MACs Based on Hash Functions: HMAC

- There has been increased interest in developing a MAC derived from a cryptographic hash function
- Motivations:
  - Cryptographic hash functions such as MD5 and SHA generally execute faster in software than symmetric block ciphers such as DES
  - Library code for cryptographic hash functions is widely available

- HMAC has been chosen as the mandatory-to-implement MAC for IP security

- Has also been issued as a NIST standard (FIPS 198)

HMAC Design Objectives

- RFC 2104 lists the following objectives for HMAC:
  - To use, without modifications, available hash functions
  - To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required
  - To preserve the original performance of the hash function without incurring a significant degradation
  - To use and handle keys in a simple way
  - To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function

HMAC Algorithm

- **HMAC Algorithm**
- Figure 12.5 illustrates the overall operation of HMAC. Define the following terms.
- *H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)*
- *IV = initial value input to hash function*
- *M = message input to HMAC (including the padding specified in the embedded* hash function)
- *Yi = i th block of M, 0 … i … (L - 1)*
- *L = number of blocks in M*
- *b = number of bits in a block*
- *n = length of hash code produced by embedded hash function*
- *K = secret key; recommended length is Ú n; if key length is greater than b, the* key is input to the hash function to produce an *n-bit key*
- *K^+ = K padded with zeros on the left so that the result is b bits in length*
- ipad = 00110110 (36 in hexadecimal) repeated *b/8 times*
- opad = 01011100 (5C in hexadecimal) repeated *b/8 times*
- Then HMAC can be expressed as

$$\mathrm{HMAC}(K, M) = \mathrm{H}[(K^+ \oplus \mathrm{opad}) \,\|\, \mathrm{H}[(K^+ \oplus \mathrm{ipad}) \,\|\, M]]$$

HMAC Structure

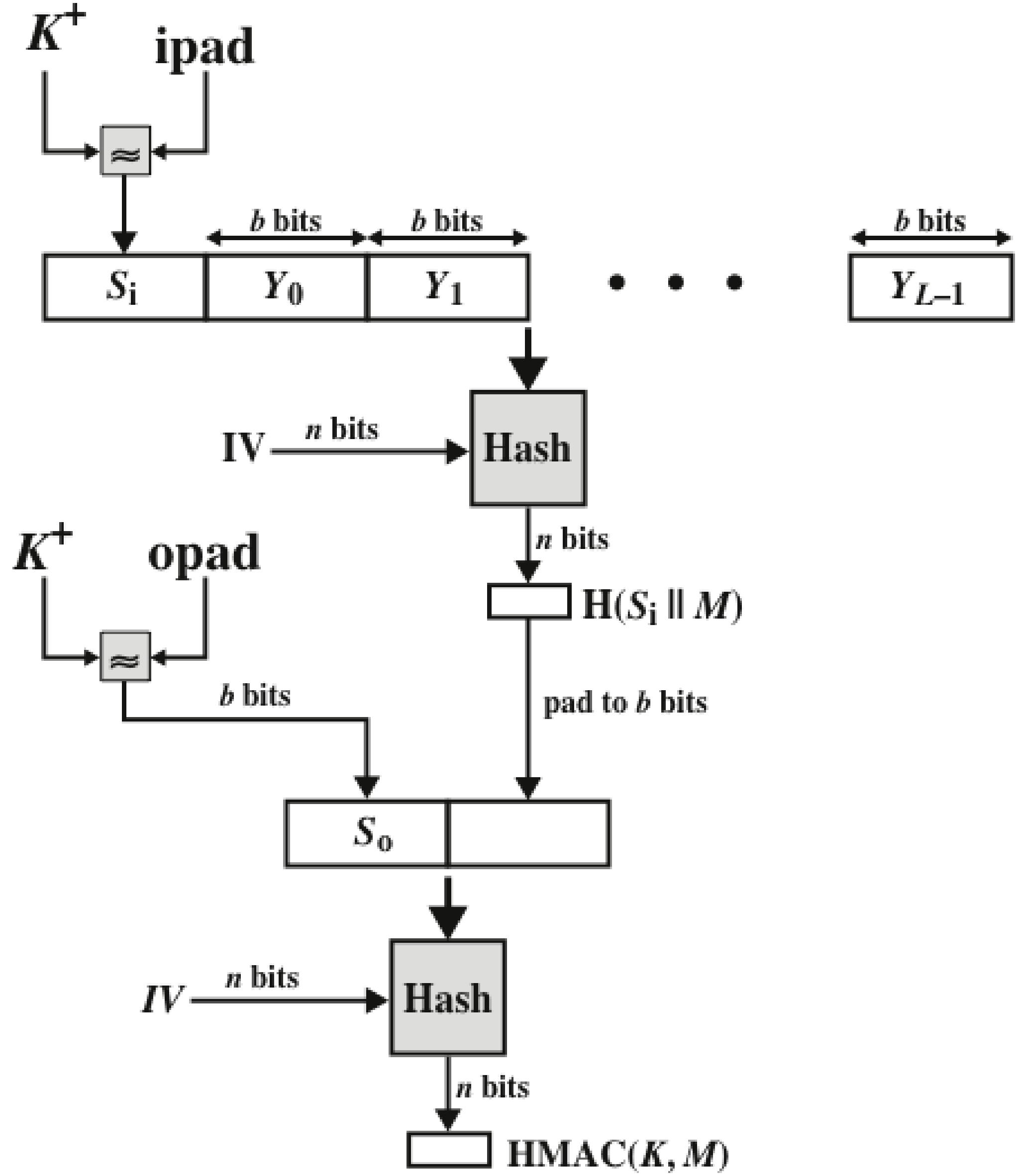
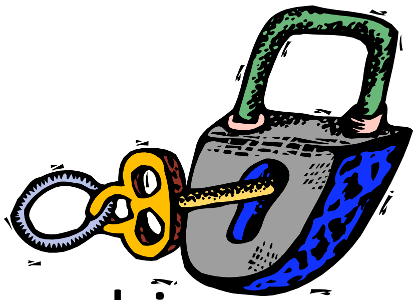


Figure 12.5 HMAC Structure

- We can describe the algorithm as follows.
- **1. Append zeros to the left end of *K to create a b-bit string K+ (e.g., if K is of*** length 160 bits and *b = 512, then K will be appended with 44 zeroes).*
- **2. XOR (bitwise exclusive-OR) *K+ with ipad to produce the b-bit block Si.***
- **3. Append *M to Si.***
- **4. Apply H to the stream generated in step 3.**
- **5. XOR *K+ with opad to produce the b-bit block So.***
- **6. Append the hash result from step 4 to *So.***
- **7. Apply H to the stream generated in step 6 and output the result.**

Security of HMAC



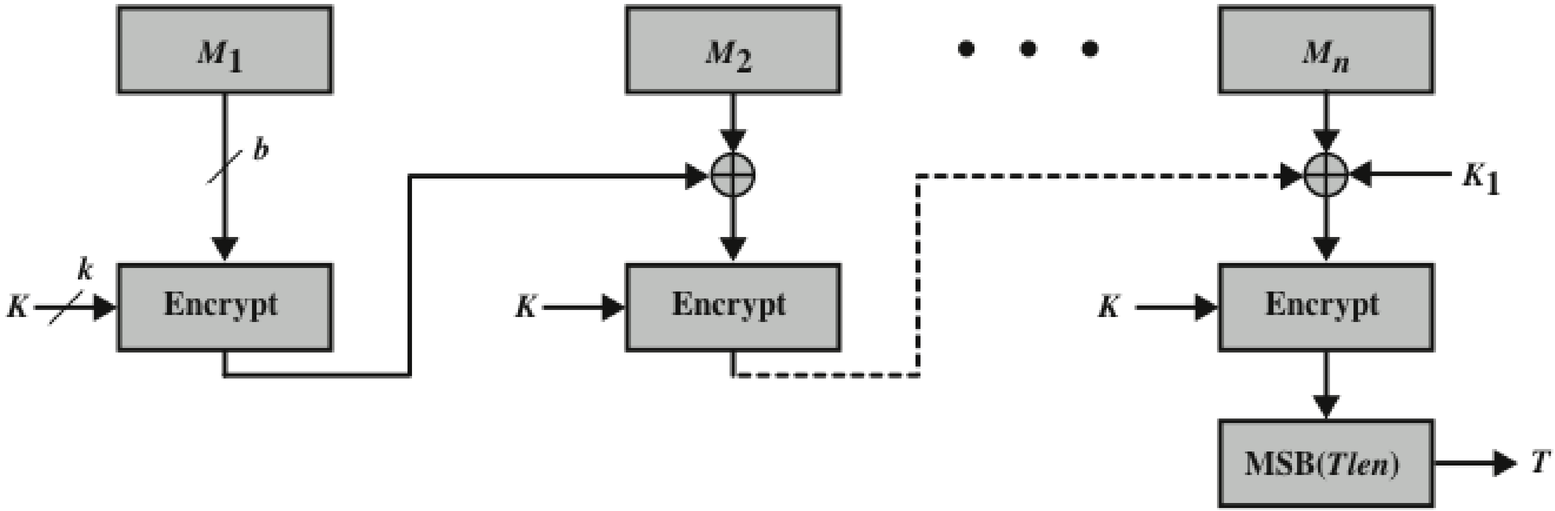
- Depends in some way on the cryptographic strength of the underlying hash function
- Appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC
- Generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message-tag pairs created with the same key

Cipher Block based MAC(CMAC)

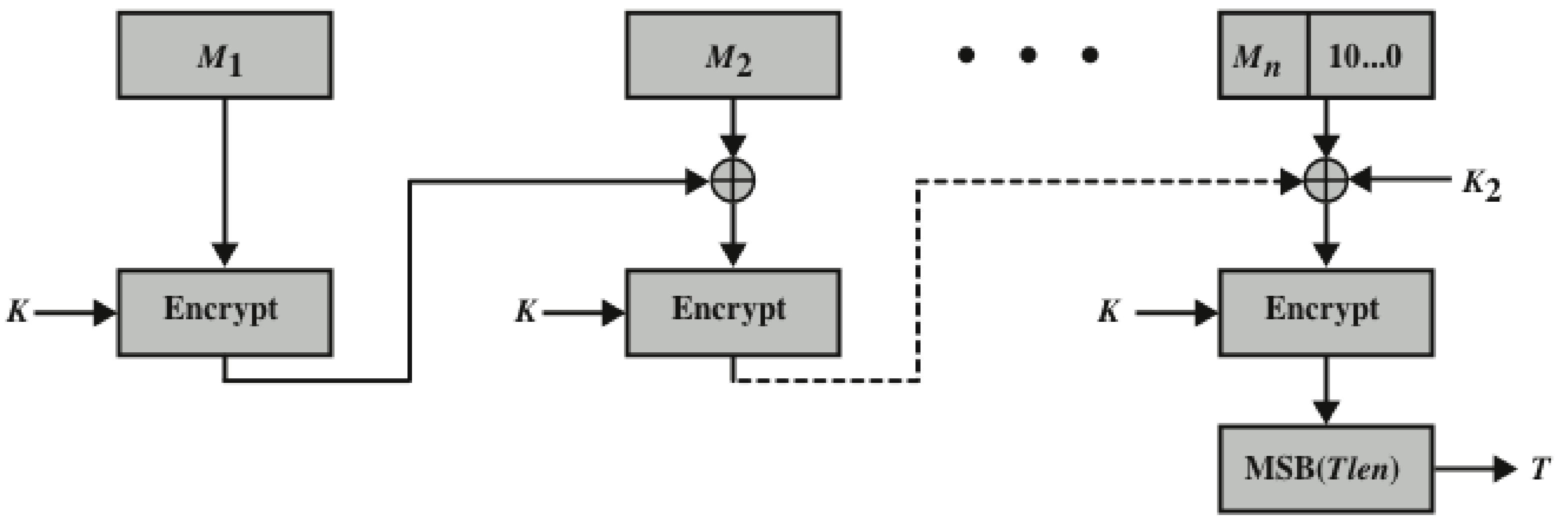
$$
\begin{aligned}
C_1 &= \mathrm{E}(K, M_1) \\
C_2 &= \mathrm{E}(K, [M_2 \oplus C_1]) \\
C_3 &= \mathrm{E}(K, [M_3 \oplus C_2]) \\
&\cdot \\
&\cdot \\
&\cdot \\
C_n &= \mathrm{E}(K, [M_n \oplus C_{n-1} \oplus K_1]) \\
T &= \mathrm{MSB}_{Tlen}(C_n)
\end{aligned}
$$

where

T	= message authentication code, also referred to as the tag
$Tlen$	= bit length of T
$\mathrm{MSB}_s(X)$	= the s leftmost bits of the bit string X



(a) Message length is integer multiple of block size



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

Figure 12.8 Cipher-Based Message Authentication Code (CMAC)

Authenticated Encryption (AE)

- A term used to describe encryption systems that simultaneously protect confidentiality and authenticity of communications
- Approaches:
  - Hash-then-encrypt: E(K, (M || *h*))
  - MAC-then-encrypt: T = MAC(K_1, M), E(K_2, [M || T])
  - Encrypt-then-MAC: $C = E(K_2, M)$, T = MAC(K_1, C)
  - Encrypt-and-MAC: C = E(K_2, M), T = MAC(K_1, M)
- Both decryption and verification are straightforward for each approach
- There are security vulnerabilities with all of these approaches