12.3: Merkle-Damgård Construction

Constructing a hash function seems like a challenging task, especially given that it must accept strings of arbitrary length as input. In this section, we'll see one approach for constructing hash functions, called the Merkle-Damgård construction.

Instead of a full-fledged hash function, imagine that we had a collision-resistant function (family) whose inputs were of a single fixed length, but longer than its outputs. In other words, suppose we had a family \( \mathcal{H} \) of functions \( h : \{0,1\}^{n+t} \rightarrow \{0,1\}^n \), where \( t > 0 \). We call such an \( h \) a compression function. This is not compression in the usual sense of data compression — we are not concerned about recovering the input from the output. We call it a compression function because it "compresses" its input by \( t \) bits (analogous to how a pseudorandom generator "stretches" its input by some amount).

We can apply the standard definition of collision-resistance to a family of compression functions, by restricting our interest to inputs of length exactly \( n + t \). The functions in the family are not defined for any other input length.

The following construction is one way to extend a compression function into a full-edged hash function accepting arbitrary-length inputs:

**Construction 12.4: Merkle-Damgård**

Let \( h : \{0,1\}^{n+t} \rightarrow \{0,1\}^n \) be a compression function. Then the Merkle-Damgård transformation of \( h \) is \( MD_h : \{0,1\}^* \rightarrow \{0,1\}^n \), where:
The idea of the Merkle-Damgård construction is to split the input $x$ into blocks of size $t$. The end of the string is filled out with 0s if necessary. A final block called the “padding block” is added, which encodes the (original) length of $x$ in binary.

We are presenting a simplified version, in which $\text{MD}_h$ accepts inputs whose maximum length is $2^t - 1$ bits (the length of the input must fit into $t$ bits). By using multiple padding blocks (when necessary) and a suitable encoding of the original string length, the construction can be made to accommodate inputs of arbitrary length (see the exercises).

The value $y_0$ is called the initialization vector (IV), and it is a hard-coded part of the algorithm. In practice, a more “random-looking” value is used as the initialization vector. Or one can think of the Merkle-Damgård construction as defining a family of hash functions, corresponding to the different choices of IV.

Claim 12.5

Let $\mathcal{H}$ be a family of compression functions, and define $\text{MD}_{\mathcal{H}} = \{\text{MD}_h | h \in \mathcal{H}\}$ (a family of hash functions). If $\mathcal{H}$ is collision-resistant, then so is $\text{MD}_{\mathcal{H}}$.

Proof

While the proof can be carried out in the style of our library-based security definitions, it’s actually much easier to simply show the following: given any collision under $\text{MD}_h$, we can efficiently find a collision under $h$. This means that any successful adversary violating the collision-resistance of $\text{MD}_{\mathcal{H}}$ can be transformed into a successful adversary violating the collision resistance of $\mathcal{H}$. So if $\mathcal{H}$ is collision-resistant, then so is $\text{MD}_{\mathcal{H}}$.

Suppose that $x, x'$ are a collision under $\text{MD}_h$. Define the values $x_1, \ldots, x_k + 1$ and $y_1, \ldots, y_k + 1$ as in the computation of $\text{MD}_h(x)$. Similarly, define $x'_1, \ldots, x'_{k'} + 1$ and $y'_1, \ldots, y'_{k'} + 1$ as in the computation of $\text{MD}_h(x')$. Note that, in general, $k$ may not equal $k'$.

Recall that:

$$\text{MD}_h(x) = y_k + 1 = h(y_k \| x_k + 1)$$

$$\text{MD}_h(x') = y'_{k'} + 1 = h(y'_{k'} \| x'_{k'} + 1)$$

Since we are assuming $\text{MD}_h(x) = \text{MD}_h(x')$, we have $y_k + 1 = y'_{k'} + 1$. We consider two cases:

Case 1: If $|x| \neq |x'|$, then the padding blocks $x_k + 1$ and $x'_{k'} + 1$ which encode $|x|$ and $|x'|$ are not equal. Hence we have $y_k \| x_k + 1 \neq y'_{k'} \| x'_{k'} + 1$, so $y_k \| x_k + 1$ and $y'_{k'} \| x'_{k'} + 1$ are a collision under $h$ and we are done.
Case 2: If $|x| = |x'|$, then $x$ and $x'$ are broken into the same number of blocks, so $k = k'$. Let us work backwards from the final step in the computations of $MD_h(x)$ and $MD_h(x')$. We know that:

$$
y_{k+1} = h(y_k || x_{k+1})$$
$$y'_{k+1} = h(y'_k || x'_{k+1})$$

If $y_k || x_k + 1$ and $y'_k || x'_k + 1$ are not equal, then they are a collision under $h$ and we are done. Otherwise, we can apply the same logic again to $y_k$ and $y'_k$, which are equal by our assumption.

More generally, if $y_i = y'_i$, then either $y_i - 1 || x_i$ and $y'_i - 1 || x'_i$ are a collision under $h$ (and we say we are “lucky”), or else $y_i - 1 = y'_i - 1$ (and we say we are “unlucky”). We start with the premise that $y_k = y'_k$. Can we ever get “unlucky” every time, and not encounter a collision when propagating this logic back through the computations of $MD_h(x)$ and $MD_h(x')$? The answer is no, because encountering the unlucky case every time would imply that $x_i = x'_i$ for all $i$. That is, $x = x'$. But this contradicts our original assumption that $x \neq x'$. Hence we must encounter some “lucky” case and therefore a collision in $h$. 


Updated: Wed, 29 Jul 2020 22:46:00 GMT
Powered by 3