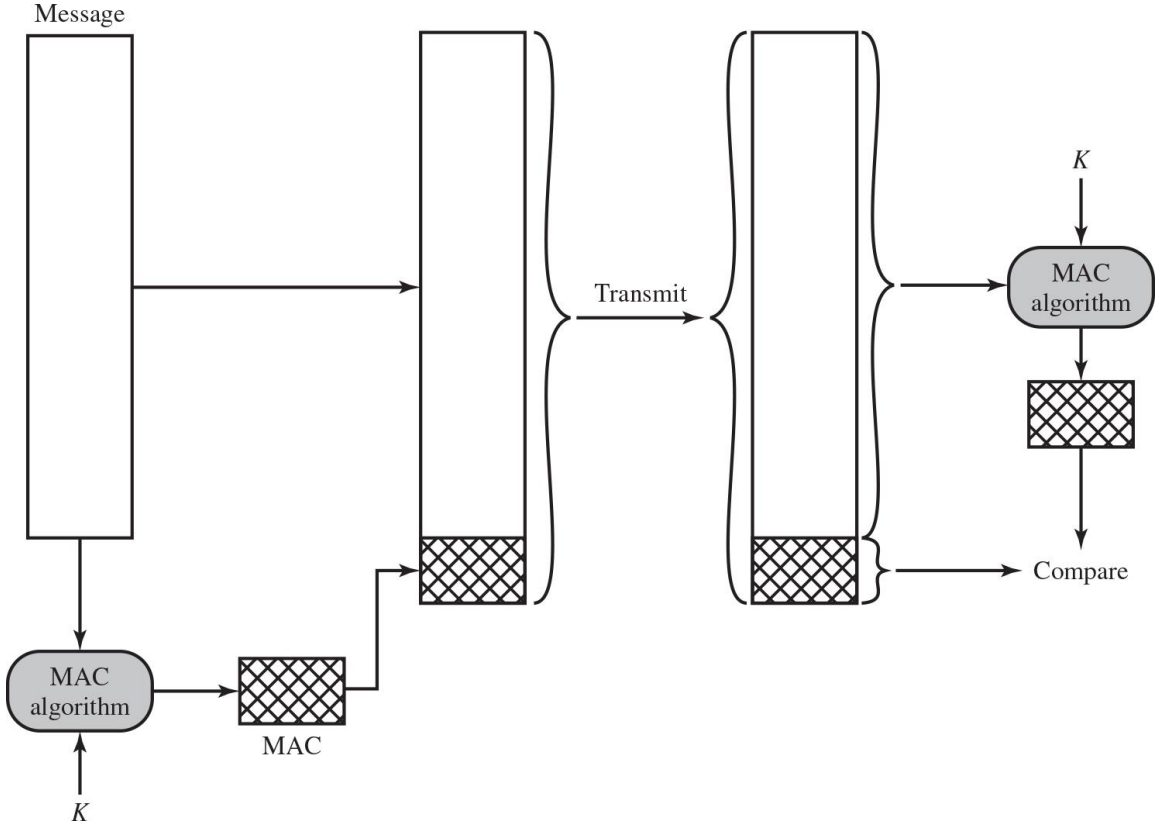
The message authentication code (MAC) is a complex function using as input 1) the message and 2) the secret key K_{AB} exchanged between sender and receiver: $MAC_M = F(K_{AB}, M)$.

MAC guarantees **integrity** AND **authentication**. Use of a key as an input to the function F guarantees **authentication**.
Use of a message content M guarantees **integrity**.

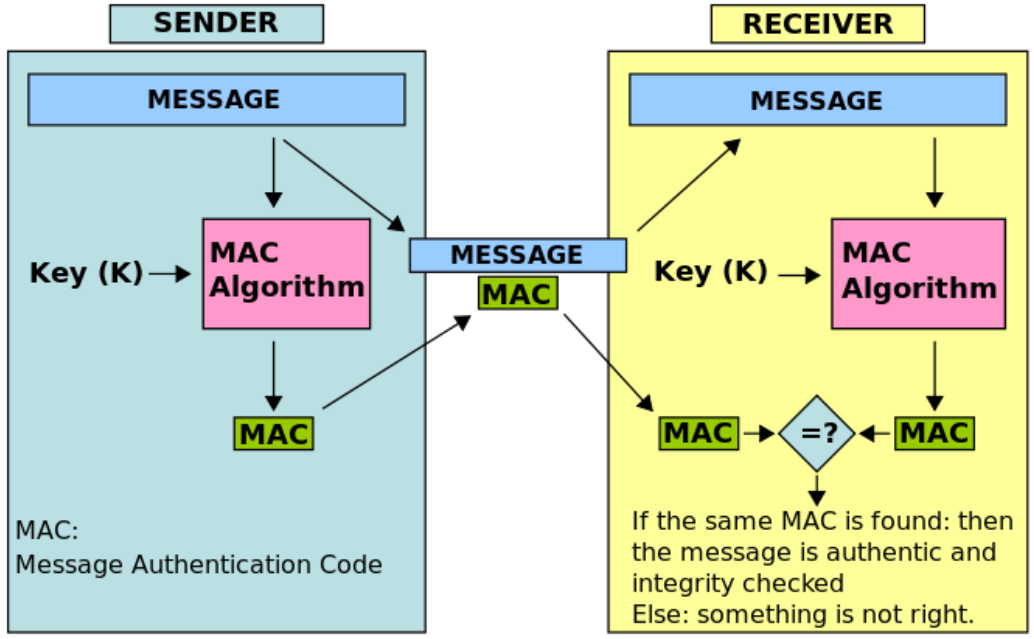
Symmetric key is still used but only for the MAC tag

1 - secret key is used to create a MAC tag (authenticity)
2 - no alteration of message is possible (integrity because nobody has secret key but legitimate ends of the communication)
3 – if the message contains a *sequence number*, any alteration of the sequence number will be detected as a integrity violation; moreover, sequence number will protect from| *replay attacks* (next slide)

Examples of common algorithms for MAC (key + message, no content encryption): CMAC, CBC-MAC.



Approach 1 (another example)



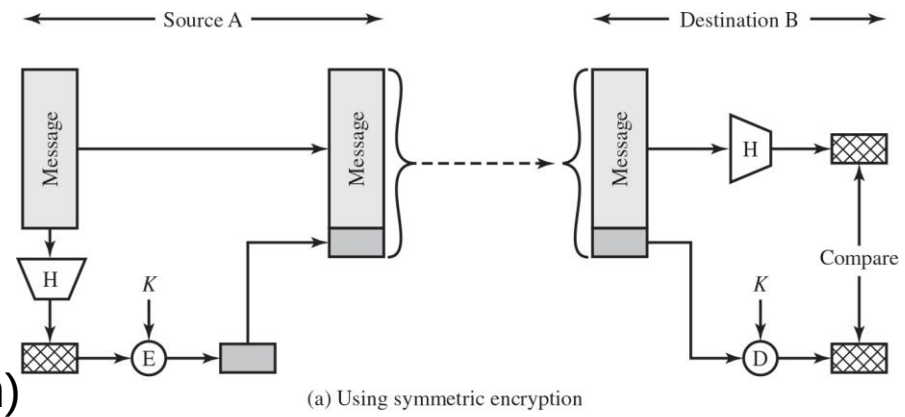
In this example, the sender of a message runs it through a MAC algorithm to produce a MAC data tag. The message and the MAC tag are then sent to the receiver. The receiver in turn runs the message portion of the transmission through the same MAC algorithm using the same key, producing a second MAC data tag. The receiver then compares the first MAC tag received in the transmission to the second generated MAC tag. If they are identical, the receiver can safely assume that the message was not altered or tampered with during transmission (data integrity).

However, to allow the receiver to be able to detect replay attacks, the message itself must contain data that assures that this same message can only be sent once (e.g. time stamp, sequence number or use of a one-time MAC). Otherwise an attacker could – without even understanding its content – record this message and play it back at a later time, producing the same result as the original sender.

Approach 2 (A/B) and Approach 3: Message Authentication Using a One-Way Hash Function

One-way hash functions
aka
Secure Hash functions

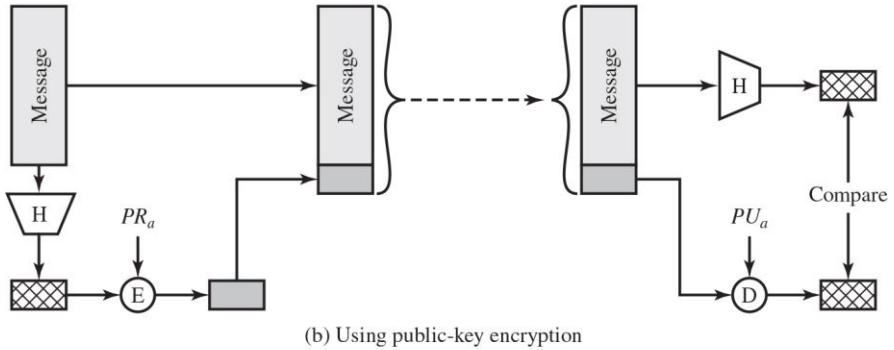
Approach 2A
(Symmetric key for encryption)



(a) Using symmetric encryption

Approach 2B
(Public key for encryption)

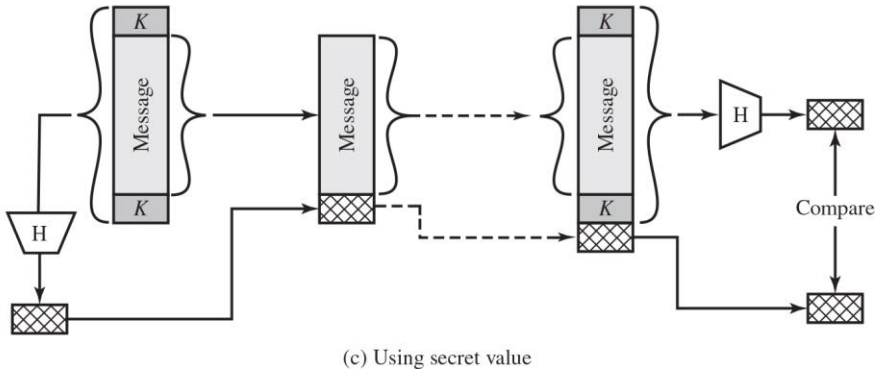
Public key on hash is used for digital signatures



(b) Using public-key encryption

Approach 3
(Keyed-Hash MAC)

Keyed-Hash avoid encryption completely



(c) Using secret value

Secure Hash Functions: To Be Useful for Message Authentication, a Hash Function H Must Have the Following Properties:

1. Can be applied to a block of data of any size
2. Produces a fixed-length output
3. H(x) is relatively easy to compute for any given *x*
4. **One-way** or **pre-image resistant**
   - Computationally infeasible to find *x* such that H(x) = h, i.e., nobody should be able to find the inverse function of H because, otherwise $H^{-1}(y) = x$;
     x might be K || M || K and the key K would be revealed if you also have M
5. **Second pre-image resistant** or ***weak* collision resistant**
   - Computationally infeasible to find y≠ x such that H(y) = H(x)
   - Note: here you get the hash of a message, but you cannot create another one that has the same hash of the first (*anti-forgery protection*)
6. **Collision resistant or *strong* collision resistance**
   - Computationally infeasible to find any pair (*x*, *y*) such that H(y) = H(x)
   - Note: Here you don't have any starting message; the property states you should not be able to create two messages with the same hash

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
- MD5 which generates 128 bit hash, has been found to be breakable and it's not secure anymore
- SHA most widely used hash algorithm family. Currently SHA-2 (256. 384, 512 bit hashes) are the most used and SHA-3 will be in the future
- Additional secure hash function applications:
  - Passwords
    - Hash of a password is stored by an operating system
  - Intrusion detection
    - Store H(File) for each file on a system and secure the hash values

In summary

- MAC function:
  - Calculates an output Tag from the message in combination with a key
  - Hence provides integrity and authenticity
- One-way hash function or Secure-hash function:
  - message Tag is calculated only using the message content, not a key
  - Hence, it provides integrity but not authenticity
  - You need to combine with encryption for authenticity (symmetric or public encryption) to obtain authenticity
  - It has applicable to any blocks size
    - In contrast to stream ciphers that cannot be applied to blocks shorter than the key size
  - Generates the same output, whatever the input size
  - It's not computationally stressful
  - It can have several levels of security based on the resistance level:
    - preimage resistance: for essentially all pre-specified outputs, it is computationally infeasible to find any input that hashes to that output; i.e., given y, it is difficult to find an x such that $h(x) = y$.
    - second-preimage resistance: for a specified input, it is computationally infeasible to find another input which produces the same output; i.e., given x, it is difficult to find a second input $x' \neq x$ such that $h(x) = h(x')$.
    - collision resistance (strong resistance): it is computationally infeasible to find any two distinct inputs x, x′ that hash to the same output; i.e., such that $h(x) = h(x')$

Deeper view of secure hash functions

CalPolyPomona

Figure 21.2 Message Digest Generation Using SHA-512

SHA-2 has three versions:
SHA-256, SHA-384, SHA-512

Message + Padding
(1 to 1024 bits added:
100…0)

L = Length of the initial message
IV = initialization vector as H0

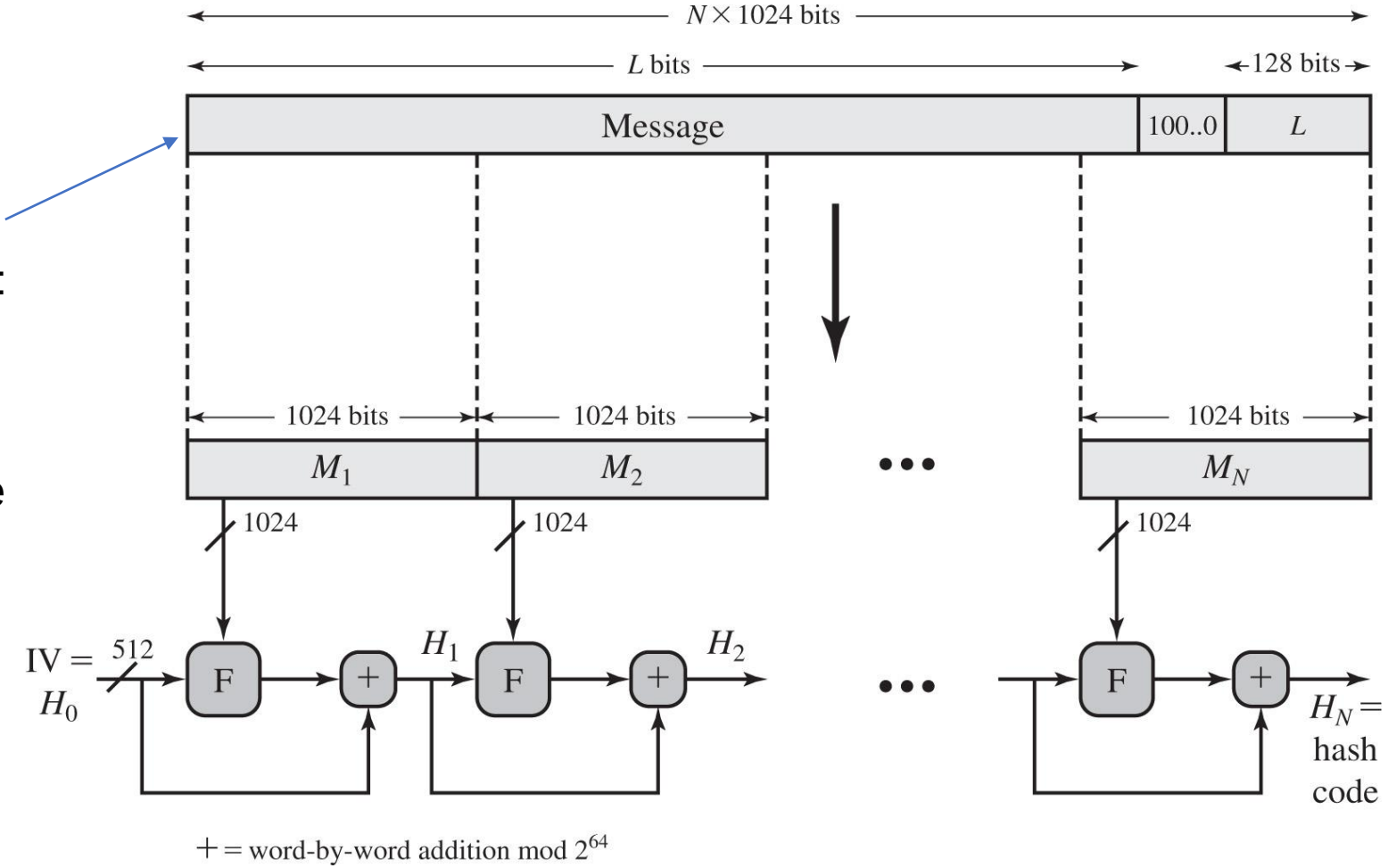


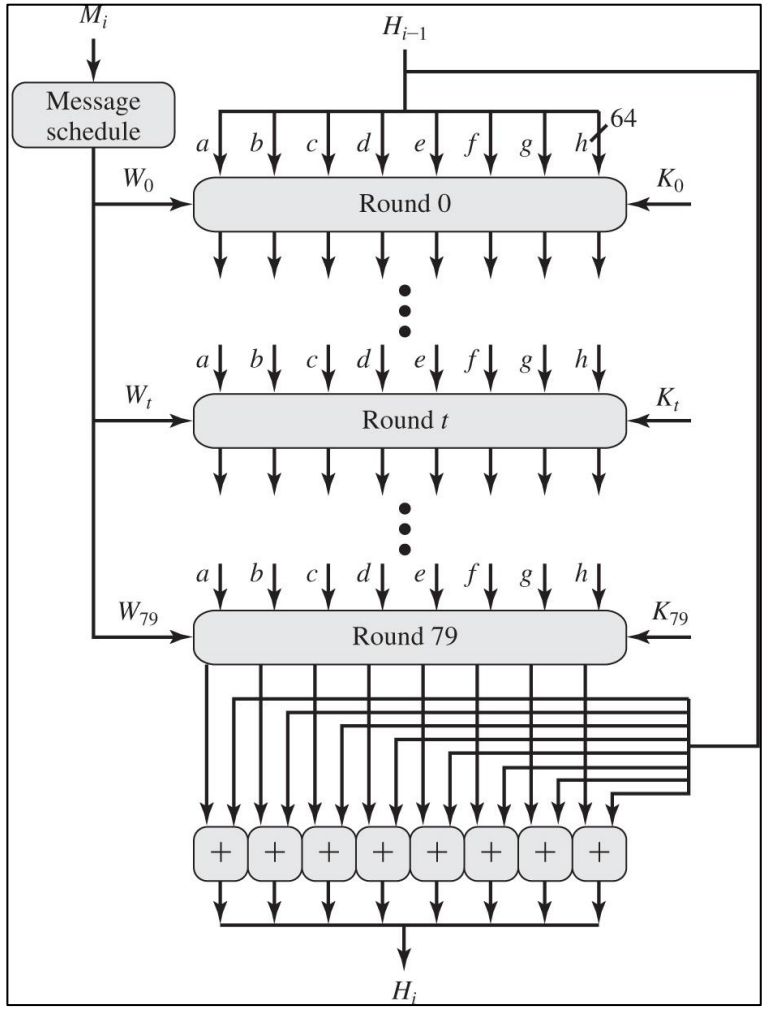
Figure 21.3 SHA-512 Processing of a Single 1024-Bit Block

W_t=64 bit data derived from the block being processed

K_t = additive constant for each round of the 80

Initialization buffer at the beginning is always the same
H0 = a|b|c|d|e|f|g|h

```
h[0..7] := 0x6a09e667f3bcc908, 0xbb67ae8584caa73b, 0x3c6ef372fe94f82b,
0xa54ff53a5f1d36f1, 0x510e527fade682d1, 0x9b05688c2b3e6c1f,
0x1f83d9abfb41bd6b, 0x5be0cd19137e2179
```



F block from previous slide

SHA-3

- SHA-2 shares same structure and mathematical operations as its predecessors and causes concern
  - SHA-512 is pretty solid so far
- 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

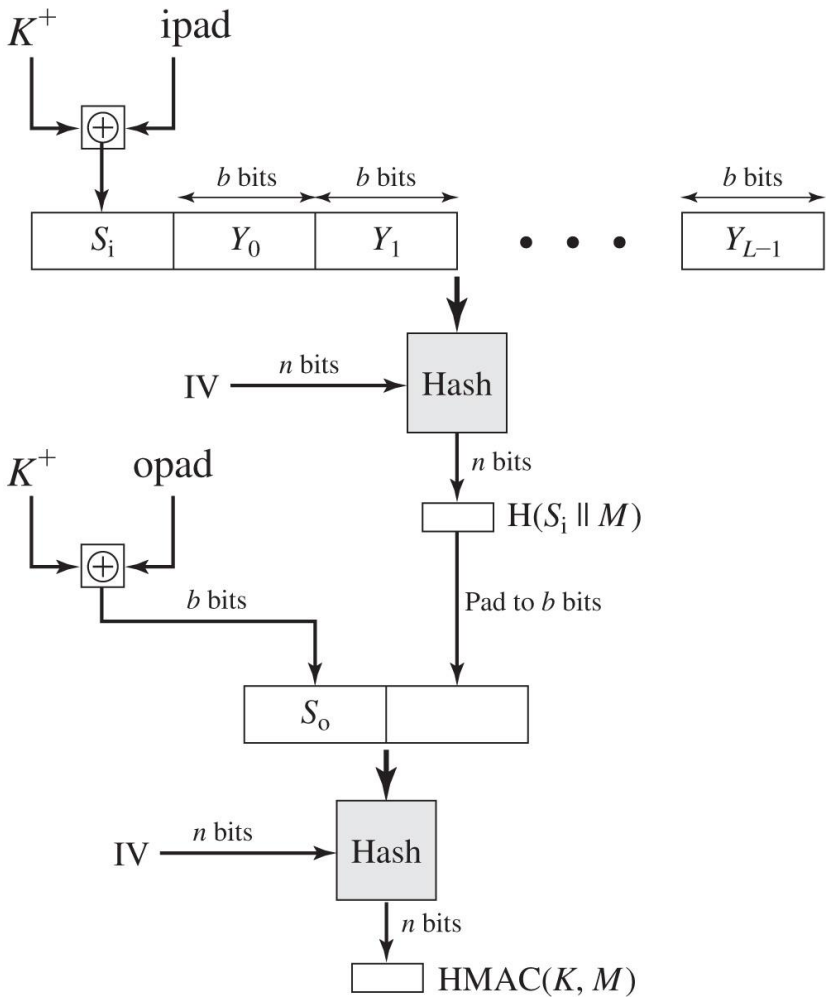
HMAC

- 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 allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required
- To preserve the original performance of the hash function without incurring a significant degradation
- 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



- H = embedded hash function (e.g., SHA)
- M = message input to HMAC (including the padding specified in the embedded hash function)
- Y_i = i-th block of M, $0 \le i \le (L-1)$
- L = number of blocks in M
- b = number of bits in a block (e.g., 512)
- 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 ≥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
- IV = initialization vector (unused)

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

Where XOR is the XOR operation between two sequences

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: same difficulty of a brute-force attack to guess a secret key with $O(2^n)$
  - The attacker finds collisions in the hash function even when the IV is random and secret. *H(M) = H(M')*. This attack is also known as *birthday attack*