Chapter 12
Message Authentication Codes (MACs)

A *Message Authentication Code* (MAC), also known as a *cryptographic checksum* or a *keyed hash function*, is widely used in practice. In terms of security functionality, MACs share some properties with digital signatures, since they also provide message integrity and message authentication. However, unlike digital signatures, MACs are symmetric-key schemes and they do not provide nonrepudiation. One advantage of MACs is that they are much faster than digital signatures since they are based on either block ciphers or hash functions.

In this chapter you will learn:

- The principle behind MACs
- The security properties that can be achieved with MACs
- How MACs can be realized with hash functions and with block ciphers

C. Paar, J. Pelzl, *Understanding Cryptography*,
DOI 10.1007/978-3-642-04101-3_12,

12.1 Principles of Message Authentication Codes

Similar to digital signatures, MACs append an authentication tag to a message. The crucial difference between MACs and digital signatures is that MACs use a symmetric key k for both generating the authentication tag and verifying it. A MAC is a function of the symmetric key k and the message x. We will use the notation

$$m = \mathrm{MAC}_k(x)$$

for this in the following. The principle of the MAC calculation and verification is shown in Figure 12.1.

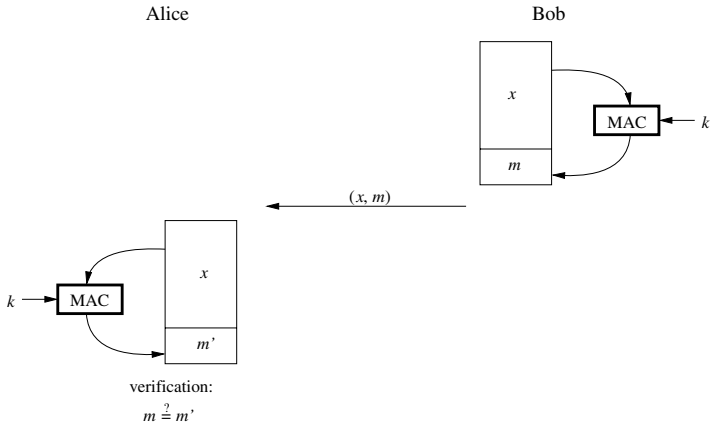


Fig. 12.1 Principle of message authentication codes (MACs)

The motivation for using MACs is typically that Alice and Bob want to be assured that any manipulations of a message x in transit are detected. For this, Bob computes the MAC as a function of the message and the shared secret key k. He sends both the message and the authentication tag m to Alice. Upon receiving the message and m, Alice verifies both. Since this is a symmetric set-up, she simply repeats the steps that Bob conducted when sending the message: She merely recomputes the authentication tag with the received message and the symmetric key.

The underlying assumption of this system is that the MAC computation will yield an incorrect result if the message x was altered in transit. Hence, *message integrity* is provided as a security service. Furthermore, Alice is now assured that Bob was the originator of the message since only the two parties with the same secret key k have the possibility to compute the MAC. If an adversary, Oscar, changes the message during transmission, he cannot simply compute a valid MAC since he lacks the secret key. Any malicious or accidental (e.g., due to transmission errors) forgery of the message will be detected by the receiver due to a failed verification of the MAC.

That means, from Alice's perspective, Bob must have generated the MAC. In terms of security services, message authentication is provided.

In practice, a messages x is often much larger than the corresponding MAC. Hence, similar to hash functions, the output of a MAC computation is a fixed-length authentication tag which is independent of the length of the input.

Together with earlier discussed characteristics of MACs, we can summarize all their important properties:

Properties of Message Authentication Codes

1. **Cryptographic checksum** A MAC generates a cryptographically secure authentication tag for a given message.
2. **Symmetric** MACs are based on secret symmetric keys. The signing and verifying parties must share a secret key.
3. **Arbitrary message size** MACs accept messages of arbitrary length.
4. **Fixed output length** MACs generate fixed-size authentication tags.
5. **Message integrity** MACs provide message integrity: Any manipulations of a message during transit will be detected by the receiver.
6. **Message authentication** The receiving party is assured of the origin of the message.
7. **No nonrepudiation** Since MACs are based on symmetric principles, they do not provide nonrepudiation.

The last point is important to keep in mind: MACs do not provide nonrepudiation. Since the two communicating parties share the same key, there is no possibility to prove towards a neutral third party, e.g., a judge, whether a message and its MAC originated from Alice or Bob. Thus, MACs offer no protection in scenarios where either Alice or Bob is dishonest, like the car-buying example we described in Section 10.1.1. A symmetric secret key is not tied to a certain person but rather to two parties, and hence a judge cannot distinguish between Alice and Bob in case of a dispute.

In practice, message authentication codes are constructed in essentially two different ways from block ciphers or from hash functions. In the subsequent sections of this chapter we will introduce both options for realizing MACs.

12.2 MACs from Hash Functions: HMAC

An option for realizing MACs is to use cryptographic hash functions such as SHA-1 as a building block. One possible construction, named HMAC, has become very popular in practice over the last decade. For instance, it is used in both the Transport Layer Security (TLS) protocol (indicated by the little lock symbol in your Web browser) as well as in the IPsec protocol suite. One reason for the widespread use of

the HMAC construction is that it can be proven to be secure if certain assumptions are made.

The basic idea behind all hash-based message authentication codes is that the key is hashed together with the message. Two obvious constructions are possible. The first one:

$$m = \text{MAC}_k(x) = h(k||x)$$

is called *secret prefix MAC*, and the second one:

$$m = \text{MAC}_k(x) = h(x||k)$$

is known as *secret suffix MAC*. The symbol "||" denotes concatenation. Intuitively, due to the one-wayness and the good "scrambling properties" of modern hash functions, both approaches should result in strong cryptographic checksums. However, as is often the case in cryptography, assessing the security of a scheme can be trickier than it seems at first glance. We now demonstrate weaknesses in both constructions.

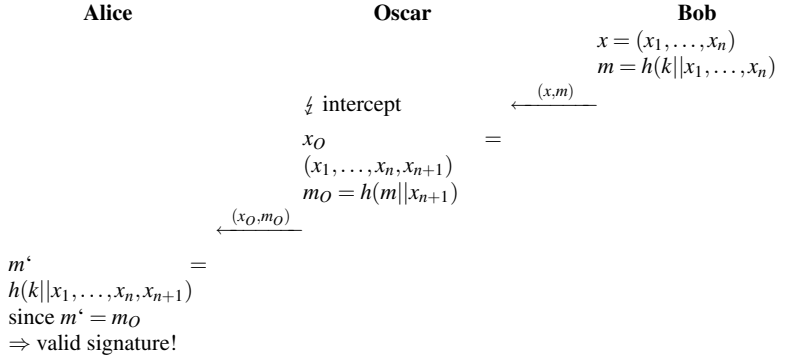
Attacks Against Secret Prefix MACs

We consider MACs realized as $m = h(k||x)$. For the attack we assume that the cryptographic checksum m is computed using a hash construction as shown in Figure 11.5. This iterated approach is used in the majority of today's hash functions. The message x that Bob wants to sign is a sequence of blocks $x = (x_1, x_2, \ldots, x_n)$, where the block length matches the input width of the hash function. Bob computes an authentication tag as:

$$m = \text{MAC}_K(x) = h(k||x_1, x_2, \ldots, x_n)$$

The problem is that the MAC for the message $x = (x_1, x_2, \ldots, x_n, x_{n+1})$, where x_{n+1} is an arbitrary additional block, can be constructed from m without knowing the secret key. The attack is shown in the protocol below.

Attack Against Secret Prefix MACs



Note that Alice will accept the message $(x_1, \ldots, x_n, x_{n+1})$ as valid, even though Bob only authenticated $(x_1, \ldots, x_n)$. The last block x_{n+1} could, for instance, be an appendix to an electronic contract, a situation that could have serious consequences.

The attack is possible since the MAC of the additional message block only needs the previous hash output, which is equal to Bob's m, and x_{n+1} as input but not the key k.

Attacks Against Secret Suffix MACs

After studying the attack above, it seems to be safe to use the other basic construction method, namely $m = h(x||k)$. However, a different weakness occurs here. Assume Oscar is capable of constructing a collision in the hash function, i.e., he can find x and x_O such that:

$$h(x) = h(x_O).$$

The two messages x and x_O can be, for instance, two versions of a contract which are different in some crucial aspect, e.g., the agreed upon payment. If Bob signs x with a message authentication code

$$m = h(x||k)$$

m is also a valid checksum for x_O, i.e.,

$$m = h(x||k) = h(x_O||k)$$

The reason for this is again given by the iterative nature of the MAC computation.

Whether this attack presents Oscar with an advantage depends on the parameters used in the construction. As a practical example, let's consider a secret suffix MAC which uses SHA-1 as hash function, which has an output length of 160 bits, and a 128-bit key. One would expect that this hash offers a security level of 128 bits,

i.e., an attacker cannot do better than brute-forcing the entire key space to forge a message. However, if an attacker exploits the birthday paradox (cf. Section 11.2.3), he can forge a signature with about $\sqrt{2^{160}} = 2^{80}$ computations. There are indications that SHA-1 collisions can be constructed with even fewer steps, so that an actual attack might be even easier. In summary, we conclude that the secret suffix method also does not provide the security one would like to have from a MAC construction.

HMAC

A hash-based message authentication code which does not show the security weakness described above is the HMAC construction proposed by Mihir Bellare, Ran Canetti and Hugo Krawczyk in 1996. The scheme consists of an inner and outer hash and is visualized in Figure 12.2.

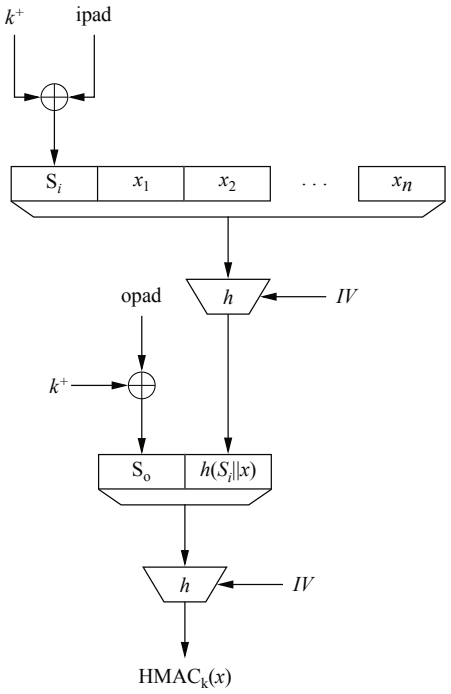


Fig. 12.2 HMAC construction

The MAC computation starts with expanding the symmetric key k with zeros on the left such that the result k^+ is b bits in length, where b is the input block width of the hash function. The expanded key is XORed with the inner pad, which consists of the repetition of the bit pattern:

$$\text{ipad} = 00110110, 00110110, \ldots, 00110110$$

so that a length of b bit is achieved. The output of the XOR forms the first input block to the hash function. The subsequent input blocks are the message blocks $(x_1, x_2, \ldots, x_n)$.

The second, outer hash is computed with the padded key together with the output of the first hash. Here, the key is again expanded with zeros and then XORed with the outer pad:

$$\text{opad} = 01011100, 01011100, \ldots, 01011100.$$

The result of the XOR operation forms the first input block for the outer hash. The other input is the output of the inner hash. After the outer hash has been computed, its output is the message authentication code of x. The HMAC construction can be expressed as:

$$\text{HMAC}_k(x) = h\left[(k^+ \oplus \text{opad}) || h\left[(k^+ \oplus \text{ipad}) || x\right]\right].$$

The hash output length l is in practice longer than the width b of an input block. For instance, SHA-1 has an $l = 160$ bit output but accepts $b = 512$ bit inputs. It does not pose a problem that the inner hash function output does not match the input size of outer hash because hash functions have preprocessing steps to match the input string to the block width. As an example, Section 11.4.1 described the preprocessing for SHA-1.

In terms of computational efficiency, it should be noted that the message x, which can be very long, is only hashed once in the inner hash function. The outer hash consists of merely two blocks, namely the padded key and the inner hash output. Thus, the computational overhead introduced through the HMAC construction is very low.

In addition to its computational efficiency, a major advantage of the HMAC construction is that there exists a *proof of security*. As for all schemes which are provable secure, HMAC is not secure per se, but its security is related to the security of some other building block. In the case of the HMAC construction it can be shown that if an attacker, Oscar, can break the HMAC, he can also break the hash function used in the scheme. Breaking HMAC means that even though Oscar does not know the key, he can construct valid authentication tags for messages. Breaking the hash function means that he can either find collisions or that he can compute a hash function output even though he does not know the initial value IV (which was the value H_0 in the case of SHA-1).

12.3 MACs from Block Ciphers: CBC-MAC

In the preceding section we saw that hash functions can be used to realize MACs. An alternative method is to construct MACs from block ciphers. The most popular

approach in practice is to use a block cipher such as AES in cipher block chaining (CBC) mode, as discussed in Section 5.1.2.

Figure 12.3 depicts the complete setting for the application of a MAC on basis of a block cipher in CBC mode. The left side shows the sender, the right side the receiver. This scheme is also referred to as CBC-MAC.

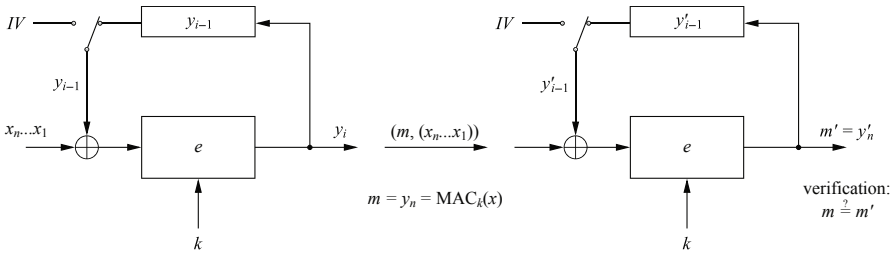


Fig. 12.3 MAC built from a block cipher in CBC mode

MAC Generation

For the generation of a MAC, we have to divide the message x into blocks $x_i, i = 1,...,n$. With the secret key k and an initial value IV, we can compute the first iteration of the MAC algorithm as

$$y_1 = e_k(x_1 \oplus IV),$$

where the IV can be a public but random value. For subsequent message blocks we use the XOR of the block x_i and the previous output y_{i-1} as input to the encryption algorithm:

$$y_i = e_k(x_i \oplus y_{i-1}).$$

Finally, the MAC of the message $x = x_1x_2x_3...x_n$ is the output y_n of the last round:

$$m = \mathrm{MAC}_k(x) = y_n$$

In contrast to CBC encryption, the values $y_1, y_2, y_3, \ldots, y_{n-1}$ are *not* transmitted. They are merely internal values which are used for computing the final MAC value $m = y_n$.

MAC Verification

As with every MAC, verification involves simply repeating the operation that were used for the MAC generation. For the actual verification decision we have to com-