Sketch of Lecture 33

Fri, 4/12/2019

Some popular hash functions:

	published	output bits	comment
CRC32	1975	32	not secure but common for checksums
MD5	1992	128	common; used to be secure (now broken)
SHA-1	1995	160	common; used to be secure (collision found in 2017)
SHA-2	2001	256/512	considered secure
SHA-3	2015	arbitrary	considered secure

- CRC is short for **Cyclic Redundancy Check**. It was designed for protection against common transmission errors, not as a cryptographic hash (for instance, CRC is a linear function).
- SHA is short for **Secure Hash Algorithm** and (like DES and AES) is a federal standard selected by NIST. SHA-2 is a family of 6 functions, including SHA-256 and SHA-512 as well as truncations of these.

  SHA-3 is not meant to replace SHA-2 but to provide a different alternative (especially following successful attacks on MD5, SHA-1 and other hash functions, NIST initiated an open competition for SHA-3 in 2007). SHA-3 is based on Keccak (like AES is based on Rijndael; Joan Daemen involved in both). Although the ouput of SHA-3 can be of arbitrary length, the number of security bits is as for SHA-2.

  https://en.wikipedia.org/wiki/NIST_hash_function_competition
- MD is short for **Message Digest**. These hash functions are due to Ron Rivest (MIT), the "R" in RSA. Collision attacks on MD5 can now produce collisions within seconds. For a practical exploit, see: https://en.wikipedia.org/wiki/Flame_(malware)

  MD6 was submitted as a candidate for SHA-3, but later withdrawn.

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 189. (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 $x = x_1 x_2 x_3 \ldots x_n$, 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, $H(x) = h_{n+1}$ (b bits).

[In $\tilde{H}(h_i, x_i)$, we mean that the b bits for h_i are concatenated with the c bits for x_i, for a total of $b + c$ bits.]

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 190. Consider the compression function $\tilde{H}\colon \{3\,\text{bits}\} \to \{2\,\text{bits}\}$ defined by

x	000	001	010	011	100	101	110	111
$\tilde{H}(x)$	00	10	11	01	10	00	01	11

[This was not chosen randomly: the first output bit is the sum of the digits, and the second output bit is just the second input bit.]

(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 $H(1101)$.

(c) Find a collision with $H(1101)$.

Solution.

(a) For instance, $\tilde{H}(001) = \tilde{H}(100)$.

(b) Here, $b = 2$ and $c = 1$, so that each x_i is 1 bit: $x_1x_2x_3x_4 = 1101$.
$h_1 = 00$
$h_2 = \tilde{H}(h_1, x_1) = \tilde{H}(001) = 10$
$h_3 = \tilde{H}(h_2, x_2) = \tilde{H}(101) = 00$
$h_4 = \tilde{H}(h_3, x_3) = \tilde{H}(000) = 00$
$h_5 = \tilde{H}(h_4, x_4) = \tilde{H}(001) = 10$
Hence, $H(1101) = h_5 = 10$.

(c) Our computation above shows that, for instance, $H(1) = 10 = H(1101)$.

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 191. (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 192. 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 193. 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 194. 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.