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

Christof Paar et al., *Understanding Cryptography*

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.

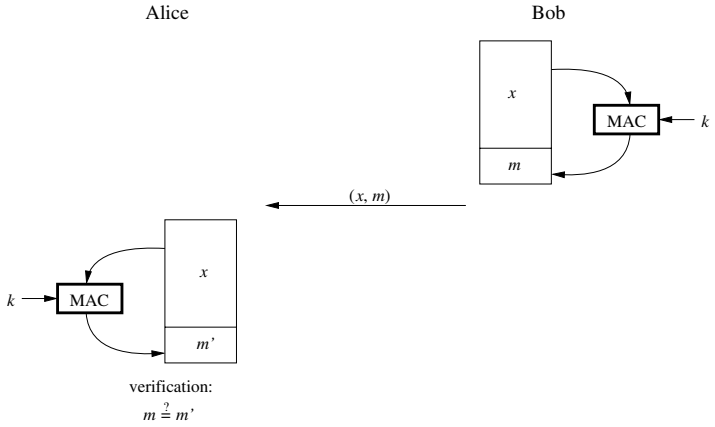

Fig. 12.1 Principle of message authentication codes (MACs)

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

Alice	Oscar	Bob
		$x = (x_1, \ldots, x_n)$ $m = h(k\|\|x_1, \ldots, x_n)$
		$\xleftarrow{(x,m)}$
	↯ intercept	
	$x_O = (x_1, \ldots, x_n, x_{n+1})$ $m_O = h(m\|\|x_{n+1})$	
	$\xleftarrow{(x_O, m_O)}$	
$m' = h(k\|\|x_1, \ldots, x_n, x_{n+1})$ since $m' = m_O$ $\Rightarrow$ valid signature!		

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.

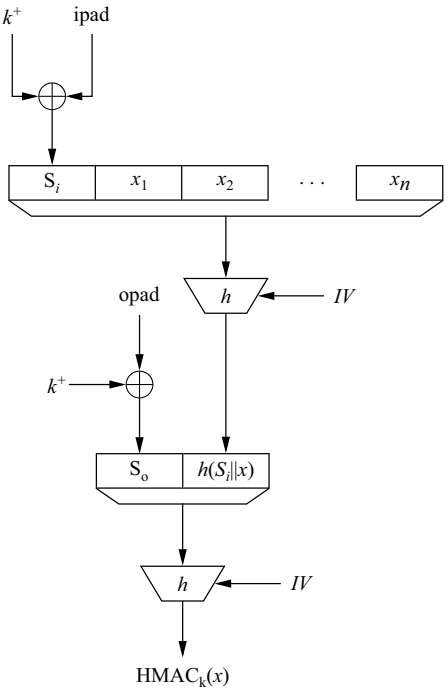

Fig. 12.2 HMAC construction

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.

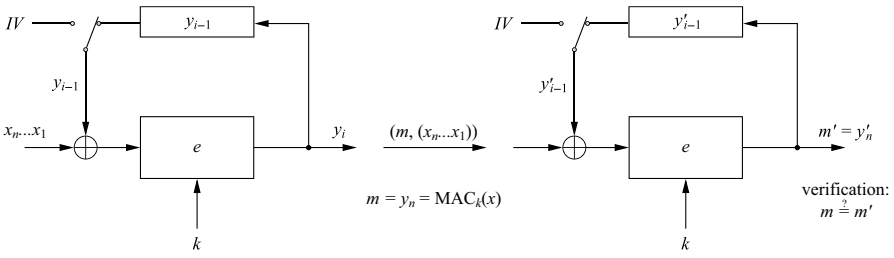

Fig. 12.3 MAC built from a block cipher in CBC mode

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:

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

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 compare the computed MAC m' with the received MAC value m. In case $m' = m$, the message is verified as correct. In case $m' \neq m$, the message and/or the MAC value m have been altered during transmission. We note that the MAC verification is different from CBC decryption, which actually reverses the encryption operation.

The output length of the MAC is determined by the block size of the cipher used. Historically, DES was widely used, e.g., for banking applications. More recently, AES is often used; it yields a MAC of length 128 bit.

12.4 Galois Counter Message Authentication Code (GMAC)

GMAC is a variant of the Galois Counter Mode (GCM) introduced in Section 5.1.6. GMAC is specified in [160] and is a mode of operation for an underlying symmetric key block cipher. In contrast to the GCM mode, GMAC does not encrypt data but only computes a message authentication code. GMAC is easily parallelizable, which is attractive for high-speed applications. The use of GMAC in IPsec Encapsulating Security Payload (ESP) and Authentication Header (AH) is described in the RFC 4543 [119]. The RFC describes how to use AES in GMAC to provide data

origin authentication without confidentiality within the IPsec ESP and AH. GMAC can be efficiently implemented in hardware and can reach a speed of 10 Gbit/sec and above.

12.5 Discussion and Further Reading

Block Cipher-Based MACs Historically, block cipher-based MACs have been the dominant method for constructing message authentication codes. As early as in 1977, i.e., only a couple of years after the announcement of the Data Encryption Standard (DES), it was suggested that DES could be used to compute cryptographic checksums [39]. In the following years, block cipher-based MACs were standardized in the US and became popular for assuring the integrity of financial transactions, see, e.g., the ANSI X9.17 standard [3]. Much more recently, the NIST recommendation [65] specifies a message authentication code algorithm based on a symmetric key block cipher (*CMAC*), which is similar to CBC-MAC. The AES-CMAC algorithm is specified in RFC 4493 [159].

In this chapter the CBC-MAC was introduced. In addition to the CBC-MAC, there are the OMAC and PMAC, which are both constructed with block ciphers. Counter with CBC-MAC (*CCM*) is a mode for authenticated encryption and is defined for use with a 128-bit block cipher [173]. It is described in the NIST recommendation [64]. The GMAC construction is standardized in IPSec [119] and in the NIST recommendation for Block Cipher Modes of Operation [66].

Hash Function-Based MACs The HMAC construction was originally proposed at the Crypto 1996 conference [14]. A very accessible treatment of the scheme can be found in [15]. HMAC was turned into an Internet RFC, and was quickly adopted in many Internet security protocols, including TLS and IPsec. In both cases it protects the integrity of a message during transmission. It is widely used with the hash functions SHA-1 and MD5, and its use with RIPEMD-160 has also been often discussed. It seems likely that the switch to more modern hash functions such as SHA-2 and SHA-3 will result in more and more HMAC constructions with these hash functions.

Other MAC Constructions Another type of message authentication code is based on *universal hashing* and is called *UMAC*. UMAC is backed by a formal security analysis, and the only internal cryptographic component is a block cipher used to generate the pseudorandom pads and internal key material. The universal hash function is used to produce a short hash value of fixed length. This hash is then XORed with a key-derived pseudorandom pad. The universal hash function is designed to be very fast in software (e.g., as low as one cycle per byte on contemporary processors) and is mainly based on additions of 32-bit and 64-bit numbers and multiplication of 32-bit numbers. Based on the original idea by Wegman and Carter [40], numerous schemes have been proposed, e.g., the schemes Multilinear-Modular-Hashing (MMH) and UMAC [89, 23].

12.6 Lessons Learned

- MACs provide two security services, *message integrity* and *message authentication*, using symmetric techniques. MACs are widely used in protocols.
- Both of these services are also provided by digital signatures, but MACs are much faster.
- MACs do not provide nonrepudiation.
- In practice, MACs are either based on block ciphers or on hash functions.
- HMAC is a popular MAC used in many practical protocols such as TLS.

Problems

12.1. As we have seen, MACs can be used to authenticate messages. With this problem, we want to show the difference between two protocols—one with a MAC, one with a digital signature. In the two protocols, the sending party performs the following operation:

1. Protocol A:
$$y = e_{k_1}[x||h(k_2||x)]$$
where x is the message, $h()$ is a hash function such as SHA-1, e is a private-key encryption algorithm, "||" denotes simple concatenation, and k_1, k_2 are secret keys which are only known to the sender and the receiver.
2. Protocol B:
$$y = e_k[x||sig_{k_{pr}}(h(x))]$$

Provide a step-by-step description (e.g., with an itemized list) of what the receiver does upon receipt of y. You may want to draw a block diagram for the process on the receiver's side, but that's optional.

12.2. For hash functions it is crucial to have a sufficiently large number of output bits, with, e.g., 160 bits, in order to thwart attacks based on the birthday paradox. Why are much shorter output lengths of, e.g., 80 bits, sufficient for MACs?

For your answer, assume a message x that is sent in clear together with its MAC over the channel: $(x, MAC_k(x))$. Exactly clarify what Oscar has to do to attack this system.

12.3. We study two methods for integrity protection with encryption.

1. Assume we apply a technique for combined encryption and integrity protection in which a ciphertext c is computed as
$$c = e_k(x||h(x))$$
where $h()$ is a hash function. This technique is not suited for encryption with stream ciphers if the attacker knows the whole plaintext x. Explain *exactly* how an active attacker can now *replace* x by an arbitrary x' of his/her choosing and compute c' such that the receiver will verify the message correctly. Assume that x and x' are of equal length. Will this attack work too if the encryption is done with a one-time pad?
2. Is the attack still applicable if the checksum is computed using a keyed hash function such as a MAC:
$$c = e_{k_1}(x||MAC_{k_2}(x))$$
Assume that $e()$ is a stream cipher as above.

12.4. We will now discuss some issues when constructing an efficient MAC.

1. The messages X to be authenticated consists of z independent blocks, so that $X = x_1||x_2||\ldots||x_z$, where every x_i consists of $|x_i| = 8$ bits. The input blocks are consecutively put into the compression function

$$c_i = h(c_{i-1}, x_i) = c_{i-1} \oplus x_i$$

   At the end, the MAC value

$$MAC_k(X) = c_z + k \bmod 2^8$$

   is calculated, where k is a 64-bit long shared key. Describe how exactly the (effective part of the) key k can be calculated with only one known message X.
2. Perform this attack for the following parameters and determine the key k:

$$\begin{aligned} X &= \texttt{HELLO ALICE!} \\ c_0 &= 11111111_2 \\ MAC_k(X) &= 10011101_2 \end{aligned}$$

3. What is the effective key length of k?
4. Although two different operations ($[\oplus, 2^8]$ and $[+, 2^8]$) are utilized in this MAC, this MAC-based signature possesses significant weaknesses. To which property of the design can these be ascribed, and where should one take care when constructing a cryptographic system? This essential property also applies for block ciphers and hash functions!

12.5. MACs are, in principle, also vulnerable against collision attacks. We discuss the issue in the following.

1. Assume Oscar found a collision between two messages, i.e.,

$$\text{MAC}_k(x_1) = \text{MAC}_k(x_2)$$

   Show a simple protocol with an attack that is based on a collision.
2. Even though the birthday paradox can still be used for constructing collisions, why is it in a practical setting much harder to construct them for MACs than for hash functions? Since this is the case: what security is provided by a MAC with 80-bit output compared to a hash function with 80-bit output?