Unit 4

1. Authentication requirement – Authentication function – MAC – Hash function – Security of hash function: HMAC, CMAC – SHA
2. Digital signature and authentication protocols – DSS – Schnorr Digital Signature Scheme – ElGamal cryptosystem
3. Entity Authentication: Biometrics, Passwords, Challenge Response protocols – Authentication applications – Kerberos MUTUAL
4. TRUST: Key management and distribution – Symmetric key distribution using symmetric and asymmetric encryption – Distribution of public keys – X.509 Certificates.

Requirements for MAC:

When A has a message to send to B, it calculates the MAC as a function of the message and the key:

$$\text{MAC} = \text{C}(K, M)$$

where

M = input message
C = MAC function
K = shared secret key
MAC = message authentication code

- The message plus MAC are transmitted to the intended recipient.
- The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC.
- The received MAC is compared to the calculated MAC

How MAC confidentiality is ensured?

1 → 00 01 10 11

n 2^n $n \to$ large

- To break symmetric / asymmetric algorithm => **Secured Key is required.**
- How to find confidential key ??? —-> **Use brute force attack Eg if key is of two digit, try all two digit numbers.**
- MAC is different.

msg
MAC
0000

C(K,M)

Key	MAC Code
abcde	oooo
fghij	oooo
tijhy	asbd

⇒ many to one mapping

- Even in brute force attack, multiple keys might map to same MAC/tag. So hacker cannot succeed in one run of brute force also.

abcde / fghij

① keys
② step 1 res

msg
xoyz

Iterative Attack - when key of MAC is of length k

- **Round 1**

  Given: $M_1, T_1 = \text{MAC}(K, M_1)$
  Compute $T_i = \text{MAC}(K_i, M_1)$ for all 2^k keys
  Number of matches $\approx 2^{(k-n)}$

- **Round 2**

  Given: $M_2, T_2 = \text{MAC}(K, M_2)$
  Compute $T_i = \text{MAC}(K_i, M_2)$ for the $2^{(k-n)}$ keys resulting from Round 1
  Number of matches $\approx 2^{(k-2 \times n)}$

And so on. On average, α rounds will be needed $k = \alpha \times n$. For example, if an 80-bit key is used and the tag is 32 bits, then the first round will produce about 2^{48} possible keys. The second round will narrow the possible keys to about 2^{16} possibilities. The third round should produce only a single key, which must be the one used by the sender.

$\underline{2}$ $2^2 \rightarrow$ 00
01
10
11

Other Attacks without finding key

Consider the following MAC algorithm. Let $M = (X_1 \| X_2 \| \ldots \| X_m)$ be a message that is treated as a concatenation of 64-bit blocks X_i. Then define

Sender

$$\Delta(M) = \boxed{X_1 \oplus X_2 \oplus \cdots \oplus X_m}$$
$$\mathrm{MAC}(K, M) = \mathrm{E}(K, \Delta(M))$$

Attack is made in:

system by replacing X_1 through X_{m-1} with any desired values Y_1 through Y_{m-1} and replacing X_m with Y_m, where Y_m is calculated as

attacker

$$Y_m = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{m-1} \oplus \Delta(M)$$

The opponent can now concatenate the new message, which consists of Y_1 through Y_m, using the original tag to form a message that will be accepted as authentic by the receiver. With this tactic, any message of length $64 \times (m - 1)$ bits can be fraudulently inserted.

Main Requirements of MAC function

8 $2^{-8} = \frac{1}{2^8}$

1. If an opponent observes M and $\text{MAC}(K, M)$, it should be computationally infeasible for the opponent to construct a message M' such that

$$\text{MAC}(K, M') = \text{MAC}(K, M)$$

2. $\text{MAC}(K, M)$ should be uniformly distributed in the sense that for randomly chosen messages, M and M', the probability that $\text{MAC}(K, M) = \text{MAC}(K, M')$ is 2^{-n}, where n is the number of bits in the tag.
3. Let M' be equal to some known transformation on M. That is, $M' = f(M)$. For example, f may involve inverting one or more specific bits. In that case,

$$\Pr[\text{MAC}(K, M) = \text{MAC}(K, M')] = 2^{-n}$$

Security of MAC

A brute-force attack on a MAC is a more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs.

- **Computation resistance:** Given one or more text-MAC pairs $[x_i, \mathrm{MAC}(K, x_i)]$, it is computationally infeasible to compute any text-MAC pair $[x, \mathrm{MAC}(K, x)]$ for any new input $x \neq x_i$.

In other words, the attacker would like to come up with the valid MAC code for a given message x. There are two lines of attack possible: attack the key space and attack the MAC value. We examine each of these in turn.

msg $\longleftrightarrow$ MAC

MAC code

Attack Key Space	Attack MAC value
1. Search brute force for the key and check with text-tag pair. 2. But the problem is many to one mapping 3. So iteratively repeat for multiple text-tag pairs.(computationally difficult)	1. objective is to generate a valid tag for a given message or to find a message that matches a given tag. 2. the attack cannot be conducted off line without further input; the attacker will require chosen text-tag pairs or knowledge of the key

Cryptanalysis:

- As with encryption algorithms and hash functions, cryptanalytic attacks on MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search.
- The way to measure the resistance of a MAC algorithm to cryptanalysis is to compare its strength to the effort required for a bruteforce attack.
- That is, an ideal MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.


thank you


LIKE
COMMENT
SHARE
SUBSCRIBE

Hashed MAC (HMAC)

$H \Rightarrow n \bmod 5$

$n \to 9 \Rightarrow H(9) = 9 \bmod 5 = 4$

$n \to 10 \to H(10) = 10 \bmod 5 = 0$

HMAC Design Objectives

RFC 2104 lists the following design objectives for HMAC.

1. ■ To use, without modifications, available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
2. ■ To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
3. ■ To preserve the original performance of the hash function without incurring a significant degradation.
4. ■ To use and handle keys in a simple way.
5. ■ To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

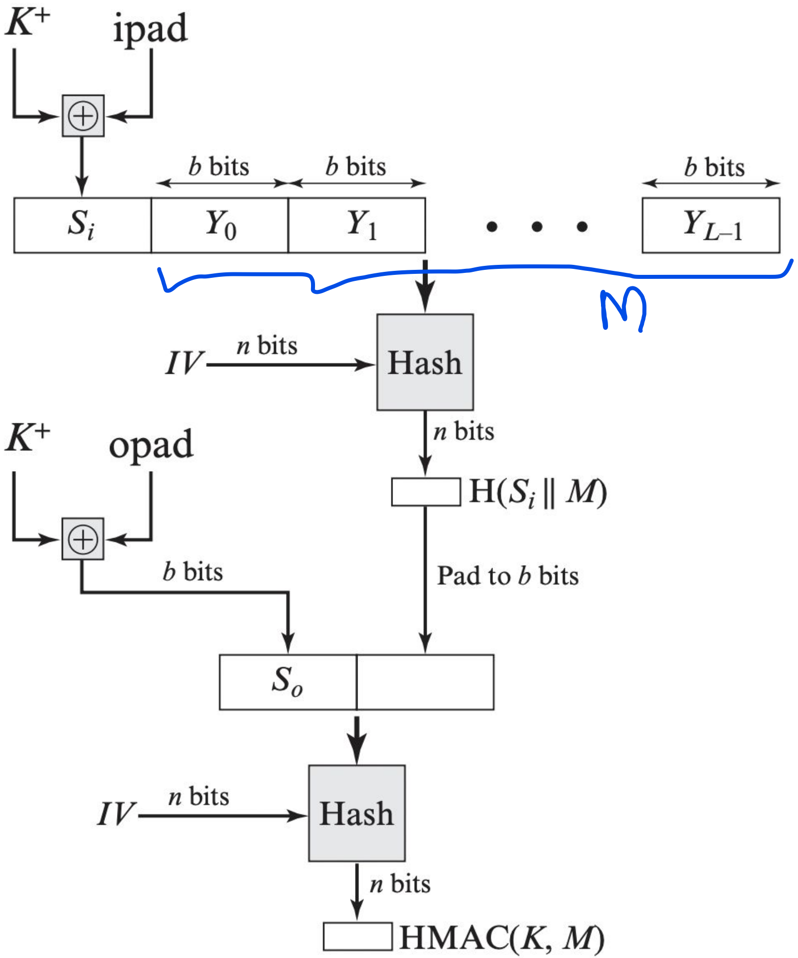

Figure 12.5 HMAC Structure

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)
Y_i = i th block of M, $0 \le i \le (L-1)$
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 $\ge n$; 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

$\underbrace{1011}_{Y_0}\underbrace{0011}_{Y_1}$

$L = 2$
$b = 4$

$K = \underbrace{011}_{3}$ $K^+ \Rightarrow \underbrace{0}_{pad}\underbrace{011}_{K}$

ipad = 00110110 (36 in hexadecimal) repeated $b/8$ times
opad = 01011100 (5C in hexadecimal) repeated $b/8$ times

Then HMAC can be expressed as

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

Steps in HMAC

1. Append zeros to the left end of K to create a b-bit string K^+
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o.
6. Append the hash result from step 4 to S_o.
7. Apply H to the stream generated in step 6 and output the result.

Efficient HMAC

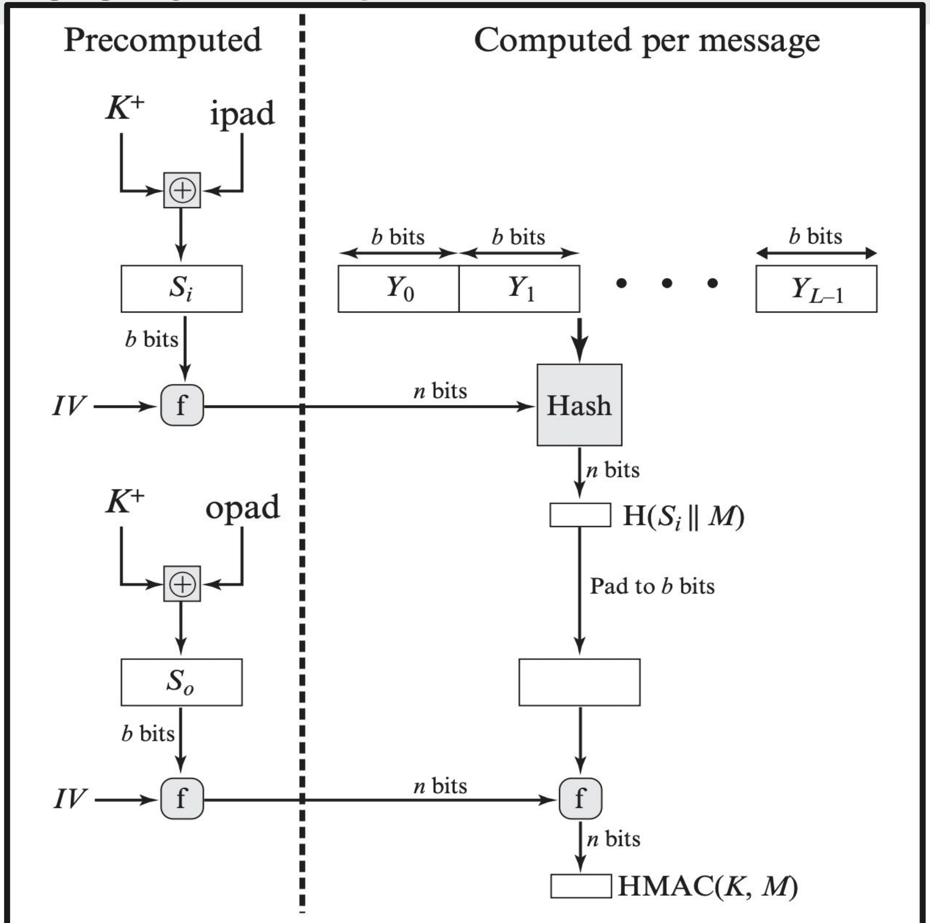

Figure 12.6 Efficient Implementation of HMAC

A more efficient implementation is possible, as shown in Figure 12.6. Two quantities are precomputed:

$$f(IV, (K^+ \oplus \text{ipad}))$$
$$f(IV, (K^+ \oplus \text{opad}))$$

Security of HMAC

SHA | MD-5

1. The security of any MAC function based on an embedded hash function depends in some way on the cryptographic strength of the underlying hash function.
2. The security of a MAC function is 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.
3. The probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded hash function.

1. The attacker is able to compute an output of the compression function even with an IV that is random, secret, and unknown to the attacker.

2. The attacker finds collisions in the hash function even when the IV is random and secret.

n mods ⇒ 10 mods = 0
5 mods = 0

Attack 1	Attack 2
• the compression function as equivalent to the hash function applied to a message consisting of a single b-bit block. • For this attack, the IV of the hash function is replaced by a secret, random value of n bits. • An attack on this hash function requires either a brute-force attack on the key, which is a level of effort on the order of 2^n $\frac{n}{2}$	• the attacker is looking for two messages M and M′ that produce the same hash: H(M) = H(M′) • requires a level of effort of $2^{n/2}$ for hash of len n • the attacker cannot generate message/code pairs off line because the attacker does not know K.


thank you


LIKE
COMMENT
SHARE
SUBSCRIBE