Overview

Today:
- Quiz (based on HW 6)
- Graded HW 2 due
- Grad/honors students: Project topic selection due
- Discuss cryptographic hash functions (today and next Tuesday)

Next:
- Complete homework 7 (due Tuesday, March 27)
- Read Sections 12.1-12.6 before next Thursday

Hash Function Basics and Terminology

General Definition: A hash function maps a large domain into a small, fixed-size range. Domain often generalized to all binary strings.

\[ H: \{0,1\}^* \rightarrow R \]

Fixed size range

Use in data structures: \( R \) is set of hash table indices.

Important properties:
- Efficient to compute
- Uniform distribution ("apparently random")

If \( H(x) = h \), then we say "\( x \) is a preimage of \( h \)"

If \( x \neq y \), but \( H(x) = H(y) \), then the pair \( (x,y) \) is a collision

Question: Do all hash functions have collisions?
Cryptographic Hash Functions

Cryptographic hash functions map to fixed-length bit-vectors, sometimes called message digests.

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

For cryptographic applications, need one or more of these properties:

- **Preimage resistance**: Given \( h \), it's infeasible to find \( x \) such that \( H(x) = h \)
  - Also called the "one-way property"

- **Second preimage resistance**: Given \( x \), it's infeasible to find \( y \neq x \) such that \( H(x) = H(y) \)
  - Also called "weak collision resistance"

- **Collision resistance**: It's infeasible to find any two \( x \) and \( y \) such that \( x \neq y \) and \( H(x) = H(y) \)
  - Also called "strong collision resistance"

The SHA Family of Algorithms

SHA is the "Standard Hash Algorithm"

Table 11.3 from the textbook:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Message Size</th>
<th>Block Size</th>
<th>Word Size</th>
<th>Digest Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>(&lt; 2^{64})</td>
<td>512</td>
<td>32</td>
<td>160</td>
</tr>
<tr>
<td>SHA-224</td>
<td>(&lt; 2^{64})</td>
<td>512</td>
<td>32</td>
<td>224</td>
</tr>
<tr>
<td>SHA-256</td>
<td>(&lt; 2^{64})</td>
<td>512</td>
<td>32</td>
<td>256</td>
</tr>
<tr>
<td>SHA-384</td>
<td>(&lt; 2^{128})</td>
<td>1024</td>
<td>64</td>
<td>384</td>
</tr>
<tr>
<td>SHA-512</td>
<td>(&lt; 2^{128})</td>
<td>1024</td>
<td>64</td>
<td>512</td>
</tr>
<tr>
<td>SHA-512/224</td>
<td>(&lt; 2^{128})</td>
<td>1024</td>
<td>64</td>
<td>224</td>
</tr>
<tr>
<td>SHA-512/256</td>
<td>(&lt; 2^{128})</td>
<td>1024</td>
<td>64</td>
<td>256</td>
</tr>
</tbody>
</table>

Note: MD5 is an older algorithm with a 128-bit digest - don't use MD5 or SHA-1.

Thinking about Collisions

If hashing \( b \)-bit inputs to \( n \)-bit digests, how many preimages per digest?

- Worst case ("at least \( c \) preimages for some digest...")?
- On average?
Thinking about Collisions

If hashing \( b \)-bit inputs to \( n \)-bit digests, how many preimages per digest?
- **Worst case ("at least \( c \) preimages for some digest...")?**
- **On average?**

For worst case:

If there are \( m \) items to be put into \( n \) bins, then one bin must contain at least \( \lceil \frac{m}{n} \rceil \) items (generalization of the pigeonhole principle).

\( 2^b \) preimages "placed in" \( 2^n \) preimage bins
\( \Rightarrow \) One digest must have at least \( \lceil 2^b/2^n \rceil = 2^b - n \) preimages

Thinking about Collisions

If hashing \( b \)-bit inputs to \( n \)-bit digests, how many preimages per digest?
- **Worst case ("at least \( c \) preimages for some digest...")?**
- **On average?**

For average case:

Let \( p_h \) be the number of preimages for hash value (digest) \( h \).
Since each of the \( 2^b \) preimages is the preimage to exactly one digest,
\[ \sum p_h = 2^b. \]

The average number of preimages for any digest is therefore
\[ \frac{\sum p_h}{2^n} = \frac{2^b}{2^n} = 2^{b-n}. \]

Thinking about Collisions

Some real numbers

Using SHA-1 to hash 256-bit (32-byte) inputs:
\( \Rightarrow \) A digest has on average \( 2^{256-160} = 2^{96} \) different preimages

Bottom line: Lots and lots and lots and lots of collisions!

Looking for \( 2^{96} \) needles in a size \( 2^{256} \) haystack still is hard...

MD5 was introduced in 1992
- Not a single collision found until 2004
- Now finding collisions in MD5 is fairly routine

SHA-1 was introduced in 1995
- Not a single collision found until... Feb 23, 2017
- Recommendations to not use since 2010
- Don't use any more!
Brute Force Attacks
On Preimage and Second Preimage Resistance

Brute force attack to find a preimage:

\[
\text{find-preimage}(h) \quad // \quad h \text{ is } n \text{ bits}
\]

\[
\text{repeat}
\] 

\[
x \leftarrow \text{random input}
\]

\[
\text{until } H(x) = h
\]

If \( H \) is uniformly distributed: prob \( 1/2^n \) of finding preimage each time

This is a Bernoulli trial with success probability \( 1/2^n \)

➔ Repeat until success gives a geometric distribution

➔ Expected number of trials is \( 2^n \)

Question: What about a brute force attack to find a second preimage?

Answer: Same analysis... expected number of test hashes is \( 2^n \)

Brute Force Attacks
On Collision Resistance

Free to match up any two preimages for a collision, so:

\[
S \leftarrow \{
\]

while true:

\[
x \leftarrow \text{random input}
\]

if a pair \((y,H(x))\) is in \( S \) with \( y \neq x \) then

\[
\text{return } (x,y)
\]

Add \((x,H(x))\) to \( S \)

Looking for any duplicate pair is the “Birthday Problem”

➔ Picking randomly from \( m \) items

➔ Expect a duplicate after \( \sqrt{m} \) selections

➔ For \( n \)-bit hash function, collision after \( 2^{n/2} \) random tests

Question: Given what you know about feasible/borderline/safe times for attacks, what digest size do you need to be safe against brute force against each property?
Attacks via Cryptanalysis

тин: Use structure of hash function - don’t just guess randomly!

Success of a cryptanalytic attack is measured by how much faster it is than brute force.

Good summary on Wikipedia “Hash function security summary” page:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preimage Resistance</th>
<th>Collision Resistance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>2^{123.4}</td>
<td>2^{128}</td>
</tr>
<tr>
<td>SHA-1</td>
<td>No attack</td>
<td>2^{18}</td>
</tr>
<tr>
<td>SHA-256</td>
<td>No attack</td>
<td>2^{64}</td>
</tr>
</tbody>
</table>

*No attack* means no attack is known that substantially improves upon brute force for the full-round version of the hash function.

Application 1: Password Storage

**Problem:** Need to store passwords in a database for checking logins

**Goal:** Passwords are checkable, but can’t be stolen if DB compromised

**Idea:** Don’t store password - store \( H(\text{password}) \)

What property of cryptographic hash functions must be satisfied?

- Preimage resistance? **Yes**
- Second preimage resistance? **No**
- Collision resistance? **No**
Application 1: Password Storage

Additional issues with password storage:

Issue 1: Would be easy to make a dictionary of hashes of popular passwords.
Solution: Add “salt” - random values prepended to password before hashing
- Like an IV - must be stored with hash
- If set of salts is $10^{15}$ or larger, destroys possibility of dictionaries - see why?

Issue 2: Given salt and hash, can brute force password (hash fins are fast!)
Solution: Purposely slow down hash function by iterating
- Compute $H(H(H(H(...H(salt+password)...))))$
- Using SHA256, can hash around 10,000,000 passwords/second
- Iterate 1,000,000 times to slow down to 0.1 seconds per test

Question 1: How long to test 1,000,000 most common passwords with SHA256?
Question 2: What about with iterated SHA256?

Application 2: Detecting File Tampering

Problem: Detect if a file has been modified without a copy of original
Goal: Can check if file is the original from a “fingerprint”
Idea: Store $H(file)$ as fingerprint - for any file, $SHA256(file)$ just 32 bytes

What property of cryptographic hash functions must be satisfied?
- Preimage resistance? **No**
- Second preimage resistance? **Yes**
- Collision resistance? **No**

Practical note: Can’t store hashes with files without additional protections!
Application 3: Verifying a message

Problem: I give you a contract, you verify what you agreed to with fingerprint of contract.

Example: Bank calls and asks “Did you agree to fingerprint xybqasd?”

Goal: I can’t trick you into verifying a different contract than you saw

What property of cryptographic hash functions must be satisfied?

Preimage resistance?

Second preimage resistance? Yes

Collision resistance? Yes

Practical note: Seems esoteric, but this is precisely what happened when an MD5-based certification authority was compromised in 2008

Relation Between Different Properties

Some basic questions

- Does a function with collision resistance have second preimage resistance?
- Does a function with second preimage resistance have preimage resistance?
- Can you construct a function with preimage resistance but not collision resistance?

These questions will be explored in your next homework!
A sampling of other applications

Hash functions have been used for:

- Fast, secure pseudorandom number generation
- Disk deduplication
  - Similar: content-addressable storage as in Dropbox
- Forensic analysis (hashes of known files)
- Commitment protocols (commit to a value and reveal later)

A new(-ish) application with a different property - proof of work

- Partial preimage: A preimage in which only part of the digest bits match
  - Example: Find SHA1 preimage in which first 40 bits of hash are 0
  - Should not be able to do this faster than $2^{40}$ tests on average
  - Smaller match requirement makes problem tractable - still hard though!

- Problem: Find $x$ such that $H(x \parallel \text{message})$ starts with $b$ 0-bits
  - Invest time in finding $x$ - changing message requires similar time
  - Link to future messages - changing a past message now very expensive
  - This is the key concept behind Bitcoin mining and blockchain integrity

Classical hash function construction

Merkle-Damgard construction

*Used in MD5, SHA1, SHA256, SHA512, ...*

![Diagram of Merkle-Damgard construction](image)

<table>
<thead>
<tr>
<th>Function</th>
<th>(b)</th>
<th>(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1</td>
<td>512</td>
<td>160</td>
</tr>
<tr>
<td>SHA256</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>SHA512</td>
<td>1024</td>
<td>512</td>
</tr>
</tbody>
</table>

Classical hash function construction

Repeating compression function for long inputs

![Diagram of repeating compression function](image)

Notice that internal state is completely given in output if you stop early - this causes a problem with some later constructions, such as creating message authentication codes (MACs).
SHA-3

SHA-3 was selection process similar to that used for AES

- Competition announced/started in 2006
- Context: Attacks had been made on MD4, SHA-0, and MD5, as well as on general structure - try to avoid “all designs alike”
  - From the competition announcement: “NIST also desires that the SHA-3 hash functions will be designed so that a possibly successful attack on the SHA-2 hash functions is unlikely to be applicable to SHA-3.”
- Selection after rounds of proposal/evaluate/narrow rounds
  - 51 submissions!
  - 14 hash functions selected for round 2 in 2009
  - 5 finalists selected in 2010
  - Winner was named Keccak - announced in 2012
    - Designed by Guido Bertoni, Joan Daemen, and Michal Peeters, and Gilles Van Assche

SHA-3

Based on a “sponge function” (not Merkle-Damgard):

Input is “absorbed” into the sponge - output is “squeezed out”

Input:

\[
\begin{array}{c}
\vdots
\end{array}
\]

Output:

\[
\begin{array}{c}
\vdots
\end{array}
\]

Notice: state include “unused capacity” bits (c) - can’t recover internal state to continue from output.