Lecture 6

Cryptographic Hash Functions

1

Cryptographic HASH Functions

- Purpose: produce a fixed-size "fingerprint" or digest of arbitrarily long input data
- Why? To guarantee integrity
- Properties of a "good" cryptographic HASH function H():
  1. Takes on input of any size
  2. Produces fixed-length output
  3. Easy to compute (efficient)
  4. Given any h, computationally infeasible to find any x such that
     H(x) = h
  1. For any x, computationally infeasible to find y such that
     H(y) = H(x) and y<>x
  1. Computationally infeasible to find any (x, y) such that
     H(x) = H(y) and x<>y

2

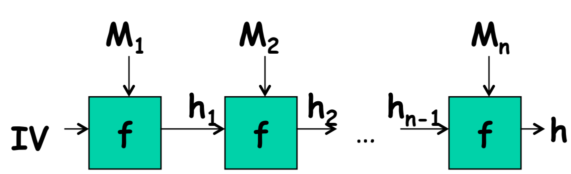
Same properties re-stated:

- 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)

Construction

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



- Sort of like a Chained Block Cipher
  - Produces a hash value for each fixed-size block based on (1) its content and (2) hash value for the previous block
  - "Avalanche" effect: 1-bit change in input produces "catastrophic" changes in output

Simple Hash Functions

- Bitwise-XOR

	bit 1	bit 2	• • •	bit *n*
block 1	b_{11}	b_{21}		b_{n1}
block 2	b_{12}	b_{22}		b_{n2}
	• • •	• • •	• • •	• • •
block *m*	b_{1m}	b_{2m}		b_{nm}
hash code	C_1	C_2		C_n

Figure 3.3 Simple Hash Function Using Bitwise XOR

- Not 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

5

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?
- Or: what is the probability of selecting at random k DISTINCT numbers from Y?

- $P0=1*(1-1/n)*(1-2/n)*…*(1-(k-1)/n)) == e^{(k(1-k)/2n)}$
- P1=1-P0 ---> probability of at least one collision
- Set P1 to be at least 0.5 and solve for k
- k == 1.17 * SQRT(n)
- k = 22.3 for n=365

So, what's the point?

6

The Birthday Paradox

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

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

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 >=50% probability)
- When m=64, it takes 2^{32} trials to find a collision (doable in very little time)
- Today, need at least m=160, requiring about 2^{80} trials

8

Hash Function Examples

	SHA-1 (or SHA-160)	MD5 (defunct)	RIPEMD-160 (unloved) ☺
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	∞	∞

Other (stronger) variants of SHA are SHA-256 and SHA-512

See: http://en.wikipedia.org/wiki/SHA_hash_functions

9

MD5

- Author: R. Rivest, 1992
- 128-bit hash

    based on earlier, weaker MD4 (1990)

- **Collision resistance (B-day attack resistance)**

    only 64-bit

- Output size not long enough today (due to various attacks)

10

MD5: Message Digest Version 5

Input message

Output: 128-bit digest

11

Overview of MD5

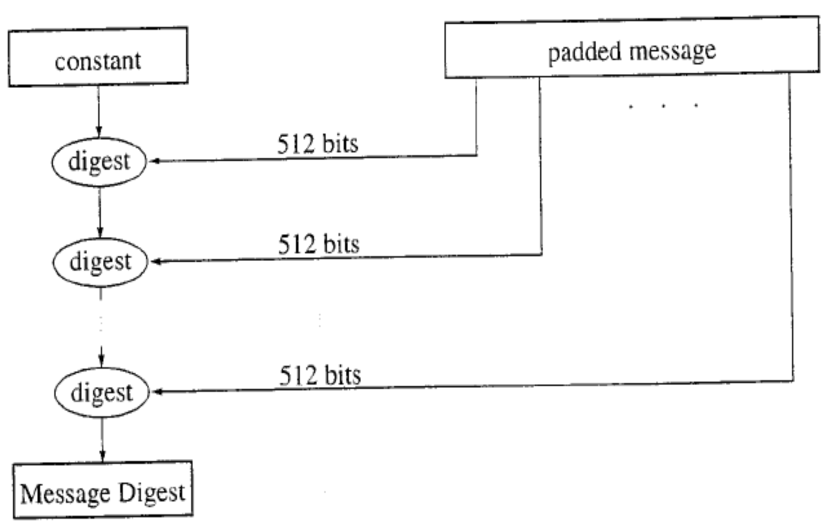


Figure 5-8. Overview of MD4, MD5, SHA-1

MD5 Padding

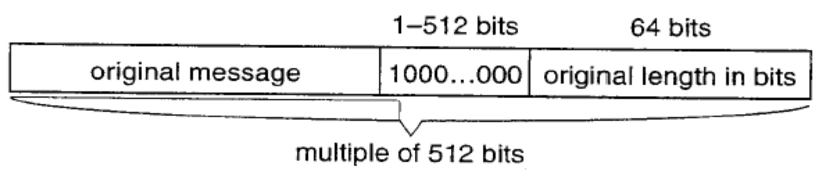
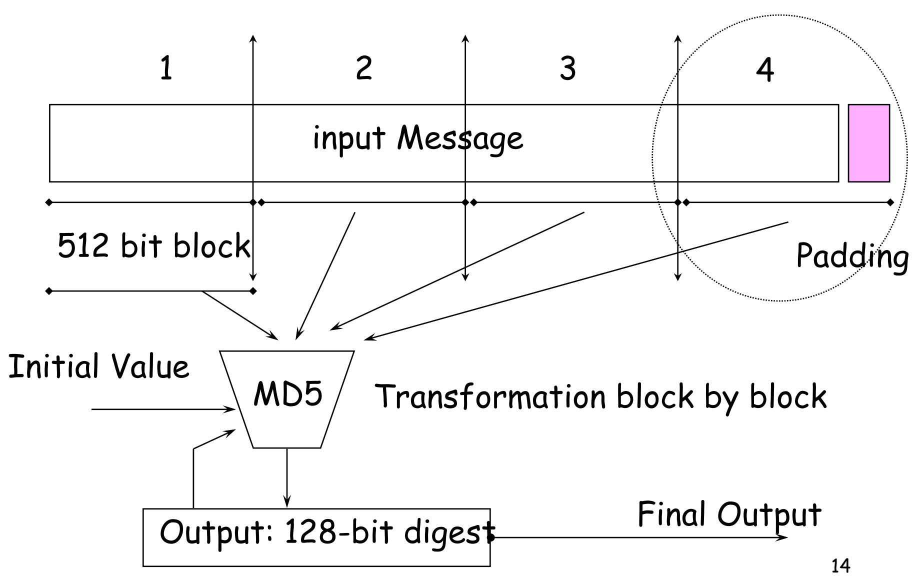


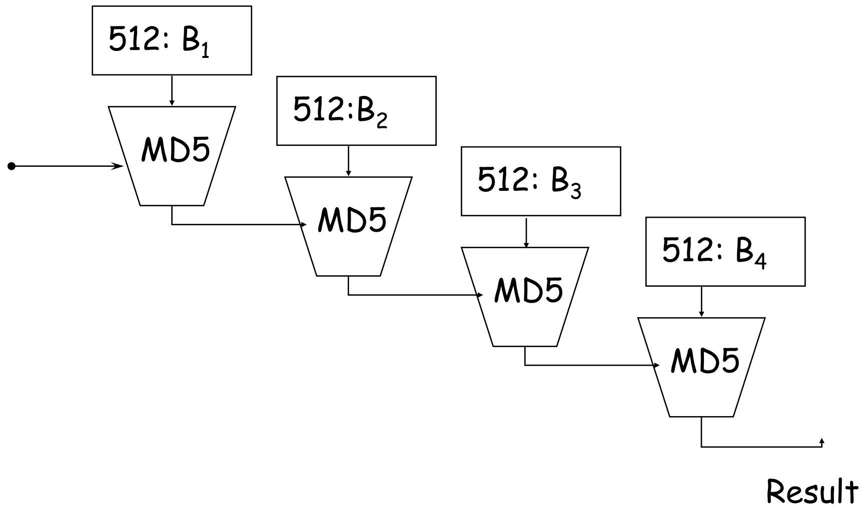
Figure 5-7. Padding for MD4, MD5, SHA-1

- 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* to the padded message
- Final message chopped into 512-bit blocks

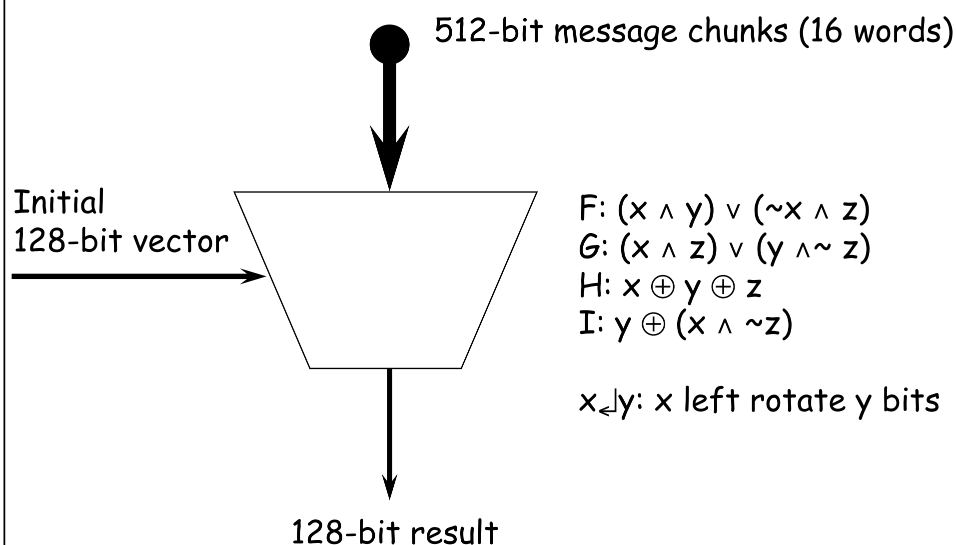
MD5: Padding



MD5 Blocks



MD5 Box



F: $(x \wedge y) \vee (\sim x \wedge z)$
G: $(x \wedge z) \vee (y \wedge \sim z)$
H: $x \oplus y \oplus z$
I: $y \oplus (x \wedge \sim z)$

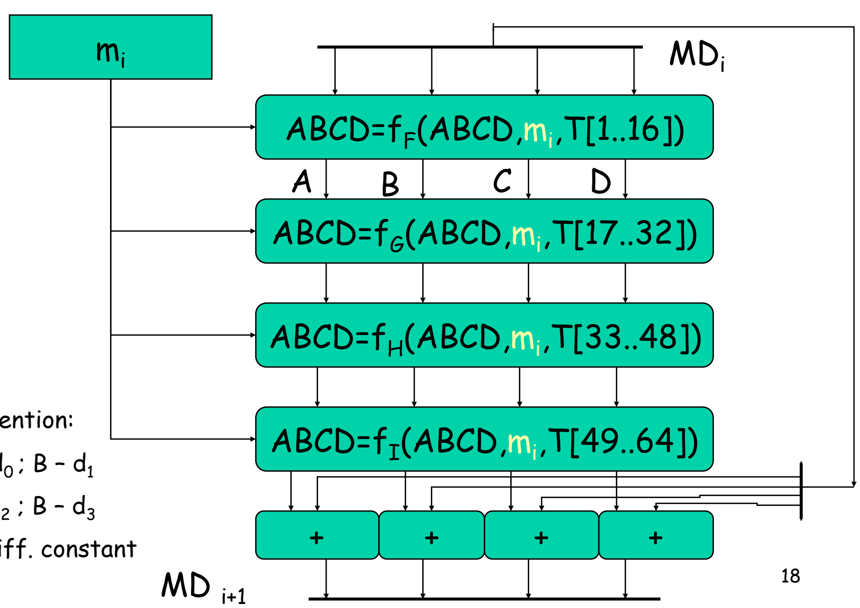
$x \hookleftarrow y$: x left rotate y bits

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: $m_0|m_1|m_2...|m_{15}$
  - Digest MD_0 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

17

Processing of Block m_i - *4 Passes*



Convention:

A – d_0 ; B – d_1

C – d_2 ; B – d_3

T_i :diff. constant

18

Different Passes...

- Different functions and constants
- Different set of m_i-s
- Different sets of shifts

19

Functions and Random Numbers

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

20

Secure Hash Algorithm (SHA)

- SHA-0 was published by NIST 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…0 || <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 function
    - registers are shifted and switched

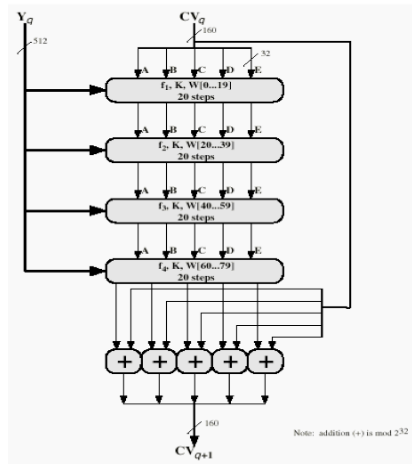
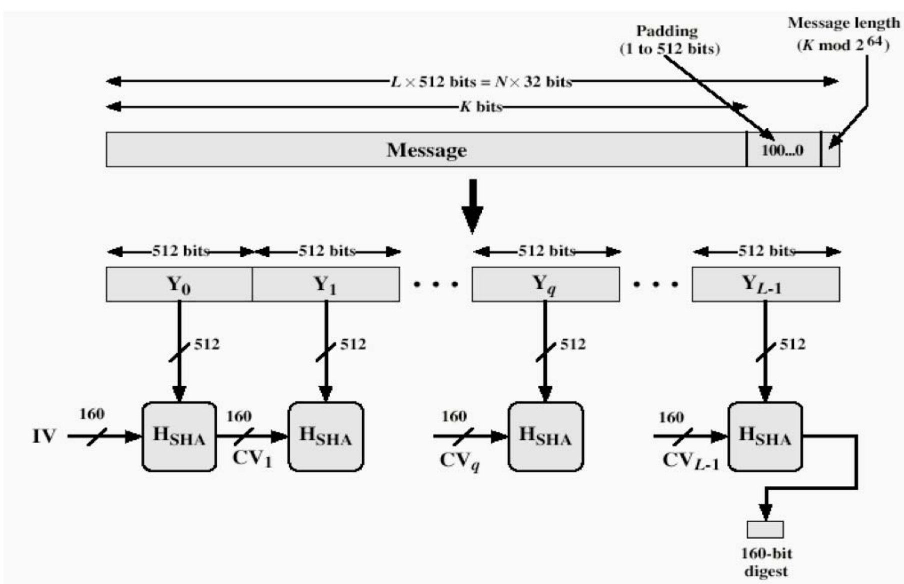


Figure 3.5 SHA-1 Processing of a Single 512-bit Block

21

Digest Generation with SHA-1



SHA-1 of a 512-Bit Block

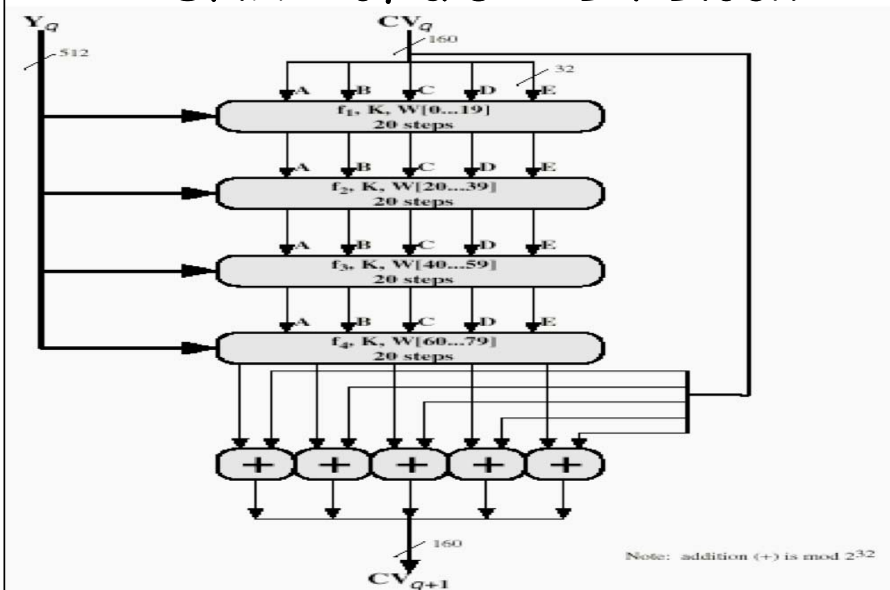


Figure 3.5 SHA-1 Processing of a Single 512-bit Block

General Logic

- Input message must be < 2^{64} bits
  - not a realistic limitation
- Message processed in 512-bit blocks sequentially
- Message digest (hash) is 160 bits
- SHA design is similar to MD5, but a lot stronger

24

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

25

Basic Steps...

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

Each step t (0 <= t <= 79):

- ❖Input:
  - ➢W_t – 32-bit word from the message
  - ➢K_t – constant
  - ➢ABCDE: current MD
- ❖Output:
  - ➢ABCDE: new MD

26

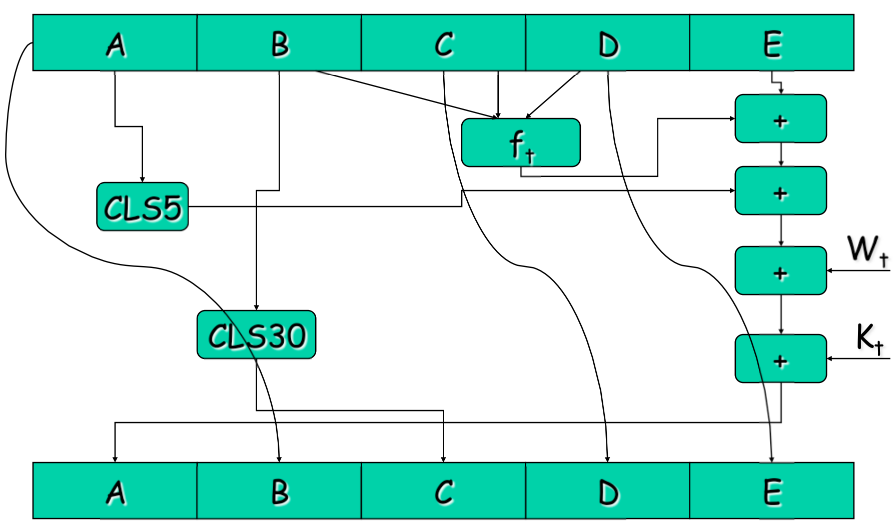
Basic Steps...

- Only 4 per-round distinctive additive constants:

0 <= t <= 19	K_t = 5A827999
20<=t<=39	K_t = 6ED9EBA1
40<=t<=59	K_t = 8F1BBCDC
60<=t<=79	K_t = CA62C1D6

27

Basic Steps – Zooming in



Basic Logic Functions

- Only 3 different functions

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

29

Twist With W_t's

- Additional mixing used with input message 512-bit block

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

  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 produces very extensive and random mixing!

30

SHA Versus MD5

- SHA is a stronger algorithm:
  - A birthday attack requires on the order of 2^{80} operations, in contrast to 2^{64} for MD5
- SHA has 80 steps and yields a 160-bit hash (vs. 128) - involves more computation

31

What are hash functions

good for?

32

Message Authentication Using a Hash Function

Use symmetric encryption such as AES or 3-DES

- Generate H(M) of same size as E() block
- Use $E_K(H(M))$ as the MAC (instead of, say, DEC MAC)
- Alice sends $E_K(H(M))$, M
- Bob receives C,M' decrypts C with k, hashes result

$$H(D_K(C)) =?= H(M')$$

Collision → MAC forgery!

33

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

34

Using Hash to Compute MAC: integrity

- **Cannot just compute and append H(m)**
- **Need "Keyed Hash":**
  - **Prefix:**
    - **MAC: $H(K_{AB} \mid m)$, almost works, but…**
    - **Allows concatenation with arbitrary message:**

      $$H(\ K_{AB} \mid m \mid m'\ )$$
  - **Suffix:**
    - **MAC: $H(m \mid K_{AB})$, works better, but what if m' is found such that H(m)=H(m')?**
  - **HMAC:**
    - **$H\ (\ K_{AB}\ \mid\ H\ (K_{AB}\ \mid m)\ )$**

35

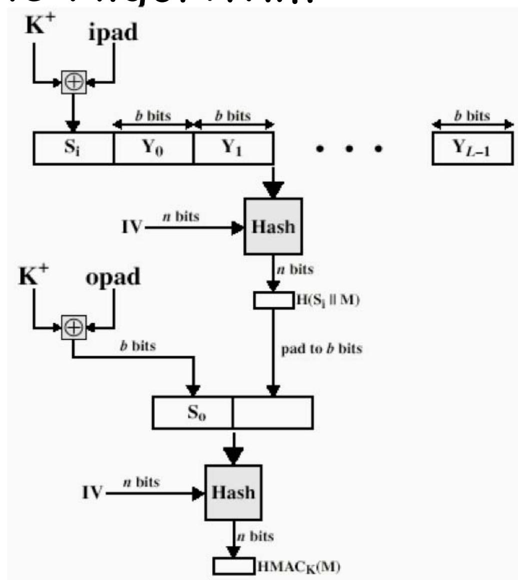
Hash Function MAC (HMAC)

- **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 US government export restrictions (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

36

HMAC Algorithm

- Compute H1 = H() of the concatenation of M and K1
- To prevent an "additional block" attack, compute again H2= H() of the concatenation of H1 and K2
- K1 and K2 each use half the bits of K
- Notation:
  - K^+ = K padded with 0's
  - ipad=00110110 x b/8
  - opad=01011100 x b/8
- Execution:
  - Same as H(M), plus 2 blocks



37

Just for fun…
Using a Hash to Encrypt

- (Almost) One-time pad: similar to OFB
  - compute bit streams using H(), K, and IV
    - b_1=H(K_{AB} | IV) , …, b_i=H(K_{AB} | b_{i-1}), …
    - c_1= $p_1 \oplus b_1$, … , c_i= $p_i \oplus b_i$, …
- Or, mix in the plaintext
  - similar to cipher feedback mode (CFB)
    - b_1=H(K_{AB} | IV), …, b_i=H(K_{AB} | c_{i-1}), …
    - c_1= $p_1 \oplus b_1$, … , c_i= $p_i \oplus b_i$, …

38