UNIT V

HASH FUNCTION AND DIGITAL SIGNATURE
AUTHENTICATION REQUIREMENTS

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

• Message Encryption:
  – Ciphertext itself serves as authenticator

• Message Authentication Code (MAC):
  – A function of message and a secret key that produces a fixed-length value that serves as authenticator

• Hash Function:
  – A function that maps a message of any length into a fixed length hash value, which serves as authenticator
Encryption for Authentication

(a) Conventional encryption: confidentiality and authentication

(b) Public-key encryption: confidentiality
Encryption for Authentication

(c) Public-key encryption: authentication and signature

(d) Public-key encryption: confidentiality, authentication and signature
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 know key used
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
An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a cryptographic checksum or MAC. It is appended to the message. Two communicating parties, say A and B, share a common secret key K. A MAC function is similar to encryption, except that the MAC algorithm need not be reversible, as it must for decryption.
MAC Properties

- a MAC is a cryptographic checksum
  \[ \text{MAC} = C_K(M) \]
  - 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

• consider the types of attacks against MAC.
• Message replacement attacks, in which an opponent is able to construct a new message to match a given MAC, even though the opponent does not know and does not learn the key.
• a brute-force attack based on chosen plaintext.
• authentication algorithm should not be weaker with respect to certain parts or bits of the message than others.
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 ($16 \leq M \leq 64$) of final block

- but final MAC is now too small for security
Data Authentication Algorithm

Diagram showing the flow of data encryption and decryption processes for different time values (Time = 1, Time = 2, Time = N - 1, Time = N). The diagram includes DES encryption and decryption processes with keys and output data (O₁, O₂, Oₙ₋₁, Oₙ) for each time step.
Hash Functions

• condenses arbitrary message to fixed size
  \[ h = H(M) \]

• usually assume that the hash function is public and not keyed
  – MAC which is keyed

• hash used to detect changes to message

• most often to create a digital signature
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

- \( C_i = b_{i1} \ XOR b_{i2} \ XOR \ldots \ XOR b_{im} \)
  - \( C_i = \) \( i \)th bit in 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

- The input is processed one block at a time in an iterative fashion to produce an \( n \)-bit hash function

- Can improve the performance, have a one bit circular shift on hash value after each block processed
Simple Hash Functions

• Initially set the n-bit value to zero
• Process each successive n-bit block of data as follows
  – Rotate the current hash value to the left by one bit
  – XOR the block into the hash value
Birthday Attacks

• might think a 64-bit hash is secure
• but by **Birthday Paradox** is not
• **birthday attack** works thus:
  – The source A is prepared to sign by appending the msg with approx m-bit hash code and encrypting with A’s private key
  – 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
Birthday Attacks (cont)

• two sets of messages are compared to find pair with same hash
• (probability > 0.5 by birthday paradox)
• Opponent offers the valid variation for sign to A
• The signature is attached with the fraudulent variation for transmission
• Since both have same hash code, they will produce same signature
• Opponent assumes though key not known red success ev
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
Hash Algorithm Structure

IV = Initial value
CV_i = chaining variable
Y_i = ith input block
f = compression algorithm

L = number of input blocks
n = length of hash code
b = length of input block
MD5

- designed by Ronald Rivest (the R in RSA)
- latest in a series of MD2, MD4
- produces a 128-bit hash value
- until recently was the most widely used hash algorithm
  - in recent times have both brute-force & cryptanalytic concerns
- specified as Internet standard RFC1321
MD5 Overview

1. pad message
   - so if its length is 448 then pad msg is 448 mod 512
     • Padding of 1-512 bits is always used.
     • Padding: 1000....0
2. append a 64-bit length value to message
   • Generate a message with 512L bits in length
3. initialise 4-word (128-bit) MD buffer (A,B,C,D)
4. process message in 16-word (512-bit) blocks:
5. output hash value is the final buffer value
MD5 Overview

- $L \times 512$ bits = $N \times 32$ bits
- $K$ bits
- Padding (1 to 512 bits)
- Message length ($k \mod 2^{64}$)

Message

$Y_0$, $Y_1$, ..., $Y_q$, ..., $Y_{L-1}$

$H_{MD5}$, $H_{MD5}$, $H_{MD5}$

IV, $CV_1$, $CV_q$, $CV_{L-1}$

128-bit digest
Figure 12.2 MD5 Processing of a Single 512-bit Block
MD5 Compression Function
MD5 Compression Function

• each round has 16 steps of the form:
  \[ a = b + ((a + g(b, c, d) + X[k] + T[i]) \ll s) \]
• a,b,c,d refer to the 4 words of the buffer, but used in varying permutations
  — note this updates 1 word only of the buffer
  — after 16 steps each word is updated 4 times
• where \( g(b,c,d) \) is a different nonlinear function in each round \((F,G,H,I)\)
• \( X[k] \) is the kth 32-bit word in the current message block.
• \( T[i] \) is the ith entry in the matrix of constants \( T \).
Secure Hash Algorithm

- SHA was designed by NIST & NSA in 1993, revised 1995 as SHA-1
- US standard for use with DSA signature scheme
- produces **160-bit** hash values
- now the generally preferred hash algorithm
- based on design of MD4 with key differences
- In 2002 SHA revised- three new versions of SHA,
  — with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512.
SHA-512 OVERVIEW

- N \times 1024 \text{ bits}
- L \text{ bits}
- 128 \text{ bits}

Message

\[ M_1, M_2, \ldots, M_N \]

IV = H_0

\[ H_0, H_1, H_2 \]

\[ H_N = \text{hash code} \]

\[ + = \text{word-by-word addition mod } 2^{64} \]
STEPS IN SHA-512

• Step 1: Append padding bits
• Step 2: Append length
• Step 3: Initialize hash buffer
• Step 4: Process the message in 1024-bit (128-word) blocks, which forms the heart of the algorithm
• Step 5: Output the final state value as the resulting hash
Figure 12.5 SHA-1 Processing of a Single 512-bit Block
(SHA-1 Compression Function)
SHA-512 Compression Function
SHA-512 Compression Function

- In this Step 4, it processes the message in 1024-bit (128-word) blocks, using a module that consists of 80 rounds, labeled F.
- Each round takes as input the 512-bit buffer value, and updates the contents of the buffer.
- Each round \( t \) makes use of a 64-bit value \( W_t \) derived using a message schedule from the current 1024-bit block being processed.
- Each round also makes use of an additive constant \( K_t \), based on the fractional parts of the cube roots of the first eighty prime numbers.
- The output of the 18th round is added to the input to the first round to produce the final hash value for this message block, which forms the input to the next iteration of this compression function.
SHA-512 Round Function
SHA-512 Round Function

- The structure of each of the 80 rounds is shown in Stallings Figure 12.3. Each 64-bit word shuffled along one place, and in some cases manipulated using a series of simple logical functions (ANDs, NOTs, ORs, XORs, ROTates), in order to provide the avalanche & completeness properties of the hash function. The elements are:
  - $\text{Ch}(e,f,g) = (e \text{ AND } f) \text{ XOR } (\text{NOT } e \text{ AND } g)$
  - $\text{Maj}(a,b,c) = (a \text{ AND } b) \text{ XOR } (a \text{ AND } c) \text{ XOR } (b \text{ AND } c)$
  - $\sum(a) = \text{ROTR}(a,28) \text{ XOR } \text{ROTR}(a,34) \text{ XOR } \text{ROTR}(a,39)$
  - $\sum(e) = \text{ROTR}(e,14) \text{ XOR } \text{ROTR}(e,18) \text{ XOR } \text{ROTR}(e,41)$
  - $+ = \text{addition modulo } 2^{64}$
  - $K_t = \text{a } 64\text{-bit additive constant}$
  - $W_t = \text{a } 64\text{-bit word derived from the current } 512\text{-bit block}$
Figure 12.7  Creation of 80-word Input Sequence for SHA-1 Processing of Single Block
• Stallings Figure 12.4 details how the 64-bit word values $W_t$ are derived from the 1024-bit message. The first 16 values of $W_t$ are taken directly from the 16 words of the current block. The remaining values are defined as a function of the earlier values using ROTates, SHIFTs and XORs as shown. The function elements are:

- $\partial_0(x) = \text{ROTR}(x,1) \text{ XOR} \ \text{ROTR}(x,8) \text{ XOR} \ \text{SHR}(x,7)$
- $\partial_1(x) = \text{ROTR}(x,19) \text{ XOR} \ \text{ROTR}(x,61) \text{ XOR} \ \text{SHR}(x,6)$. 