Cryptographic hash functions I

Practical aspects and generic hardness

Tanja Lange

Eindhoven University of Technology

2MMC10 – Cryptology

Motivation

Want a short handle to some larger piece of data such that:

- even a small change in the large data leads to a very different handle; handle can serve as fingerprint
- it (probably uniquely) identifies the larger piece of data; (think of PGP fingerprints)
- one cannot compute the fingerprint without knowing all the data; fingerprint forms a commitment to the data.
- the fingerprints are (close to) uniformly distributed; (can use them – or parts thereof – to assign data to buckets or next steps to random walks.)
- one cannot reconstruct the data from the fingerprint. (at least sometimes that's desired.)

Cryptographic hash functions - practical definition

A cryptographic hash function H maps bit strings of arbitrary length to bit strings of length n.

$$H : \{0,1\}^* \to \{0,1\}^n$$

The input space might be further restricted.

A secure hash function satisfies the following 3 properties:

Preimage resistance: Given $y \in H(\{0,1\}^*)$ finding $x \in \{0,1\}^*$ with $H(x) = y$ is hard.

Second preimage resistance: Given $x \in \{0,1\}^*$ finding $x' \in \{0,1\}^*$ with $x \neq x'$ and $H(x') = H(x)$ is hard.

Collision resistance: Finding $x, x' \in \{0,1\}^*$ with $x \neq x'$ and $H(x') = H(x)$ is hard.

Cryptographic hash functions - practical definition

A cryptographic hash function H maps bit strings of arbitrary length to bit strings of length n.

$$H : \{0,1\}^* \to \{0,1\}^n$$

The input space might be further restricted.

A secure hash function satisfies the following 3 properties:

Preimage resistance: Given $y \in H(\{0,1\}^*)$ finding $x \in \{0,1\}^*$ with $H(x) = y$ is hard.

y is fixed and known to be the image of some $x \in \{0,1\}^*$. Typically there are many such x, but it should be computationally hard to find any.

Second preimage resistance: Given $x \in \{0,1\}^*$ finding $x' \in \{0,1\}^*$ with $x \neq x'$ and $H(x') = H(x)$ is hard.

Collision resistance: Finding $x, x' \in \{0,1\}^*$ with $x \neq x'$ and $H(x') = H(x)$ is hard.

Cryptographic hash functions - practical definition

A cryptographic hash function H maps bit strings of arbitrary length to bit strings of length n.

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

The input space might be further restricted.

A secure hash function satisfies the following 3 properties:

Preimage resistance: Given $y \in H(\{0,1\}^*)$ finding $x \in \{0,1\}^*$ with $H(x) = y$ is hard.

y is fixed and known to be the image of some $x \in \{0,1\}^*$. Typically there are many such x, but it should be computationally hard to find any.

Second preimage resistance: Given $x \in \{0,1\}^*$ finding $x' \in \{0,1\}^*$ with $x \neq x'$ and $H(x') = H(x)$ is hard.

$x \in \{0,1\}^*$ fixes $H(x) = y$. Typically there are many other $x' \neq x$ with the same image, but it should be computationally hard to find any.

Collision resistance: Finding $x, x' \in \{0,1\}^*$ with $x \neq x'$ and $H(x') = H(x)$ is hard.

Cryptographic hash functions - practical definition

A cryptographic hash function H maps bit strings of arbitrary length to bit strings of length n.

$$H : \{0,1\}^* \to \{0,1\}^n$$

The input space might be further restricted.

A secure hash function satisfies the following 3 properties:

Preimage resistance: Given $y \in H(\{0,1\}^*)$ finding $x \in \{0,1\}^*$ with $H(x) = y$ is hard.

y is fixed and known to be the image of some $x \in \{0,1\}^*$. Typically there are many such x, but it should be computationally hard to find any.

Second preimage resistance: Given $x \in \{0,1\}^*$ finding $x' \in \{0,1\}^*$ with $x \neq x'$ and $H(x') = H(x)$ is hard.

$x \in \{0,1\}^*$ fixes $H(x) = y$. Typically there are many other $x' \neq x$ with the same image, but it should be computationally hard to find any.

Collision resistance: Finding $x, x' \in \{0,1\}^*$ with $x \neq x'$ and $H(x') = H(x)$ is hard.

This property leaves full flexibility to choose any target y. Nevertheless it should be computationally hard to find any $x \neq x'$ with the same image.

Generic hardness

If the output of H is distributed uniformly then each y has a $1/2^n$ chance of being the image.

Hence it takes about 2^n calls to H to find a preimage.

Generic hardness

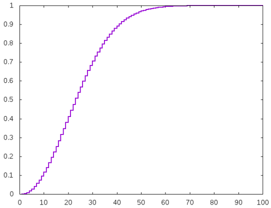
If the output of H is distributed uniformly then
each y has a $1/2^n$ chance of being the image.

Hence it takes about 2^n calls to H
to find a preimage.

The same approach works to find second preimages.
The probability that same x is found is negligible.

Hence it takes about 2^n calls to H to find a second preimage.

Generic hardness



If the output of H is distributed uniformly then each y has a $1/2^n$ chance of being the image.

Hence it takes about 2^n calls to H to find a preimage.

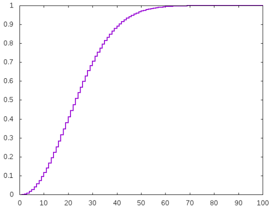
The same approach works to find second preimages.
The probability that same x is found is negligible.

Hence it takes about 2^n calls to H to find a second preimage.

The birthday paradox implies that if one draws elements at random from a set of m elements, then with 50% probability one has picked one element twice after about $\sqrt{\pi m/2}$ picks.
Hence it takes $O(2^{n/2})$ calls to H to find a collision.

This number is much lower than the other two because there is no restriction on the target.

Generic hardness



If the output of H is distributed uniformly then each y has a $1/2^n$ chance of being the image.

Hence it takes about 2^n calls to H to find a preimage.

The same approach works to find second preimages.
The probability that same x is found is negligible.

Hence it takes about 2^n calls to H to find a second preimage.

The birthday paradox implies that if one draws elements at random from a set of m elements, then with 50% probability one has picked one element twice after about $\sqrt{\pi m/2}$ picks.
Hence it takes $O(2^{n/2})$ calls to H to find a collision.

This number is much lower than the other two because there is no restriction on the target.

These are the *highest possible* complexities one can hope for.
Some hash functions require far fewer operation to break.

Practical use hash functions

Hash functions are often called the Swiss-army knife of cryptography. They are used in

- key-derivation functions
- public-key signatures
- symmetric-key authentication

Cryptographic libraries support several hash functions:

- In use and probably OK: SHA-256, SHA-384, SHA-512; SHA-3, SHAKE, other SHA-3 finalists.
- SHA-1 is still in use for fingerprints, e.g. for git and PGP. Collisions were computed in 2017 `https://shattered.io/`. Practical attack (chosen prefix collision) in 2020 `https://sha-mbles.github.io/`
- MD5: collisions (2004) and chosen-prefix collisions (2008). Flame malware (2012) used MD5 collision to create signature on fake Windows update.
- MD4: collisions (1995), very efficient collisions (2004).