CSEN 1001

Computer and Network Security

Amr El Mougy
Reham Ayman
Abdelrahman Anwar

Lecture (6)

Public Key Cryptography – Cont’d

Diffie-Hellman Key Exchange

- First public-key type scheme proposed
- By Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
- Is a practical method for public exchange of a secret key
- Used in a number of commercial products

Diffie-Hellman Key Exchange

- A public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
- Value of key depends on the participants (and their private and public key information)
- Based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- Security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Diffie-Hellman Setup

- All users agree on global parameters:
  - large prime integer or polynomial q
  - a being a primitive root mod q
  - ***a*** is a **primitive root modulo *q*** if the powers of *a* mod *q* generate all the integers from 1 to *q* - 1
  - *Ex:* 3 is a primitive root mod 7 because $3 \equiv 3 \bmod 7$, $3^2 \equiv 2 \bmod 7$, $3^3 \equiv 6 \bmod 7$, $3^4 \equiv 4 \bmod 7$, $3^5 \equiv 5 \bmod 7$, $3^6 \equiv 1 \bmod 7$
- Each user (eg. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$
- Each user makes public that key y_A

Diffie-Hellman Key Exchange

- Shared session key for users A & B is K_{AB}:

  $K_{AB} = a^{x_A . x_B} \bmod q$

  $= y_A^{x_B} \bmod q$ (which **B** can compute)

  $= y_B^{x_A} \bmod q$ (which **A** can compute)

- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- If Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- Attacker needs an x, must solve discrete log

Diffie-Hellman Example

- Users Alice & Bob who wish to swap keys:
- Agree on prime $q=353$ and $a=3$
- Select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- Compute respective public keys:
  - $y_A=\mathbf{3}^{97} \bmod 353 = 40$ (Alice)
  - $y_B=\mathbf{3}^{233} \bmod 353 = 248$ (Bob)
- Compute shared session key as:
  - $K_{AB}= y_B^{x_A} \bmod 353 = \mathbf{248}^{97} \bmod 353 = 160$ (Alice)
  - $K_{AB}= y_A^{x_B} \bmod 353 = \mathbf{40}^{233} \bmod 353 = 160$ (Bob)

Key Exchange Protocols

- Users could create random private/public D-H keys each time they communicate
- Users could create a known private/public D-H key and publish in a directory, then consulted and used to securely communicate with them
- Both of these are vulnerable to a Man-in-the-Middle Attack
- Authentication of the keys is needed

Message Authentication

- Message authentication is concerned with:

  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)

Security Requirements

- Disclosure
- Traffic analysis
- Masquerade
- Content modification
- Sequence modification
- Timing modification
- Source repudiation
- Destination repudiation

Message Authentication

- Virtually all authentication mechanisms rely on an **authenticator** that is sent with the message

- Three alternative functions used:
  - message encryption
  - message authentication code (MAC)
  - hash function

Message Encryption

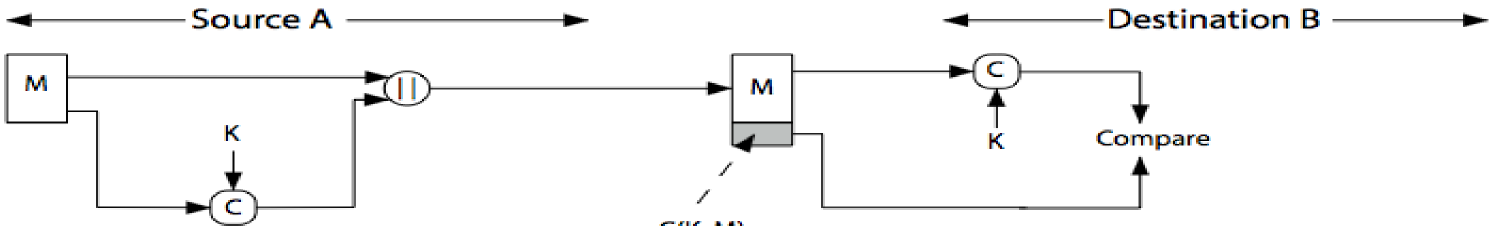
- Message encryption by itself also provides a measure of authentication
- If symmetric encryption is used then:
  - receiver know sender must have created it
  - since only sender and receiver now key used
  - if message has suitable structure, redundancy or a checksum to detect any changes

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 recipient’s public key
    - have both secrecy and authentication
  - Again need to recognize corrupted messages
  - But at cost of two public-key uses on message

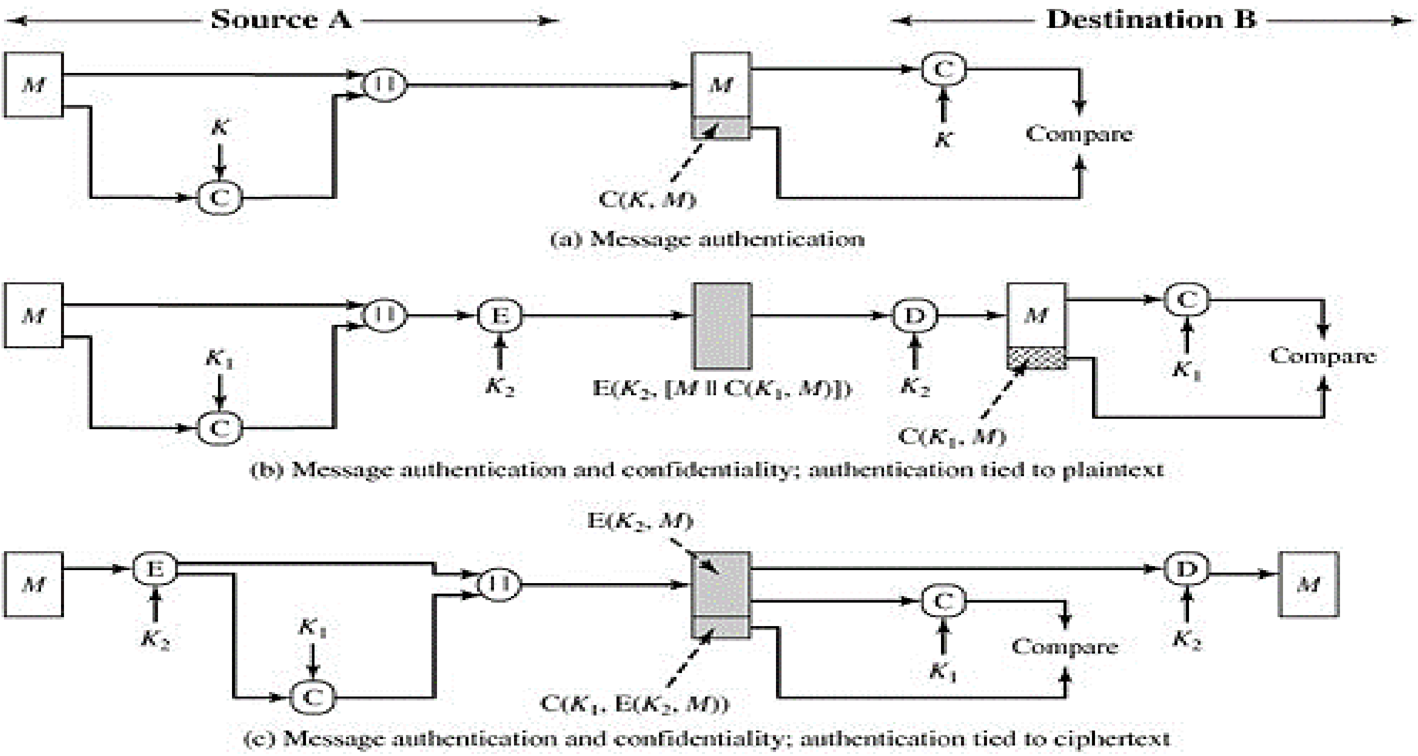
Message Authentication Codes (MAC)

- 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



(a) Message authentication

Message Authentication Codes (MAC)



Message Authentication Codes (MAC)

- 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

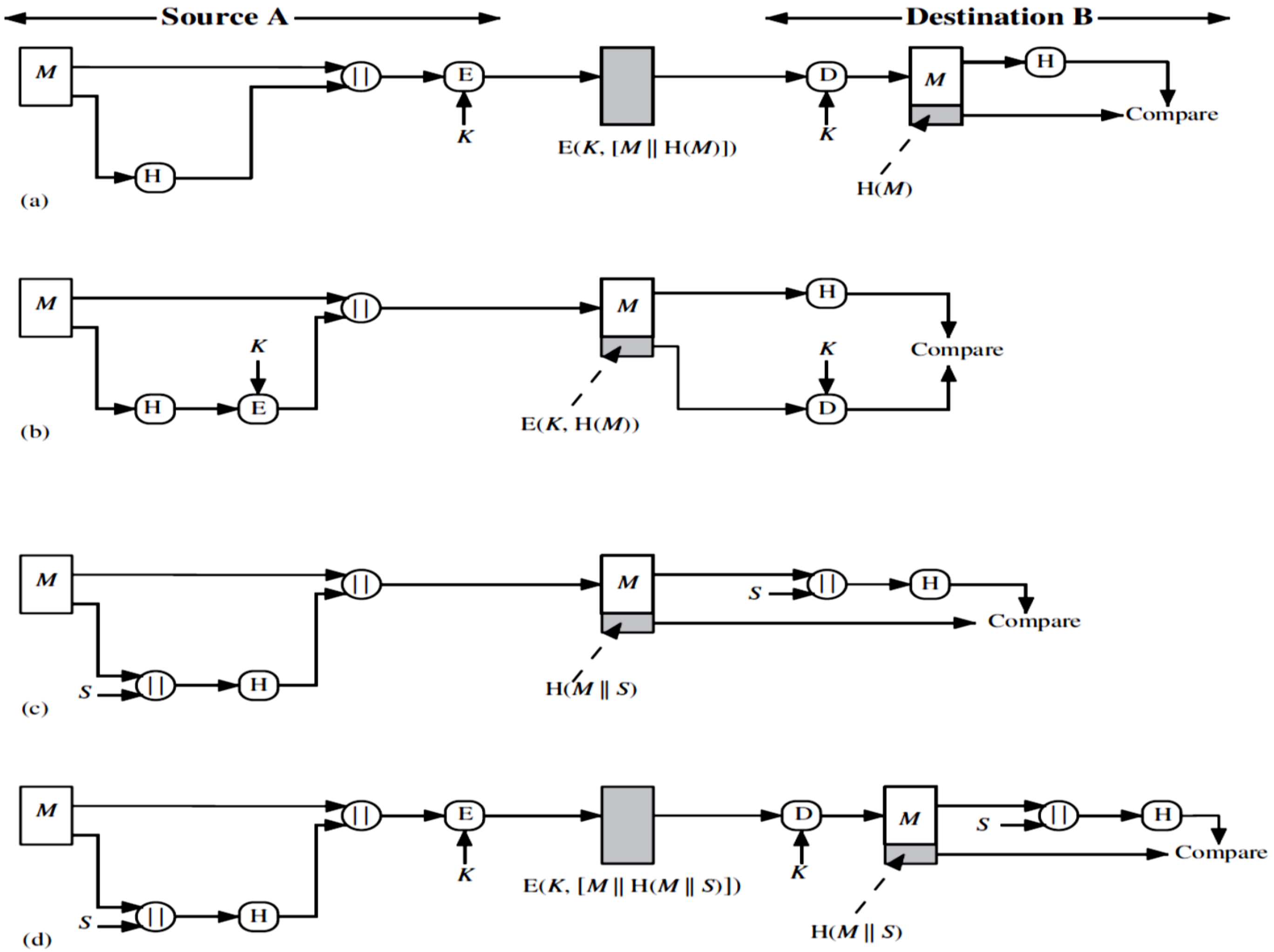
- A MAC is a cryptographic checksum
  - $MAC = C_K(M)$
  - condenses a variable-length message M
  - using a secret key K
  - to a fixed-sized authenticator
- Is a many-to-one function
  - potentially many messages have same MAC
  - but finding these needs to be very difficult

MAC Requirements

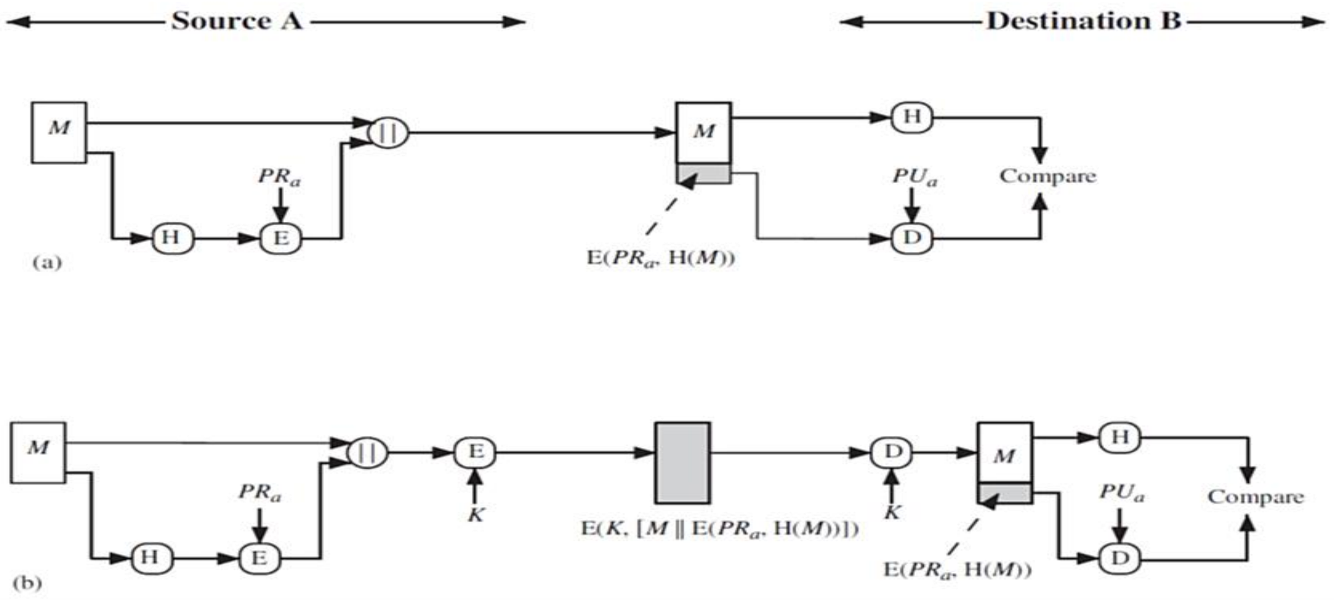
- Taking into account the types of attacks
- Need the MAC to satisfy the following:
  1. knowing a message and MAC, is infeasible to find another message with same MAC
  2. MACs should be uniformly distributed
  3. MAC should depend equally on all bits of the message

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



Source A
Destination B
M
||
E
K
H
E(K, [M || H(M)])
D
Compare
H(M)
(a)
E(K, H(M))
(b)
S
H(M || S)
(c)
E(K, [M || H(M || S)])
(d)



Source A
Destination B
M
||
H
PRa
E
M
E(PRa, H(M))
H
PUa
D
Compare
(a)
M
||
E
K
H
PRa
E
E(K, [M || E(PRa, H(M))])
D
K
M
E(PRa, H(M))
H
PUa
D
Compare
(b)

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

Birthday Attack

- 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

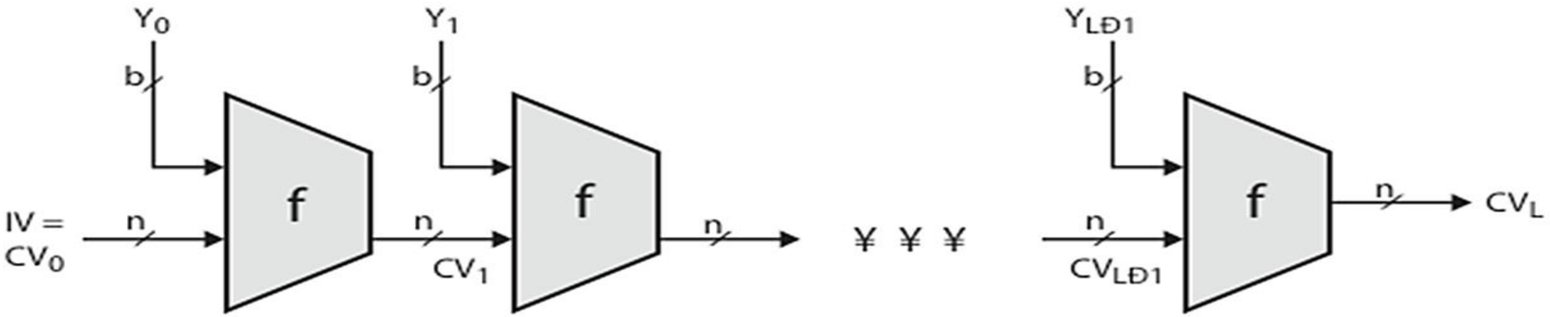
Dear Anthony,

{This letter is / I am writing} to introduce {you to / to you} {Mr. / --} Alfred {P. / --} Barton, the {new / newly appointed} {chief / senior} jewellery buyer for {our / the} Northern {European / Europe} {area / division}. He {will take / has taken} over {the / --} responsibility for {all / the whole of} our interests in {watches and jewellery / jewellery and watches} in the {area / region}. Please {afford / give} him {every / all the} help he {may need / needs} to {seek out / find} the most {modern / up to date} lines for the {top / high} end of the market. He is {empowered / authorized} to receive on our behalf {samples / specimens} of the {latest / newest} {watch and jewellery / jewellery and watch} products, {up / subject} to a {limit / maximum} of ten thousand dollars. He will {carry / hold} a signed copy of this {letter / document} as proof of identity. An order with his signature, which is {appended / attached} {authorizes / allows} you to charge the cost to this company at the {above / head office} address. We {fully / --} expect that our {level / volume} of orders will increase in the {following / next} year and {trust / hope} that the new appointment will {be / prove} {advantageous / an advantage} to both our companies.

Hash Function Security

- Like block ciphers have:
- **Brute-force** attacks exploiting
  - strong collision resistance hash have cost $2^{m/2}$
  - MACs with known message-MAC pairs
    - can either attack keyspace (cf key search) or MAC
- **Cryptanalytic attacks** exploit structure
  - like block ciphers want brute-force attacks to be the best alternative
  - Exploiting properties of round functions of block ciphers

Hash Algorithm Structure

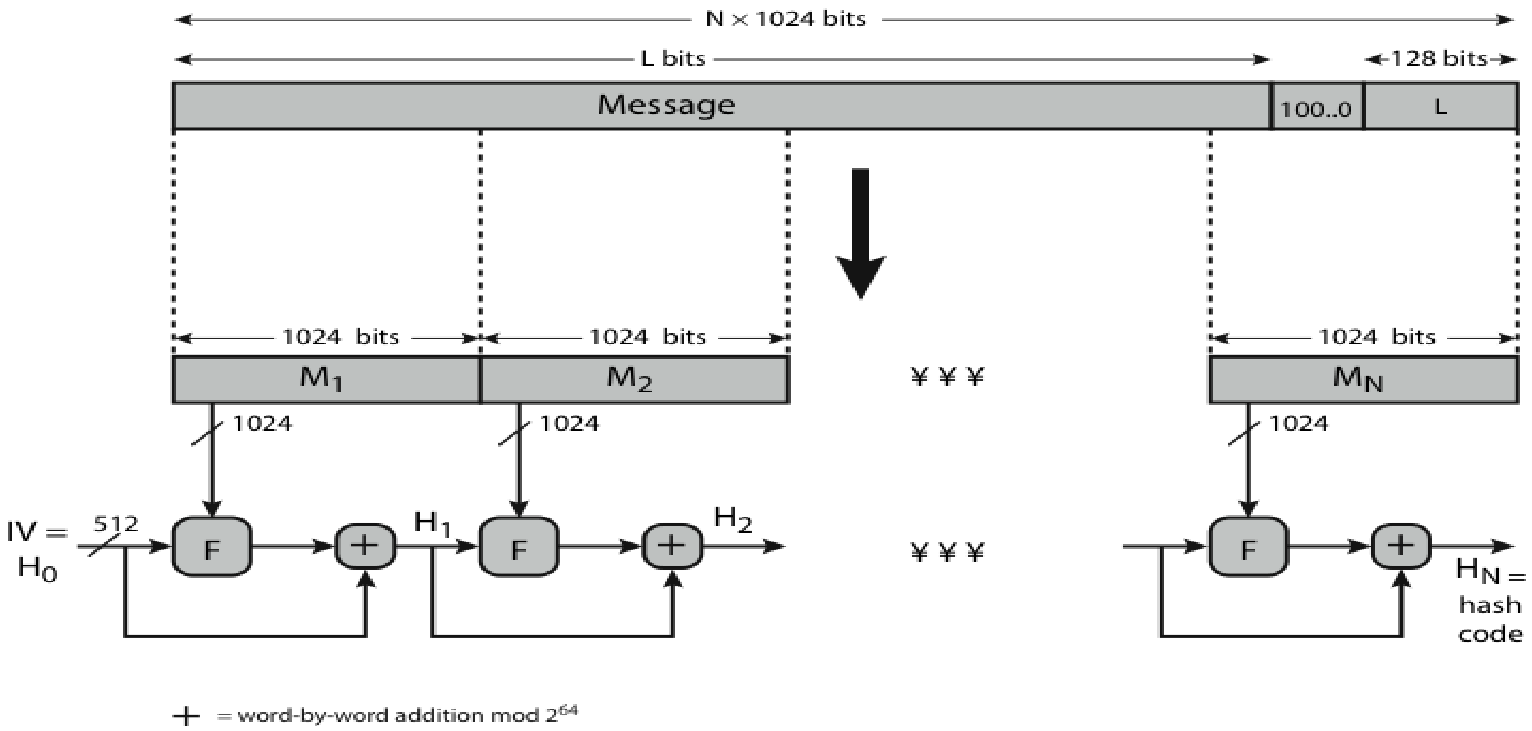


IV	=	Initial value	L	=	number of input blocks
CV_i	=	chaining variable	n	=	length of hash code
Y_i	=	ith input block	b	=	length of input block
f	=	compression algorithm			

Secure Hash Algorithm (SHA)

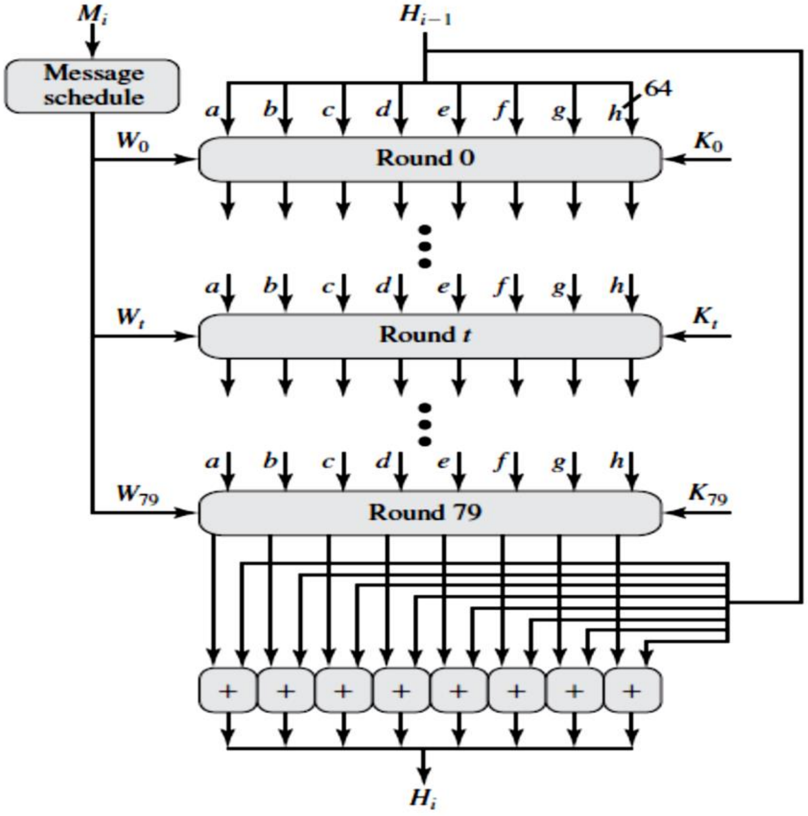
- SHA originally designed by NIST & NSA in 1993
- Was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- Based on design of MD4 with key differences
- Produces 160-bit hash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

Secure Hash Algorithm (SHA)

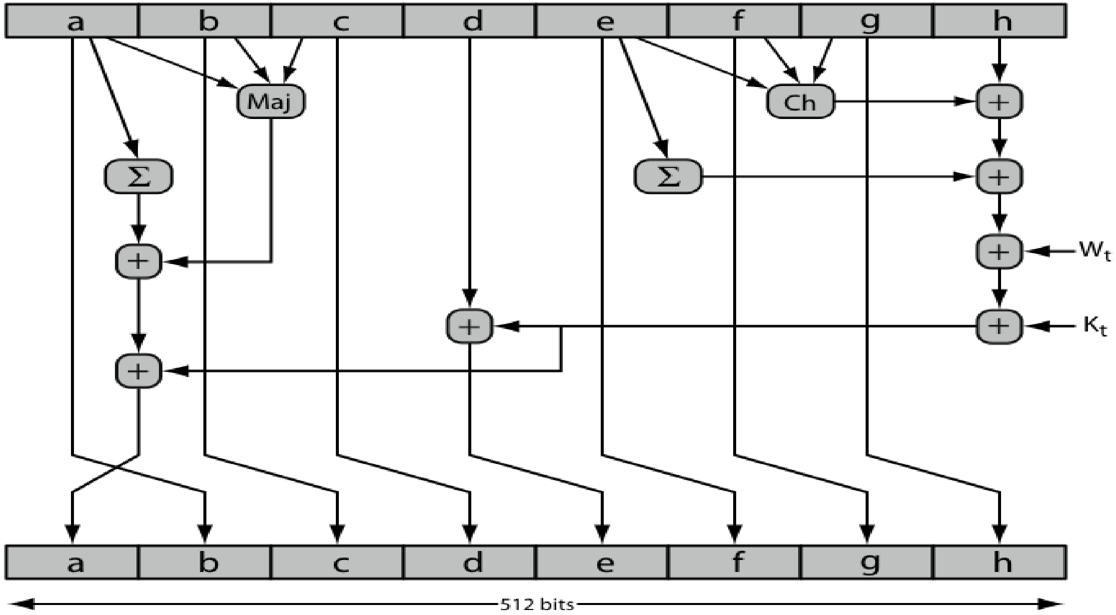


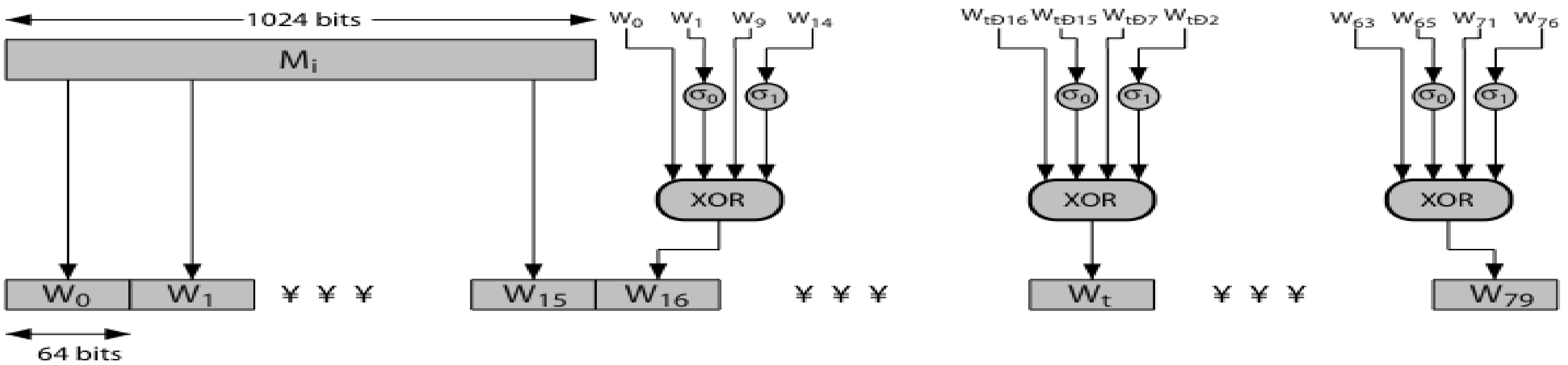
SHA-512 Compression Function

- ❑Heart of the algorithm
- ❑Processing message in 1024-bit blocks
- ❑Consists of 80 rounds
  - ❑updating a 512-bit buffer
  - ❑using a 64-bit value W_t derived from the current message block
  - ❑and a round constant based on cube root of first 80 prime numbers



SHA-512 Round Function





$$
\begin{aligned}
T_1 &= h + \mathrm{Ch}(e, f, g) + \left(\textstyle\sum_1^{512} e\right) + W_t + K_t \\
T_2 &= \left(\textstyle\sum_0^{512} a\right) + \mathrm{Maj}(a, b, c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

where

t = step number; $0 \le t \le 79$

$\mathrm{Ch}(e, f, g) = (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$
the conditional function: If e then f else g

$\mathrm{Maj}(a, b, c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$
the function is true only of the majority (two or three) of the arguments are true

$\left(\sum_0^{512} a\right) = \mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a)$

$\left(\sum_1^{512} e\right) = \mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e)$

$\mathrm{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

W_t = a 64-bit word derived from the current 512-bit input block

K_t = a 64-bit additive constant

$+$ = addition modulo 2^{64}

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

where

$\sigma_0^{512}(x) = \mathrm{ROTR}^1(x) \oplus \mathrm{ROTR}^8(x) \oplus \mathrm{SHR}^7(x)$

$\sigma_1^{512}(x) = \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^6(x)$

$\mathrm{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

$\mathrm{SHR}^n(x)$ = left shift of the 64-bit argument x by n bits with padding by zeros on the right

$+$ = addition modulo 2^{64}

Keyed-Hash Message Authentication Code (HMAC)

- specified as Internet standard RFC2104
- uses hash function on the message:
  - $HMAC_K$ = Hash[(K$^+$ XOR opad) || Hash[(K$^+$ XOR ipad)||M)]]
- where K$^+$ is the key padded out to size
- and opad, ipad are specified padding constants
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC Overview

$HMAC_K$ = Hash[(K+ XOR opad) ||
Hash[(K+ XOR ipad) || M)]

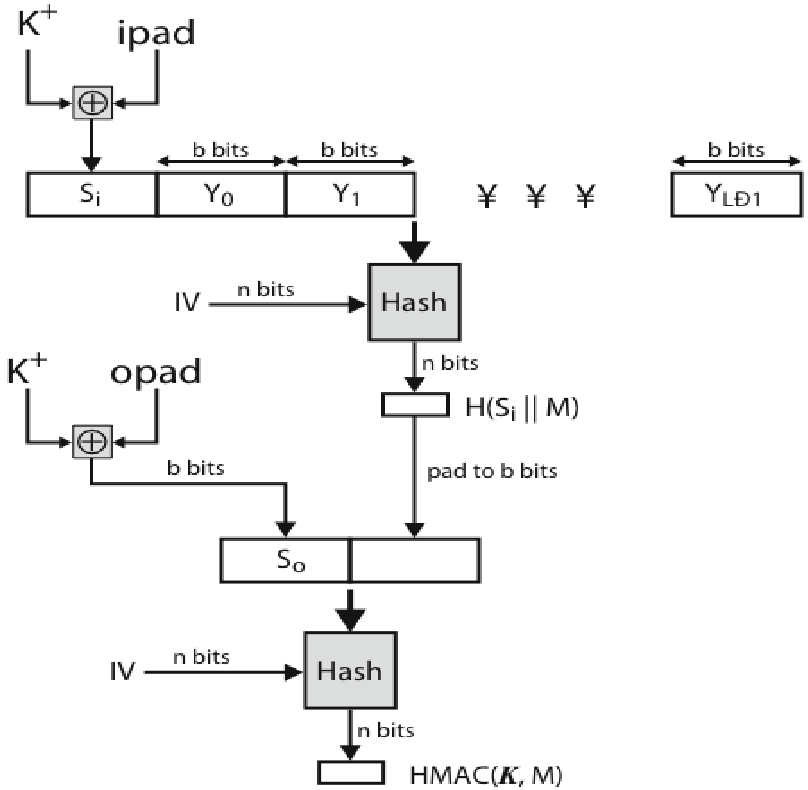
elements are:

K^+ is K padded with zeros on the left so that the result is b bits in length

ipad is a pad value of 36 hex repeated to fill block

opad is a pad value of 5C hex repeated to fill block

M is the message input to HMAC (including the padding specified in the embedded hash function)

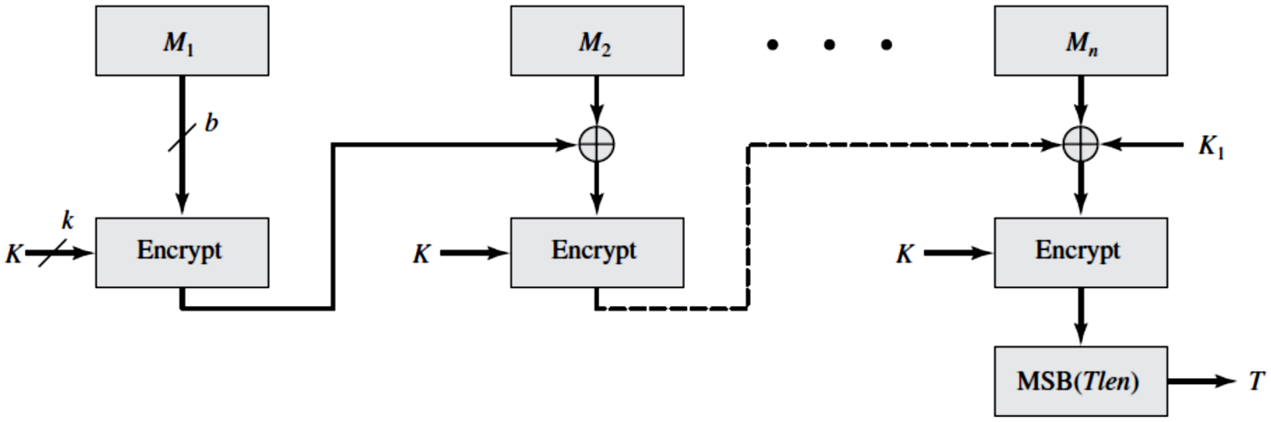


HMAC Security

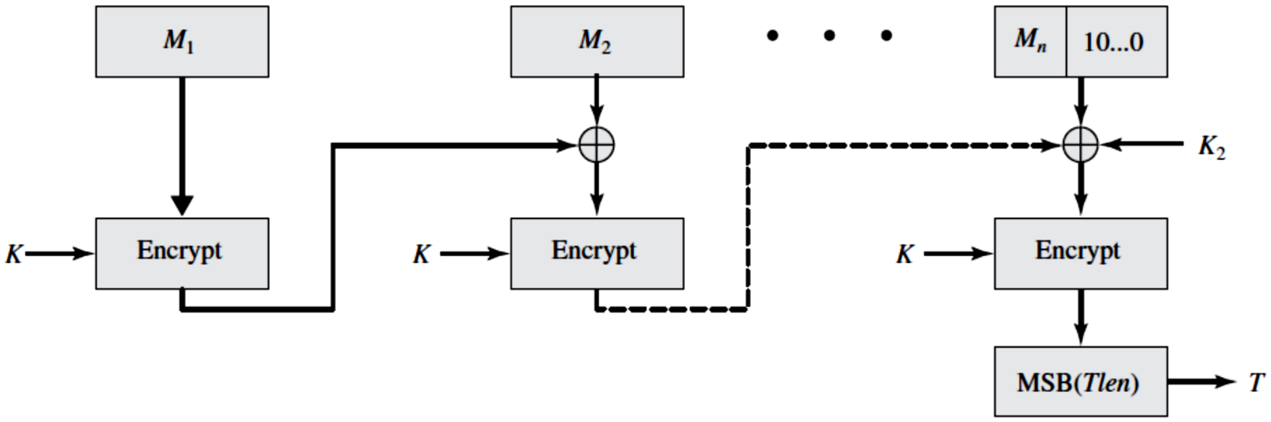
- proved security of HMAC relates to that of the underlying hash algorithm
- attacking HMAC requires either:
  - brute force attack on key used which has work of order 2^n
  - birthday attack (but since keyed would need to observe a very large number of messages) requires work of order 2^(n/2) - but which requires the attacker to observe 2^(n/2) blocks of messages using the same key - very unlikely
- choose hash function used based on speed versus security constraints

Cipher-Based MAC (CMAC)

$$L = E(K, 0^n)$$
$$K_1 = L \cdot x$$
$$K_2 = L \cdot x^2 = (L \cdot x) \cdot x$$



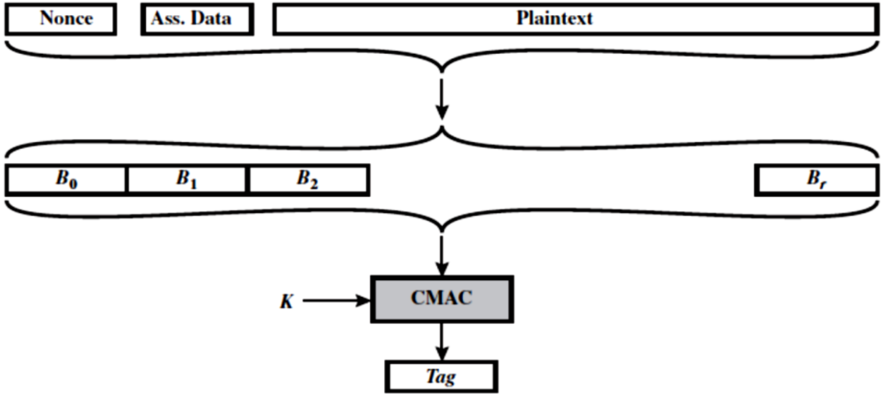
(a) Message length is integer multiple of block size



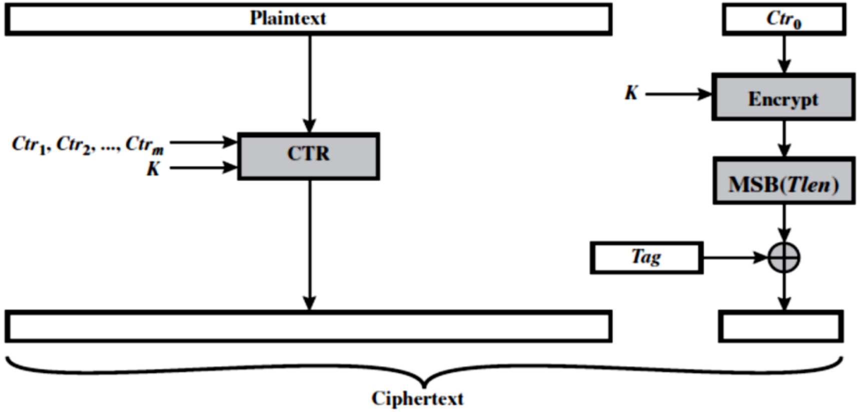
(b) Message length is not integer multiple of block size

Authenticated Modes of Encryption:

Counter with CBC Mode (CCM)



(a) Authentication



(b) Encryption

Authenticated Modes of Encryption:

Galois Counter Mode (GCM)