# UNIT-III

Message Authentication and Hash Functions: Authentication Requirements - Authentication Functions - Message Authentication Codes - Hash Functions - Security of Hash Functions and MACs. Hash and MAC Algorithms: Secure Hash Algorithm. Digital Signatures and Authentication Protocols: Digital Signatures - Authentication Protocols - Digital Signature Standard. Authentication Applications: Kerberos -X.509 Authentication Service - Public-key infrastructure

# Message Authentication

- message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- will consider the security requirements
- then three alternative functions used:
  - message encryption
  - message authentication code (MAC)
  - hash function

# Security Requirements

- disclosure
- traffic analysis
- masquerade
- content modification
- sequence modification
- timing modification
- source repudiation
- destination repudiation

# Message Encryption

- message encryption by itself also provides a measure of authentication
- if symmetric encryption is used then:
  - receiver know sender must have created it
  - since only sender and receiver now key used
  - know content cannot of been altered
  - if message has suitable structure, redundancy or a checksum to detect any changes

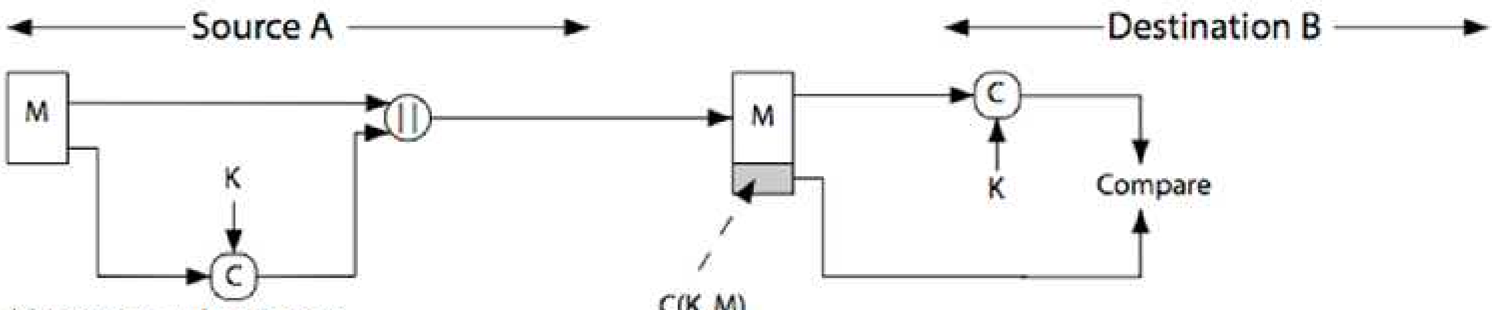
# Message Encryption

- if public-key encryption is used:
  - encryption provides no confidence of sender
  - since anyone potentially knows public-key
  - however if
    - sender **signs** message using their private-key
    - then encrypts with recipients public key
    - have both secrecy and authentication
  - again need to recognize corrupted messages
  - but at cost of two public-key uses on message

# Message Authentication Code (MAC)

- generated by an algorithm that creates a small fixed-sized block
  - depending on both message and some key
  - like encryption though need not be reversible
- appended to message as a **signature**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

# Message Authentication Code




(a) Message authentication

# Message Authentication Codes

- as shown the MAC provides authentication
- can also use encryption for secrecy
  - generally use separate keys for each
  - can compute MAC either before or after encryption
  - is generally regarded as better done before
- why use a MAC?
  - sometimes only authentication is needed
  - sometimes need authentication to persist longer than the encryption (eg. archival use)
- note that a MAC is not a digital signature

# MAC Properties

- a MAC is a cryptographic checksum

  $MAC = C_K(M)$

  – condenses a variable-length message M
  – using a secret key K
  – to a fixed-sized authenticator
- is a many-to-one function
  – potentially many messages have same MAC
  – but finding these needs to be very difficult

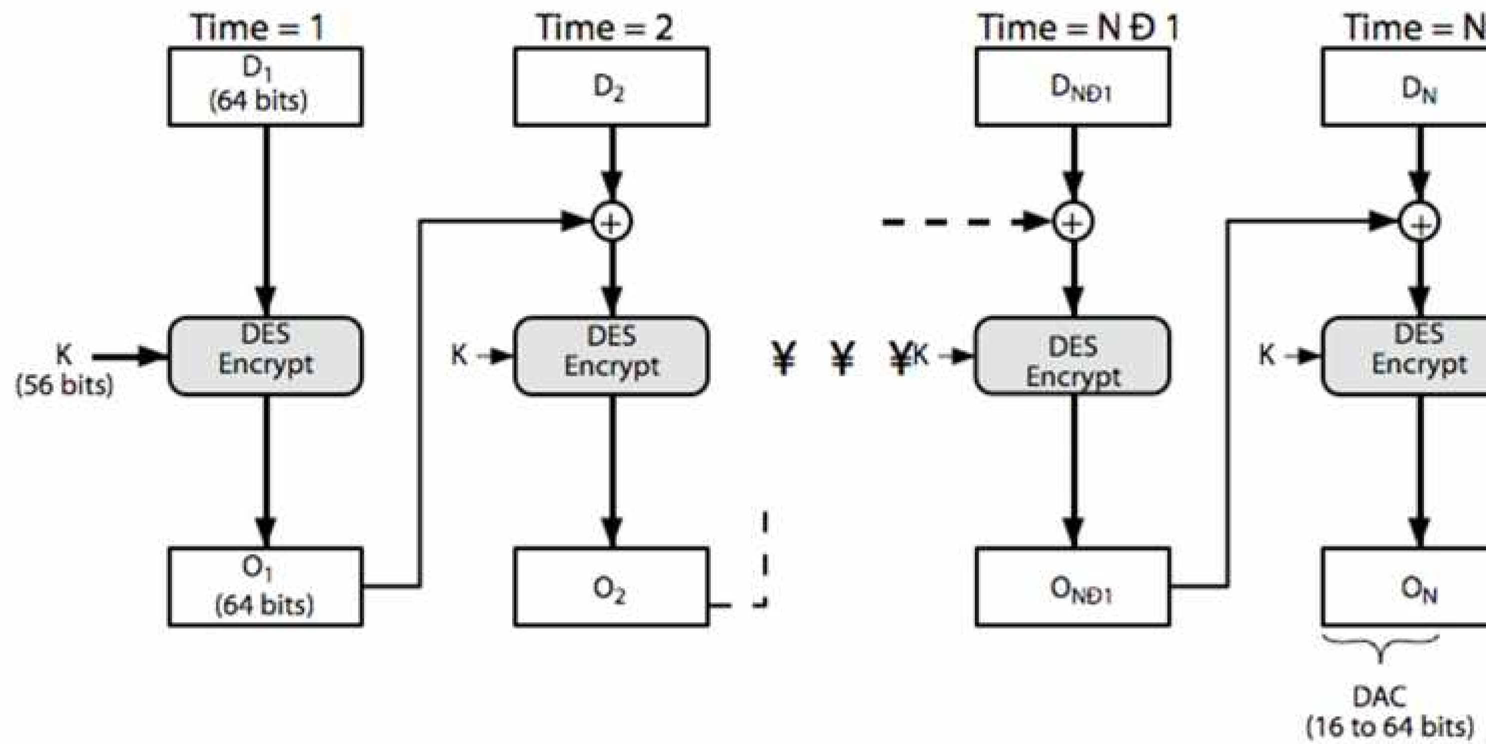
# Requirements for MACs

- taking into account the types of attacks
- need the MAC to satisfy the following:
  1. knowing a message and MAC, is infeasible to find another message with same MAC
  2. MACs should be uniformly distributed
  3. MAC should depend equally on all bits of the message

# Using Symmetric Ciphers for MACs

- can use any block cipher chaining mode and use final block as a MAC
- **Data Authentication Algorithm (DAA)** is a widely used MAC based on DES-CBC
  - using IV=0 and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final block as the MAC
    - or the leftmost M bits ($16 \le M \le 64$) of final block
- but final MAC is now too small for security

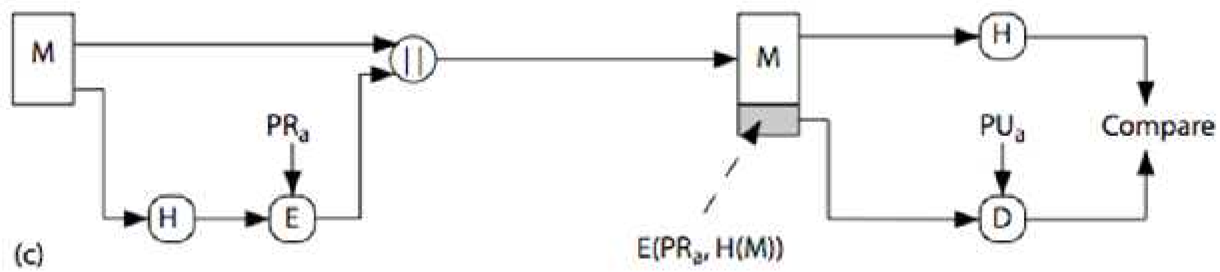
# Data Authentication Algorithm



# Hash Functions

- condenses arbitrary message to fixed size
  `h = H(M)`
- usually assume that the hash function is public and not keyed
  - cf. MAC which is keyed
- hash used to detect changes to message
- can use in various ways with message
- most often to create a digital signature

# Hash Functions & Digital Signatures



# Requirements for Hash Functions

1. can be applied to any sized message `M`
2. produces fixed-length output `h`
3. is easy to compute `h=H(M)` for any message `M`
4. given `h` is infeasible to find `x` s.t. `H(x)=h`
   - one-way property
5. given `x` is infeasible to find `y` s.t. `H(y)=H(x)`
   - weak collision resistance
6. is infeasible to find any `x,y` s.t. `H(y)=H(x)`
   - strong collision resistance

# Simple Hash Functions

- are several proposals for simple functions
- based on XOR of message blocks
- not secure since can manipulate any message and either not change hash or change hash also
- need a stronger cryptographic function (next chapter)

# Birthday Attacks

- might think a 64-bit hash is secure
- but by **Birthday Paradox** is not
- **birthday attack** works thus:
  - opponent generates $2^{m/2}$ variations of a valid message all with essentially the same meaning
  - opponent also generates $2^{m/2}$ variations of a desired fraudulent message
  - two sets of messages are compared to find pair with same hash (probability > 0.5 by birthday paradox)
  - have user sign the valid message, then substitute the forgery which will have a valid signature
- conclusion is that need to use larger MAC/hash

# Block Ciphers as Hash Functions

- can use block ciphers as hash functions
  - using $H_0$=0 and zero-pad of final block
  - compute: $H_i = E_{M_i} [H_{i-1}]$
  - and use final block as the hash value
  - similar to CBC but without a key
- resulting hash is too small (64-bit)
  - both due to direct birthday attack
  - and to "meet-in-the-middle" attack
- other variants also susceptible to attack

# Hash Functions & MAC Security

- like block ciphers have:
- **brute-force** attacks exploiting
  - strong collision resistance hash have cost $2^{m/2}$
    - have proposal for h/w MD5 cracker
    - 128-bit hash looks vulnerable, 160-bits better
  - MACs with known message-MAC pairs
    - can either attack keyspace (cf key search) or MAC
    - at least 128-bit MAC is needed for security

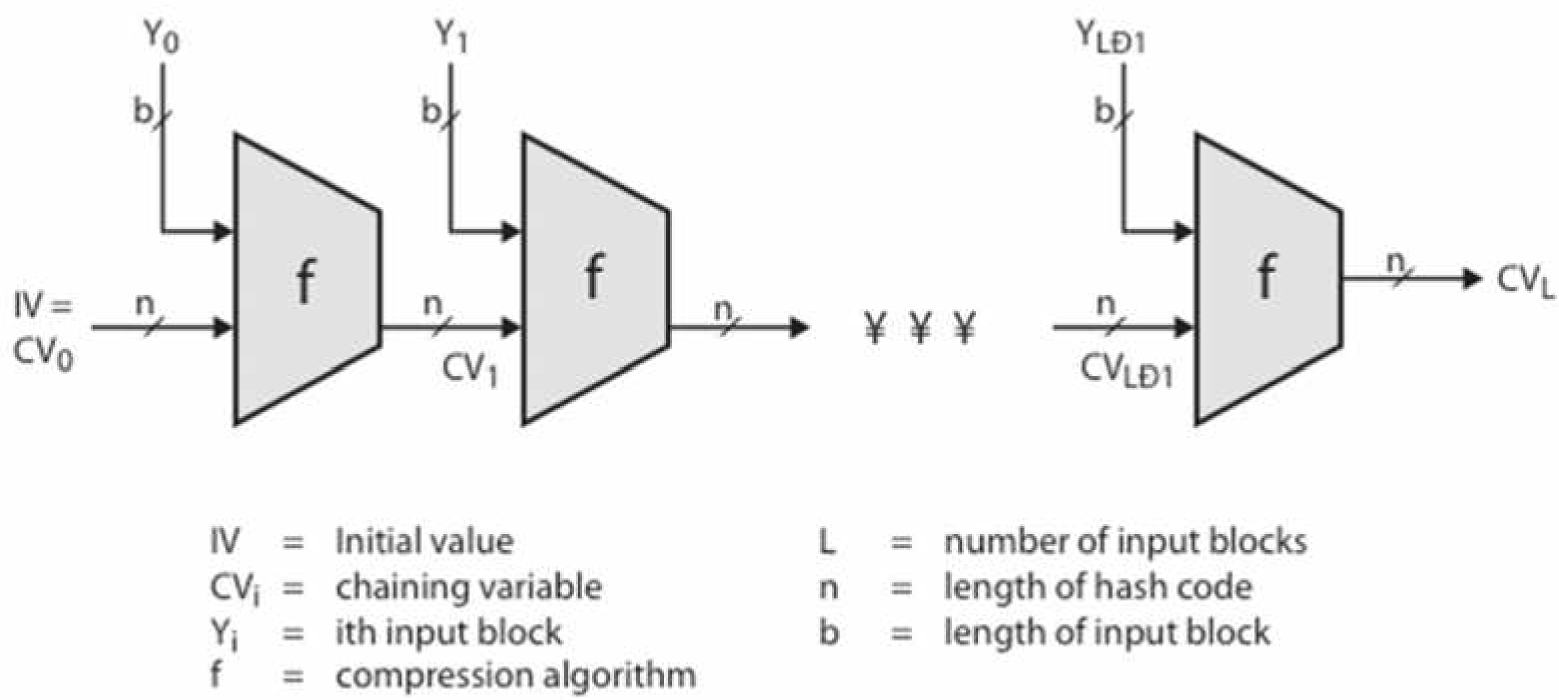
# Hash Functions & MAC Security

- **cryptanalytic attacks** exploit structure
  - like block ciphers want brute-force attacks to be the best alternative
- have a number of analytic attacks on iterated hash functions
  - $CV_i = f[CV_{i-1}, M_i]$; $H(M)=CV_N$
  - typically focus on collisions in function f
  - like block ciphers is often composed of rounds
  - attacks exploit properties of round functions

# Hash and MAC Algorithms

- Hash Functions
  - condense arbitrary size message to fixed size
  - by processing message in blocks
  - through some compression function
  - either custom or block cipher based
- Message Authentication Code (MAC)
  - fixed sized authenticator for some message
  - to provide authentication for message
  - by using block cipher mode or hash function

# Hash Algorithm Structure



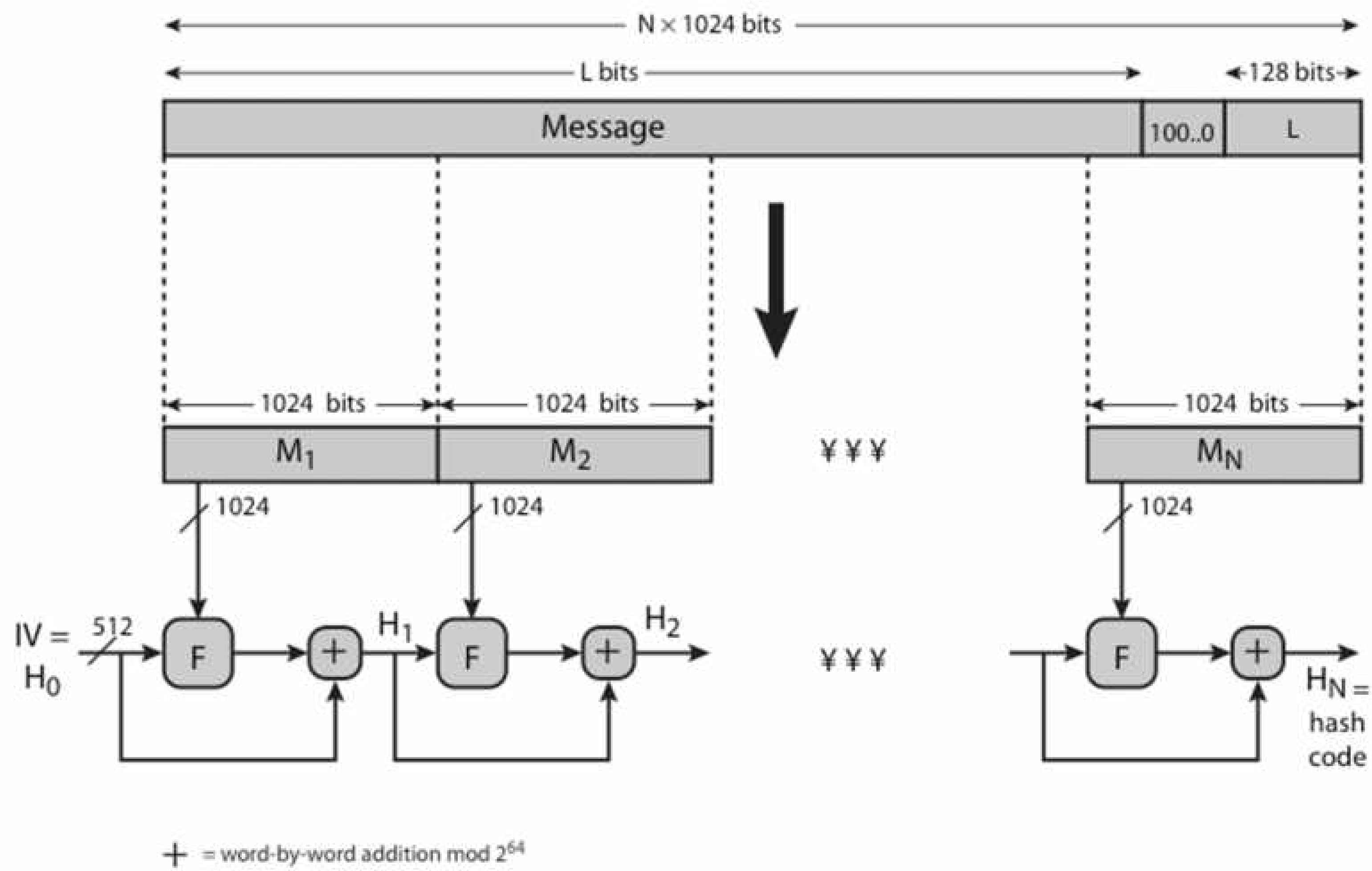
# Secure Hash Algorithm

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- based on design of MD4 with key differences
- produces 160-bit hash values
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

# Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

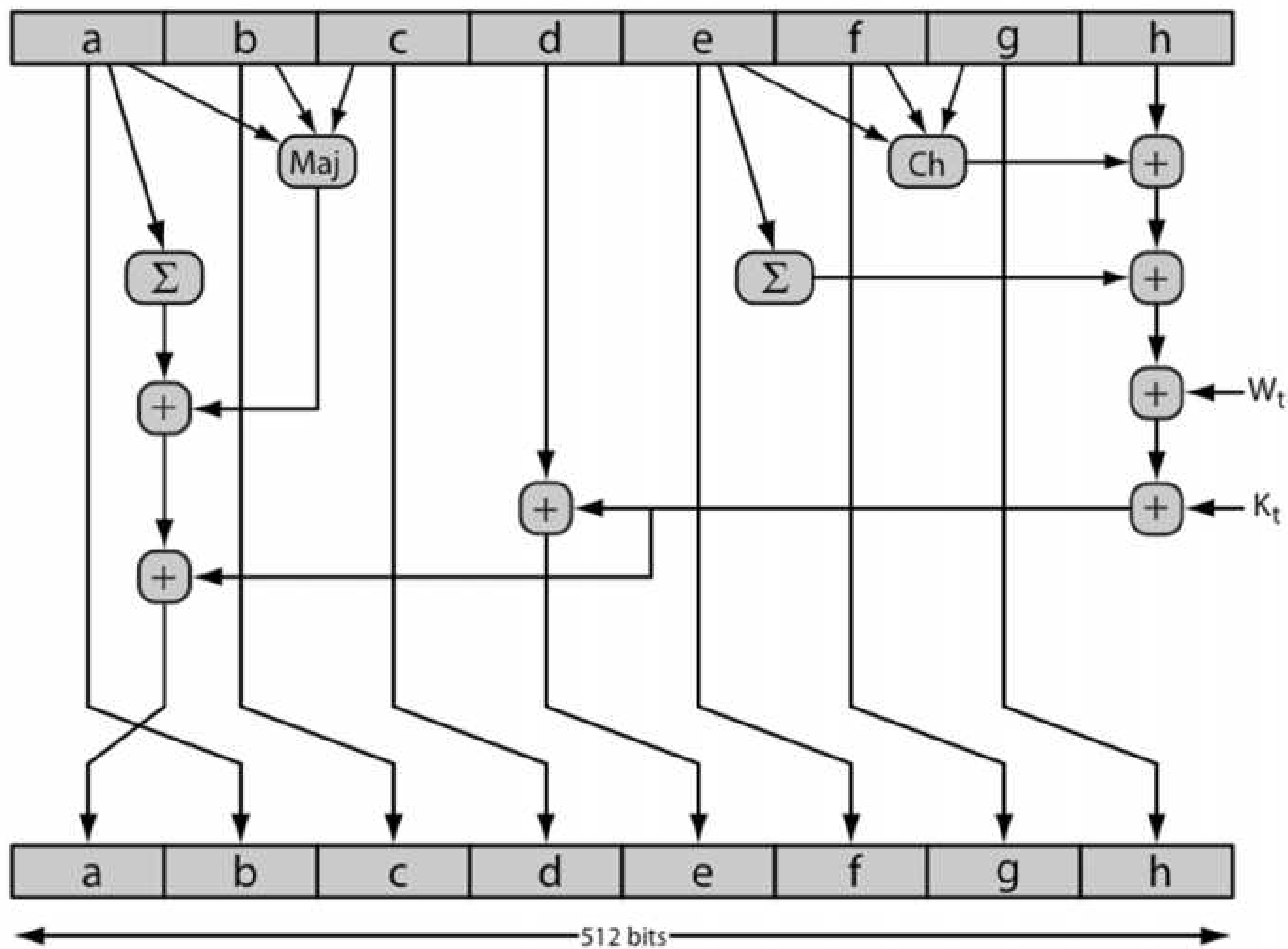
# SHA-512 Overview



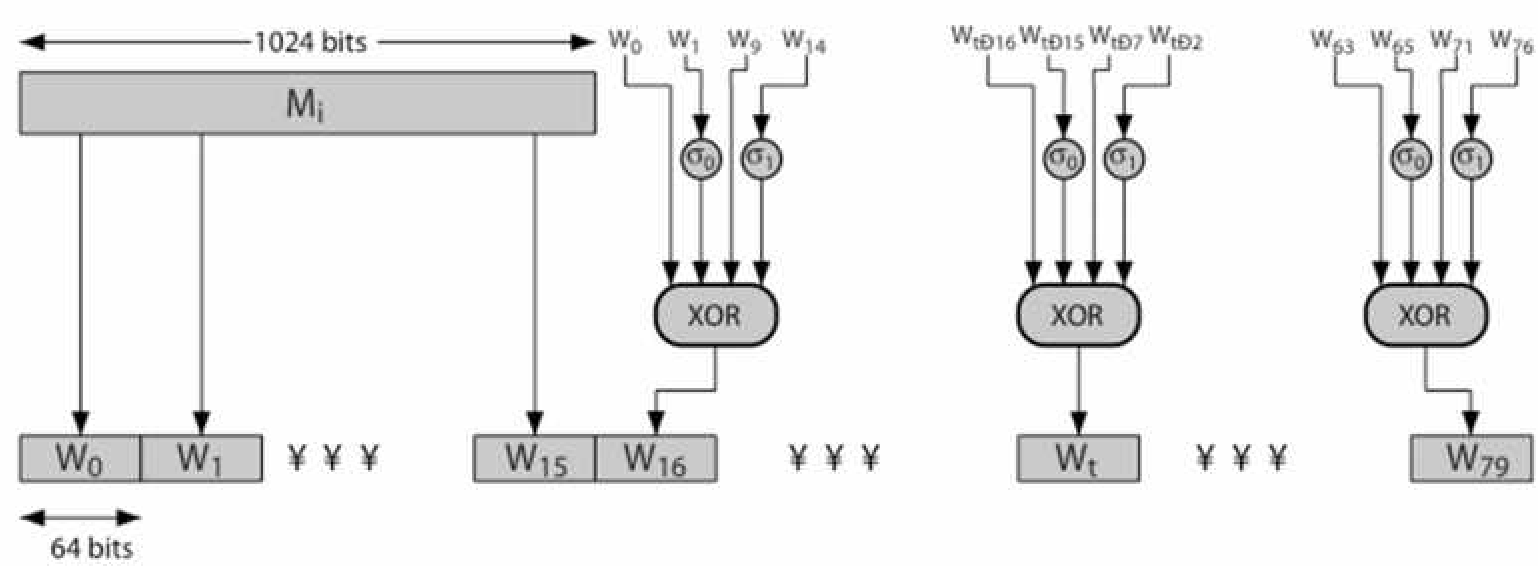
# SHA-512 Compression Function

- heart of the algorithm
- processing message in 1024-bit blocks
- consists of 80 rounds
  - updating a 512-bit buffer
  - using a 64-bit value Wt derived from the current message block
  - and a round constant based on cube root of first 80 prime numbers

# SHA-512 Round Function



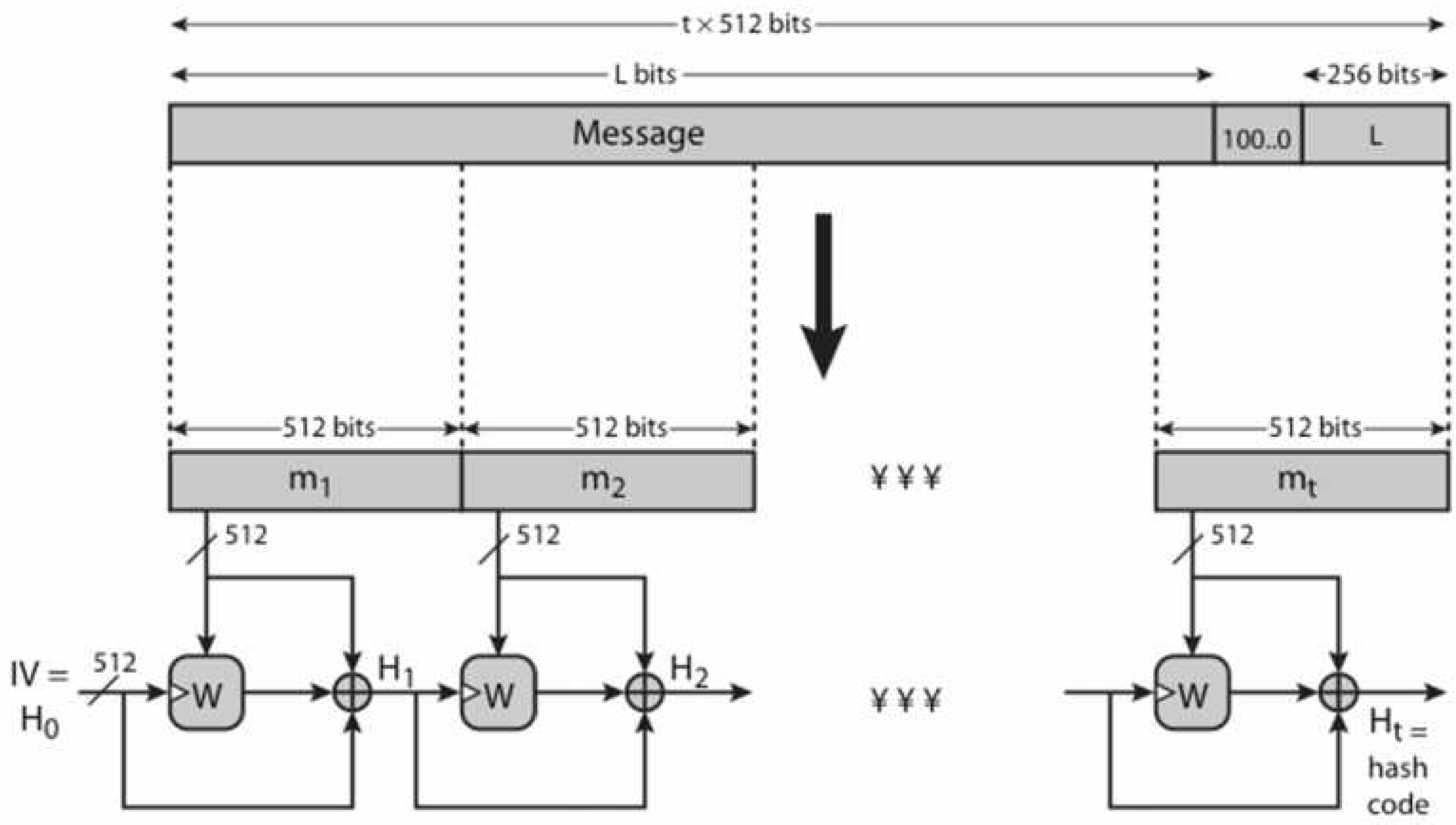
# SHA-512 Round Function



# Whirlpool

- now examine the Whirlpool hash function
- endorsed by European NESSIE project
- uses modified AES internals as compression function
- addressing concerns on use of block ciphers seen previously
- with performance comparable to dedicated algorithms like SHA

# Whirlpool Overview

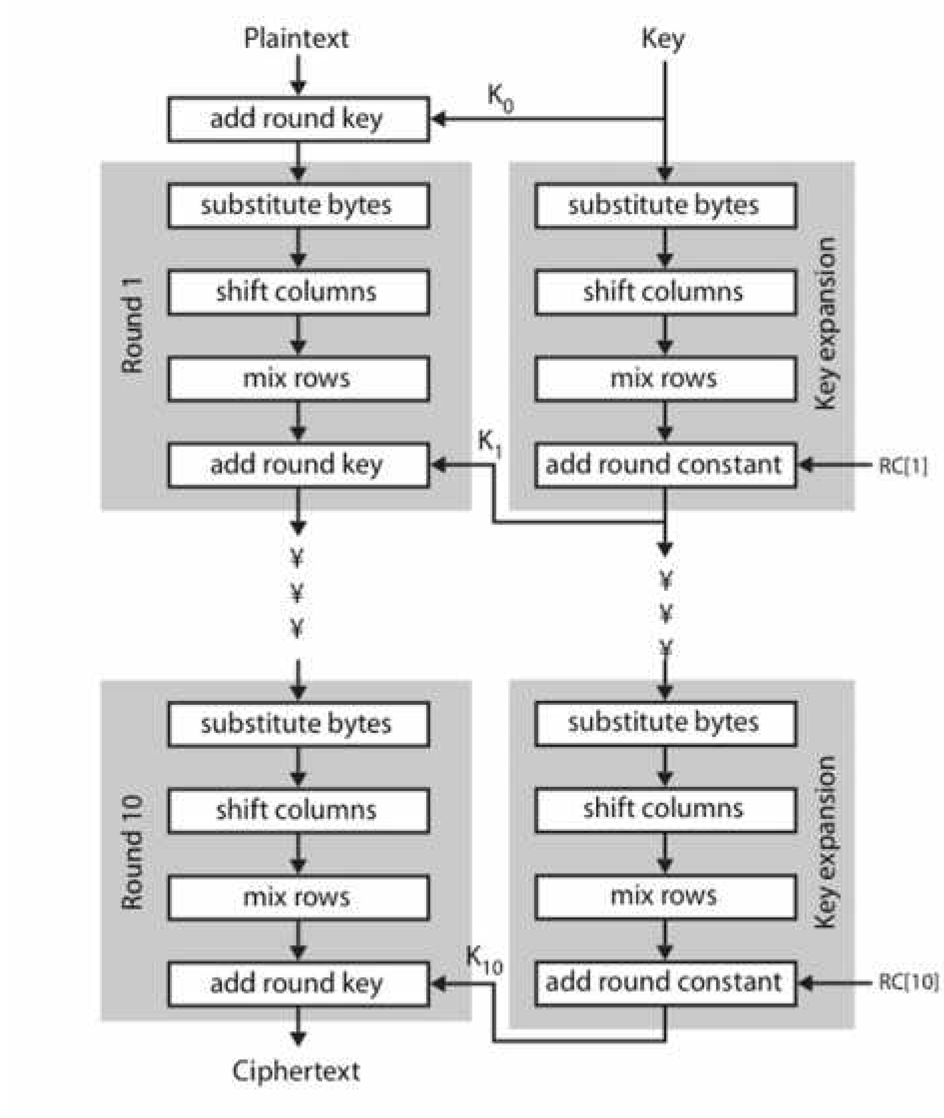



Note: triangular hatch marks key input

# Whirlpool Block Cipher W

- designed specifically for hash function use
- with security and efficiency of AES
- but with 512-bit block size and hence hash
- similar structure & functions as AES but
  - input is mapped row wise
  - has 10 rounds
  - a different primitive polynomial for GF(2^8)
  - uses different S-box design & values

# Whirlpool Block Cipher W



# Whirlpool Performance & Security

- Whirlpool is a very new proposal
- hence little experience with use
- but many AES findings should apply
- does seem to need more h/w than SHA, but with better resulting performance

# Keyed Hash Functions as MACs

- want a MAC based on a hash function
  - because hash functions are generally faster
  - code for crypto hash functions widely available
- hash includes a key along with message
- original proposal:

  `KeyedHash = Hash(Key|Message)`

  - some weaknesses were found with this
- eventually led to development of HMAC

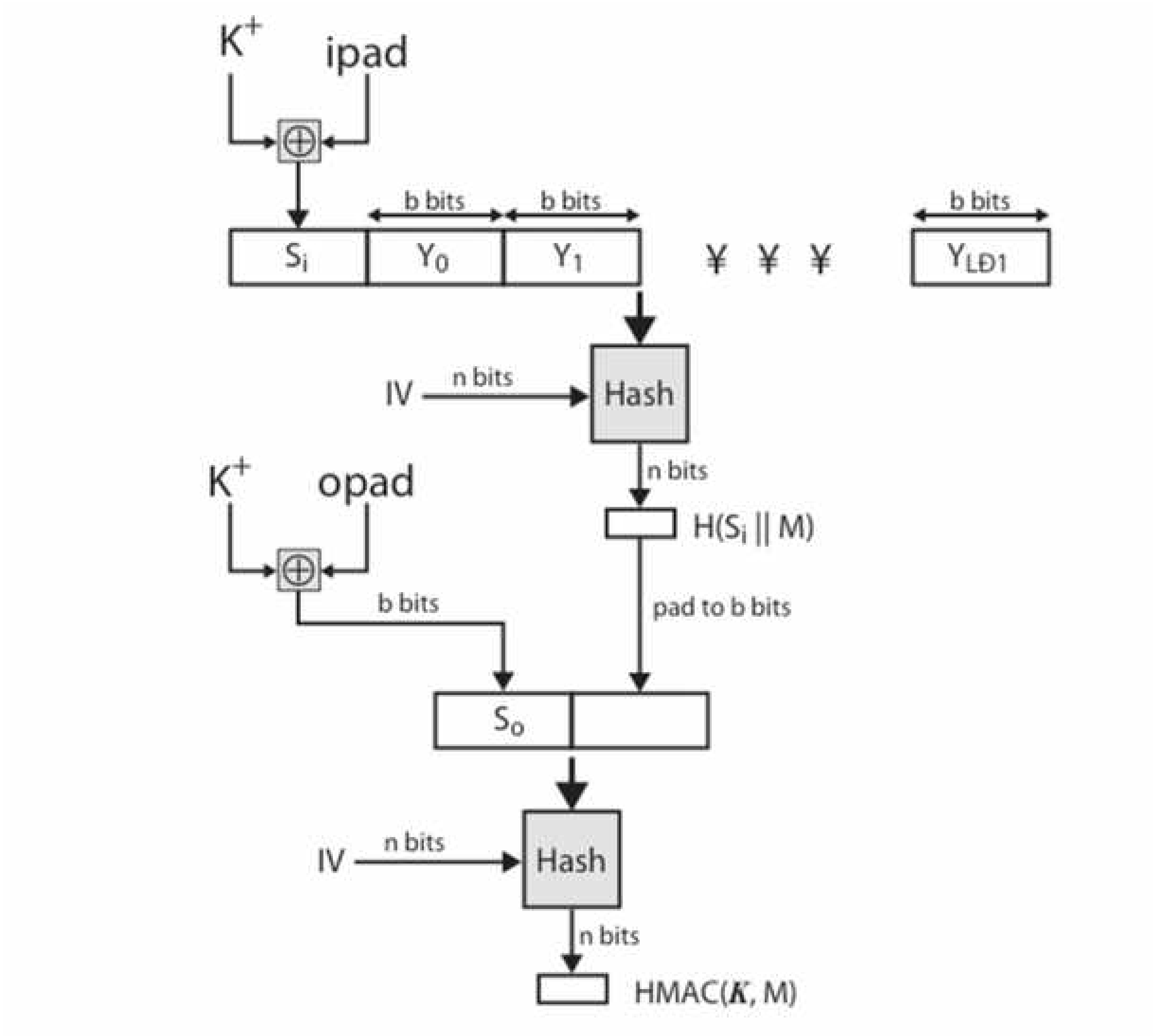
# HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:

  $$HMAC_K = Hash[(K^+ \text{ XOR } opad) \,||\, Hash[(K^+ \text{ XOR } ipad)||M)]]$$

- where $K^+$ is the key padded out to size
- and opad, ipad are specified padding constants
- overhead is just 3 more hash calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

# HMAC Overview



# HMAC Security

- proved security of HMAC relates to that of the underlying hash algorithm
- attacking HMAC requires either:
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)
- choose hash function used based on speed verses security constraints

# CMAC

- previously saw the DAA (CBC-MAC)
- widely used in govt & industry
- but has message size limitation
- can overcome using 2 keys & padding
- thus forming the Cipher-based Message Authentication Code (CMAC)
- adopted by NIST SP800-38B

# CMAC Overview

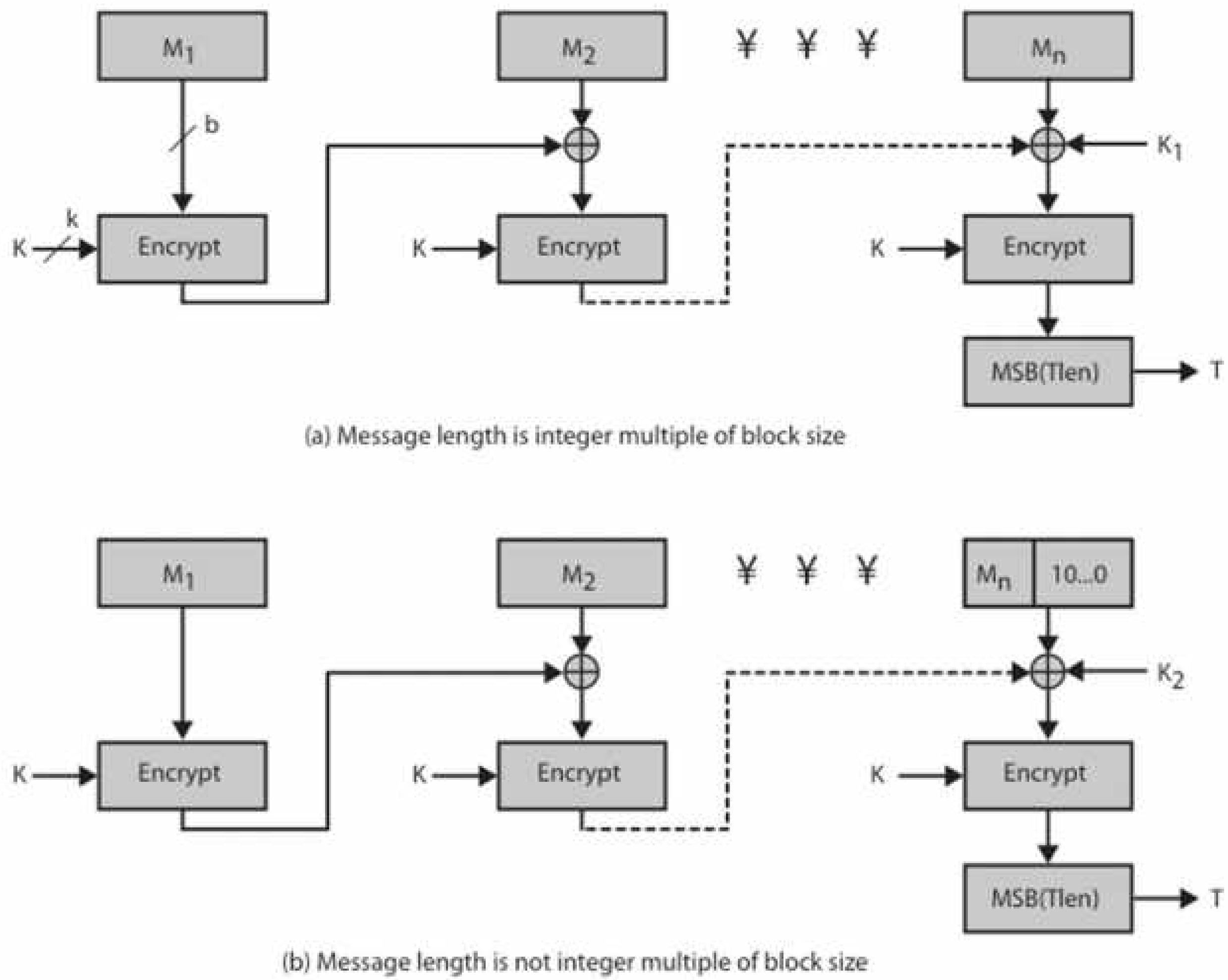



Figure 12.12 Cipher-Based Message Authentication Code (CMAC)

# Digital Signatures

- have looked at message authentication
  - but does not address issues of lack of trust
- digital signatures provide the ability to:
  - verify author, date & time of signature
  - authenticate message contents
  - be verified by third parties to resolve disputes
- hence include authentication function with additional capabilities

# Digital Signature Properties

- must depend on the message signed
- must use information unique to sender
  - to prevent both forgery and denial
- must be relatively easy to produce
- must be relatively easy to recognize & verify
- be computationally infeasible to forge
  - with new message for existing digital signature
  - with fraudulent digital signature for given message
- be practical save digital signature in storage

# Direct Digital Signatures

- involve only sender & receiver
- assumed receiver has sender’s public-key
- digital signature made by sender signing entire message or hash with private-key
- can encrypt using receivers public-key
- important that sign first then encrypt message & signature
- security depends on sender’s private-key

# Arbitrated Digital Signatures

- involves use of arbiter A
  - validates any signed message
  - then dated and sent to recipient
- requires suitable level of trust in arbiter
- can be implemented with either private or public-key algorithms
- arbiter may or may not see message

# Authentication Protocols

- used to convince parties of each others identity and to exchange session keys
- may be one-way or mutual
- key issues are
  - confidentiality – to protect session keys
  - timeliness – to prevent replay attacks
- published protocols are often found to have flaws and need to be modified

# Replay Attacks

- where a valid signed message is copied and later resent
  - simple replay
  - repetition that can be logged
  - repetition that cannot be detected
  - backward replay without modification
- countermeasures include
  - use of sequence numbers (generally impractical)
  - timestamps (needs synchronized clocks)
  - challenge/response (using unique nonce)

# Using Symmetric Encryption

- as discussed previously can use a two-level hierarchy of keys
- usually with a trusted Key Distribution Center (KDC)
  - each party shares own master key with KDC
  - KDC generates session keys used for connections between parties
  - master keys used to distribute these to them

# Needham-Schroeder Protocol

- original third-party key distribution protocol
- for session between A B mediated by KDC
- protocol overview is:
  **1.** A->KDC: $ID_A \,||\, ID_B \,||\, N_1$
  **2**. KDC -> A: $E_{Ka}[Ks \,||\, ID_B \,||\, N_1 \,||\, E_{Kb}[Ks||ID_A]\,]$
  **3.** A -> B: $E_{Kb}[Ks||ID_A]$
  **4.** B -> A: $E_{Ks}[N_2]$
  **5.** A -> B: $E_{Ks}[f(N_2)]$

# Needham-Schroeder Protocol

- used to securely distribute a new session key for communications between A & B
- but is vulnerable to a replay attack if an old session key has been compromised
  - then message 3 can be resent convincing B that is communicating with A
- modifications to address this require:
  - timestamps (Denning 81)
  - using an extra nonce (Neuman 93)

# Using Public-Key Encryption

- have a range of approaches based on the use of public-key encryption
- need to ensure have correct public keys for other parties
- using a central Authentication Server (AS)
- various protocols exist using timestamps or nonces

# Denning AS Protocol

- Denning 81 presented the following:
  **1.** A -> AS: $ID_A$ || $ID_B$
  **2.** AS -> A: $E_{PRas}[ID_A||PU_a||T]$ || $E_{PRas}[ID_B||PU_b||T]$
  **3.** A -> B: $E_{PRas}[ID_A||PU_a||T]$ || $E_{PRas}[ID_B||PU_b||T]$ || $E_{PUb}[E_{PRas}[K_s||T]]$
- note session key is chosen by A, hence AS need not be trusted to protect it
- timestamps prevent replay but require synchronized clocks

# One-Way Authentication

- required when sender & receiver are not in communications at same time (eg. email)
- have header in clear so can be delivered by email system
- may want contents of body protected & sender authenticated

# Using Symmetric Encryption

- can refine use of KDC but can't have final exchange of nonces, vis:
  **1.** A->KDC: $ID_A \,||\, ID_B \,||\, N_1$
  **2**. KDC -> A: $E_{Ka}[Ks \,||\, ID_B \,||\, N_1 \,||\, E_{Kb}[Ks||ID_A]\,]$
  **3.** A -> B: $E_{Kb}[Ks||ID_A] \,||\, E_{Ks}[M]$
- does not protect against replays
  – could rely on timestamp in message, though email delays make this problematic

# Public-Key Approaches

- have seen some public-key approaches
- if confidentiality is major concern, can use:

  A->B: $E_{PUb}[Ks] || E_{Ks}[M]$

  – has encrypted session key, encrypted message
- if authentication needed use a digital signature with a digital certificate:

  A->B: $M || E_{PRa}[H(M)] || E_{PRas}[T||ID_A||PU_a]$

  – with message, signature, certificate

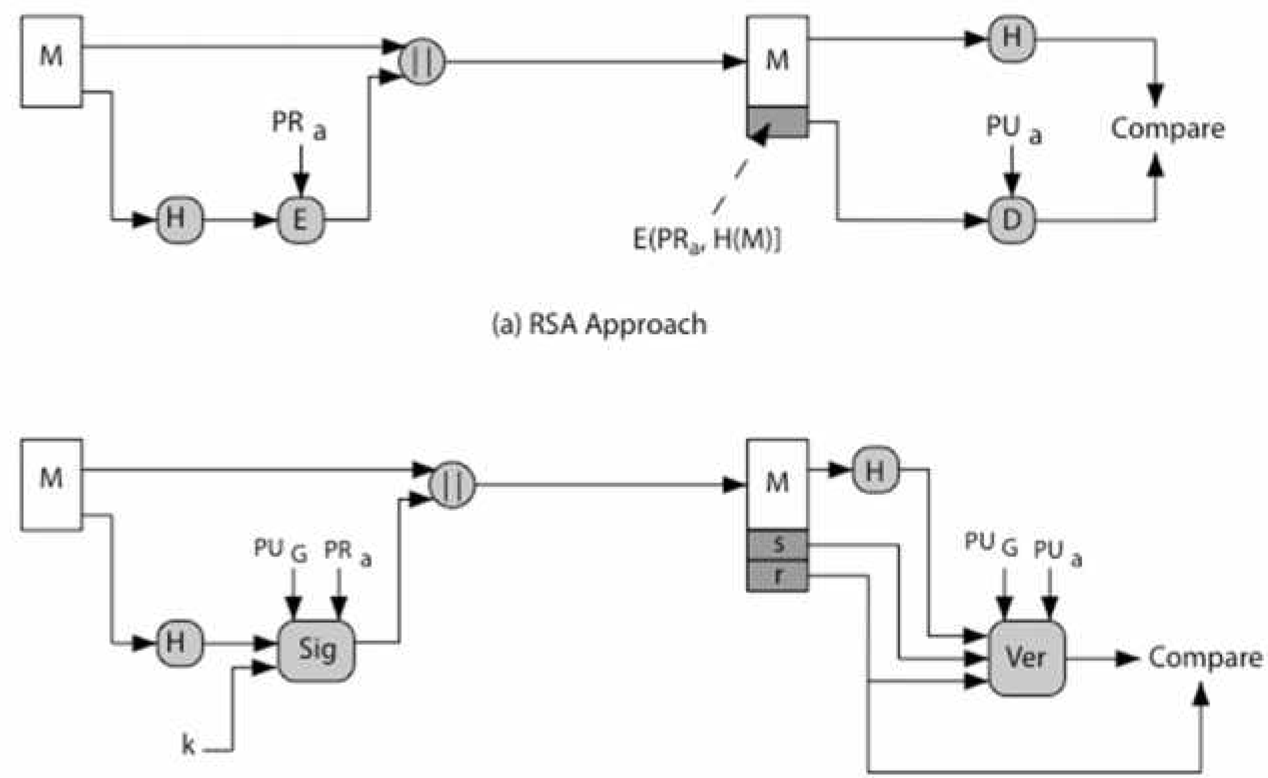
# Digital Signature Standard (DSS)

- US Govt approved signature scheme
- designed by NIST & NSA in early 90's
- published as FIPS-186 in 1991
- revised in 1993, 1996 & then 2000
- uses the SHA hash algorithm
- DSS is the standard, DSA is the algorithm
- FIPS 186-2 (2000) includes alternative RSA & elliptic curve signature variants

# Digital Signature Algorithm (DSA)

- creates a 320 bit signature
- with 512-1024 bit security
- smaller and faster than RSA
- a digital signature scheme only
- security depends on difficulty of computing discrete logarithms
- variant of ElGamal & Schnorr schemes

# Digital Signature Algorithm (DSA)




(a) RSA Approach

(b) DSS Approach

# DSA Key Generation

- have shared global public key values (p,q,g):
  - choose q, a 160 bit
  - choose a large prime $p = 2^L$
    - where L= 512 to 1024 bits and is a multiple of 64
    - and q is a prime factor of (p-1)
  - choose $g = h^{(p-1)/q}$
    - where $h<p-1$, $h^{(p-1)/q} \pmod{p} > 1$
- users choose private & compute public key:
  - choose $x<q$
  - compute $y = g^x \pmod{p}$

# DSA Signature Creation

- to **sign** a message M the sender:
  - generates a random signature key k, $k<q$
  - nb. k must be random, be destroyed after use, and never be reused
- then computes signature pair:

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

  $s = (k^{-1}.H(M) + x.r) \pmod q$
- sends signature (r,s) with message M

# DSA Signature Verification

- having received M & signature (r,s)
- to **verify** a signature, recipient computes:

$$w = s^{-1} \pmod q$$

$$u1 = (H(M).w) \pmod q$$

$$u2 = (r.w) \pmod q$$

$$v = (g^{u1}.y^{u2} \pmod p) \pmod q$$

- if v=r then signature is verified
- see book web site for details of proof why