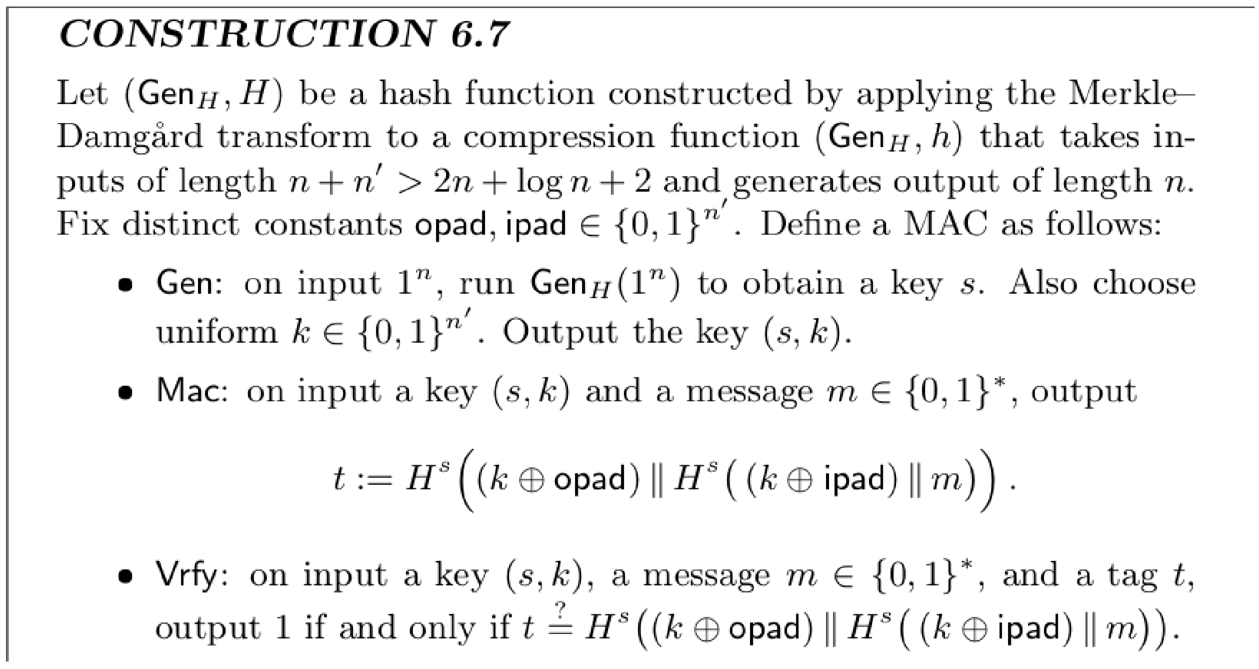
HMAC CONCEPTS

Overview

We can use collision-resistant hash functions for domain extension of message authentication codes.
Say we have a fixed-length MAC for l(n)-bit messages, and a collision-resistant hash function with l(n)-bit output
length. Then we can authenticate an arbitrary-length message m by using the MAC to authenticate the hash of m.
Intuitively, this is secure because the MAC ensures that the attacker cannot authenticate any new hash value, while collision resistance ensures that the attacker will be unable to find any new message that hashes to a previously used hash value.
In principle, the hash-and-MAC approach from the previous section could be instantiated by combining an arbitrary collision-resistant hash function with the fixed-length MAC

Code construction:



CONSTRUCTION 6.7

Let (Gen_H, H) be a hash function constructed by applying the Merkle–Damgård transform to a compression function (Gen_H, h) that takes inputs of length $n + n' > 2n + \log n + 2$ and generates output of length n. Fix distinct constants $\mathsf{opad}, \mathsf{ipad} \in \{0,1\}^{n'}$. Define a MAC as follows:

- Gen: on input 1^n, run $\mathsf{Gen}_H(1^n)$ to obtain a key s. Also choose uniform $k \in \{0,1\}^{n'}$. Output the key (s, k).
- Mac: on input a key (s, k) and a message $m \in \{0,1\}^*$, output

$$t := H^s\Big((k \oplus \mathsf{opad}) \,\|\, H^s\big((k \oplus \mathsf{ipad}) \,\|\, m\big)\Big).$$

- Vrfy: on input a key (s, k), a message $m \in \{0,1\}^*$, and a tag t, output 1 if and only if $t \overset{?}{=} H^s\big((k \oplus \mathsf{opad}) \,\|\, H^s\big((k \oplus \mathsf{ipad}) \,\|\, m\big)\big)$.

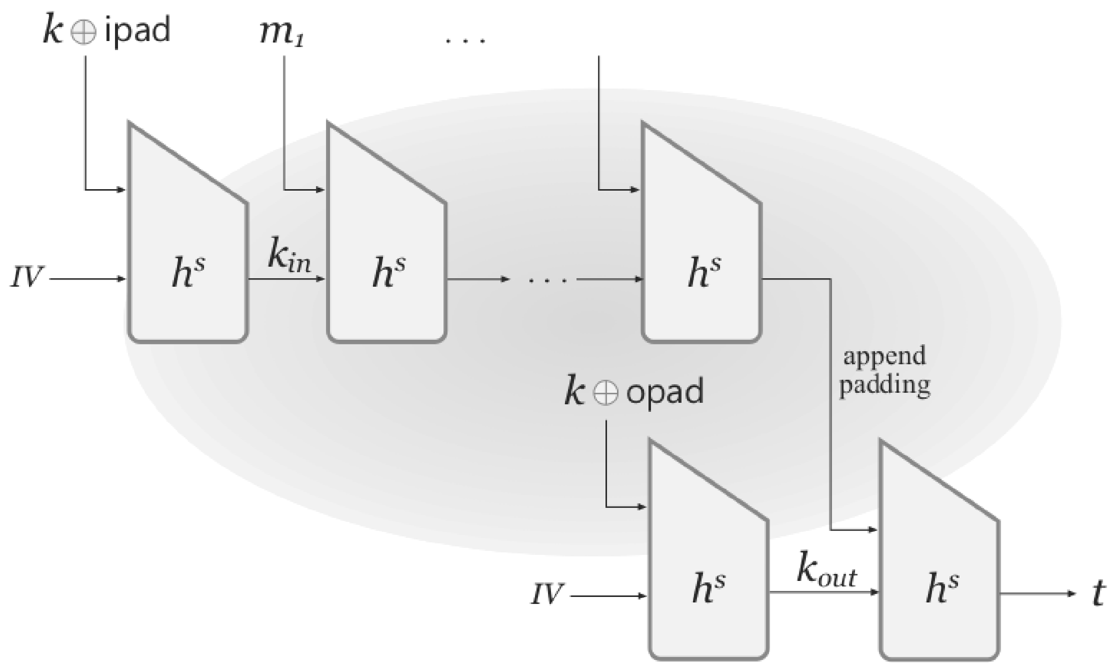


FIGURE 6.2: HMAC, pictorially.

THEOREM 6.8 *Assume G^s is a pseudorandom generator, $\widetilde{\Pi}^s$ is a secure fixed-length MAC for messages of length n', and (Gen_H, h) is collision resistant. Then HMAC is a secure MAC (for arbitrary-length messages).*

(We require the first two assumptions in the theorem to hold for all s. Even if G^s is not expanding, it is still meaningful to speak of its output as being pseudorandom.) Because of the way the compression function h is typically designed (see Section 7.3.1), the first two assumptions are reasonable.

The roles of ipad and opad. One might wonder why it is necessary to incorporate k_{in} (or k itself) in the "inner" computation at all. In particular, for the hash-and-MAC approach all that is required is for the inner computation to be collision resistant, which does not require any secret key. The reason for including a secret key as part of the inner computation is that this allows security of HMAC to be based on the assumption that (Gen_H, H) is *weakly* collision resistant, where (informally) this refers to an experiment in which an attacker needs to find collisions in a secretly keyed hash function. This is a weaker condition than collision resistance, and hence is potentially easier to satisfy. The defensive design strategy of HMAC paid off when it was discovered that the hash function MD5 (see Section 7.3.2) used in HMAC–MD5 was *not* collision resistant. The attacks on MD5 did not violate *weak* collision resistance, and so HMAC–MD5 was not broken even though MD5 was. (Despite this, HMAC–MD5 should no longer be used now that weaknesses in MD5 are known.) This gave developers time to replace MD5 in HMAC implementations, without immediate fear of attack.

Ideally, *independent* keys k_{in}, k_{out} should have been used in the inner and outer computations. To reduce the key length of HMAC, a single key k is used to derive k_{in} and k_{out} using ipad and opad. (Moreover, in practice it is typical for the length of k to be much shorter than n'—in which case k is simply padded with 0s before being XORed with ipad and opad.) If we assume that G^s (as defined above) is a pseudorandom generator for any s, then k_{in} and k_{out} can be treated as independent, uniform keys when k is uniform.