Hash functions

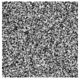
Tanja Lange

Eindhoven University of Technology

2WF80: Introduction to Cryptology

Motivation

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

- it (probably) uniquely identifies the larger piece of data;
  (think of PGP fingerprints)
- even a small change in the large data leads to a different handle;
  (think of  vs.  as some bits flip in the data)
- 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.)
- one cannot reconstruct the data from the fingerprint.
  (at least sometimes that's desired.)

Cryptographic hash functions

A cryptographic hash function H maps

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

bit strings of arbitrary length to bit strings of length n.

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

A cryptographic hash function H maps

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

bit strings of arbitrary length to bit strings of length n.

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

A cryptographic hash function H maps

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

bit strings of arbitrary length to bit strings of length n.

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

A cryptographic hash function H maps

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

bit strings of arbitrary length to bit strings of length n.

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.

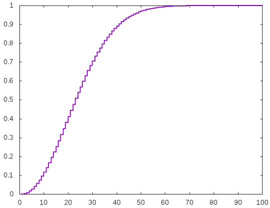
The same approach works to find second preimages.
The probability that same x is found is negligible.

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.

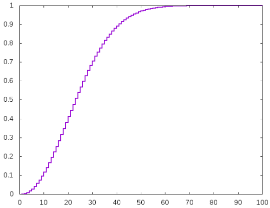
The birthday paradox implies that if one draws elements at random from a set of m elements, then with 50% probability one has picked one element twice after about $\sqrt{m}$ picks.
Hence it takes $O(2^{n/2})$ calls to H to find a collision.

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 $\sqrt{m}$ 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.

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

Merkle-Damgård construction

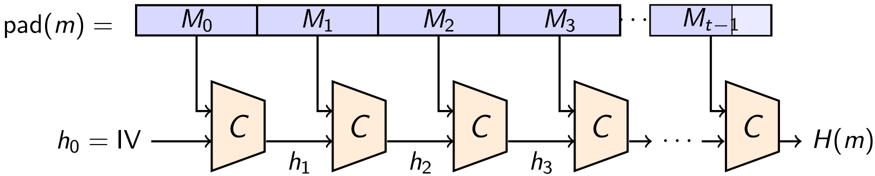
While the definition says $H : \{0,1\}^* \to \{0,1\}^n$
most constructions take data in blocks of a fixed number of bits.

Let $\text{pad}(m) = M_0\, M_1\, M_1\, \ldots\, M_{t-1}$ be the message padded up to a multiple of the block length n so that $m = m_0\, m_1\, m_2 \ldots m_{\ell-1}$ turns into
$M_0 = m_0\, m_1\, m_2 \ldots m_{n-1}, M_1 = m_n\, m_{n+1}\, m_{n+2} \ldots m_{2n-1}, \ldots$
$M_{t-1} = m_{(t-1)n}\, m_{(t-1)n+1}\, m_{(t-1)n+2} \ldots m_{\ell-1}\, p_0\, p_1\, \ldots p_{j-1}$, where
$t = \lceil \ell/n \rceil$, $p_0, p_1, \ldots, p_{j-1}$ are padding bits and $j = tn - \ell$

Merkle-Damgård construction

While the definition says $H : \{0,1\}^* \to \{0,1\}^n$
most constructions take data in blocks of a fixed number of bits.

Let $\text{pad}(m) = M_0\, M_1\, M_1\, \ldots\, M_{t-1}$ be the message padded up to a multiple of the block length n so that $m = m_0\, m_1\, m_2 \ldots m_{\ell-1}$ turns into
$M_0 = m_0\, m_1\, m_2 \ldots m_{n-1}, M_1 = m_n\, m_{n+1}\, m_{n+2} \ldots m_{2n-1}, \ldots$
$M_{t-1} = m_{(t-1)n}\, m_{(t-1)n+1}\, m_{(t-1)n+2} \ldots m_{\ell-1}\, p_0\, p_1\, \ldots p_{j-1}$, where
$t = \lceil \ell/n \rceil$, $p_0, p_1, \ldots, p_{j-1}$ are padding bits and $j = tn - \ell$



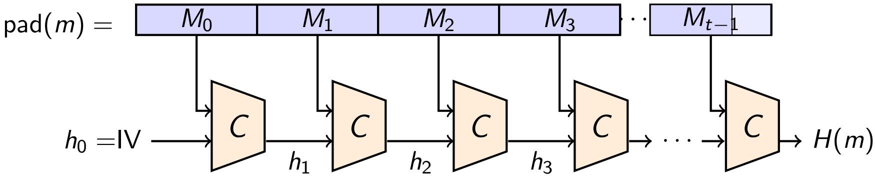
C in the Merkle-Damgård construction is a compression function

$$C : \{0,1\}^{2n} \to \{0,1\}^n.$$

Each step takes the n-bit h_{i-1} (previous output or $h_0 = \text{IV}$) and n message bits and compresses these to $h_i = C(M_{i-1}, h_{i-1})$ of n bits.

Image credit: adapted from Jérémy Jean.

Properties of Merkle-Damgård construction



The iterative design makes analysis easier.

- If $C : \{0,1\}^{2n} \rightarrow \{0,1\}^n$ is collision resistant then so is H.
- H is only collision resistant if C is.

The construction means that hashes can be computed incrementally, e.g., one can stream data one block at a time into a small hashing device.

We used this as a feature in finding partial SHA-1 collisions, see our write up for details.

Image credit: adapted from Jérémy Jean.

Summary of hash functions

Hash functions are used in

- public-key signatures
  (see video Public-key and symmetric-key cryptology);
- symmetric-key authentication
  (see video Message authentication codes (MACs)).

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: efficient collisions (1995), very efficient collisions (2004).