FIXED LENGTH HASH FUNCTION CONCEPTS

Overview

Hash functions are simply functions that take inputs of some length and compress them into short, fixed-length outputs.

At the most basic level, a hash function H provides a way to deterministically map a long input string to a shorter output string sometimes called a digest.

A collision occurs when two elements end up being stored in the same cell, increasing the lookup time

DEFINITION 6.2 *A hash function* $\mathcal{H} = (\mathsf{Gen}, H)$ *is* collision resistant *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\Pr\left[\mathsf{Hash\text{-}coll}_{\mathcal{A},\mathcal{H}}(n) = 1\right] \leq \mathsf{negl}(n).$$

A hash function is collision-resistant if it is infeasible for any probabilistic polynomial-time algorithm to find a collision in the given function.

Formally this is defined below:

DEFINITION 6.1 *A* hash function *(with output length $\ell(n)$) is a pair of probabilistic polynomial-time algorithms* (Gen, H) *satisfying the following:*

- Gen *is a probabilistic algorithm that takes as input a security parameter 1^n and outputs a key s. We assume that n is implicit in s.*
- *H is a deterministic algorithm that takes as input a key s and a string $x \in \{0,1\}^*$ and outputs a string $H^s(x) \in \{0,1\}^{\ell(n)}$ (where n is the value of the security parameter implicit in s).*

If H^s is defined only for inputs x of length $\ell'(n) > \ell(n)$, then we say that (Gen, H) *is a* fixed-length hash function for inputs of length $\ell'(n)$. *In this case, we also call H a* compression function.

In the fixed-length case we require that l' be greater than l. This ensures that H s compresses its input. In the general case, the function takes as input strings of arbitrary length; thus, it also compresses (albeit only inputs of length greater than l(n)). Note that without compression, collision resistance
is trivial (since one can just take the identity function $H^s(x) = x$).

Code construction:

Let $\mathcal{G}$ be a polynomial-time algorithm that on input 1^n outputs a cyclic group $\mathbb{G}$ of prime order q (with $n = \|q\|$) and generator g. Define a fixed-length hash function (Gen, H) as follows:

- Gen: On input 1^n, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$ and then select $h \leftarrow \mathbb{G}$. Output $s := \langle \mathbb{G}, q, g, h \rangle$.
- H: given a key $s = \langle \mathbb{G}, q, g, h \rangle$ and input $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$, output $H^s(x_1, x_2) := g^{x_1} h^{x_2}$.

PROOF:

Theorem 8.79. If the discrete logarithm problem is hard relative to $\mathcal{G}$, the Constuction 8.78 is a fixed-length collision-resistant hash function.

Proof. Let $\Pi = (\mathsf{Gen}, H)$ as in Construction 8.78, and let $\mathcal{A}$ be a PPT algorithm with

$$\epsilon \stackrel{\text{def}}{=} \Pr[\mathsf{Hash\text{-}coll}_{\mathcal{A},\Pi}(n) = 1]$$

We show how $\mathcal{A}$ can be used by an algorithm $\mathcal{A}'$ to solve the discrete logarithm problem with success probability ϵ.

Recall the discrete logarithm problem:

The discrete logarithm experiment $\mathsf{Dlog}_{\mathcal{A},\mathcal{G}}(n)$:

1. Run $\mathcal{G}(1^n)$ to obtain $((G), q, g)$, where $\mathbb{G}$ is a cyclic group of order q (with $\|q\| = n$), and g is a generator of $\mathbb{G}$.
2. Choose $h \leftarrow \mathbb{G}$. (This can be done by choosing $x' \leftarrow \mathbb{Z}_q$ and set $h := g^{x'}$).
3. $\mathcal{A}$ is given $\mathbb{G}, q, g, h$, and outputs $x \in \mathbb{Z}_q$.
4. The output of the experiment is defined to be 1, if $g^x = h$, and 0 otherwise.

Definition 7.59 We say that *the discrete logarithm problem is hard relative to $\mathcal{G}$* if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Dlog}_{\mathcal{A},\mathcal{G}}(n) = 1] \leq \mathsf{negl}(n).$$

Back to the proof of 8.79

Algorithm $\mathcal{A}'$:

The algorithm is given $\mathbb{G}, q, g, h$ as input.

1. Let $s := \angle \mathbb{G}, q, g, h\rangle$. Run $\mathcal{A}(s)$ and obtain output x and x'.
2. If $x \neq x'$ and $H^s(x) = H^s(x')$ then:
   2.1 If $h = 1$ return 0.
   2.2 Otherwise, parse x as (x_1, x_2) and parse x' as (x_1', x_2'). Return $(x_1 - x_1'), \cdot (x_2 - x_2')^{-1} \mod q]$.

Clearly, $\mathcal{A}'$ runs in polynomial time. Furthermore, the input s given to $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ is distributed exactly as in experiment Hash-coll$_{\mathcal{A},\Pi}$. So with probability precisely $\epsilon(n)$ there is a *collision*.

We claim that whenever there is a collision, $\mathcal{A}'$ returns the correct answer $\log_g h$.

$\mathit{Collision} \Rightarrow \log_g h$

If $h = 1$, then $log_g h = 0$ which is previously what $\mathcal{A}'$ returns.

Otherwise, the collision implies

$$\begin{aligned} H^s(x_1, x_2) = H^s(x_1', x_2') &\Rightarrow g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'} \\ &\Rightarrow g^{x_1 - x_1'} = h^{x_2' - x_2}. \end{aligned}$$

If $x_2' - x_2 = 0 \mod q$, then $g^{x_1 - x_1'} = h^{x_2' - x_2} = h^0 = 1$ and $x_1 - x_1' = 0 \mod q$. But then $x = (x_1, x_2) = (x_1', x_2') = x'$ in contradiction. Thus, $x_2' - x_2 \neq 0 \mod q$ and has an inverse.

$$g^{(x_1 - x_1')\cdot[(x_2' - x_2)^{-1} \mod q]} = \left(h^{(x_2' - x_2)}\right)^{[(x_2' - x_2)^{-1} \mod q]} = h^1 = h,$$

and so

$$\log_g h = [(x_1 - x_1'), \cdot (x_2 - x_2')^{-1} \mod q]$$