Constructions of hash functions

Recall that a hash function H is a function, which takes an input x of arbitrary length, and produces an output $H(x)$ of fixed length, say, b bit.

Example 177. (Merkle–Damgård construction) Similarly, a **compression function** $\tilde{H}$ takes input x of length $b+c$ bits, and produces output $\tilde{H}(x)$ of length b bits. From such a function, we can easily create a hash function H. How?

Importantly, it can be proved that, if $\tilde{H}$ is collision-resistant, then so is the hash function H.

Solution. Let x be an arbitrary input of any length. Let's write $x = x_1x_2x_3...x_n$, where each x_i is c bits (if necessary, pad the last block of x so that it can be broken into c bit pieces).
Set $h_1 = 0$ (or any other initial value), and define $h_{i+1} = \tilde{H}(h_i, x_i)$ for $i \geqslant 1$. Then, $H(x) = h_{n+1}$.
Comment. This construction is known as a Merkle–Damgård construction and is used in the design of many hash functions, including MD5 and SHA-1/2.
Careful padding. Some care needs to be applied to the padding. Just padding with zeroes would result in easy collisions (why?), which we would like to avoid. For more details:
https://en.wikipedia.org/wiki/Merkle-Damgård_construction

Example 178. Consider the (silly) compression function

$$\tilde{H}\colon \{4\,\text{bits}\} \to \{3\,\text{bits}\}, \quad x \pmod{16} \mapsto 3x \pmod 8.$$

For instance, $\tilde{H}((1010)_2) = (110)_2$ because $(1010)_2 = 10$, $3 \cdot 10 \equiv 6 \pmod 8$ and $6 = (110)_2$.

(a) Find a collision of $\tilde{H}$.

(b) Let $H(x)$ be the hash function obtained from $\tilde{H}$ using the Merkle–Damgård construction (using initial value $h_1 = 0$). Compute $H((1100)_2)$.

Solution.

(a) Let's produce a collision with $\tilde{H}((1010)_2) = (110)_2$.
For that, we need another value $x \pmod{16}$, besides $x = 10$, such that $3x \equiv 6 \pmod 8$.
Since $3x \equiv 6 \pmod 8$ is equivalent to $x \equiv 3^{-1} \cdot 6 \equiv 3 \cdot 6 \equiv 2 \pmod 8$, another such value is $x = 2$.
In conclusion, we have the collision $\tilde{H}((0010)_2) = \tilde{H}((1010)_2) = (110)_2$.
Shortcut. We could have just noticed that changing x by 8 does not change $\tilde{H}$.

(b) Here, $x_1x_2x_3x_4 = 1100$.
$h_1 = (000)_2$
$h_2 = \tilde{H}(h_1, x_1) = \tilde{H}((0001)_2) = \tilde{H}(1) = 3 = (011)_2$
$h_3 = \tilde{H}(h_2, x_2) = \tilde{H}((0111)_2) = \tilde{H}(7) = 5 = (101)_2$
$h_4 = \tilde{H}(h_3, x_3) = \tilde{H}((1010)_2) = \tilde{H}(10) = 6 = (110)_2$
$h_5 = \tilde{H}(h_4, x_4) = \tilde{H}((1100)_2) = \tilde{H}(12) = 4 = (100)_2$
Hence, $H((1100)_2) = (100)_2$.

Armin Straub
straub@southalabama.edu

The construction of good hash algorithms is linked to the construction of good ciphers. Below, we indicate how to use a block cipher to construct a hash.

Why linked? The ciphertext produced by a good cipher should be indistinguishable from random bits. Similarly, the output of a cryptographic cipher should look random, because the presence of patterns would likely allow us to compute preimages or collisions.

However. The design goals for a hash are somewhat different than for a cipher. It is therefore usually advisable to not crossbreed these constructions and, instead, to use a specially designed hash like SHA-2 when a hash is needed for cryptographic purposes.

First, however, a cautionary example.

Example 179. (careful!) Let E_k be encryption using a block cipher (like AES). Is the compression function $\tilde{H}$ defined by

$$\tilde{H}(x,k) = E_k(x)$$

one-way?

Solution. No, it is not one-way.

Indeed, given y, we can produce many different (x,k) such that $\tilde{H}(x,k) = y$ or, equivalently, $E_k(x) = y$. Namely, pick any k, and then choose $x = D_k(y)$.

Example 180. Let E_k be encryption using a block cipher (like AES). Then the compression function $\tilde{H}$ defined by

$$\tilde{H}(x,k) = E_k(x) \oplus x$$

is usually expected to be collision-resistant.

Let us only briefly think about whether $\tilde{H}$ might have the weaker property of being one-way (as opposed to the previous example). For that, given y, we try to find (x,k) such that $\tilde{H}(x,k) = y$ or, equivalently, $E_k(x) \oplus x = y$. This seems difficult.

Just getting a feeling. We could try to find such (x,k) with $x = 0$. In that case, we need to arrange k such that $E_k(0) = y$. For a block cipher like AES, this seems difficult. In fact, we are trying a known-plaintext attack on the cipher here: assuming that $m = 0$ and $c = y$, we are trying to determine the key k. A good cipher is designed to resist such an attack, so that this approach is infeasible.

Comment. Combined with the Merkle–Damgård construction, you can therefore use AES to construct a hash function with 128 bits output size. However, as indicated before, it is advisable to use a hash function designed specifically for the purpose of hashing.

For several other (more careful) constructions of hash functions from block ciphers, you can check out Chapter 9.4.1 in the *Handbook of Applied Cryptography* (Menezes, van Oorschot and Vanstone, 2001), freely available at: http://cacr.uwaterloo.ca/hac/

We have seen how hash functions can be constructed from block ciphers (though this is usually not advisable). Similarly, hash functions can be used to build PRGs (and hence, stream ciphers).

Example 181. A hash function $H(x)$, producing b bits of output, can be used to build a PRG as follows. Let x_0 be our b bit seed. Set $x_n = H(x_{n-1})$, for $n \geqslant 1$, and $y_n = x_n \pmod 2$. Then, the output of the PRG are the bits $y_1 y_2 y_3 \ldots$

Comment. As for the B-B-S PRG, if b is large, it might be OK to extract more than one bit from each x_n.

Comment. Technically speaking, we should extract a "hardcore bit" y_n from x_n.

Comment. It might be a little bit better to replace the simple rule $x_n = H(x_{n-1})$ with $x_n = H(x_0, x_{n-1})$. Otherwise, collisions would decrease the range during each iteration. However, if b is large, this should not be a practical issue. (Also, think about how this alleviates the issue in the next example.)

Comment. Of course, one might then use this PRG as a stream cipher (though this is probably not a great idea, since the design goals for hashes and secure PRGs are not quite the same). Our book lists a similar construction in Section 8.7: starting with a seed $x_0 = k$, bytes x_n are created as follows $x_1 = H(x_0)$ and $x_n = L_8(H(x_0, x_{n-1}))$, where L_8 extracts the leftmost 8 bits. The output of the PRG is $x_1 x_2 x_3 \ldots$ However, can you see the flaw in this construction? (Hint: it repeats very soon!)

Example 182. Suppose, with the same setup as in the previous example, we let our PRG output $x_1 x_2 x_3 \ldots$, where each x_n is b bits. What is your verdict?

Solution. This PRG is not unpredictable (at all). After b bits have been output, x_1 is known and $x_2 = H(x_1)$ can be predicted perfectly. Likewise, for all the following output.

Comment. While completely unacceptable for cryptographic purposes, this might be a fine PRG for other purposes that do not need unpredictability.

Finally, here is a compression function, which is provably strongly collision-free.

However, it is rather slow and so is not practical for hashing larger data. On the other hand, its slowness could be beneficial for applications like password hashing.

Example 183. (the discrete log hash) Let p be a large safe prime (that is, $q = (p-1)/2$ is also prime). Let g_1, g_2 be two primitive roots modulo p. Define the compression function h as:

$$h\colon \{0, q^2 - 1\} \to \{1, \ldots, p-1\}, \quad h(m_1 + m_2 q) = g_1^{m_1} g_2^{m_2} \pmod p.$$

[Note that, although not working with inputs and outputs of certain size in bits, this is a compression function, because the input space is much larger than the output space.]

Show that finding a collision of $h(x)$ is as difficult as determining the discrete logarithm x in $g_1^x = g_2 \pmod p$.

Solution. Suppose we have a collision: $g_1^{m_1} g_2^{m_2} \equiv g_1^{m_1'} g_2^{m_2'} \pmod p$

Hence, $g_1^{(m_1 - m_1') + (m_2 - m_2')x} \equiv 1 \pmod p$ or, equivalently, $(m_1 - m_1') + (m_2 - m_2')x \equiv 0 \pmod{p-1}$ (because g_1 is a primitive root and so has order $p-1$).

This final congruence can now be solved for x.

More precisely, if $d = \gcd(m_2 - m_2', p-1)$, there are actually d solutions for x. Since we chose p to be safe, the only factors of $p-1$ are $1, 2, (p-1)/2, p-1$.

Since $|m_2 - m_2'| < q$, the only possibilities are $d = 1, 2$ (unless $m_2 = m_2'$; however, this cannot be the case since then also $m_1 = m_1'$, so that we wouldn't have a collision in the first place).