MACing longer messages

Recommended reading:

KL, Sections 5.1, 5.3

Suppose that we have a MAC that lets us authenticate messages of length l. How can we extend it to authenticate longer messages?

Natural approach: (this worked for encryption!!) Given long message $m \in \{0,1\}^{t \cdot l}$, split it into $m = m_1 \| m_2 \| \cdots \| m_t$ each of length l, then MAC it as

$$MAC'(k, m) = (MAC(k, m_1), \ldots, MAC(k, m_t))$$

Problem: This allows an attacker to reorder the m_i's as he wishes! This is really bad.

How can we avoid this?

Natural solution: append unique identifier to each m_i!

$$MAC'(k, m) = MAC(k, (1, m_1)), \ldots, MAC(k, (t, m_t))$$

↳ now the m_i's need to be slightly shorter.

Problem: If adversary gets tags for $m = m_1 \| \dots \| m_t$ and $m' = m'_1 \| \dots \| m'_t$, they can get a correct tag for $m'' = m_1 \| m'_2 \| m_3 \| \dots \| m'_t$ ...

How can we overcome this?

Something that actually works:

$MAC'(k, m)$: $m = m_1 \| \dots \| m_t$, each m_i of length $\ell \approx n/2 - 2\lg t$

→ sample $r \leftarrow \{0,1\}^{n/2}$

→ output $(MAC(k, (r, 1, \ell, m_1)), \dots, MAC(k, (r, t, \ell, m_t)))$

↳ now MAC' is randomized

This is too slow in practice ...

In the real world: Hash-then-MAC

→ Hash m to small digest $H(m)$ of length ℓ

→ MAC $H(m)$.

What properties do we require of our hash function?

↳ something that maps long inputs to short outputs

If we could find m, m' s.t. $H(m) = H(m')$, then we would break the MAC, because the tags of m and m' coincide.

$\Rightarrow$ H should be collision-resistant

Intuitively, no efficient adversary should be able to find a collision, ie, distinct inputs m, m' s.t. $H(m) = H(m')$.

How to formalize this?

Claim: No hash function H is "collision-resistant".

Proof: Let H be any hash function with output length $\ell(n)$ for a security parameter n. We know that there exist distinct $x, x' \in \{0,1\}^{\ell(n)+1}$ such that $H(x) = H(x')$. Consider the PPT adversary A that simply outputs (x, x'). ▨

So ... Hmm...

In theory, we must consider keyed families of hash functions!
More on theory vs. practice below.

Def (hash function): A hash function (with output length $\ell(n)$) is a pair of PPT algorithms (Gen, H) s.t.:

- Gen takes as input 1^n and outputs a key s.
  Intuitively, s describes the hash function chosen from the family.
- H takes as input the key s and $x \in \{0,1\}^*$ and outputs $H_s(x) = H(s,x) \in \{0,1\}^{\ell(n)}$.

If H_s is only defined on inputs of length $\ell'(n)$, for some $\ell'(n) > \ell(n)$, we call (Gen, H) a fixed-length hash function.

Def (collision-resistance): A hash function (Gen, H) is collision-resistant if for every PPT adversary A there exists a negligible function $\varepsilon(n)$ such that A wins the following collision finding game with probability at most $\varepsilon(n)$:

→ We sample $s \leftarrow Gen(1^n)$ and give s to A

→ $A(1^n, s)$ outputs (x, x') and wins if $x \neq x'$ and $H_s(x) = H_s(x')$.

Note: We reveal the "key" s to the adversary. This is actually not needed for the MAC application, but it's a very useful notion elsewhere!

Thm (Hash-then-MAC): Suppose that (Gen_1, H) is a collision-resistant hash function with output length $\ell(n)$ and (Gen_2, MAC, Ver) is a secure MAC for messages of length $\ell(n)$. Then, (Gen', MAC', Ver') defined below is a secure MAC for arbitrary-length messages:

- $Gen'(1^n)$ samples $s \leftarrow Gen_1(1^n)$ and $k \leftarrow Gen_2(1^n)$
- $MAC'(s,k,m) = MAC(k, H_s(m))$
- $Ver'(s,k,m,t) = Ver(k, H_s(m), t)$

Proof: Suppose that (Gen', MAC', Ver') is not a secure MAC. This means that there is a PPT adversary A' that wins the MAC forgery game for (Gen', MAC', Ver') with non-negligible probability $\geq \frac{1}{p(n)}$:

- $A'^{\mathcal{O}_{MAC',s,k}}(1^n)$ queries the authentication oracle $\mathcal{O}_{MAC',s,k}$ on various messages $m_1, \ldots, m_{q(n)}$ and learns tags $t_1, \ldots, t_{q(n)}$. Let Q be the set of queried messages.
- A' outputs (m^*, t^*) such that

$$Pr(m^* \notin Q, Ver'(s,k,m^*,t^*) = 1) \geq \frac{1}{p(n)}$$

Note that A' only wins the game above if one of the following happens:

Case 1: there is $m \in Q$ such that $H_s(m) = H_s(m^*)$

Case 2: $H_s(m^*) \neq H_s(m)$ for all $m \in Q$, and
$Ver(k, H_s(m^*), t^*) = 1$

Let E_i be the event that case i holds. Then,

$$\frac{1}{p(n)} \leq Pr(A' \text{ wins game above}) \leq Pr(E_1) + Pr(E_2). \quad (*)$$

(union bound.)

This means that at least one of $Pr(E_1)$ or $Pr(E_2)$ is non-negligible.

We build an adversary A_1 for the collision finding game for (Gen_2, H) such that $Pr(A_1 \text{ wins}) = Pr(E_1)$,
and an adversary A_2 for the MAC forgery game for (Gen_2, MAC, Ver) such that $Pr(A_2 \text{ wins}) = Pr(E_2)$.

A_1 behaves as follows on input $(1^n, s)$:

→ Sample $k \leftarrow Gen_2(1^n)$.

→ Run $A'(1^n)$. If A' queries the auth-oracle on m then A_1 computes $H_s(x)$ and returns $MAC(k, H_s(x))$ to A'. Let Q be the set of queried messages.

→ If A' outputs (m^*, t^*), search for $m \in Q$ such that $m^* \neq m$ but $H_s(m^*) = H_s(m)$. If such m exists, output (m^*, m). Else, abort

$A_2^{\mathcal{O}_{MAC,k}}$ behaves as follows:

→ Sample $s \leftarrow Gen_1(1^n)$.

→ Run $A'(1^n)$. If A' queries the auth-oracle on m then A_1 computes $H_s(x)$ and queries the auth oracle $\mathcal{O}_{MAC,k}$ for messages of length $\ell(n)$ on $H_s(m)$ and returns its output to A'. Let Q be the set of queried messages

→ If A' outputs (m^*, t^*), check if $H_s(m^*) \neq H_s(m)$ for all $m \in Q$. If so, output $(H_s(m^*), t^*)$.

By (*), at least one of A_1 or A_2 wins their respective game with non-negligible probability, a contradiction. ▨

In practice ... people don't use keyed families of collision-resistant hash functions, but rather just one fixed hash function, such as SHA-256

Recall that we can't hope that collision-resistance holds for a fixed hash function, so our proof doesn't apply.
So ... Why do we do this in practice, and why do we care about the theory. Here are two valid reasons (for me):

→ The "attack" that breaks collision-resistance can really only be implemented in practice if we already know a collision. We don't know how to efficiently find collisions for SHA-256, so in practice we can use it as a collision-resistant hash function.

→ Our theoretical analysis shows that the Hash-then-MAC methodology is sound, if we believe in the collision-resistance of H.