Message Authentication Codes (MACs)

Noah Stephens-Davidowitz

June 8, 2023

1 From secrecy to authenticity

In our first lecture, we met Alice and Bob. Alice wanted to communicate with Bob without revealing any information to an eavesdropper Eve. We then spent almost a month figuring out how to do this—using one-way functions to build pseudorandom generators to build pseudorandom functions to build secret-key encryption.

But, now Alice and Bob have another problem (ugh!). Bob has received a message, but he's not sure that it came from Alice. Perhaps Eve sent the message? Even if the message was encrypted using a semantically secure encryption scheme with the secret key that only Alice and Bob hold, this does not guarantee that the message was actually sent by Alice.

In other words, until now, we have only been concerned with the *secrecy* of their communication—the guarantee that "nobody else can read Alice and Bob's messages"—but we have not been concerned at all with the *authenticity* of their communication—the guarantee that "nobody can forge a message in a way that makes it look like it was sent by Alice."

Both properties are extremely important. E.g., if Alice sends the message "ATTACK AT DAWN" to Bob, she certainly wants the message to stay secret. (If Eve knew the contents of this message, that would be a huge problem!) But, perhaps just as importantly, if Bob receives a message saying "ATTACK AT DUSK", he needs to know whether this message came from Alice. (It would be a big problem if Eve could tell Bob when to attack!)

Even in situations where we don't care about secrecy, we still often care about authenticity. For example, remember that Alice and Bob got divorced in lecture 5? They then decided to flip a coin to decide who got to keep their cat Whiskers. But, what if Bob loses the coinflip and then refuses to accept defeat? How could Alice prove in divorce court that Bob really did agree to flip a coin for Whiskers?

1.1 Squiggles don't work

In the physical world, we often solve this using a squiggly signature written on a piece of paper. I.e., in order to "prove" to Bob that she was the one to send the plaintext message "ATTACK AT DAWN", Alice writes her name in some complicated squiggly way (something like "$\mathscr{ALICE}$") below the plaintext. I.e., she "signs" it. (I'm using this silly "squiggle" terminology both to point out how silly this is, and because we will want to use the word "signature" for something else.) Similarly, before Alice and Bob flip a coin for Whiskers, they both write on a piece of paper "We both agree to flip a coin for Whiskers. Bob wins if the coin comes up heads and Alice wins if the coin comes up tails." Then they each draw on the piece of paper their respective squiggles,

"$\mathscr{ALICE}$" and "$\mathscr{BOB}$". (They should also probably draw some squiggles their messages in the coin-flipping protocol.) Then, later, if the coin comes up tails, Alice can take this piece of paper with Bob's squiggle on it to divorce court and tell the judge "See?! Bob squiggled on this piece of paper!" and the judge will let her have Whiskers.

I guess the hope is that if Alice and Bob's squiggles are sufficiently squiggly, then it should be hard for anyone else to produce a squiggle that looks the same.

Frankly, this idea is just absolutely horrible (though somehow we do use it for a *lot* of things, and it leads to fewer problems than you might expect). E.g., think about the "ATTACK AT DAWN" story from Eve's perspective. Eve intercepts Alice's letter to Bob (because she's an eavesdropper; that's just what she does). The end of Alice's letter has these funny squiggles "$\mathscr{ALICE}$" on them, which Eve immediately recognizes as Alice's squiggle. Now, Eve can send whatever messages she wants to Bob, and as long as she ends the message with these funny squiggles "$\mathscr{ALICE}$", Bob will think that the message came from Alice! So, while Alice's original message was "ATTACK AT DAWN", Eve can replace it with the message "ATTACK AT DUSK" or "SURRENDER TO EVE" or "GIVE EVE WHISKERS" or whatever. (With the wonders of modern technology, she can just copy and paste the squiggle from one message to another.)

The same issue arises in the divorce story with Whiskers. If the coin comes up tails, Bob can replace the original agreement (which was "We both agree to flip a coin for Whiskers. Bob wins if the coin comes up heads and Alice wins if the coin comes up tails.") with a new message "...Bob wins if the coin comes up tails." He can then put Alice's squiggle "$\mathscr{ALICE}$" on this new message and bring it to a judge to argue that *he* won the coin flip, so *he* should get Whiskers.

The only sense in which this system of squiggly signatures is secure is if it is somehow extremely difficult to reproduce the squiggles "$\mathscr{ALICE}$" *or* even to just transfer them from one piece of paper to another, *or* even to just erase the original plaintext that Alice signed and replace it with a new one. This is, of course, absurd.

A much much better solution would be for Alice and Bob to somehow produce a signature that somehow *incorporates the plaintext that they're signing*. In other words, for different plaintexts m, Alice should produce different signatures that (1) confirm to Bob that she herself signed specifically the plaintext m; and (2) do not allow an adversary to forge a signature for some other plaintext m'.

2 MACs

We make the above formal in the secret-key setting by defining the notion of a *message authentication code*, or a MAC. (Later, we will see the public-key variant of this, which is called a *digital signature*—or just a signature. MACs are useful for dealing with some of the problems described above, but not all of them. Signatures more-or-less solve all of the problems described above (at the cost of being significantly less efficient than MACs in practice).)

As we did with encryption schemes, we will first define correctness of MACs and then worry about security. Intuitively, a MAC consists of a key-generation algorithm, a MAC algorithm, and a verification algorithm. The key generation algorithm generates a key (duh). The MAC algorithm takes as input the key and a plaintext m and outputs a *tag* τ. The verification algorithm takes as input the key, a plaintext, and a tag τ, and confirms that the tag is valid. (We use the word *tag* here because, again, w are saving the word signature for something else. Sometimes, people simply refer to the tag itself as a "MAC.")

Definition 2.1. *A MAC is a triple of PPT algorithms* $(\mathsf{Gen}, \mathsf{MAC}, \mathsf{Ver})$ *with the following behavior.*

1. Gen *takes as input* 1^n *and outputs a key* $k \in \{0,1\}^n$. *(Here, we are assuming that the key* k *is an* n*-bit string for simplicity.)*
2. MAC *takes as input a key* $k \in \{0,1\}^n$ *and a plaintext* $m \in \{0,1\}^n$ *and outputs a tag* $\tau \in \{0,1\}^*$.
3. Ver *takes as input a key* $k \in \{0,1\}^n$*, plaintext* $m \in \{0,1\}^n$*, and tag* $\tau \in \{0,1\}^*$ *and outputs either* 0 *or* 1 *(i.e., either "invalid" or "valid").*
4. **(Correctness.)** *For any* n *and any plaintext* $m \in \{0,1\}^n$,

$$\Pr_{k \leftarrow \mathsf{Gen}(1^n)}[\mathsf{Ver}(k, m, \mathsf{MAC}(k,m)) = 1] = 1 \; .$$

(In other words, "the Ver *algorithm always accepts a tag* τ *produced by the MAC algorithm on the correct plaintext* m *with the correct key* k*.") Here, all three algorithms can potentially be randomized (certainly,* Gen *should be randomized), and the probability is therefore over the random coins of all three algorithms. We often think of* Ver *as being deterministic, though this is not strictly necessary.*

Notice that both the MAC algorithm and the Ver algorithm take as input a plaintext m. I.e., a tag τ is associated with a particular plaintext m. So, if Alice provides a tag τ for the plaintext message "ATTACK AT DAWN", τ is not necessarily a valid tag for the plaintext "ATTACK AT DUSK". (In fact, to achieve security, τ probably shouldn't be a valid tag for both messages simultaneously.)

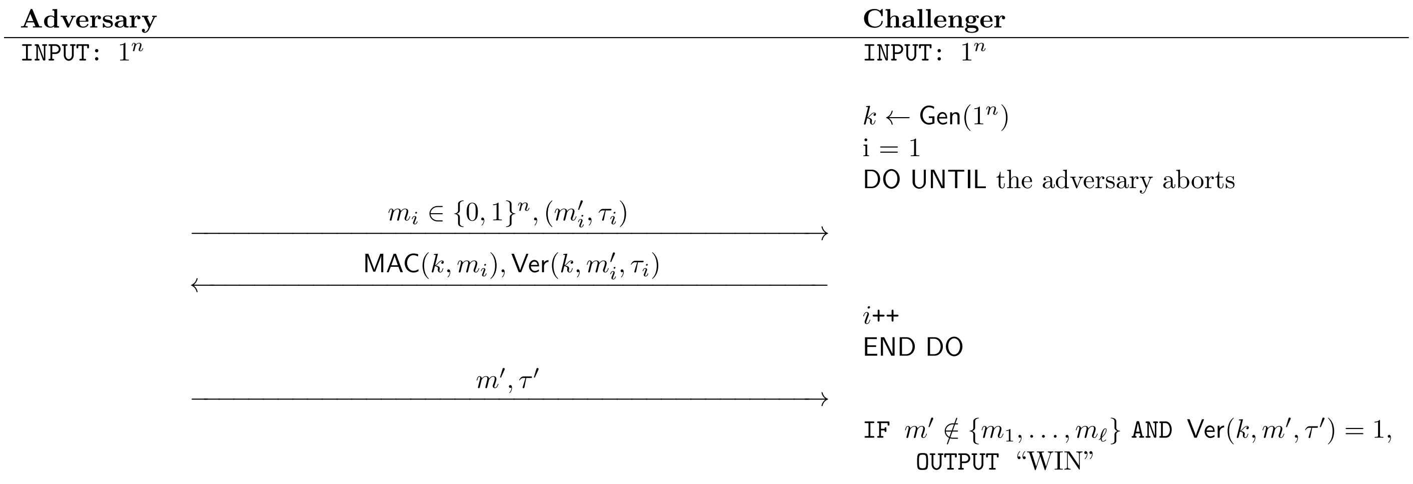
2.1 Security

Of course, a MAC is quite boring without some notion of security. (If we only wanted to satisfy correctness, then the scheme in which Ver *always* outputs 1 would suffice!) Our security definition for a MAC should formalize the intuitive idea that the adversary should not be able to forge a tag for a new message.

To make this formal, we imagine an adversary that can see many different tags τ_i of many different plaintexts m_i. (I.e., we imagine that Eve has seen many messages from Alice that include a tag.) In fact, to be extra careful, we let the adversary choose the plaintexts herself. This is similar to how in the semantic security game, we let the adversary choose the plaintexts to be encrypted. Here, we will even allow the adversary to do this adaptively—i.e., she may choose which plaintext m_{i+1} he wants to sign next *after* she has seen the tags for $m_1, \ldots, m_i$. The adversary will also have access to the Ver algorithm. We then ask the adversary to produce a tag $\tau' \in \{0,1\}^*$ for *any* plaintext $m' \in \{0,1\}^n$, and the adversary wins if (1) m' is a new message (i.e., $m' \neq m_i$ for all i); and (2) $\mathsf{Ver}(k, m', \tau') = 1$. In other words, the adversary wins if she can convince Bob (the verifier running the Ver algorithm) that Alice tagged a message m' that Alice hasn't actually tagged.

Notice that it's not interesting to produce a valid tag for a message that Alice already tagged herself. First of all, this would be trivial to do—just use the tag you already know Second, we don't want such protection—if Alice really did send the message "ATTACK AT DAWN" to Bob, then we don't mind if Eve is able to convince Bob that Alice sent the message "ATTACK AT DAWN" to Bob.

We capture all of this formally in the following game between an adversary and a challenger.


Adversary
Challenger
INPUT: 1^n
INPUT: 1^n
$k \leftarrow \mathsf{Gen}(1^n)$
i = 1
DO UNTIL the adversary aborts
$m_i \in \{0,1\}^n, (m'_i, \tau_i)$
$\mathsf{MAC}(k, m_i), \mathsf{Ver}(k, m'_i, \tau_i)$
i++
END DO
m', τ'
IF $m' \notin \{m_1, \ldots, m_\ell\}$ AND $\mathsf{Ver}(k, m', \tau') = 1$,
OUTPUT "WIN"

Definition 2.2. *A MAC is* secure *if for any PPT adversary $\mathcal{A}$ there exists a negligible $\varepsilon(n)$ such that the probability that $\mathcal{A}$ wins the above game is at most $\varepsilon(n)$.*

The following definition is *equivalent* and replaces the interactive game described above with a much more succinct oracle-based definition.

Definition 2.3. *A MAC is* secure *if for any PPT adversary $\mathcal{A}$ there exists a negligible $\varepsilon(n)$ such that*

$$\Pr_{k \leftarrow \mathsf{Gen}(1^n)}[(m', \tau') \leftarrow \mathcal{A}^{\mathsf{Mac}(k,\cdot),\mathsf{Ver}(k,\cdot,\cdot)}(1^n),\ m' \notin Q \text{ and } \mathsf{Ver}(k, m', \tau') = 1] \leq \varepsilon(n)\ ,$$

where $\mathsf{Mac}(k,\cdot)$ is an oracle that takes as input a plaintext $m \in \{0,1\}^n$ and outputs $\mathsf{Mac}(k,m)$, $\mathsf{Ver}(k,\cdot,\cdot)$ is an oracle that takes as input a plaintext $m \in \{0,1\}^n$ and a tag τ and outputs $\mathsf{Ver}(k,m,\tau)$, and Q is the list of queries that $\mathcal{A}$ makes to the oracle $\mathsf{Mac}(k,\cdot)$.

There are actually *many* variants of this definition. Formally, the above definition is known as *existential unforgeability against adaptive chosen message attacks* (EUACMA for "short"). Here, "existential unforgeability" means that the adversary wins if *there exists* a plaintext m' that it can produce a valid tag τ' for. A weaker definition is *universal unforgeability*, in which we only ask that the adversary cannot produce a valid tag for some specific plaintext (e.g., a random plaintext or one that the adversary chooses *before* making its queries). "Adaptive chosen message attack" means that the adversary can *choose* the messages m_i that are tagged, and that she may do so *adaptively*, i.e. choosing m_{i+1} after seeing the tags for $m_1, \ldots, m_i$. A *non-adaptive* chosen message attack would require the adversary to send $m_1, \ldots, m_\ell$ all at once. And, e.g., a *random message attack* only shows the adversary tags on random messages, rather than adversarially chosen messages.

So, the above definition is quite strong! Nevertheless, we can achieve it relatively easily. (There are even stronger definitions than this! For example, we could say that the adversary can win even if $m' \in \{m_1, \ldots, m_\ell\}$, as long as τ' is not one of the tags produced by the challenger—i.e., the adversary wins by producing a valid tag for a new message *or* producing a new valid tag for an old message. We actually achieve this definition too, since our MAC will actually have a unique valid tag for every plaintext, but we do not worry about this here.)

2.2 A simple construction

It turns out that, given the machinery that we have already developed for building SKE, constructing MACs is quite simple. Our MAC will even be deterministic (except, of course, for the key-generation algorithm). In fact, "a PRF is already a MAC." In particular, let $F_k : \{0,1\}^{|k|} \to \{0,1\}^{|k|}$ be a PRF. Then, here is the construction.

- $\mathsf{Gen}(1^n)$: output $k \sim \{0,1\}^n$.
- $\mathsf{MAC}(k, m)$: output $\tau := F_k(m)$.
- $\mathsf{Ver}(k, m', \tau')$: output 1 if and only if $\tau' = F_k(m')$.

Intuitively, it should be clear why this MAC should be unforgeable. For a fixed key $k \in \{0,1\}^n$, each plaintext $m \in \{0,1\}^n$ has a unique tag $F_k(m) = \tau \in \{0,1\}^n$ such that $\mathsf{Ver}(k, m, \tau) = 1$. So, in order for the adversary to forge a valid tag τ' for any message m' on which he has not queried his MAC oracle, he must guess the value $F_k(m')$. But, since F_k is a PRF, $F_k(m')$ is indistinguishable from random from the adversary's perspective.

Below, we prove this formally. Notice in particular that this implies that secure MACs exist if OWFs exist.

Theorem 2.4. *The above MAC is secure (i.e., EUACMA).*

Proof. It is trivial to see that the MAC is efficient and correct. We prove security, of course, via a reduction. For simplicity, we ignore verification queries in this proof. They do complicate things a bit.

In particular, recall the PRF security game, in which an adversary $\mathcal{A}'$ is given oracle access to some function $H_b : \{0,1\}^n \to \{0,1\}^n$, where H_0 is a random oracle and $H_1 := F_k$ for $k \sim \{0,1\}^n$. And, $\mathcal{A}'$ must guess the bit b.

So, we suppose that the statement is false, i.e., that there exists some adversary $\mathcal{A}$ that wins the above game with non-negligible probability $\varepsilon(n)$. Then, we construct an adversary $\mathcal{A}'$ in the PRF game as follows. $\mathcal{A}'$ receives repeated plaintext queries $m \in \{0,1\}^n$ from $\mathcal{A}$, and each time simply passes the query m to its own oracle H_b, and passes the response $H(m)$ back to $\mathcal{A}$. (I.e., $\mathcal{A}'$ "tells $\mathcal{A}$ that the tag of the message m is equal to $H_b(m)$.")

Eventually, $\mathcal{A}$ finishes with its queries and sends (m', τ') to $\mathcal{A}'$. We assume for simplicity that m' was not queried before (since we can always replace $\mathcal{A}$ by another adversary with at least advantage $\varepsilon(n)$ that never sends an m' that was previously queried). $\mathcal{A}'$ then queries its oracle H_b one additional time on the plaintext m', receiving as output $\tau^* := H_b(m')$. If $\tau^* = \tau'$, then $\mathcal{A}'$ outputs $b' := 1$ (i.e., $\mathcal{A}'$ guesses that $H_b = F_k$). Otherwise, $\mathcal{A}'$ outputs $b' := 0$.

Clearly, $\mathcal{A}'$ is efficient. We have

$$\begin{aligned}\Pr[b' = b] &= \Pr[b' = b \mid b = 0]/2 + \Pr[b' = b \mid b = 1]/2 \\ &= \Pr_{H \sim \{f : \{0,1\}^n \to \{0,1\}^n\}}[\tau' \neq H(m')]/2 + \Pr_{k \sim \{0,1\}^n}[\tau' = \mathsf{MAC}(k, m')]/2 \;.\end{aligned}$$

The second probability is exactly $\varepsilon(n)$. We claim that the first probability is exactly $1 - 2^{-n}$, so that the total probability is $1/2 + \varepsilon(n)/2 - 2^{-n-1}$, which is non-negligibly larger than $1/2$, a contradiction. Indeed, to see this, it suffices to notice that, since m' was never queried by $\mathcal{A}$, $H(m') \in \{0,1\}^n$ is uniformly random and independent of the view of $\mathcal{A}$. Therefore, we must have

$\Pr[\tau' = H(m)] = 2^{-n}$, regardless of how $\mathcal{A}$ behaves. (If we allowed for verification queries, we would have to worry about what happens when m' was a part of a verification query. We would need to argue that with high probability all verification queries made involving m' returned 0, and then argue that conditioned on that $H(m')$ is a uniformly random element from the set of n-bit strings τ that were not part of a verification query with m'.)

The result follows. $\square$

3 MACing longer messages

Our MAC scheme, as defined above, only handles plaintexts $m \in \{0,1\}^n$ of length n when the key $k \in \{0,1\}^n$ has length n. But, what if we want to use our key $k \in \{0,1\}^n$ to MAC a longer plaintext, say $M := (m_1, m_2, \ldots, m_\ell) \in \{0,1\}^{\ell n}$—or even an arbitrarily long plaintext $M \in \{0,1\}^*$?

The analogous problem for encryption schemes was trivial to solve. If you want to encrypt a long message, you can just encrypt the parts individually. However, for MACs, this doesn't work. I.e., suppose that MAC is a secure MAC for n-bit strings, and consider $\mathsf{MAC}'(k, M = (m_1, \ldots, m_\ell)) := (\mathsf{MAC}(k, m_1), \ldots, \mathsf{MAC}(k, m_\ell))$. This is *not* a secure MAC! For example, given a tag of $M := (m_1, \ldots, m_\ell)$ under this MAC, it is trivial to find a valid tag of, e.g., $M' := (m_\ell, m_{\ell-1}, \ldots, m_1)$, or more generally, any reordering of any subset of the plaintexts! This could be a huge problem! E.g., imagine that each of the plaintexts is a single word. Then, an adversary Eve could use a valid tag of the message "EVE GOT A C ON HER EXAM" to "EVE GOT A ON EXAM".

So, one needs to be a little more clever. One could try fixing the ordering of the m_i by trying $\mathsf{MAC}'(k, M = (m_1, \ldots, m_\ell)) = (\mathsf{MAC}(k, (1, m_1)), \ldots, \mathsf{MAC}(k, (\ell, m_\ell))$ (where here we asssume that the plaintexts m_i have length $n - \log \ell$ so that we have extra room to include the indices). But, this still doesn't work. For example, if Eve knows a valid tag for "ALICE GOT AN A ON HER EXAM" *and* a valid tag for "EVE GOT A C ON HER EXAM", it is trivial for her to construct a valid tag for "EVE GOT AN A ON HER EXAM" and also a tag for "ALICE GOT A C ON HER EXAM".

One solution that works is to use the following scheme:

$$\mathsf{MAC}'(k, (m_1, \ldots, m_\ell)) := (r, \mathsf{MAC}(k, (r, 1, \ell, m_1)), \mathsf{MAC}(k, (r, 2, \ell, m_2)), \ldots, \mathsf{MAC}(k, (r, \ell, \ell, m_\ell))) ,$$

where, e.g., $r \sim \{0,1\}^{n/2}$ is sampled uniformly at random. (This is our first example of a MAC where the MAC function is randomized. Again, we have cheated a bit by changing the length of the individual m_i—now they have length roughly $n/2 - 2\log \ell$.)

Intuitively, the uniformly random r prevents the sort of "mix and match" attack above, which allowed Eve to use a MAC of $(m_{1,0}, m_{2,0}, \ldots, m_{\ell,0})$ and a MAC of $(m_{1,1}, m_{2,1}, \ldots, m_{\ell,1})$ to create a new MAC of the plaintext $(m_{1,b_1}, m_{2,b_2}, \ldots, m_{\ell,b_\ell})$ for bits b_i of her choice. This attack will not work on the above scheme because the MACs of the two different plaintexts will have two different random nonces $r_0, r_1 \sim \{0,1\}^{n/2}$ except with probability $2^{-n/2}$. (Recall a "nonce" is what we call a string that is meant to be used only once.)

The indices $1, 2, \ldots, \ell$ prevent an attacker from applying the rearranging attack we discussed before. And, the inclusion of the length ℓ of the total message prevents an attacker from shortening a plaintext: e.g., "EVE IS OUR ALLY'S ENEMY" should not be shortened to "EVE IS OUR ALLY".

It is a nice exercise to prove that this MAC actually is secure.

3.1 Hash-then-MAC, and collision-resistant hash functions

In practice, the above technique is far too slow for MACing very long messages. E.g., it is quite common for people to MAC (or, really, to sign, using the digital signatures that we will discuss later) software that might be billions of bits long. Practical PRFs are quite efficient, but we would still prefer not to apply a PRF billions of times, and more importantly, we don't want to end up with a MAC that is a billion bits long if we can avoid it.

Instead, people use a technique called hash-then-MAC. (It's not a very clever name.) In particular, suppose we have an efficiently computable family of hash functions $H_k : \{0,1\}^* \to \{0,1\}^{|k|}$ (for now, all I require of H_k is that it be efficiently computable and maps arbitrarily long strings to $|k|$-bit strings) and a MAC scheme $(\mathsf{Gen}, \mathsf{MAC}, \mathsf{Ver})$ that allows us to tag only n-bit plaintexts. Then, we can consider the following MAC scheme that works with plaintexts $M \in \{0,1\}^*$ of arbitrary length.

- $\mathsf{Gen}'(1^n)$: Output $k_1 \sim \{0,1\}^n$, $k_2 \leftarrow \mathsf{Gen}(1^n)$.
- $\mathsf{MAC}'(k_1, k_2, M)$: Compute $m := H_{k_1}(M)$ and output $\tau := \mathsf{MAC}(k_2, m)$.
- $\mathsf{Ver}'(k_1, k_2, M, \tau)$: Compute $m := H_{k_1}(M)$ and output $\mathsf{Ver}(k_2, m, \tau)$.

In other words, instead of MACing M, we MAC a hash of M. In practice, it is typically much much faster to hash-then-MAC, rather than using the construction from the previous section. And, clearly, hash-then-MAC allows us to produce much shorter tags—tags whose length is independent of the length of the plaintext M.

Of course, not all hash functions H_k are sufficient for this task. E.g., if the hash function H_k always outputs zero, or if it just outputs the first n bits of its input, then clearly the above will not be a secure MAC.

Much more generally, suppose that an adversary can efficiently find a *collision* $M, M' \in \{0,1\}^*$ such that $M \neq M'$ but $h_k(M) = h_k(M')$. Then, such an adversary could be used to violate the security of our MAC by requesting a tag τ of M, and then using that same tag τ as a tag for M'. This motivates the definition of a *collision-resistant hash function.*

Definition 3.1. *A family of functions $H_k : \{0,1\}^* \to \{0,1\}^{|k|}$ is a* collision-resistant hash function *if the following hold.*

- **(Efficiently computable.)** *There exists a PPT algorithm $\mathcal{B}$ such that $\mathcal{B}(k, m) = H_k(M)$ for all k and M.*
- **(Collision resistant.)** *For every PPT adversary $\mathcal{A}$, there exists a negligible $\varepsilon(n)$ such that*

$$\Pr_{k \sim \{0,1\}^n}[(M, M') \leftarrow \mathcal{A}(k),\ H_k(M) = H_k(M') \text{ and } M \neq M'] \leq \varepsilon(n)\ .$$

Remark. *In the above definition of collision resistance, we give the adversary access to the key k. But, in the MAC game, the adversary is certainly not given the key k. One can therefore imagine a different definition in which the adversary is just given oracle access to H_k. This weaker definitions would be sufficient for building MACs (and is much easier to construct), but we instead give this stronger definition because it is useful in many other context (such as in the next lecture).*

Remark. *Collision-resistant hash functions are very important objects. Unfortunately, there is no known construction of a collision-resistant hash function from one-way functions. In fact, we have oracle separations proving that there can be no simple black-box construction of a CRHF from a OWF. So, in some sense, CRHFs exist outside of Minicrypt.*

In practice, though, we often think of Minicrypt as containing CRHFs, since a random oracle is a CRHF, and since in practice we think of "cryptographic hash functions" like SHA-3 as being collision resistant. (Russell Impagliazzo didn't specify whether Minicrypt contains CRHFs when he defined Minicrypt.) There are also relatively simple examples of collision-resistant hash functions that can be built assuming the hardness of the discrete logarithm or factoring (appropriately defined).

Theorem 3.2. *If H_k is a collision-resistant hash function and $(\mathsf{Gen}, \mathsf{MAC}, \mathsf{Ver})$ is a secure MAC, then $(\mathsf{Gen}', \mathsf{MAC}', \mathsf{Ver}')$ is a secure MAC (with message space $\{0,1\}^*$).*

We prove security via a simple sequence of (just two) games. In fact, this is our first example of a construction that uses two different primitives together—both a MAC and a collision-resistant hash function. In such cases, it is almost always a good idea to apply a sequence of games, which allows us to neatly apply the security of the different primitives one by one.

Proof. Let Game 1 be the EUACMA-security game against Mac'.

To define Game 2, we write $M_1, \ldots, M_q$ for the queries made by the adversary, and we write M' for the plaintext that the adversary attempts to produce a valid MAC for. Then, let Game 2 be the same as Game 1, except we replace the condition that $M' \neq M_i$ by the condition $H_{k_1}(M') \neq H_{k_2}(M_i)$. In other words, instead of requiring that the adversary tag a *new* plaintext, we require the adversary to tag a plaintext with a new *hash*. (Notice that this makes things a bit harder for the adversary.)

The result then follows from the following two claims. Intuitively, the first claim is proven via a reduction from collision resistance, in which the reduction computes MAC itself, and the second game is proven via a reduction from MAC security, in which the reduction computes H itself.

Claim 3.3. *Any PPT adversary $\mathcal{A}$,*

$$\Pr[\mathcal{A} \textit{ wins Game } 1] - \Pr[\mathcal{A} \textit{ wins Game } 2]$$

is negligible, assuming that H_k is collision resistant.

Proof. Suppose not. Then, $\mathcal{A}$ must produce a query plaintext M_i and another plaintext M' with $M_i \neq M'$ such that $H_{k_1}(M_i) = H_{k_1}(M')$ with non-negligible probability.

So, consider the adversary $\mathcal{A}'$ in the collision resistance game for H_{k_1} that behaves as follows. $\mathcal{A}'$ takes as input a key $k_1 \sim \{0,1\}^n$ and samples a MAC key $k_2 \sim \mathsf{Gen}(1^n)$. It then takes the queries $M_1, \ldots, M_q$ from $\mathcal{A}$, each time responding with $\mathsf{MAC}(k_2, H_{k_1}(M_i))$. Eventually, $\mathcal{A}$ sends the plaintext M' and tag τ' to $\mathcal{A}'$. Then, $\mathcal{A}'$ checks if $H_{k_1}(M') = H_{k_2}(M_i)$ (and $M' \neq M_i$) for any i, and if so, it simply outputs (M_i, M'). (Otherwise, it simply fails.)

Notice that the view of $\mathcal{A}$ is distributed identically to its view in an honest run Game 1. So, it must be the case that $\mathcal{A}'$ successfully finds a collision with non-negligible probability. But, this contradicts the collision resistance of H. So, such an $\mathcal{A}$ cannot exist. □

Claim 3.4. *No adversary has non-negligible advantage in Game 2.*

Proof. Suppose not. I.e., there exists an adversary $\mathcal{A}$ that has non-negligible probability of winning the EUACMA game against MAC' with the extra guarantee that $H_{k_1}(M') \neq H_{k_1}(M_i)$ for all i.

Then, we construct an adversary $\mathcal{A}'$ in the EUACMA game against the original MAC MAC as follows. It first samples a uniformly random key $k_1 \sim \{0,1\}^n$. For each query M_i made by the adversary $\mathcal{A}$, $\mathcal{A}'$ simply queries its oracle on $m_i := H_{k_1}(M_i)$, receiving as output a tag τ_i, which it passes to $\mathcal{A}$. Finally, $\mathcal{A}$ sends the message M' and tag τ', and $\mathcal{A}'$ outputs $m' := H_{k_1}(M')$ together with τ'.

Clearly, $\mathcal{A}'$ is efficient. Furthermore, notice that if $H_{k_1}(M') \neq H_{k_1}(M_i)$ for all i, then $m' \neq m_i$ for all i (by definition). Furthermore, τ' is a valid tag of M' under MAC' if and only if τ' is a valid tag of m' under MAC. Therefore, $\mathcal{A}'$ wins its game if and only if $\mathcal{A}$ wins Game 2. Since $\mathcal{A}$ wins with non-negligible probability by assumption, then $\mathcal{A}'$ wins with non-negligible probability, contradicting the fact that MAC is secure.

The result follows. $\square$

The full theorem then follows immediately by combining the two claims. $\square$

4 A note on information-theoretic one-time (and t-time) MACs

It turns out that, if you just want to MAC a *single* plaintexts (or, actually, any a priori fixed polynomial number of plaintexts), then this can be done information theoretically (i.e., in a way that works even against unbounded adversaries). This is MAC analogue of how the one-time pad encryption scheme achieves perfect Shannon security. Here is the formal definition. For simplicity, in this case we assume that the MAC algorithm is deterministic and we replace the Ver algorithm with a simple check that $\tau = \mathsf{MAC}(k, m)$—and we explicitly model the tag space $\mathcal{T}$ and the message space $\mathcal{M}$ with the assumption that $|\mathcal{T}| \geq |\mathcal{M}|$.

Definition 4.1. *A* perfect one-time MAC *is a pair of PPT algorithms* $(\mathsf{Gen}, \mathsf{MAC})$ *with plaintext space* $\mathcal{M}$ *and tag space* $\mathcal{T}$ *such that for any two plaintexts* $m_0, m_1 \in \mathcal{M}$ *with* $m_0 \neq m_1$ *and any two tags* $\tau_0, \tau_1 \in \mathcal{T}$*,*

$$\Pr_{k \leftarrow \mathsf{Gen}()}[\mathsf{MAC}(k, m_0) = \tau_0 \text{ and } \mathsf{MAC}(k, m_1) = \tau_1] = 1/|\mathcal{T}|^2 .$$

Intuitively, the above definition means that, no matter what tag τ_0 the adversary sees for the message m_0, the unique valid tag for m_1 is uniformly random. So, no matter what the adversary does, if she only sees a single tag (i.e., if the MAC is only used one time), she cannot guess the correct tag of any other plaintext with probability better than random. Another way to phrase this is to say that the tags $\mathsf{MAC}(k, m_0)$ and $\mathsf{MAC}(k, m_1)$ are uniformly random and independent, or alternatively, that the full list of tags for all possible plaintexts $\mathsf{MAC}(k, m_1), \mathsf{MAC}(k, m_2), \ldots, \mathsf{MAC}(k, m_N)$ are *pairwise independent* uniform random variables.

First, let's see a MAC scheme that *does not* achieve this definition: the one-time pad. I.e., suppose that Gen outputs a uniformly random key $k \sim \{0,1\}^n$, and for any plaintext $m \in \{0,1\}^n$, we can define the tag of m as $\tau := \mathsf{MAC}(k, m) := k \oplus m$. This is *not* a good MAC! In particular, given the tag of m_0, the tag of m_1 is *fixed*, $\mathsf{MAC}(k, m_0) = \mathsf{MAC}(k, m_1) \oplus m_0 \oplus m_1$!

For an actually secure solution, we let the plaintext space be $\mathcal{M} = \mathbb{Z}_p$ for some prime p, and we take the tag space to also equal $\mathcal{T} := \mathbb{Z}_p$. The key-generation algorithm outputs *two* uniformly random elements $a, b \sim \mathbb{Z}_p$. Then, a tag of $m \in \mathbb{Z}_p$ is

$$am + b \bmod p .$$

One can then easily check that for any two tags $\tau_0, \tau_1 \in \mathbb{Z}_p$ and any two *distinct* plaintexts $m_0, m_1 \in \mathbb{Z}_p$, there is a unique pair of elements $a, b \in \mathbb{Z}_p$ such that $am_0 + b = \tau_0 \bmod p$ *and* $am_1 + b = \tau_1 \bmod p$. Specifically, $a := (\tau_1 - \tau_0)(m_1 - m_0)^{-1} \bmod p$ (where the inverse is taken modulo p) and take $b := \tau_0 - am_0$. This immediately implies that this is a secure one-time MAC.

Much more generally, any *pairwise-independent hash family* h_k yields a one-time MAC. A pairwise independent hash family h_k (often also called a *universal hash function*) is a family of functions such that, well,

$$\Pr_k[h_k(m_0) = \tau_0,\ h_k(m_1) = \tau_1] = 1/|\mathcal{T}|^2$$

for distinct inputs m_0, m_1 and any τ_0, τ_1 in the range $\mathcal{T}$ of the function family h_k. These are very well-studied, and they're clearly equivalent to perfect one-time MACs.

More generally still, one can construct a t-time MAC using what's known as a $(t+1)$-wise independent hash family, which satisfy

$$\Pr_k[h_k(m_1) = \tau_1,\ h_k(m_2) = \tau_2, \ldots, h_k(m_{t+1})] = 1/|\mathcal{T}|^{t+1} \ .$$

Such objects can be constructed efficiently for any polynomially bounded choice of t. (Specifically, one needs key size equal to $(t+1) \cdot \log_2 |\mathcal{M}|$).) This is somehow analogous to how one can generalize the one-time pad with a "t-time pad" by taking the key to have length $t \cdot \log_2 |\mathcal{M}|$. (It's not exactly the same, though. The t-time pad must either be stateful or randomized in order to avoid encrypting the same plaintext twice with the same ciphertext. For MACs, this is not an issue.)

Remark. *MACs as defined in the previous sections (not the information-theoretic MACs from this section) imply the existence of one-way functions. The existence of information-theoretic one-time MACs (and, more generally, t-time MACs) in some sense explains why this fact isn't so easy to prove. A proof will need to somehow use the fact that your MAC isn't just a one-time MAC (or even a t-time MAC for any fixed polynomial t).*

This is similar to the subtleties that arise when trying to prove that semantically secure secret-key encryption implies the existence of one-way functions.