Cryptography: An Introduction (3rd Edition)

Nigel Smart

Chapter 10 (second half)

4. Message Authentication Codes

Given a message and its hash code, as output by a cryptographic hash function, ensures that data has not been tampered with between the execution of the hash function and its verification, by recomputing the hash. However, using a hash function in this way requires the hash code itself to be protected in some way, by for example a digital signature, as otherwise the hash code itself could be tampered with.

To avoid this problem one can use a form of keyed hash function called a message authentication code, or MAC. This is a symmetric key algorithm in that the person creating the code and the person verifying it both require the knowledge of a shared secret.

Suppose two parties, who share a secret key, wish to ensure that data transmitted between them has not been tampered with. They can then use the shared secret key and a keyed algorithm to produce a check-value, or MAC, which is sent with the data. In symbols we compute

$$\text{code} = MAC_k(m)$$

where

- MAC is the check function,
- k is the secret key,
- m is the message.

Note we do not assume that the message is secret, we are trying to protect data integrity and not confidentiality. If we wish our message to remain confidential then we should encrypt it before applying the MAC. After performing the encryption and computing the MAC, the user transmits

$$e_{k_1}(m) \| MAC_{k_2}\left(e_{k_1}(m)\right).$$

This is a form of encryption called a data encapsulation mechanism, or DEM for short. Note, that different keys are used for the encryption and the MAC part of the message and that the MAC is applied to the ciphertext and not the message.

Before we proceed on how to construct MAC functions it is worth pausing to think about what security properties we require. We would like that only people who know the shared secret are able to both produce new MACs or verify existing MACs. In particular it should be hard given a MAC on a message to produce a MAC on a new message.

4.1. Producing MACs from hash functions. A collision-free cryptographic hash function can also be used as the basis of a MAC. The first idea one comes up with to construct such a MAC is to concatenate the key with the message and then apply the hash function. For example

$$MAC_k(M) = h(k\|M).$$

However, this is not a good idea since almost all hash functions are created using methods like the Merkle–Damgård construction. This allows us to attack such a MAC as follows: We assume that first that the non-length strengthed Merkle–Damgård construction is used with compression function f. Suppose one obtains the MAC c_1 on the t block message m_1

$$c_1 = MAC_k(m_1) = h(k\|m_1)$$

We can then, without knowledge of k compute the MAC c_2 on the $t+1$ block message $m_1\|m_2$ for any m_2 of one block in length, via

$$\begin{aligned} c_2 &= MAC_k(m_1\|m_2) \\ &= f(c_1\|m_2). \end{aligned}$$

Clearly this attack can be extended to a appending an m_2 of arbitrary length. Hence, we can also apply it to the length strengthed version. If we let m_1 denote a t block message and let b denote the block which encodes the bit length of m_1 and we let m_2 denote an arbitrary new block, then from the MAC of the message m_1 one can obtain the MAC of the message

$$m_1 \| b \| m_2.$$

Having worked out that prepending a key to a message does not give a secure MAC, one might be led to try appending the key after the message as in

$$MAC_k(M) = h(M \| k).$$

Again we now can make use of the Merkle–Damgård construction to produce an attack. We first, without knowledge of k, find via a birthday attack on the hash function h two equal length messages m_1 and m_2 which hash to the same values:

$$h(m_1) = h(m_2).$$

We now try to obtain the legitimate MAC c_1 on the message m_1. From, this we can deduce the MAC on the message m_2 via

$$\begin{aligned} MAC_k(m_2) &= h(m_2 \| k) \\ &= f\left(h(m_2) \| k\right) \\ &= f\left(h(m_1) \| k\right) \\ &= h(m_1 \| k) \\ &= MAC_k(m_1) \\ &= c_1. \end{aligned}$$

assuming k is a single block in length and the non-length strengthened version is used. Both of these assumptions can be relaxed, the details of which we leave to the reader.

To produce a secure MAC from a hash function one needs to be a little more clever. A MAC, called HMAC, occurring in a number of standards documents works as follows:

$$HMAC = h(k \| p_1 \| h(k \| p_2 \| M)),$$

where p_1 and p_2 are strings used to pad out the input to the hash function to a full block.

4.2. Producing MACs from block ciphers. Apart from ensuring the confidentiality of messages, block ciphers can also be used to protect the integrity of data. There are various types of MAC schemes based on block ciphers, but the best known and most widely used by far are the CBC-MACs. These are generated by a block cipher in CBC Mode. CBC-MACs are the subject of various international standards dating back to the early 1980s. These early standards specify the use of DES in CBC mode to produce a MAC, although one could really use any block cipher in place of DES.

Using an n-bit block cipher to give an m-bit MAC, where $m \leq n$, is done as follows:

- The data is padded to form a series of n-bit blocks.
- The blocks are encrypted using the block cipher in CBC Mode.
- Take the final block as the MAC, after an optional postprocessing stage and truncation (if $m < n$).

Hence, if the n-bit data blocks are

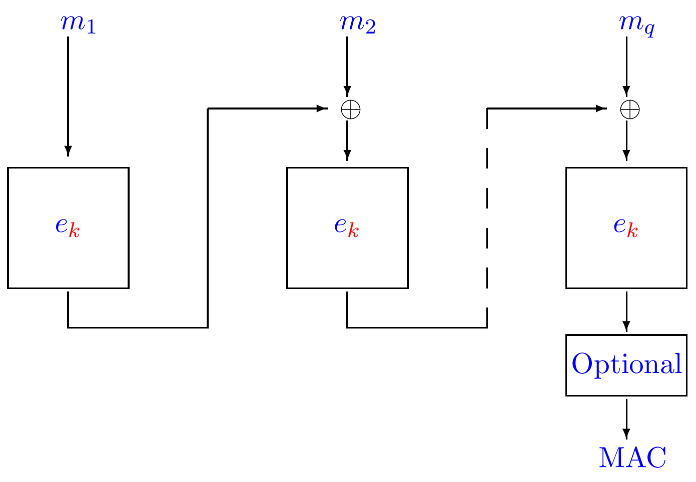
$$m_1, m_2, \ldots, m_q$$

then the MAC is computed by first setting $I_1 = m_1$ and $O_1 = e_k(I_1)$ and then performing the following for $i = 2, 3, \ldots, q$

$$I_i = m_i \oplus O_{i-1},$$
$$O_i = e_k(I_i).$$

The final value O_q is then subject to an optional processing stage. The result is then truncated to m bits to give the final MAC. This is all summarized in Fig. 1.

FIGURE 1. CBC-MAC: Flow diagram



Just as with hash functions one needs to worry about how one pads the message before applying the CBC-MAC. The three main padding methods proposed in the standards, are as follows, and are equivalent to those already considered for hash functions:

- **Method 1:** Add as many zeros as necessary to make a whole number of blocks. This method has a number of problems associated to it as it does not allow the detection of the addition or deletion of trailing zeros, unless the message length is known.
- **Method 2:** Add a single one to the message followed by as many zeros as necessary to make a whole number of blocks. The addition of the extra bit is used to signal the end of the message, in case the message ends with a string of zeros.
- **Method 3:** As method one but also add an extra block containing the length of the unpadded message.

Before we look at the "optional" post-processing steps let us first see what happens if no post-processing occurs. We first look at an attack which uses padding method one. Suppose we have a MAC M on a message

$$m_1, m_2, \ldots, m_q,$$

consisting of a whole number of blocks. Then one can the MAC M is also the MAC of the double length message

$$m_1, m_2, \ldots, m_q, M \oplus m_1, m_2, m_3, \ldots, m_q.$$

To see this notice that the input to the $(q+1)$'st block cipher envocation is equal to the value of the MAC on the original message, namely M, xor'd with the $(q+1)$'st block of the new message, namely $M \oplus m_1$. Thus the input to the $(q+1)$'st cipher envocation is equalk to m_1, and so the MAC on the double length message is also equal to M.

One could suspect that if you used padding method three above then attacks would be impossible. Let b denote the block length of the cipher and let $\mathbb{P}(n)$ denote the encoding within a block of the number n. To MAC a single block message m_1 one then computes

$$M_1 = e_k\left(e_k(m_1) \oplus \mathbb{P}(b)\right).$$

Suppose one obtains the MAC's M_1 and M_2 on the single block messages m_1 and m_2. Then one requests the MAC on the three block message

$$m_1, \mathbb{P}(b), m_3$$

for some new block m_3. Suppose the recieved MAC is then equal to M_3, i.e.

$$M_3 = e_k\left(e_k\left(e_k\left(e_k(m_1) \oplus \mathbb{P}(b)\right) \oplus m_3\right) \oplus \mathbb{P}(3b)\right).$$

Now also consider the MAC on the three block message

$$m_2, \mathbb{P}(b), m_3 \oplus M_1 \oplus M_2.$$

This MAC is equal to M_3', where

$$\begin{aligned}
M_3' &= e_k\left(e_k\left(e_k\left(e_k(m_2) \oplus \mathbb{P}(b)\right) \oplus m_3 \oplus M_1 \oplus M_2\right) \oplus \mathbb{P}(3b)\right) \\
&= e_k\left(e_k\left(e_k\left(e_k(m_2) \oplus \mathbb{P}(b)\right) \oplus m_3 \oplus e_k\left(e_k(m_1) \oplus \mathbb{P}(b)\right) \oplus e_k\left(e_k(m_2) \oplus \mathbb{P}(b)\right)\right)\right. \\
&\quad \left.\oplus \mathbb{P}(3b)\right) \\
&= e_k\left(e_k\left(m_3 \oplus e_k\left(e_k(m_1) \oplus \mathbb{P}(b)\right)\right) \oplus \mathbb{P}(3b)\right) \\
&= e_k\left(e_k\left(e_k\left(e_k(m_1) \oplus \mathbb{P}(b)\right) \oplus m_3\right) \oplus \mathbb{P}(3b)\right) \\
&= M_3.
\end{aligned}$$

Hence, we see that on their own the non-trivial padding methods do not protect against MAC forgery attacks. This is one of the reasons for introducing the post processing steps. There are two popular post-processing steps, designed to make it more difficult for the cryptanalyst to perform an exhaustive key search and to protect against attacks such as the ones explained above:

(1) Choose a key k_1 and compute

$$O_q = e_k\left(d_{k_1}(O_q)\right).$$

(2) Choose a key k_1 and compute

$$O_q = e_{k_1}(O_q).$$

Both of these post-processing steps were invented when DES was the dominant cipher, and in such a situation the first of these is equivalent to processing the final block of the message using the 3DES algorithm.

Chapter Summary

- Hash functions are required which are both preimage, collision and second-preimage resistant.
- Due to the birthday paradox the output of the hash function should be at least twice the size of what one believes to be the limit of the computational ability of the attacker.
- More hash functions are iterative in nature, although most of the currently deployed ones have recently shown to be weaker than expected.
- A message authentication code is in some sense a keyed hash function.
- MACs can be created out of either block ciphers or hash functions.

Further Reading

A detailed description of both SHA-1 and the SHA-2 algorithms can be found in the FIPS standard below, this includes a set of test vectors as well. The recent work on the analysis of SHA-1, and references to the earlier attacks on MD4 and MD5 can be found in the papers og Wang et. al., of which we list only one below.

FIPS PUB 180-2, *Secure Hash Standard (including SHA-1, SHA-256, SHA-384, and SHA-512)*. NIST, 2005.

X. Wang, Y.L. Yin and H. Yu. *Finding Collisions in the Full SHA-1* In Advances in Cryptology – CRYPTO 2005, Springer-Verlag LNCS 3621, pp 17-36, 2005.