11.3: Hash Functions vs. MACs: Length-Extension Attacks

When we discuss hash functions, we generally consider the salt s to be public. A natural question is, **what happens when we make the salt private?** Of all the cryptographic primitives we have discussed so far, a hash function with secret salt most closely resembles a MAC. So, **do we get a secure MAC** by using a hash function with private salt?

Unfortunately, the answer is no in general (although it can be yes in some cases, depending on the hash function). In particular, the method is insecure when H is constructed using the Merkle-Damgård approach. The key observation is that:

> *knowing $H(x)$ allows you to predict the hash of any string that begins with $\text{MDPAD}(x)$*

This concept is best illustrated by example.

Example

Let's return to our previous example, with a compression function $h : \{0,1\}^{48} \to \{0,1\}^{32}$. Suppose we construct a Merkle-Damgård hash out of this compression function, and use the construction $\text{MAC}(k, m) = H(k||m)$ as a MAC.

Suppose the MACkey is chosen as $k = 0110001111001101$, and an attacker sees the MAC tag t of the message $m = 0100001110010111010100000$. Then $t = H(k||m)$ corresponds exactly to the example from before:

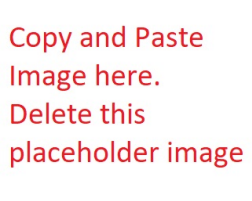

Figure 11.3.1: Copy and Paste Caption here. (Copyright; author via source)

The only difference from before is that the first block contains the $MACkey$, so its value is not known to the attacker. We have shaded it in gray here. The attacker knows all other inputs as well as the output tag t.

I claim that the attacker can now exactly predict the tag of:

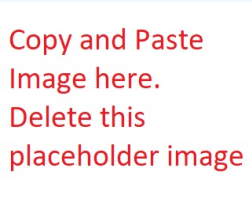

Figure 11.3.1: Copy and Paste Caption here. (Copyright; author via source)

The correct MAC tag t' of this value would be computed by someone with the key as:

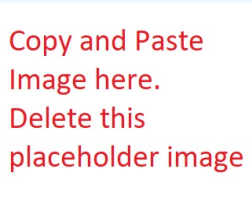

Figure 11.3.1: Copy and Paste Caption here. (Copyright; author via source)

The attacker can compute the output t' in a different way, without knowing the key. In particular, the attacker knows all inputs to the last instance of h. Since the h function itself is public, the attacker can compute this value herself as $t' = h(t||0000000001000000)$. Since she can predict the tag of m', having seen only the tag of m, she has broken the MAC scheme.

Discussion

- In our example, the attacker sees the MAC tag for m (computed as $H(k\|m)$) and then forges the tag for $m' = m\|p$, where p is the padding you must add when hashing $k\|m$. Note that the padding depends only on the length of k, which we assume is public.
- The same attack works to forge the tag of any m' that begins with $m\|p$. The attacker would simply have to compute the last several rounds (not just one round) of MerkleDamgård herself.
- **This is not an attack on collision resistance!** Length-extension does not result in collisions! We are not saying that $k\|m$ and $k\|m\|p$ have the same hash under H, only that knowing the hash of $k\|m$ allows you to also compute the hash of $k\|m\|p$.

Knowing how $H(k\|m)$ fails to be a MAC helps us understand better ways to build a secure MAC from a hash function:

- The Merkle-Damgård approach suffers from length-extension attacks because it outputs its **entire internal state**. In the example picture above, the value t is both the output of $H(k\|m)$ as well as the only information about $k\|m$ needed to compute the last call to h in the computation $H(k\|m\|p)$.

  One way to avoid this problem is to only output part of the internal state. In MerkleDamgård, we compute $y_i := h\left(y_{i-1}\|x_i\right)$ until reaching the final output y_{k+1}. Suppose instead that we only output half of y_{k+1} (the y_i values may need to be made longer in order for this to make sense). Then just knowing half of y_{k+1} is not enough to predict what the hash output will be in a length-extension scenario.

  The hash function SHA-3 was designed in this way (often called a "wide pipe" construction). One of the explicit design criteria of SHA-3 was that $H(k\|m)$ would be a secure MAC.

- Length extension with Merkle-Damgård is possible because the computation of $H(k\|m)$ exactly appears during the computation of $H(k\|m\|p)$. Similar problems appear in plain CBC-MAC when used with messages of mixed lengths. To avoid this, we can "do something different" to mark the end of the input. In a "wide pipe" construction, we throw away half of the internal state at the end. In ECBC-MAC, we use a different key for the last block of CBC chaining.

  We can do something similar to the $H(k\|m)$ construction, by doing $H\left(k_2\|H\left(k_1\|m\right)\right)$, with independent keys. This change is enough to mark the end of the input. This construction is known as NMAC, and it can be proven secure for MerkleDamgård hash functions, under certain assumptions about their underlying compression function. A closely related (and popular) construction called HMAC allows k_1 and k_2 to even be related in some way.
