Source:

Network Security

by Kaufman, Perlman, Speciner

Section 5.2.2, pages 123-125

5.2.2 Computing a MAC with a Hash

In §4.3 *Generating MACs* we described how to compute a MAC (message authentication code) with a secret key algorithm. Again, let's assume that for some reason no secret key algorithms are available. Can we still compute a MAC, using a hash function instead of something like DES?

The obvious thought is that MD(m) is a MAC for message m. But it isn't. Anyone can compute MD(m). The point of the MAC is to send something that only someone knowing the secret can compute (and verify). For instance, if Alice and Bob share a secret, then Alice can send m, plus

MAC, and since nobody except Alice and Bob can compute a MAC with their shared key, nobody but Alice or Bob would be able to send a message to Bob with an appropriate MAC. If we just simply used MD, then anyone can send any message m' together with MD(m'). So we do roughly the same trick for the MAC as we did for authentication. We concatenate a shared secret K_{AB} with the message m, and use MD($K_{AB}|m$) as the MAC.

This scheme almost works, except for some idiosyncracies of most of the popular message digest algorithms, which would allow an attacker to be able to compute a MAC of a longer message beginning with m, given message m and the correct MAC for m.

Assume MD is one of MD4, MD5, or SHA-1. The way these algorithms work is that the message is padded to a multiple of 512 bits with a pad that includes the message length. The padded message is then digested from left to right in 512-bit chunks. In order to compute the message digest through chunk n, all that you need to know is the message digest through chunk n–1, plus the value of chunk n of the padded message.

Let's assume Carol would like to send a different message to Bob, and have it look like it came from Alice. Let's say that Carol doesn't care what's in the message. She only cares that the end of the message says P.S. Give Carol a promotion and triple her salary. Alice has transmitted some message m, and MD($K_{AB}|m$). Carol can see both of those quantities. She concatenates the padding and then whatever she likes to the end of m, and initializes the message digest computation with MD($K_{AB}|m$). She does not need to know the shared secret in order to compute the MAC.

How can we avoid this flaw? Lots of techniques have been proposed, all entirely effective as far as anyone can tell. But people came up with "improvements", each one a little more complex than the one before, with the apparent winner being HMAC. Some proposals with no known weaknesses are:

- Put the secret at the end of the message instead of at the beginning. This will work. This method can be criticized for an extremely unlikely security flaw. The complaint is that if the MD algorithm were weak, and it was therefore possible to find two messages with the same digest, then those two messages would also have the same MAC.
- Use only half the bits of the message digest as the MAC. For instance, take the low-order 64 bits of the MD5 digest. This gives an attacker no information with which to continue the message digest (well, the attacker has a 1 in 2^{64} chance of guessing the rest of the message digest correctly—we assume you're not going to worry about that risk). Having only 64 bits of MAC (rather than using all 128 bits of the MD) is not any less secure, since there is no way that an attacker can generate messages and test the resulting MAC. Without knowing the secret, there is no way for the attacker to calculate the MAC. The best that can be done is to generate a random 64-bit MAC for the message you'd like to send and hope that you'll be really really lucky.

- Concatenate the secret to both the front and the back of the message. That way you get the collision resistance of putting it in front and the protection from appending that comes from putting it in back.

HMAC concatenates the secret to the front of the message, digests the combination, then concatenates the secret to the front of the digest, and digests the combination again. The actual construction is a little more complicated than this, and is described in section §5.7 *HMAC*. HMAC has lower performance than the other alternatives because it does a second digest. But the second digest is only computed over the secret and a digest, so it does not add much cost to large messages. In the worst case, if the message concatenated with the key fit into a single (512-bit) block, HMAC would be four times as expensive as one of the other alternatives described above. However, if many small messages are to be HMAC'd with the same key, it is possible to reuse the part of the computation that digests the key, so that HMAC would only be twice as slow. With a large enough message, HMAC's performance is only negligibly worse.

We call any hash combining the secret key and the data a **keyed hash**.