5. MACs and Hash Functions

This chapter deals with message integrity, one of the core areas of cryptography.

5.1 Message Authentication Codes (MACs)

Message Authentication Codes (MACs) do not address privacy of data but provide data integrity, i.e., they prevent data from being changed by unauthorized users. MACs consist of two algorithms (S,V) ("sign" and "verify"). The sign algorithm takes a message and a key and computes a *tag*, also called *authenticator*. The verify algorithm takes a message, a key, and a tag and outputs true if it thinks the tag was correctly generated for the specified message, and false otherwise. One requires that the verify algorithm always outputs true if the tag was generated correctly using the sign algorithm with the respective message and key.

Definition 5.1 (Message Authentication Code) *A* message authentication code *over* $(\mathcal{K},\mathcal{M},\mathcal{T})$ *is a tuple of functions* $\mathsf{I}=(\mathsf{S},\mathsf{V})$ *where* $\mathsf{S}:\mathcal{K}\times\mathcal{M}\to\mathcal{T}$ *and* $\mathsf{V}:\mathcal{K}\times\mathcal{M}\times\mathcal{T}\to\mathcal{BOOL}$ *such that:*

(1) S *and* V *are efficiently computable, and*

(2) *for all* $m\in\mathcal{M}$ *and* $K\in\mathcal{K}$*, and for* $t\leftarrow\mathsf{S}(K,m)$*, we have* $\mathsf{V}(K,m,t)=\mathsf{true}$.

◇

The definition of efficiency is again asymptotic, so strictly speaking we again consider sequences $\mathsf{S}=(\mathsf{S}_n)_{n\in\mathbb{N}}$ and $\mathsf{V}=(\mathsf{V}_n)_{n\in\mathbb{N}}$ with domains/ranges $\mathcal{K}=(\mathcal{K}_n)_{n\in\mathbb{N}}$, $\mathcal{M}=(\mathcal{M}_n)_{n\in\mathbb{N}}$, and $\mathcal{T}=(\mathcal{T}_n)_{n\in\mathbb{N}}$. This parameter n is again called *security parameter*. As usual, "efficient computable" is now defined as running in probabilistic polynomial time in n. In the following we will again omit to explicitly write these sequences in order to increase readability; the theorem statements also do not require explicit sequence notation.

Security of MACs is defined by requiring that no efficient adversary can *forge* a tag, i.e., intuitively no valid tag can be computed without knowing the key. This is, again, defined in terms of a cryptographic game with the following challenger.

Definition 5.2 (MAC Challenger) *Let* $\mathsf{I}=(\mathsf{S},\mathsf{V})$ *be a MAC over* $(\mathcal{K},\mathcal{M},\mathcal{T})$*. Then the* MAC challenger *for* I *is defined as follows:*

- *It first randomly chooses a key* $K\leftarrow_{\mathcal{R}}\mathcal{K}$.
- *It then receives messages* $m_i\in\mathcal{M}$ *and outputs* $t_i\leftarrow\mathsf{S}(K,m_i)$*. This stage may repeat arbitrarily but finitely often, thus yielding a sequence of pairs* $(m_1,t_1),\ldots,(m_q,t_q)$.
- *Finally, it receives a pair* (m^*,t^*)*. If* t^* *is a valid tag for* m^* *and this pair is not contained in the obtained sequence, i.e., if* $\mathsf{V}(K,m^*,t^*)=\mathsf{true}$ *and* $\forall i\in\{1,\ldots,q\}\colon(m_i,t_i)\neq(m^*,t^*)$*, then it outputs* true*, otherwise* false.

◇

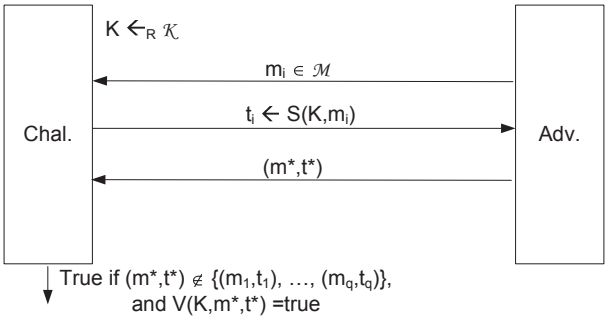

Figure 5.1: The Attack Game for MACs

The attack game between the MAC challenger and an adversary is depicted in Figure 5.1. The adversary wins the game if he is able to compute a tuple (m^*, t^*), such that the challenger outputs true. This tuple is called an *existential forgery*. Another type of forgery is *universal forgery*, where an adversary has been able to find a tag t' *for every* (given) message m'.

Definition 5.3 (MAC Advantage) *Let* $\mathsf{I} = (\mathsf{S}, \mathsf{V})$ *be a MAC over* $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. *We let* $Exp_{\mathsf{A}}^{\mathsf{MAC}}$ *denote the experiment that an adversary* A *interacts with the MAC challenger for* I, *and we let* $Exp_{\mathsf{A}}^{\mathsf{MAC}} = \mathsf{true}$ *denote the event that the challenger finally outputs* true.

The advantage of an adversary A *against the* MAC *challenger for* I *is defined as*

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}] := \mathsf{Pr}\left[Exp_{\mathsf{A}}^{\mathsf{MAC}} = \mathsf{true}\right].$$

◇

For a sequence of adversaries $\mathsf{A} = (\mathsf{A}_n)_{n \in \mathbb{N}}$ we as usual define

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}](n) := Adv^{\mathsf{MAC}}[\mathsf{A}_n, \mathsf{I}_n].$$

Definition 5.4 (Secure MAC) *Let* $\mathsf{I} = (\mathsf{S}, \mathsf{V})$ *be a MAC. Then* I *is* secure against existential forgery under chosen-message attack (CMA) *if* $Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}](n)$ *is negligible for all (sequences of) efficient adversaries* A. *We often say* secure MAC *for brevity instead of a MAC that is secure against existential forgery under chosen-message attack.* ◇

5.1.1 Constructing MACs from PRFs

We will now see a first example of a MAC: it turns out that any PRF is a MAC. This construction is of course not entirely practical as the message space of PRFs we have seen so far is very small. However, we will later see how to increase the size of a PRF's message space, i.e., how to build PRFs with larger domains out of PRFs with smaller domains. For a PRF E over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ we define the MAC $\mathsf{I}_{\mathsf{E}} = (\mathsf{S}, \mathsf{V})$ as follows:

- $\mathsf{S}(K, m) = \mathsf{E}(K, m)$,
- $\mathsf{V}(K, m, t) = \mathsf{true}$ if and only if $\mathsf{E}(K, m) = t$.

This construction yields a secure MAC, as long as E is a PRF and $\frac{1}{|\mathcal{Y}|}$ is negligible. This is captured in the following theorem.

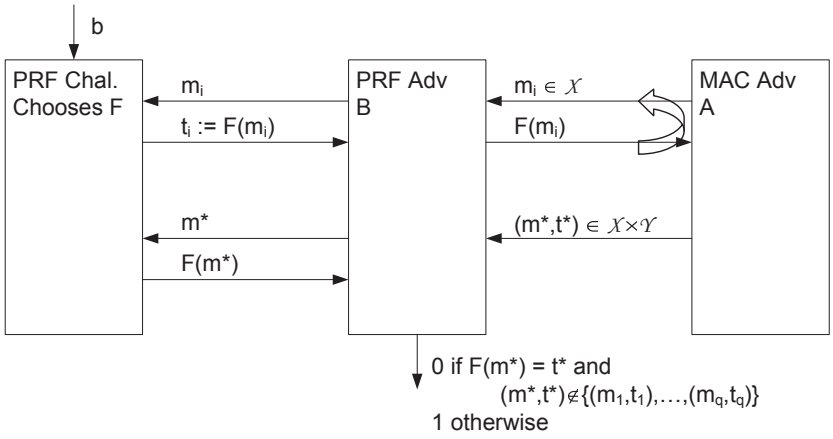

Figure 5.2: Any PRF is a MAC: Proof Sketch

Lemma 5.1 *Let* E *be a PRF over* $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ *and* I_E *as constructed above. Then for every adversary* A *attacking* I_E *there exists an adversary* B *attacking* E *such that*

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}] \leq Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{1}{|\mathcal{Y}|}.$$

In particular, if $\frac{1}{|\mathcal{Y}|}$ *is negligible, then* I_E *is a secure MAC.* □

Proof. Let us consider the adversary B as defined in Figure 5.2. Consider the event $Exp^{\mathsf{PRF}}_{\mathsf{B}}(1) = 0$. We then have $\mathsf{F}(m^*) = t^*$ and $(m^*, t^*) \notin \{(m_1, t_1), \ldots, (m_q, t_q)\}$ for a random function F. First we assume $m^* = m_i$ for some i. Thus $t^* = \mathsf{F}(m^*) = \mathsf{F}(m_i) = t_i$, and hence $(m_i, t_i) = (m^*, t^*)$, which gives us a contradiction. Hence we have $m^* \notin \{m_1, \ldots, m_q\}$. However, this means that $\mathsf{F}(m^*)$ is a random element in $\mathcal{Y}$, thus the probability that $\mathsf{F}(m^*) = t^*$ for some fixed t^* is precisely $\frac{1}{|\mathcal{Y}|}$. Thus, we obtain $\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{B}}(1) = 0\right] = \frac{1}{|\mathcal{Y}|}$.

Next we consider the event $Exp^{\mathsf{PRF}}_{\mathsf{B}}(0) = 0$. We then have $\mathsf{F}(m_i) = \mathsf{E}(K, m_i) = \mathsf{S}(K, m_i)$, thus B always hands over the correct tag to A. Moreover B outputs true if $\mathsf{F}(m^*) = t^* \ \wedge \ (m^*, t^*) \notin \{(m_1, t_1), \ldots, (m_q, t_q)\}$. This is by definition equivalent to $\mathsf{V}(K, m^*, t^*) = \mathsf{true} \ \wedge \ (m^*, t^*) \notin \{(m_1, t_1), \ldots, (m_q, t_q)\}$. Thus, we obtain $\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{B}}(0) = 0\right] = \mathsf{Adv}^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}]$.

Putting this together, we obtain

$$\begin{aligned} Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] &= \left|\mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{B}}(0) = 1\right] - \mathsf{Pr}\left[Exp^{\mathsf{PRF}}_{\mathsf{B}}(1) = 1\right]\right| \\ &= \left|(1 - Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}]) - (1 - \frac{1}{|\mathcal{Y}|})\right| \\ &= \left|Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}] - \frac{1}{|\mathcal{Y}|}\right| \end{aligned}$$

and finally

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{I}_\mathsf{E}] \leq Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{1}{|\mathcal{Y}|}.$$

■

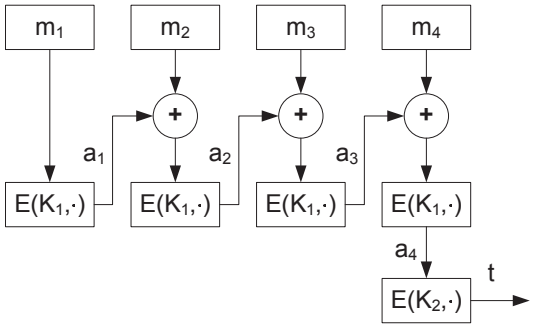

Figure 5.3: The (Encrypted) CBC-MAC Construction

5.2 Extending the Message Space of MACs

When using block ciphers such as DES or AES in the construction of Section 5.1.1, the message space consists of only 64 or 128 bits. This is of course not sufficient for most applications, so we explore several ways to increase the message space of PRFs. One of the most famous constructions for doing so is CBC-MAC, which is used in several standards. It is standardized as ANSI X9.9 and X9.19, as an ISO standard, and in FIPS 186-3, and it is also broadly used by the banking industry. Its main disadvantage is that it is fully sequential, i.e., its many block cipher invocations cannot be parallelized, e.g., by specialized hardware. This disadvantage is solved by PMAC (Parallel MAC), a recent system that also enjoys the property of being incremental, see below. A third approach is the HMAC construction, which relies on collision-resistant hash functions (CRHFs). CBC-MAC and PMAC will be introduced in this sections; the description of HMAC first requires reviewing some basic knowledge of hash functions and will be given thereafter.

Let us start with a brief observation: if a MAC $\mathsf{I}_\mathsf{E} = (\mathsf{S}, \mathsf{V})$ is a PRF over $(\mathcal{K}, \mathcal{M}, \{0,1\}^k)$, then we can shorten the tags by a certain amount without sacrificing security of the MAC. More specifically, if we delete all but the first w bits, then this "truncated PRF" is still a PRF. If additionally $(\frac{1}{2})^w$ is negligible, it also constitutes a secure MAC.

5.2.1 Encrypted CBC-MAC

The *Encrypted CBC-MAC*, which we will sometimes simply call CBC-MAC, is similar to the CBC-mode in symmetric encryption. For a given PRF E over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, one defines $\mathsf{E}^{\text{CBC-MAC}}$ over $(\mathcal{K}^2, \mathcal{X}^L, \mathcal{X})$ as follows; a graphical illustration is given in Figure 5.3.

- A CBC-MAC key consists of two PRF keys K_1, K_2.
- Given a message $m = m^{(1)}m^{(2)}\ldots m^{(s)}$ and a key (K_1, K_2), the signing algorithm computes $a_1 := \mathsf{E}(K_1, m_1)$ and $a_i := \mathsf{E}(K_1, m_i \oplus a_{i-1})$ for $i = 2, \ldots, s$.
- Finally, it computes the tag as $t := \mathsf{E}(K_2, a_s)$.
- Given a message m, a tag t, and a key (K_1, K_2), verification is done by recomputing the tag from m and by checking if the recomputed tag equals t.

This construction is secure, as stated in the following theorem.

Theorem 5.1 (Security of CBC-MAC) *Let $L > 0$, let E be a PRF over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, and let $\mathsf{E}^{\text{CBC-MAC}}$ be as constructed above. Then for each adversary A attacking $\mathsf{E}^{\text{CBC-MAC}}$ and making at*

most q queries, there exists an attacker B attacking E such that

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{E}^{\mathsf{CBC\text{-}MAC}}] < Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{q^2}{|\mathcal{X}|} \cdot L^{o(1)}.$$

(Here $L^{o(1)}$ denotes a quantity that converges to 1, and which will not matter in the following.) In particular, if $\frac{1}{|\mathcal{X}|}$ is negligible, then $\mathsf{E}^{\mathsf{CBC\text{-}MAC}}$ is a secure MAC. □

Intuitively, this means that CBC-MAC is secure as long as $q \ll \sqrt{|\mathcal{X}|}$.

Raw CBC-MAC One might wonder if the additional encryption of the output a_s is really necessary, i.e., why not let $t = a_s$? We will show that dropping the last encryption does not result in a secure MAC. We call the resulting scheme *Raw CBC-MAC*, thus letting $t := a_s$. The following instructions describe how an adversary can win a game against a MAC challenger for Raw CBC-MAC.

- Choose a message $m \in \mathcal{X}$, i.e., a message that fits into one block, and request the tag t for it, receiving $t = \mathsf{E}(K, m)$.
- Output $(m \mid\mid m \oplus t, t)$ as a candidate forgery.

It is easy to see that t is indeed a valid tag for $(m \mid\mid m \oplus t)$. Calculating the tag t' of the message $m' = m \mid\mid m \oplus t$ yields

$$t' = \mathsf{E}(K, \mathsf{E}(K, m) \oplus (m \oplus t)) = \mathsf{E}(K, t \oplus (m \oplus t)) = \mathsf{E}(K, m) = t.$$

It turns out that this works as well for messages that span more than one block; however, the resulting forgeries and their correctness are lengthier to write down. We remark however that if the message length is fixed a-priori or if the receiver prepends the message length to the message, then Raw CBC-MAC can be proven secure.

5.2.2 Padding with CBC-MAC

If the message length is not a multiple of the block length then one needs to *pad* the message, i.e., one needs to append some bits to reach a multiple of the block size. This sounds trivial (and indeed is if one is only concerned about privacy), but has some hidden caveats if integrity is the goal.

- The easiest solution seems to be to append a string of 0's of the required length, or any other fixed string of appropriate length. This, however, turns out to be a bad idea, as one cannot distinguish between trailing zeros of the message and leading zeros of the pad. In particular, the messages 0 and 00 have the same tag, as they are both padded to 0^k where k is the block-size. This allows for easy creation of existential forgeries.

- A good method for padding, which is also described in the ISO standard, is the following: Append a 1 to the message and fill the remaining space with 0's. If the message size is a multiple of the block size, then append a new block $10 \ldots 0$. The inserted 1 unambiguously determines the begin of the padding.

Note again that padding was not an issue for encryption schemes, as padding any string does not violate privacy of the string.

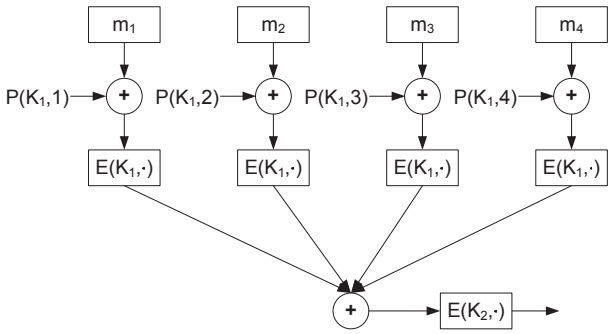

Figure 5.4: PMAC

5.2.3 PMAC

The CBC-MAC we just described is sequential, i.e., for computing the tag of a long message, all PRF invocations depend on the output of previous invocations; thus they cannot be parallelized even by specialized hardware. The PMAC (Parallel MAC) was designed to circumvent this problem. Let E a PRF over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, then we define a function $\mathsf{E}^{\mathsf{PMAC}} : \mathcal{K}^2 \times \mathcal{X}^L \to \mathcal{X}$ as follows:

- A key for PMAC consists of two PRF keys K_1, K_2.
- Furthermore it uses a special deterministic function $\mathsf{P} : \mathcal{K} \times \mathbb{N} \to \mathcal{X}$, which can be computed efficiently. It exploits properties of finite fields, which come later in the course, so that we do not describe the details here.
- Given a message $m = m_1 m_2 \dots m_r$ one computes

$$b_i := \mathsf{E}(K_1, m_i \oplus \mathsf{P}(K_1, i)) \text{ for } i = 1, \dots, r,$$

- Finally one computes $t := \mathsf{E}(K_2, \bigoplus_{i=1,\dots,r} b_i)$.

The resulting function is a secure MAC, which is formalized in the following theorem:

Theorem 5.2 (PMAC) *Let $L > 0$ and let E be a PRF over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$. For $\mathsf{E}^{\mathsf{PMAC}}$ as defined above the following holds: For each adversary A attacking $\mathsf{E}^{\mathsf{PMAC}}$ and making at most q queries there exists an attacker B attacking E such that*

$$Adv^{\mathsf{MAC}}[\mathsf{A}, \mathsf{E}^{\mathsf{PMAC}}] \leq Adv^{\mathsf{PRF}}[\mathsf{B}, \mathsf{E}] + \frac{2q^2L^2}{|\mathcal{X}|}.$$

In particular, if $\frac{1}{|\mathcal{X}|}$ is negligible, then $\mathsf{E}^{\mathsf{PMAC}}$ is a secure MAC. □

We remark that PMAC is secure as long as $q \ll \frac{\sqrt{\mathcal{X}}}{L}$.

Another nice property is that PMAC is incremental, i.e., if a small part of a message is changed or added to the message, then the creation of the tag for the new message can easily be recomputed: Say we computed $t \leftarrow \mathsf{E}^{\mathsf{PMAC}}(K, m)$ for a long message m. Now if one block changed yielding another message m^*, one can recompute the new tag t^* for m^* very efficiently (either by assuming that E is a PRP, or by storing a little bit of intermediary information from the signing algorithm). This will be explored in detail in the homework exercises.

5.3 Hash Functions

Hash functions constitute a core cryptographic primitive. They map very long messages to a fixed-size value called *message digest* or *hash value*. One of the most important properties of hash functions is that one cannot find *collisions*, i.e., two different messages that are mapped to the same hash value. Hash functions often serve as basic building blocks for more complex primitives such as MACs, signatures, and so on.

Defining collision resistance of hash functions turns out to be a quite unsatisfactory task. The intuitively expected definition "no efficient adversary can find collisions" cannot be fulfilled by any hash function: As the message space is necessarily larger than the digest space, there always exist collisions $m, m^* \in \mathcal{M}$. Now take a look at the collection of adversaries A_{m_0,m_1} for $m_0, m_1 \in \mathcal{M}$, that simply output m_0 and m_1. Clearly, the adversary A_{m,m^*} exists (but nobody might know how to find it)! For this reason the following "definition" of collision-resistant hash functions is inherently imprecise, but it turns out that subsequent reduction proofs based on such hash functions can be turned into rigorous mathematical arguments again.

"Definition" 5.5 (Collision-Resistant Hash Function (CRHF)) *A (non-keyed)* hash function *is an "efficient" function* $\mathsf{H} : \mathcal{M} \to \mathcal{T}$. *A* collision *for* H *is a tuple* (m_0, m_1) *with* $\mathsf{H}(m_0) = \mathsf{H}(m_1)$ *and* $m_0 \neq m_1$. *A hash function* H *is* collision-resistant *if no "efficient"* A *adversary is known that finds a collision.*

5.3.1 Examples

Many proposals for collision-resistant hash functions exist; however, most of them have been broken already. Even the widely deployed hash functions MD5 and SHA-1 have recently been successfully attacked: While MD5 is definitely insecure (collisions can be found in roughly 1 hour on an ordinary PCs), the attacks on SHA-1 are rather theoretical in that they are still not efficient enough to find useful collisions in a short time. However, history has shown that it usually does not take long to further decrease the complexity of such attacks, once the first theoretical glitch has been found. Examples of well-known hash functions include

- MD5 (broken): 128-bits digest, hashes roughly 216 MB per second
- SHA-1 (broken) 160-bits digest, hashes roughly 68 MB per second
- SHA-256: 256-bits digest, hashes roughly 44.5 MB per second
- Whirlpool: 512-bits digest, hashes roughly 12.1 MB per second

5.3.2 Constructing Big-MACs from Small-MACs and CRHFs

We now present a generic construction how a MAC with a small message space can be turned into a MAC with a big message space by additionally exploiting a collision-resistant hash function. Let $\mathsf{I} = (\mathsf{S}, \mathsf{V})$ be a MAC over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ and let H be a collision-resistance hash function $\mathsf{H} : \mathcal{M}^{\mathsf{big}} \to \mathcal{M}$. Then one can construct a MAC $\mathsf{I}^* = (\mathsf{S}^*, \mathsf{V}^*)$ by first hashing the message $m \in \mathcal{M}^{\mathsf{big}}$ and then applying the MAC:

- Let $\mathsf{S}^*(K, m) := \mathsf{S}(K, \mathsf{H}(m))$, and
- let $\mathsf{V}^*(K, m, t) := \mathsf{true}$ iff $\mathsf{V}(K, \mathsf{H}(m), t) = \mathsf{true}$.

One can prove the security of this construction assuming the security of the small MAC and the collision-resistance of the hash function:

Theorem 5.3 *Let* $\mathsf{H} : \mathcal{M}^{\mathsf{big}} \to \mathcal{M}$ *be a collision-resistant hash function and* $\mathsf{I} = (\mathsf{S}, \mathsf{V})$ *be a secure MAC over* $(\mathcal{K}, \mathcal{M}, \mathcal{T})$*. Then* $\mathsf{I}^* = (\mathsf{S}^*, \mathsf{V}^*)$ *as constructed above is a secure MAC over* $(\mathcal{K}, \mathcal{M}^{\mathsf{big}}, \mathcal{T})$*.* □

5.3.3 Generic Attack on CRHFs and the Birthday Paradox

The birthday paradox derives its name from the following question: if there are p people in a room, what is the probability that at least two of them have their birthday on the same day? Thus we are looking for the probability that there exist $i, j \in \{1, \ldots, n\}$, $i \neq j$, such that $t_i = t_j$, where t_i denotes the birthday of the i-th person. The somewhat surprising result is that for about 23 people the probability is already bigger than 50%. This result is stated and generalized in the following theorem.

Theorem 5.4 (Birthday Paradox) *Let* $t_1, \ldots, t_n$ *be independent randomly chosen integers in the set* $\{1, \ldots, B\}$*.*

$$\mathsf{Pr}\left[\exists i, j \in \{1, \ldots, n\}, i \neq j \colon t_i = t_j\right] \geq 1 - e^{\frac{-n(n-1)}{2B}}$$

□

In particular, for $n \geq 1.2 \cdot \sqrt{B}$ we have $\mathsf{Pr}\left[\exists i \neq j \in \{1, \ldots, n\} \colon t_i = t_j\right] \geq \frac{1}{2}$.

Proof. First we calculate the probability that all n values are different: For the first value this is 1, for the second $1 - \frac{1}{B}$, and so on. We thus obtain

$$\begin{aligned}
\mathsf{Pr}\left[\exists i, j \in \{1, \ldots, n\}, i \neq j \colon t_i = t_j\right] &= 1 - 1 \cdot \left(1 - \frac{1}{B}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{B}\right) \\
&= 1 - \prod_{i=1}^{n-1} (1 - \frac{i}{B}) \\
&\overset{(1)}{\geq} 1 - \prod_{i=1}^{n-1} e^{-\frac{i}{B}} \\
&= 1 - e^{-\sum_{i=1}^{n-1} \frac{i}{B}} \\
&= 1 - e^{-\frac{n(n-1)}{2B}}.
\end{aligned}$$

where in step (1) we exploited that $1 - x \leq e^{-x}$ for all $x \geq 0$. ∎

Note that the birthday paradox gives rise to a generic attack against all hash functions $\mathsf{H} : \mathcal{M} \to \mathcal{T}$:

- Let $p = 1.2 \cdot \sqrt{|\mathcal{T}|}$.
- Randomly choose $m_1, \ldots, m_n \in \mathcal{M}$,
- Compute hash values $t_1 := \mathsf{H}(m_1), \ldots, t_n := \mathsf{H}(m_n) \in \mathcal{T}$.
- With probability at least $\frac{1}{2}$, there exist $i \neq j \in \{1, \ldots, n\}$ such that $t_i = t_j$. Thus the adversary can output (m_i, m_j).

Consequently, if the length of hash values was 64 bits, then $|\mathcal{T}| = 2^{64}$ and the attack would require 2^{32} applications of the hash function. Typically, the length of hash values is 160 bits (SHA-1) or 256 bits (SHA-256): thus the attack would require 2^{80} or 2^{128} applications of the hash function, respectively, which is infeasible. We only mention here that there is already a specific attack on SHA-1 requiring only 2^{63} applications.

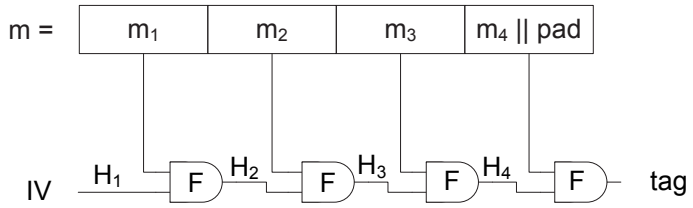

Figure 5.5: Merkle-Damgard Construction for Hash Functions

5.3.4 Merkle-Damgard Construction

Given a collision-resistant hash function F (called compression function in this subsection) that maps a fixed-length bitstring to a shorter bitstring, one can construct a collision-resistant hash function with a huge domain using the Merkle-Damgard construction.

Note that there are variations of the described construction. The one given here can be applied to bitstrings up to a length of 2^{64} bits. This bound, however, is big enough for any practical purpose, since $2^{64} \approx 10^{19}$ bits or 2305843 Terrabytes can be hashed.

Construction Let $\mathsf{F}\colon \{0,1\}^b \times \{0,1\}^n \to \{0,1\}^n$ be a collision-resistant compression (hash) function, and $IV = H_1 \in \{0,1\}^n$ be an initial value. This IV is a fixed string whose precise value does not matter here.

Given $m = m_1 \dots m_r$, where each m_i is a b-bit block, the hash-value $\mathsf{H}(m)$ is computed as follows, see the graphical illustration in Figure 5.5:

- $H_{i+1} := \mathsf{F}(m_i, H_i)$ for $i = 2, \dots, r$,
- $\mathsf{H}(m) := H_{r+1} := \mathsf{F}(m_r \mid\mid pad, H_r)$.

The padding pad is computed as follows: If $|m|$ is not a multiple of b then a block $pad = 10\dots0||msg\_len$ is appended, where msg_len is the bitstring representation of the message length (message length uses exactly the last 64 bits of the padded message), and the number of 0's is chosen so that $|m| + |pad|$ becomes a multiple of the block size b. If it already is a multiple then a new block $pad = 10\dots0||msg\_len$ with sufficiently many 0's is appended.

Lemma 5.2 *If the compression function* $\mathsf{F} : \{0,1\}^b \times \{0,1\}^n \to \{0,1\}^n$ *is collision-resistant, then the Merkle-Damgard Hash Function (MD Hash)* $\mathsf{H} : \{0,1\}^{2^{64}} \to \{0,1\}^n$ *as defined above is also collision-resistant.* □

Proof. We proof this lemma by reduction, i.e., we prove that if one finds a collision for the hash function H, then one also has a collision for the compression function F.

Let $m, m' \in \{0,1\}^{2^{64}}$ be a collision, i.e., $m \neq m'$ and $\mathsf{H}(M) = \mathsf{H}(m')$. Let us write $m = m_1 \dots m_t$ and $m' = m'_1 \dots m'_r$ as in the construction, and let us denote the intermediate values appearing in the computation of $\mathsf{H}(m)$ and $\mathsf{H}(m')$ with $H_1, \dots, H_t, H_{t+1}$ and $H_1, \dots, H'_r, H'_{r+1}$, respectively. Then we have $\mathsf{H}(m) = \mathsf{H}(m') \Rightarrow H_{t+1} = H'_{r+1} \Rightarrow \mathsf{F}(m_t \mid\mid pad, H_t) = \mathsf{F}(m'_r \mid\mid pad', H'_r)$. This means that either the arguments of the compression function are equal, or we found a collision, in which case we are done. So let us assume $(m_t \mid\mid pad, H_t) = (m'_r \mid\mid pad', H'_r)$. This in particular implies $pad = pad'$; consequently the length of m, m' are equal, thus $t = r$. Now we can proceed iteratively backwards as follows: From $H_i = H'_i$ it follows that $\mathsf{F}(m_{i-1}, H_{i-1}) = \mathsf{F}(m'_{i-1}, H'_{i-1})$, so either we found a collision for F, thus we are done, or the arguments are equal $(m_{i-1}, H_{i-1}) = (m'_{i-1}, H'_{i-1})$.

Thus, finally, we have $t = r$ and $m_i = m'_i$ for all $i = 1, \ldots, t$, thus $m = m'$, contradicting the assumption. So we had to find a collision for F in one of the above steps. ∎

5.3.5 Davies-Meyer

For using the Merkle-Damgard construction, it remains to come up with a suitable compression function. One possibility would be to use the so-called *Davies-Meyer construction*: Starting with a PRP E on $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, one defines the compression function $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ as $\mathsf{F}(M, H) := \mathsf{E}(M, H) \oplus H$. Note, however, that security of this construction can only be proved when E is an "ideal cipher", i.e., a collection of random permutations.

Theorem 5.5 (Davies-Meyer yields Collision-Resistant Compression Functions) *If E is a collection of random permutations on $(\{0,1\}^k, \{0,1\}^n, \{0,1\}^n)$, then finding collision for F takes time $2^{n/2}$.* □

An example for a hash function using this construction is SHA-256, which is an MD hash function using a Davies-Meyer compression function based on a cipher called SHACAL-2.

5.3.6 Miyaguichi-Preneel

An alternative construction is the Miyaguichi-Preneel construction, which is similar to Davies-Meyer: Let a PRP E on $(\mathcal{X}, \mathcal{X}, \mathcal{X})$ be given, and let $g : \mathcal{X} \to \mathcal{X}$ be a function mapping chaining variables from $\mathcal{X}$ to other elements of $\mathcal{X}$ (details do not matter here). Then one defines $\mathsf{F} : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ as $\mathsf{F}(M, H) := \mathsf{E}(g(H), M) \oplus H \oplus M$.

An examples for a hash function using this construction is Whirlpool, an MD hash function using a Miyaguichi-Preneel compression function based on a cipher called W (derived from AES).

5.4 HMAC

We have seen how to construct MACs with a large message space from collision-resistant hash functions and secure MACs that themselves have only a small message space. We will finally investigate how one can omit the small-MAC function and construct MACs with large message spaces directly from hash functions. The idea is to combine the message and the key in a special manner and input the result to the hash function. However, it is important to combine them in a suitable manner, as naive attempts are insecure. Let $\mathsf{H} : \mathsf{M} \to \mathsf{T}$ a Merkle-Damgard hash function. One may try the following constructions:

- $\mathsf{S}(K, M) := \mathsf{H}(K||m)$: This is completely insecure, as you will prove in the homework exercises.
- $\mathsf{S}(K, m) := \mathsf{H}(m||K)$: This can be proven secure if H is a collision-resistant hash function *and* F is a PRF. So you need two assumptions, which is not the best you can achieve.
- $\mathsf{S}((K_1, K_2), m) := \mathsf{H}(K_1||m||K_2)$: This so-called *envelope method* is secure if F is a PRF. However, it is rarely used in practice.

The method usually used in practice is HMAC. Let H be a MD-hash function with block-size n, K the key padded with 0's to the block-size n, $ipad = 3636\ldots36$, and $opad = 5c5c\ldots5c$, both in hexadecimal notation. Then signing in HMAC is defined as

- $\mathsf{HMAC} : \mathsf{S}(K, m) := \mathsf{H}(K \oplus opad \;||\; \mathsf{H}(K \oplus ipad \;||\; M))$

The verify algorithm again works by recomputing the tag.

Theorem 5.6 *If the compression function F is a PRF (when either input is used as the key), then the HMAC construction yields a secure MAC.* □