

# Lecture 6

## Cryptographic Hash Functions and Message Digests

1

### Cryptographic HASH Functions

- Purpose: produce a fingerprint or digest of input data
- Properties of a "good" HASH function  $H()$ :
  1.  $H()$  takes on input of any size
  2.  $H()$  produces fixed-length output
  3.  $H(x)$  is easy to compute (efficient)
  4. Given any  $h$ , it is computationally infeasible to find  $x$  such that  $H(x) = h$
  5. For any  $x$ , it is computationally infeasible to find  $y$  such that  $H(y) = H(x)$  and  $y \neq x$
  6. It is computationally infeasible to find any  $(x, y)$  such that  $H(x) = H(y)$  and  $x \neq y$

2

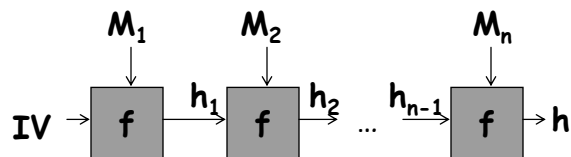
## HASH Functions properties restated:

- ❖ Cryptographic properties of a "good" HASH function:
  - ❖ One-way-ness (#4)
  - ❖ Weak Collision-Resistance (#5)
  - ❖ Strong Collision-Resistance (#6)
- ❖ Non-cryptographic properties of a "good" HASH function
  - ❖ Efficiency (#3)
  - ❖ Fixed output (#1)
  - ❖ Arbitrary-length input (#2)

3

## Construction

- A hash function is typically based on internal compression function  $f()$  that works on fixed-size input blocks ( $M_i$ )



- Works sort of like a Chained Block Cipher
  - ❖ Produces a hash value for each fixed-size block based on its content and based on the hash value for the previous block
  - ❖ "Avalanche" effect (1-bit change in input produces "catastrophic changes in output")
- In fact, can use symmetric encryption as  $f=E$ , and use  $M_i$  as the key

4

# Simple Hash Functions

➤ Bitwise-XOR

	bit 1	bit 2	• • •	bit n
block 1	b <sub>11</sub>	b <sub>21</sub>		b <sub>n1</sub>
block 2	b <sub>12</sub>	b <sub>22</sub>		b <sub>n2</sub>
	•	•	•	•
	•	•	•	•
	•	•	•	•
block m	b <sub>1m</sub>	b <sub>2m</sub>		b <sub>nm</sub>
hash code	C <sub>1</sub>	C <sub>2</sub>		C <sub>n</sub>

Figure 3.3 Simple Hash Function Using Bitwise XOR

- Not very secure, e.g., for English text (ASCII<128) the high-order bit is almost always zero
- Can be improved by rotating the hash code after each block is XOR-ed into it
- If message itself is not encrypted, it is easy to modify the message and append one block that would set the hash code as needed

# The Birthday Paradox

- ❖ Example hash function:  $y=H(x)$
- ❖ where:  $x$ =person and  $H()$  is Bday()
- ❖  $y$  ranges over set  $Y=[1...365]$ , let  $n$  = size of  $Y$ , i.e., number of distinct values in the range of  $H()$
- ❖ how many people do we need to 'hash' to have a collision?
- ❖ what is the probability of selecting at random  $k$  DISTINCT numbers from  $Y$ ?
- ❖  $P_0=1*(1-1/n)*(1-2/n)*...*(1-(k-1)/n) == e^{(k(1-k)/2n)}$
- ❖  $P_1=1-P_0$  ----> probability of at least one collision
- ❖ Set  $P_1$  to be at least 0.5 and solve for  $k$
- ❖  $k == 1.17 * \text{SQRT}(n)$
- ❖  $k = 22.3$  for  $n=365$

**So, what's the point?**

## The Birthday Paradox

$m = \log(n) = \text{size of } H()$

$\sqrt{2^m} = 2^{m/2}$  trials must  
be infeasible!

7

## How long should a hash be?

- Many input messages yield the same hash
  - ❖ E.g., 1024-bit message, 128-bit hash
  - ❖ On average,  $2^{896}$  messages map into one hash
- With  $m$ -bit hash, it takes about  $2^{m/2}$  trials to find a collision (with  $\geq 50\%$  probability)
- When  $m=64$ , it takes  $2^{32}$  trials to find a collision (do-able in very little time)
- Today, need at least  $m=128$ , requiring about  $2^{64}$  trials

8

## Some Popular (Secure) HASH functions

	SHA-1	MD5	RIPEMD-160
Digest length	160 bits	128 bits	160 bits
Block size	512 bits	512 bits	512 bits
# of steps	80 (4 rounds of 20)	64 (4 rounds of 16)	160 (5 paired rounds of 16)
Max message size	$2^{64}-1$ bits	$\infty$	$\infty$

9

## MD2

- Rivest, 1992
- 128-bit message digest
- Especially designed for application use (such as email)
- Arbitrary-length input message (in bytes)
  1. First pad to multiple of 16 bytes
  2. Append MD2 checksum (16 bytes) to the end  
The checksum is almost a hash, but not cryptographically secure yet..
  3. Final pass: process whole message, 16 bytes at a time

10

## MD2 Padding

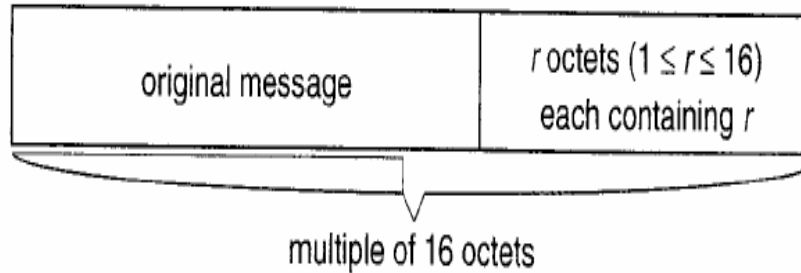


Figure 5-3. MD2 Padded Message

- If message length is a multiple of 16, 16 bytes are added
- Otherwise, between 1 and 15 bytes are added

11

## MD2 Checksum

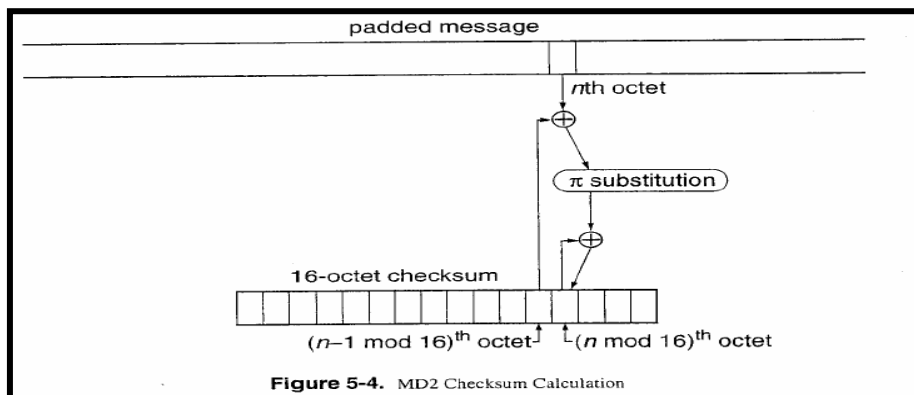
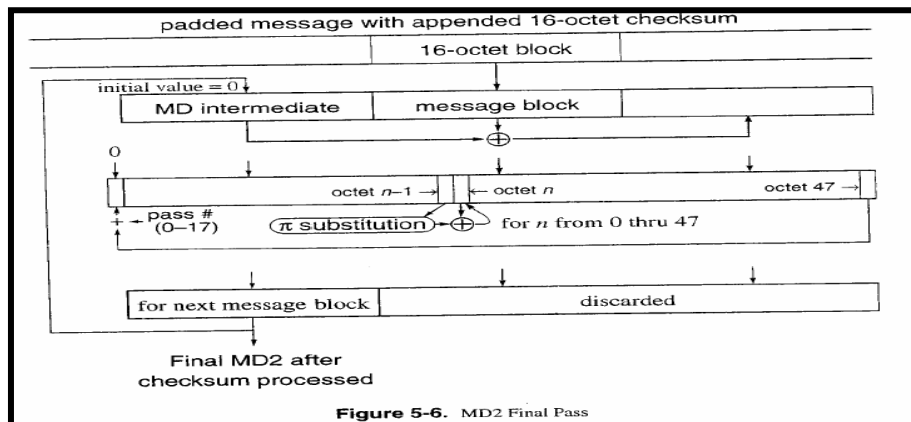


Figure 5-4. MD2 Checksum Calculation

- One byte at a time,  $k \times 16$  steps
- $m_{nk}$ : byte  $nk$  of message,  $n: 1 \sim 16$
- $c_n = \pi(m_{nk} \oplus c_{n-1}) \oplus c_n$
- $\pi: 0 \rightarrow 41, 1 \rightarrow 46, \dots$ 
  - ❖ Substitution on 0-255 (value of the byte)

12

## MD2 Final Process



- Operate on 16-byte chunks
- 48-byte quantity  $q$ .
  - ❖ (current digest|chunk|digest@chunk)
- 18 passes of massaging over  $q$ , and one byte at a time:
  - ❖  $c_i = \pi(c_{n,i}) \oplus c_n$  for  $n = 0, \dots, 47$ ;  $c_i = 0$  for pass 0;  $c_i = (c_{47} + \text{pass \#}) \bmod 256$
- After pass 17, use first 16 bytes as new digest
  - ❖  $16 \times 8 = 128$
- Repeat till the entire padded message is processed

13

## MD5

- Rivest, 1992
- 128-bit hash
  - based on earlier, weaker MD4, 1990
- Simplified versions have been crypto-analyzed but not MD5 itself
- But: strong collision resistance (B-day attack resistance) only 64-bit
- Output length not really long enough today

14

# MD5: Message Digest Version 5

Input message



Output: 128-bit digest

# Overview of MD5

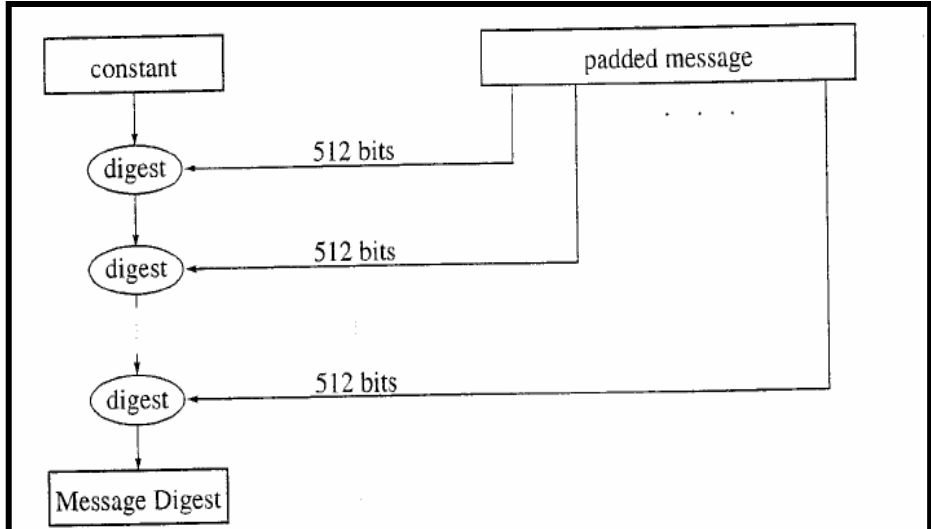
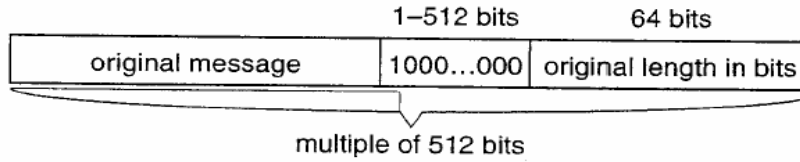


Figure 5-8. Overview of MD4, MD5, SHA-1

## MD5 Padding

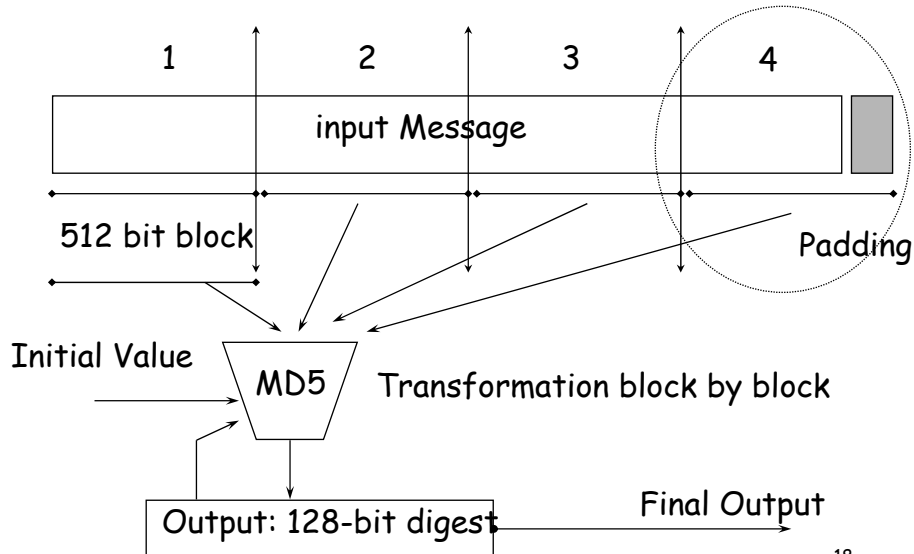


**Figure 5-7.** Padding for MD4, MD5, SHA-1

- Given original message  $M$ , add padding bits "100..." such that resulting length is 64 bits less than a multiple of 512 bits.
- Append (*original length in bits mod  $2^{64}$* ), represented in 64 bits to the padded message
- Final message is chopped into 512-bit blocks

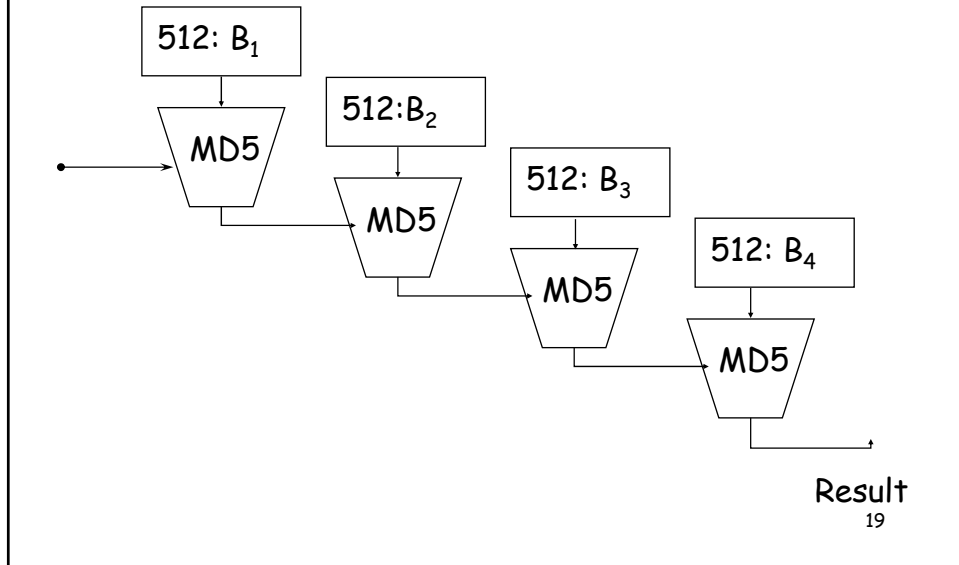
17

## MD5: Padding

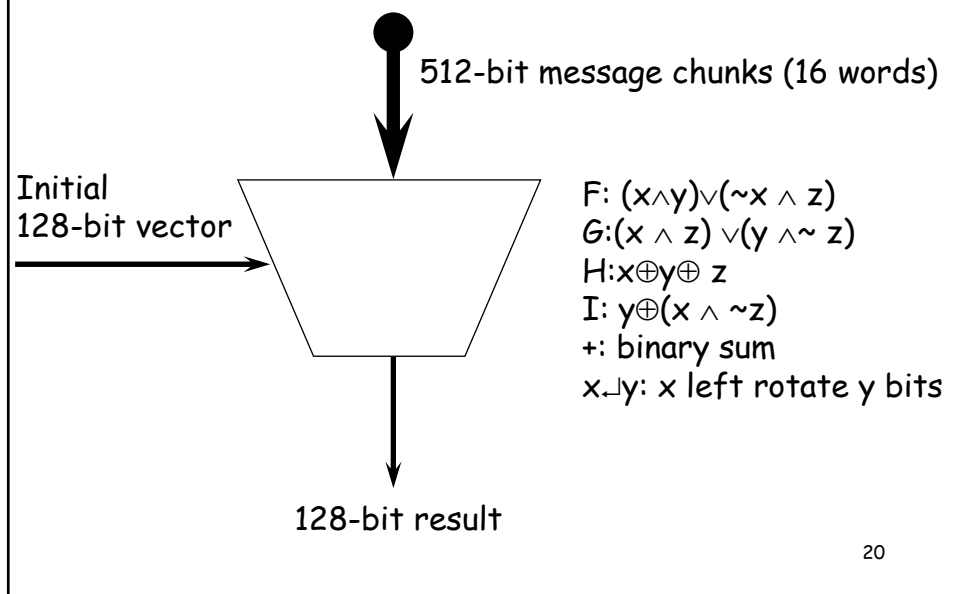


18

## MD5 Blocks



## MD5 Box

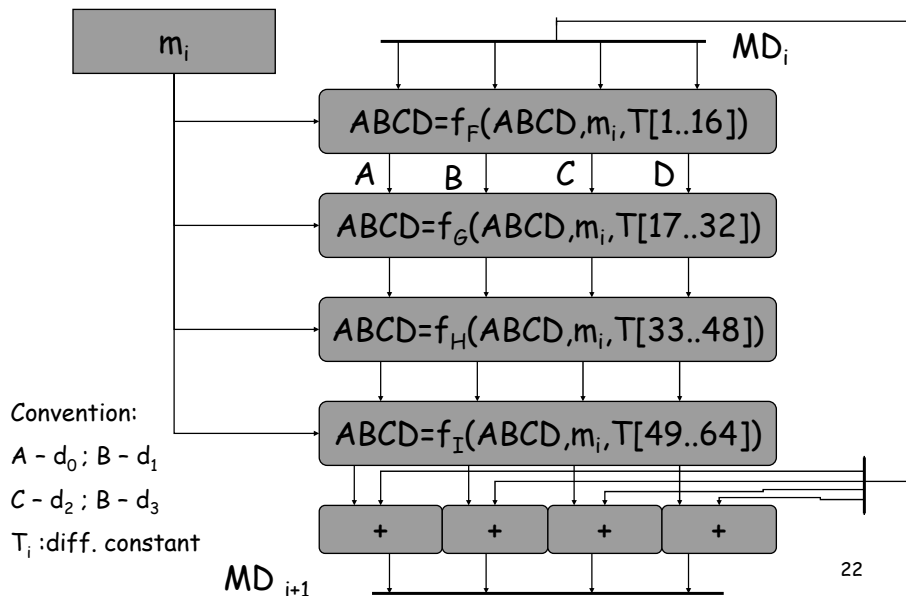


## MD5 Process

- As many stages as the number of 512-bit blocks in the final padded message
- Digest: 4 32-bit words: MD=A|B|C|D
- Every message block contains 16 32-bit words:  $m_0|m_1|m_2\dots|m_{15}$ 
  - ❖ Digest MD<sub>0</sub> initialized to:  
A=01234567, B=89abcdef, C=fedcba98, D=76543210
  - ❖ Every stage consists of 4 passes over the message block, each modifying MD; each pass involves different operation

21

## Processing of Block $m_i$ - 4 Passes



## Different Passes...

- Different functions and constants are used
- Different set of  $m_i$  is used
- Different set of shift amount is used

23

## Functions and Random Numbers

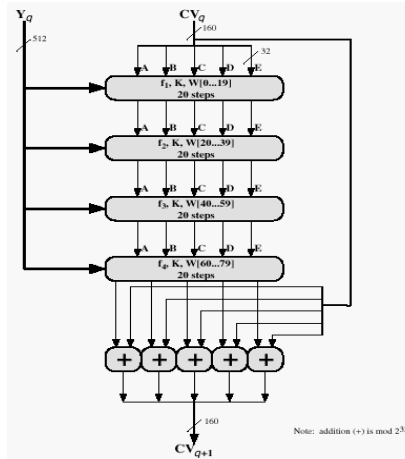
- $F(x,y,z) == (x \wedge y) \vee (\sim x \wedge z)$
- $G(x,y,z) == (x \wedge z) \vee (y \wedge \sim z)$
- $H(x,y,z) == x \oplus y \oplus z$
- $I(x,y,z) == y \oplus (x \wedge \sim z)$
- $T_i = \text{int}(2^{32} * \text{abs}(\sin(i))), 0 < i < 65$

24

# Secure Hash Algorithm (SHA)

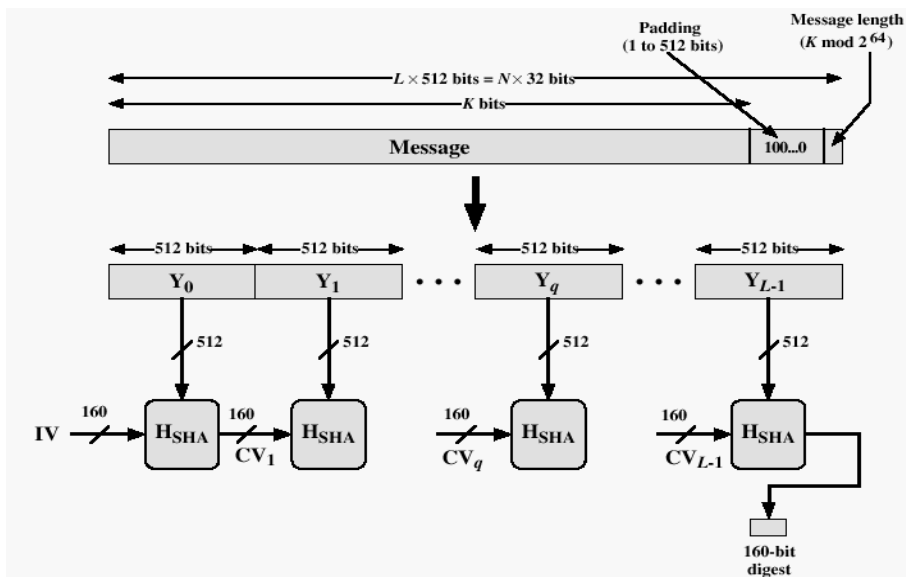
➤ SHA was published by NIST as a standard in 1993

- Revised in 1995 as SHA-1
  - ❖ Input: Up to  $2^{64}$  bits
  - ❖ Output: 160 bit digest
  - ❖ 80-bit collision resistance
- Pad with at least 64 bits to resist padding attack
  - ❖  $1000\dots 0 < \text{message length}$
- Processes 512-bit block
  - ❖ Initiate 5x32bit MD registers
  - ❖ Apply compression function
    - 4 rounds of 20 steps each
    - each round uses different non-linear  $f_i$
    - registers are shifted and switched

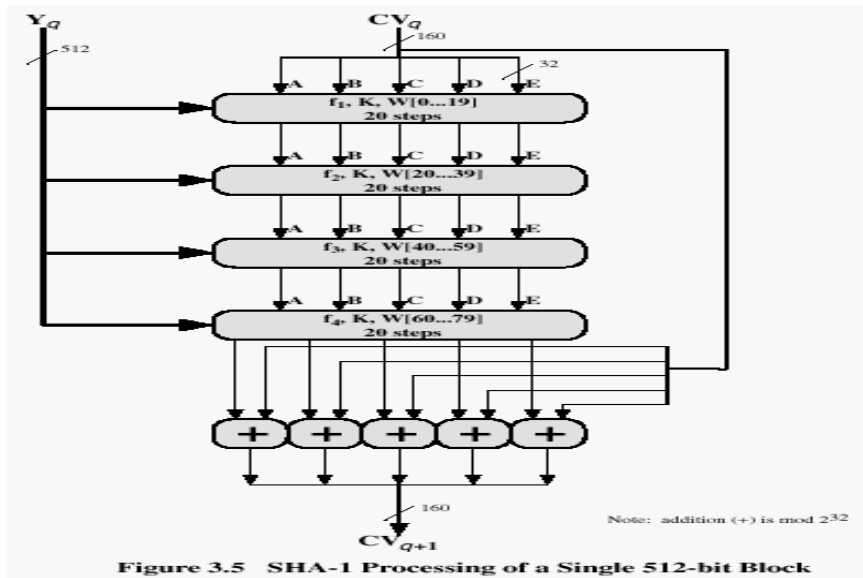


25

# Digest Generation with SHA-1



## SHA-1 of a 512-Bit Block



## General Logic

- Input message must be  $< 2^{64}$  bits
  - ❖ not really a problem
- Message is processed in 512-bit blocks sequentially
- Message digest is 160 bits
- SHA design is similar to MD5, but a lot stronger

## Basic Steps

Step1: Padding

Step2: Appending length as 64 bit unsigned

Step3: Initialize MD buffer 5 32-bit words

A|B|C|D|E

A = 67452301

B = efcdab89

C = 98badcfe

D = 10325476

E = c3d2e1f0

29

## Basic Steps...

Step 4: the 80-step processing of 512-bit blocks: 4 rounds, 20 steps each

Each step  $t$  ( $0 \leq t \leq 79$ ):

❖ Input:

➤  $W_t$  - a 32-bit word from the message

➤  $K_t$  - a constant

➤ ABCDE: current MD

❖ Output:

➤ ABCDE: new MD

30

## Basic Steps...

- Only 4 per-round distinctive additive constants

$0 \leq t \leq 19$   $K_t = 5A827999$

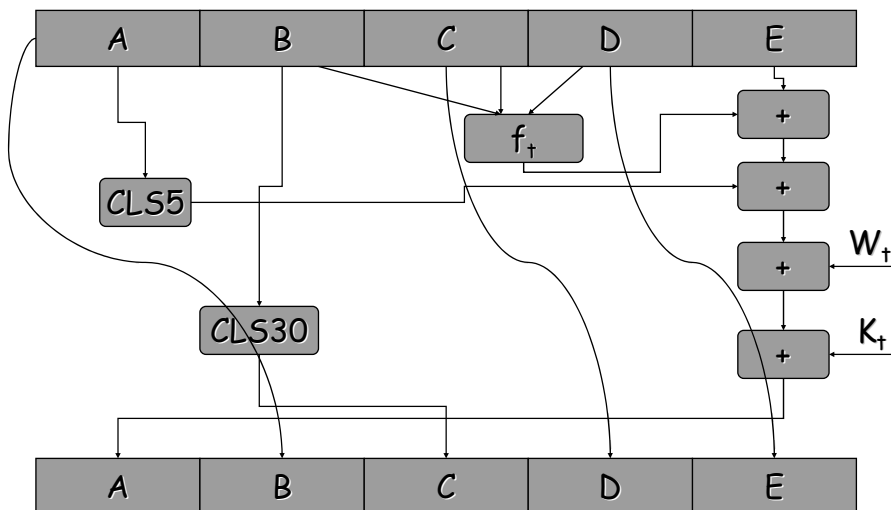
$20 \leq t \leq 39$   $K_t = 6ED9EBA1$

$40 \leq t \leq 59$   $K_t = 8F1BBCDC$

$60 \leq t \leq 79$   $K_t = CA62C1D6$

31

## Basic Steps - The Heart Of The Matter



## Basic Logic Functions

- Only 3 different functions

Round	Function $f_t(B,C,D)$
$0 \leq t \leq 19$	$(B \wedge C) \vee (\sim B \wedge D)$
$20 \leq t \leq 39$	$B \oplus C \oplus D$
$40 \leq t \leq 59$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$60 \leq t \leq 79$	$B \oplus C \oplus D$

33

## Twist With $W_t$ 's

- Additional mixing used with input message 512-bit block

$$W_0 | W_1 | \dots | W_{15} = m_0 | m_1 | m_2 \dots | m_{15}$$

For  $15 < t < 80$ :

$$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$$

- XOR is a very efficient operation, but with multilevel shifting, it should produce very extensive and random mixing!

34

## SHA Versus MD5

- SHA is a stronger algorithm:
  - ❖ Brute-force birthday attack requires on the order of  $2^{80}$  operations, in contrast to  $2^{64}$  for MD5
- SHA's 80 steps and 160 bits hash (vs. 128) require a little more computation

35

## RIPE-MD

- RIPE-MD developed in Europe (1996-7), funded by EC
- RIPEMD-160: 160-bit hash, 512-bit blocks (same as SHA-1)
- Comparable to SHA-1 in speed, security
  
- SHA and RIPE-MD are about half the speed of MD5
- American standard is SHA-1 (for now)
- SHA-128, SHA-256 match key lengths in AES

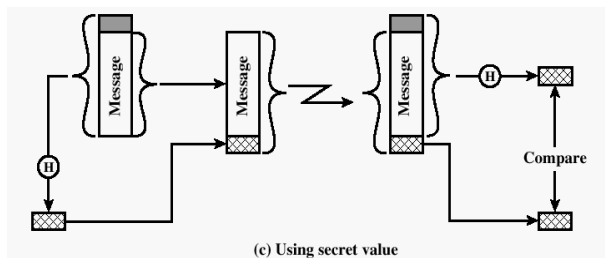
36

# What are hash functions good for?

37

## Message Authentication Protocol Using a Hash Function

### 1. Using a symmetric key



### 2. Using symmetric encryption (flawed!)

- Generate  $H(M)$  of preferably same size as  $E()$  block
- Use  $E_K(H(M))$  as the MAC

**Collision → MAC forgery!**

38

## Using Hash for Authentication

- Alice to Bob: random challenge  $r_A$
- Bob to Alice:  $H(K_{AB} | r_A)$
- Bob to Alice: random challenge  $r_B$
- Alice to Bob:  $H(K_{AB} | r_B)$
- Only need to compare  $H()$  results

39

## Using Hash to Compute MAC: integrity

- **Cannot just compute and append  $H(m)$**
- **Need "Keyed Hash":**
  - ❖ **Prefix:**
    - **MAC:**  $H(K_{AB} | m)$ , almost works, but...
    - **Allows concatenation with arbitrary message:**  
 $H(K_{AB} | m | m')$
  - ❖ **Suffix:**
    - **MAC:**  $H(m | K_{AB})$ , works better, but what if  $m'$  is found such that  $H(m) = H(m')$ ?
  - ❖ **HMAC:**
    - $H(K_{AB} | H(K_{AB} | m))$

40

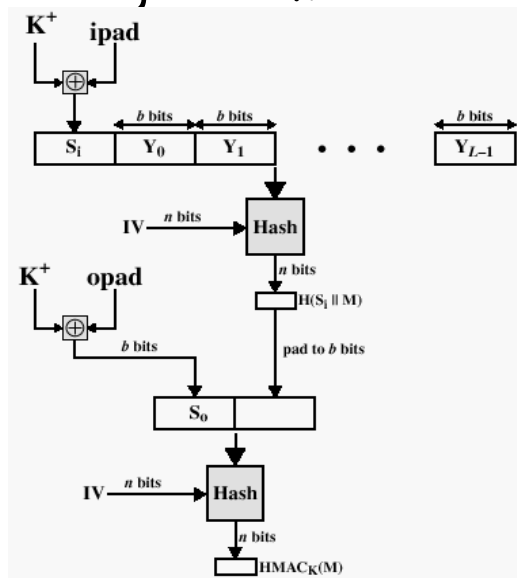
# Hash Function MAC (HMAC)

- **Main Idea:** Use a MAC derived from any cryptographic hash function
  - ❖ Note that hash functions do not use a key, and therefore cannot serve directly as a MAC
- **Motivations for HMAC:**
  - ❖ Cryptographic hash functions execute faster in software than encryption algorithms such as DES
  - ❖ No need for the reverseability of encryption
  - ❖ No export restrictions from the US (was important in the past)
- **Status:** designated as mandatory for IP security
  - ❖ Also used in Transport Layer Security (TLS), which will replace SSL, and in SET

41

# HMAC Algorithm

- Compute  $H_1 = H()$  of the concatenation of  $M$  and  $K_1$
- To prevent an "additional block" attack, compute again  $H_2 = H()$  of the concatenation of  $H_1$  and  $K_2$
- $K_1$  and  $K_2$  each use half the bits of  $K$
- Notation:
  - ❖  $K^+ = K$  padded with 0's
  - ❖  $ipad = 00110110 \times b/8$
  - ❖  $opad = 01011100 \times b/8$
- Execution:
  - ❖ Same as  $H(M)$ , plus 2 blocks



42

## Using Hash to Encrypt : confidentiality

### ➤ (Almost) One-time pad: similar to OFB

❖ compute bit streams using  $H()$ ,  $K$ , and  $IV$

➤  $b_1 = H(K_{AB} \mid IV)$ , ...,  $b_i = H(K_{AB} \mid b_{i-1})$ , ...

➤  $c_i = p_i \oplus b_i$ , ...,  $c_i = p_i \oplus b_i$ , ...

### ➤ Or mix in the plaintext

❖ similar to cipher feedback mode (CFB)

➤  $b_1 = H(K_{AB} \mid IV)$ , ...,  $b_i = H(K_{AB} \mid c_{i-1})$ , ...

➤  $c_i = p_i \oplus b_i$ , ...,  $c_i = p_i \oplus b_i$ , ...