CHAPTER 12 MESSAGE AUTHENTICATION CODES

A message authentication code (MAC) is an algorithm that requires the use of a secret key. A MAC takes a variable-length message and a secret key as input and produces an authentication code. A recipient in possession of the secret key can generate an authentication code to verify the integrity of the message.

12.1. Authentication Requirements

In the context of communications across a network, the following attacks can be identified:

1. **Disclosure:** Release of message contents to any person or process not possessing the appropriate cryptographic key.
2. **Traffic analysis:** Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined.
3. **Masquerade:** Insertion of messages into the network from a fraudulent source.
4. **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition, and modification.
5. **Sequence modification:** Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.
6. **Timing modification:** Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed.
7. **Source repudiation:** Denial of transmission of message by source.
8. **Destination repudiation:** Denial of receipt of message by destination.

In summary, Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as

sent by (i.e., contain no modification, insertion, deletion, or replay) and that the purported identity of the sender is valid.

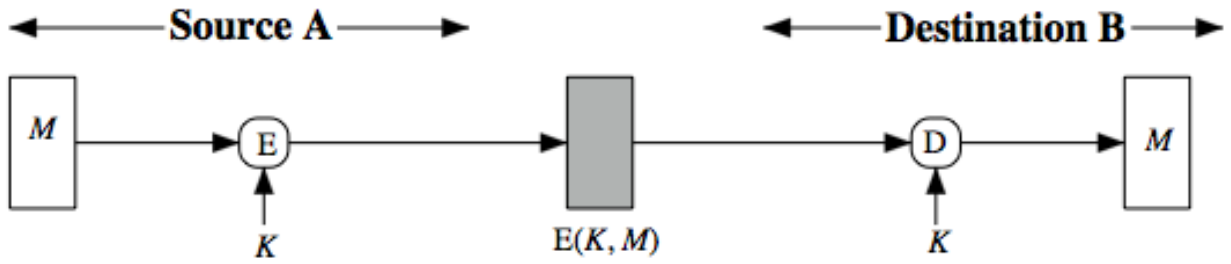
There are three types of functions that may be used to produce an authenticator:

1. **Hash function** - A function that maps a message of any length into a fixed length hash value, which serves as the authenticator
2. **Message encryption** - The ciphertext of the entire message serves as its authenticator
3. **Message Authentication Code (MAC)** - A function of the message and a secret key that produces a fixed-length value that serves as the authenticator.

Hash functions, and how they may serve for message authentication, are discussed in Chapter 11. The remainder of this section briefly examines the remaining two topics.

Message Encryption:

Symmetric Encryption:



(a) Symmetric encryption: confidentiality and authentication

Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

Consider the straightforward use of symmetric encryption (Figure 12.1a). A message transmitted from source A to destination B is encrypted using a secret key

shared by A and B. If no other party knows the key, then confidentiality is provided: No other party can recover the plaintext of the message.

In addition, B is assured that the message must have come from A. Furthermore, if M is recovered, B knows that none of the bits of have been altered, because an opponent that does not know how to alter bits in the ciphertext to produce the desired changes in the plaintext. So we may say that symmetric encryption provides authentication as well as confidentiality.

Public key Message Encryption:

(d) Public-key encryption: confidentiality, authentication, and signature

With public-key techniques, one can use a digital signature which can only be created by key owner to validate the integrity of the message contents. To provide both confidentiality and authentication, A can encrypt M first using its private key, which provides the digital signature, and then using B's public key, which provides confidentiality (Figure 12.1d). The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

Message Authentication Code:

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, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key K.

MAC = MAC(K, M)

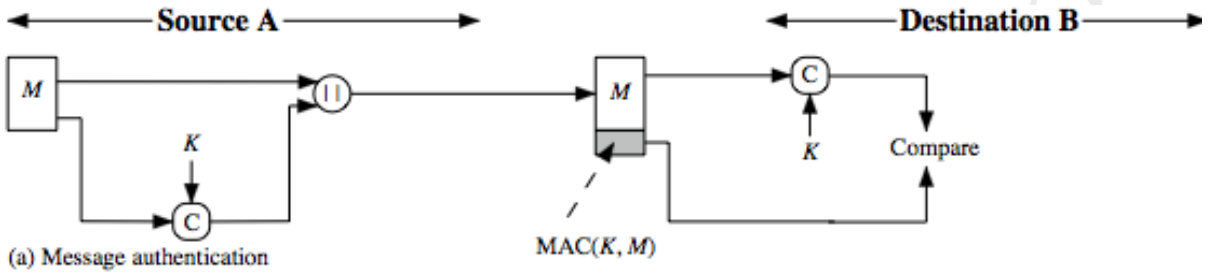
where

M = input message

C = MAC function

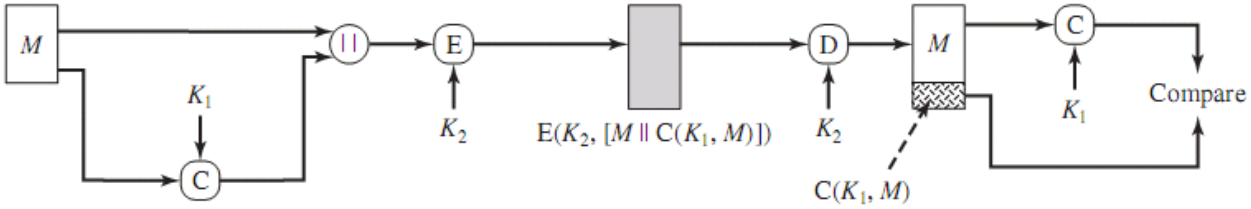
K = shared secret key

MAC = message authentication code

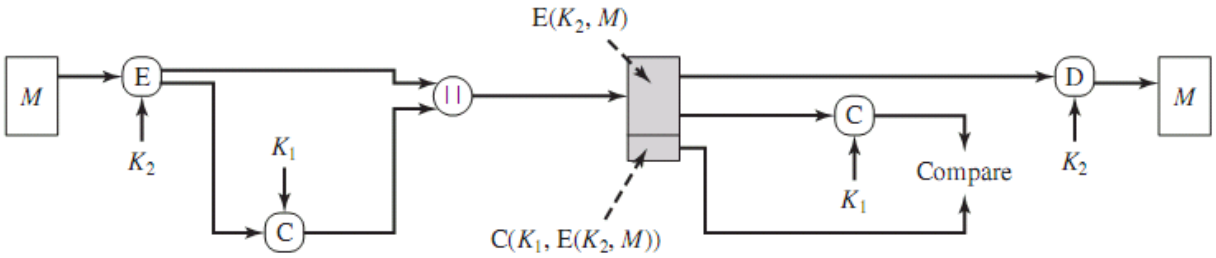


When A has a message to send to B, it calculates the MAC as a function of the message and the key: ***MAC = C(K, M).*** The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC (Figure 12.4a). If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then the receiver is assured that the message has not been altered, is from the alleged sender, and if the message includes a sequence number then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

A MAC function is similar to encryption. One **difference** is that the MAC algorithm need not be reversible, as it must for decryption. In general, the MAC function is a many-to-one function.



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

The process depicted (Figure 12.4a) provides authentication but not confidentiality, because the message as a whole is transmitted in the clear. Confidentiality can be provided by performing message encryption either after (Figure 12.4b) or before (Figure 12.4c) the MAC algorithm. In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver. MAC can be used in circumstances where just authentication is needed, (e.g. such as when the same message is broadcast to a number of destinations; when one side has a heavy load and cannot afford the time to decrypt all incoming messages; or do not need to keep messages secret, but must authenticate messages). Finally, note that the MAC does not provide a digital signature because both sender and receiver share the same key.

12.3 REQUIREMENTS FOR MESSAGE AUTHENTICATION CODES

The MAC is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the MAC. The MAC function is a many-to-one function, since potentially many arbitrarily long messages can be condensed to the same summary value, but don't want finding them to be easy.

Let us state the requirements for the function. Assume that an opponent knows the MAC function but does not know K. Then the MAC function should satisfy the following requirements.

1. If an opponent observes and, it should be computationally infeasible for the opponent to construct a message such that

   MAC (K, M') = MAC (K, M)

2. MAC(K, M) should be uniformly distributed in the sense that for randomly chosen messages, M and M' , the probability that is MAC(K, M) = 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 [MAC(K, M) = MAC(K, M')] = 2^{-n}

12.4 SECURITY OF MACS:

Just as with symmetric and public-key encryption, we can group attacks on hash functions and MACs into two categories: **brute-force attacks** and **cryptanalysis.**

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. The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm, with cost ($2^{m/2}$). A brute-force attack on a MAC has cost related to min(2^k, 2^n), similar to symmetric encryption algorithms. It would appear reasonable to require that the key length and MAC length satisfy a relationship such as min(*k, n)* >= *N*, where N is perhaps in the range of 128 bits.

As with encryption algorithms, cryptanalytic attacks on hash functions and 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 hash or MAC algorithm to cryptanalysis is to compare its strength to the effort required for a brute-

force attack. That is, an ideal hash or MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

HMAC:

In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash function, because they generally execute faster than symmetric block ciphers, and because code for cryptographic hash functions is widely available.

A hash function such as SHA was not designed for use as a MAC and cannot be used directly for that purpose because it does not rely on a secret key. There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm, originally by just pre-pending a key to the message. Problems were found with these earlier, simpler proposals, but they resulted in the development of HMAC.

HMAC Design Objectives:

- 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.
- 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:

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, $0 \leq i \leq L - 1$

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 ≥ 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

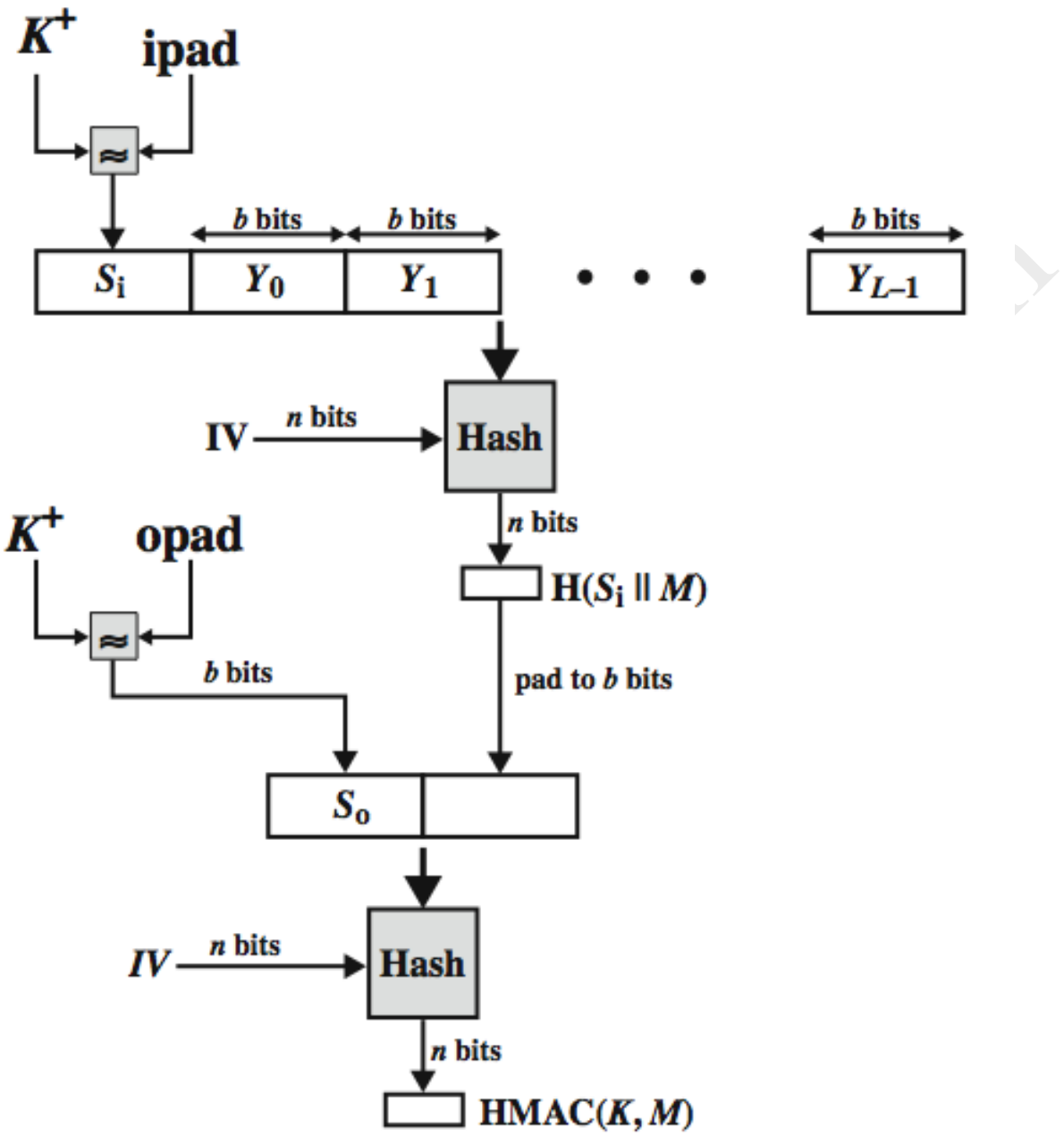
opad = 01011100 (5C in hexadecimal) repeated b/8 times

Then HMAC can be expressed as

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

We can describe the algorithm as follows

1. Append zeros to the left end of to create a b-bit string K^+ (e.g., if Kis of length 160 bits and b=512, then will be appended with 44 zeroes).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
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_0 .
6. Append the hash result from step 4 to S_0.
7. Apply H to the stream generated in step 6 and output the result.



Note that the XOR with ipad results in flipping one-half of the bits of *K*. Similarly, the XOR with opad results in flipping one-half of the bits of K, but a different set of bits. In effect, pseudorandomly generated two keys from K. HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the hash compression function (for *Si, So*, and the block produced from the inner hash).

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

$$f(IV, (K^+ \oplus ipad))$$

$$f(IV, (K^+ \oplus opad))$$

where f(cv,block) is the compression function for the hash function, which takes as arguments a chaining variable of n bits and a block of b bits and produces a chaining variable of n bits. These quantities only need to be computed initially and every time the key changes. In effect, the precomputed quantities substitute for the initial value (IV) in the hash function. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This more efficient implementation is especially worthwhile if most of the messages for which a MAC is computed are short.

Security of HMAC:

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. for a given level of effort (time, message–tag pairs) on messages generated by a legitimate user and seen by the attacker, 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 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.

****