# CPSC 467: Cryptography and Security

Michael J. Fischer

Lecture 15
October 20, 2020

## Cryptographic Hash Functions

- Properties of random functions
- Message digest functions

## Properties of Hash Functions

- Hash functions do not always look random
- Relations among hash function properties

## Constructing New Hash Functions from Old

- Extending a hash function
- A general chaining method

# Cryptographic Hash Functions

# Cryptographic use of random functions

Let $\mathcal{M}$ be a message space and $\mathcal{H}$ a hash value space, and assume $|\mathcal{M}| \gg |\mathcal{H}|$. A random function $h$ chosen uniformly from $\mathcal{M} \to \mathcal{H}$ gives a way to protect the integrity of messages.

Suppose Bob knows $h(m)$ for Alice's message $m$, and Bob receives $m'$ from Alice. If $h(m') = h(m)$, then with very high probability, $m' = m$, and Bob can be assured of the integrity of $m'$.

One problem with this approach is that we have no succinct way of describing random functions, so there is no way for Bob to compute $h(m')$. The other problem is that $h$ should be chosen anew for each message. Otherwise, there is a small chance being stuck with a bad $h$ (for example a constant function) forever and ever.

# Message digest functions

A *message digest* (also called a *cryptographic hash* or *fingerprint*) function is a fixed (non-random) function that is designed to "look like" a random function.

The goal is to preserve the integrity-checking property of random functions: If Bob knows $h(m)$ and he receives $m'$, then if $h(m') = h(m)$, he can reasonably assume that $m' = m$.

We now try to formalize what we require of a message digest function in order to have this property.

We also show that message digest functions do not necessarily "look random", so one should not assume such functions share other properties with random functions.

# Formal definition of message digest functions

Let $\mathcal{M}$ be a message space and $\mathcal{H}$ a hash value space, and assume $|\mathcal{M}| \gg |\mathcal{H}|$.

A *message digest* (or *cryptographic one-way hash* or *fingerprint*) function $h$ maps $\mathcal{M} \to \mathcal{H}$.

A *collision* is a pair of messages $m_1, m_2$ such that $h(m_1) = h(m_2)$, and we say that $m_1$ and $m_2$ *collide*.

Because $|\mathcal{M}| \gg |\mathcal{H}|$, $h$ is very far from being one-to-one, and there are many colliding pairs. Nevertheless, it should be hard for an adversary to find collisions.

# Collision-avoidance properties

We consider three increasingly strong versions of what it means to be hard to find collisions:

- One-way: Given $y \in \mathcal{H}$, it is hard to find $m \in \mathcal{M}$ such that $h(m) = y$.
- Weakly collision-free: Given $m \in \mathcal{M}$, it is hard to find $m' \in \mathcal{M}$ such that $m' \neq m$ and $h(m') = h(m)$.
- Strongly collision-free: It is hard to find colliding pairs $(m, m')$.

These definitions are rather vague, for they ignore issues of what we mean by "hard" and "find".

# What does "hard" mean?

Intuitively, "hard" means that Mallory cannot carry out the computation in a feasible amount of time on a realistic computer.

# What does "find" mean?

The term "find" may mean

- "always produces a correct answer", or
- "produces a correct answer with high probability", or
- "produces a correct answer on a significant number of possible inputs with non-negligible probability".

The latter notion of "find" says that Mallory every now and then can break the system. For any given application, there is a maximum acceptable rate of error, and we must be sure that our cryptographic system meets that requirement.

# One-way function

What does it mean for $h$ to be one-way?

It means that no probabilistic polynomial time algorithm $A_h(y)$ produces a message $m$ such that $h(m) = y$ with non-negligible success probability.

(Such an $m$ is called a *pre-image* of $y$ under $h$.)

This is only required for random $y$ chosen according to a particular hash value distribution. There might be particular values of $y$ on which $A_h$ does succeed with high probability.

# Hash value distribution

The hash value distribution we have in mind is the one induced by $h$ applied to the assumed distribution on the message space $\mathcal{M}$.

Thus, the probability of $y$ is the probability that a message $m$ chosen according to the assumed message distribution satisfies $h(m) = y$.

This means that $h$ can be considered one-way even though algorithms might exist that succeed on low-probability subsets of $\mathcal{H}$.

# Properties of Hash Functions

# Collision-resistance

Recall the three collision-resistance properties for a hash function $\mathcal{H}$:

- One-way: Given $y \in \mathcal{H}$, it is hard to find $m \in \mathcal{M}$ such that $h(m) = y$. [1]
- Weakly collision-free: Given $m \in \mathcal{M}$, it is hard to find $m' \in \mathcal{M}$ such that $m' \neq m$ and $h(m') = h(m)$.
- Strongly collision-free: It is hard to find colliding pairs $(m, m')$.

[1] More precisely, no probabilistic polynomial-time algorithm $A(y)$ succeeds with non-negligible probability at finding a pre-image $m \in h^{-1}(y)$, where $y$ is chosen at random from $\mathcal{H}$ with probability proportional to $|h^{-1}(y)|$.

# Hash values can look non-random

Intuitively, we like to think of $h(m)$ as being *"random-looking"*, with no obvious pattern.

Indeed, it would seem that obvious patterns and structure in $h$ would provide a means of finding collisions, violating the property of being strong collision-free.

However, hash functions don't necessarily look random or share other properties of random functions, as I now show.

# Example of a non-random-looking hash function

Suppose $h$ is a strong collision-free hash function.

Define $H(m) = 0 \cdot h(m)$.

If $(m, m')$ is a colliding pair for $H$, then $(m, m')$ is also a colliding pair for $h$.

Hence, if we could find colliding pairs for $H$, we could find colliding pairs for $h$, contradicting the assumption that $h$ is strong collision-free.

We conclude that $H$ is strong collision-free, despite the fact that $H(m)$ always begins with 0.

# A one-way function that is sometimes easy to invert

Let $h(m)$ be a cryptographic hash function that produces hash values of length $n$. Define a new hash function $H(m)$ as follows:

$$H(m) = \begin{cases} 0 \cdot m & \text{if } |m| = n \\ 1 \cdot h(m) & \text{otherwise.} \end{cases}$$

Thus, $H$ produces hash values of length $n + 1$.

- $H(m)$ is clearly collision-free since the only possible collisions are for $m$'s of lengths different from $n$.
- Any colliding pair $(m, m')$ for $H$ is also a colliding pair for $h$.
- Since $h$ is collision-free, then so is $H$.

# $H$ is one-way

$H$ is one-way, assuming uniformly distributed messages.

This is true, *even though $H$ can be inverted for 1/2 of all possible hash values $y$*, namely, those that begin with 0.

The reason this doesn't violate the definition of one-wayness is that only $2^n$ values of $m$ map to hash values that begin with 0, and all the rest map to values that begin with 1.

Since we are assuming $|\mathcal{M}| \gg |\mathcal{H}|$, the probability is small that a uniformly sampled $m \in \mathcal{M}$ has length exactly $n$.

We see that $H$ is a cryptographic hash function, even though $H$ does not look random.

# Strong implies weak collision-free

There are some obvious relationships between properties of hash functions that can be made precise once the underlying definitions are made similarly precise.

## Fact

*If h is strong collision-free, then h is weak collision-free, assuming uniformly distributed messages.*

# Proof that strong ⇒ weak collision-free

## Proof (Sketch).

Suppose $h$ is *not* weak collision-free. We show that it is not strong collision-free by showing how to enumerate colliding message pairs.

The method is straightforward:

- Pick a random message $m \in \mathcal{M}$.
- Try to find a colliding message $m'$.
- If we succeed, then output the colliding pair $(m, m')$.
- If not, try again with another randomly-chosen message.

Since $h$ is not weak collision-free, we will succeed in finding $m'$ for a significant number of $m$. Each success yields a colliding pair $(m, m')$. □

# Speed of finding colliding pairs

How fast the pairs are enumerated depends on how often the algorithm succeeds and how fast it is.

These parameters in turn may depend on how large $\mathcal{M}$ is relative to $\mathcal{H}$.

It is always possible that $h$ is one-to-one on some subset $U$ of elements in $\mathcal{M}$, so it is not necessarily true that every message has a colliding partner.

However, an easy counting argument shows that $U$ has size at most $|\mathcal{H}| - 1$.

Since we assume $|\mathcal{M}| \gg |\mathcal{H}|$, the probability that a randomly-chosen message from $\mathcal{M}$ lies in $U$ is correspondingly small.

# Strong implies one-way

In a similar vein, we argue that strong collision-free implies one-way.

## Fact

*If $h$ is strong collision-free, then $h$ is one-way.*

# Proof that strong ⇒ one-way

## Proof (Sketch).

Suppose $h$ is *not* one-way. Then there is an algorithm $A(y)$ for finding $m$ such that $h(m) = y$, and $A(y)$ succeeds with non-negligible probability when $y$ is chosen randomly with probability proportional to the size of its preimage. Assume that $A(y)$ returns $\perp$ to indicate failure.

A randomized algorithm to enumerate colliding pairs:

1. *Choose random $m$.*
2. *Compute $y = h(m)$.*
3. *Compute $m' = A(y)$.*
4. *If $m' \notin \{\perp, m\}$ then output $(m, m')$.*
5. *Start over at step 1.*

# Proof (cont.)

## Proof (continued).

Each iteration of this algorithm succeeds with significant probability $\varepsilon$ that is the product of the probability that $A(y)$ succeeds on $y$ and the probability that $m' \neq m$.

The latter probability is at least $1/2$ except for those values $m$ which lie in the set of $U$ of messages on which $h$ is one-to-one (defined in the previous proof).

Thus, assuming $|\mathcal{M}| \gg |\mathcal{H}|$, the algorithm outputs each colliding pair in expected number of iterations that is only slightly larger than $1/\varepsilon$. □

# Weak implies one-way

These same ideas can be used to show that weak collision-free implies one-way, but now one has to be more careful with the precise definitions.

## Fact

*If $h$ is weak collision-free, then $h$ is one-way.*

# Proof that weak ⇒ one-way

## Proof (Sketch).

Suppose as before that $h$ is *not* one-way, so there is an algorithm $A(y)$ for finding $m$ such that $h(m) = y$, and $A(y)$ succeeds with significant probability when $y$ is chosen randomly with probability proportional to the size of its preimage.

Assume that $A(y)$ returns $\perp$ to indicate failure. We want to show this implies that the weak collision-free property does not hold, that is, there is an algorithm that, for a significant number of $m \in \mathcal{M}$, succeeds with non-negligible probability in finding a colliding $m'$.

# Proof that weak ⇒ one-way (cont.)

We claim the following algorithm works:

*Given input $m$:*

1. *Compute $y = h(m)$.*
2. *Compute $m' = A(y)$.*
3. *If $m' \notin \{\perp, m\}$ then output $(m, m')$ and halt.*
4. *Otherwise, start over at step 1.*

This algorithm fails to halt for $m \in U$, but the number of such $m$ is small (= insignificant) when $|\mathcal{M}| \gg |\mathcal{H}|$.

# Proof that weak ⇒ one-way (cont.)

It may also fail even when a colliding partner $m'$ exists if it happens that the value returned by $A(y)$ is $m$. (Remember, $A(y)$ is only required to return some preimage of $y$; we can't say which.)

However, corresponding to each such bad case is another one in which the input to the algorithm is $m'$ instead of $m$. In this latter case, the algorithm succeeds, since $y$ is the same in both cases. With this idea, we can show that the algorithm succeeds in finding a colliding partner on at least half of the messages in $\mathcal{M} - U$.

# Constructing New Hash Functions from Old

# Extending a hash function

Suppose we are given a strong collision-free hash function

$$h : 256\text{-bits} \rightarrow 128\text{-bits}.$$

How can we use $h$ to build a strong collision-free hash function

$$H : 512\text{-bits} \rightarrow 128\text{-bits}?$$

We consider several methods.

In the following, $M$ is 512 bits long.
We write $M = m_1 m_2$, where $m_1$ and $m_2$ are 256 bits each.

# Method 1

First idea. Define

$$H(M) = H(m_1 m_2) = h(m_1) \oplus h(m_2).$$

Unfortunately, this fails to be either strong or weak collision-free.

Let $M' = m_2 m_1$. $(M, M')$ is always a colliding pair for $H$ except in the special case that $m_1 = m_2$.

Recall that $(M, M')$ is a colliding pair iff $H(M) = H(M')$ and $M \neq M'$.

# Method 2

Second idea. Define

$$H(M) = H(m_1 m_2) = h(h(m_1)h(m_2)).$$

$m_1$ and $m_2$ are suitable arguments for $h()$ since $|m_1| = |m_2| = 256$.

Also, $h(m_1)h(m_2)$ is a suitable argument for $h()$ since $|h(m_1)| = |h(m_2)| = 128$.

## Theorem

*If $h$ is strong collision-free, then so is $H$.*

# Correctness proof for Method 2

Assume $H$ has a colliding pair $(M = m_1 m_2,\ M' = m_1' m_2')$.
Then $H(M) = H(M')$ but $M \neq M'$.

Case 1: $h(m_1) \neq h(m_1')$ or $h(m_2) \neq h(m_2')$.
Let $u = h(m_1)h(m_2)$ and $u' = h(m_1')h(m_2')$.
Then $h(u) = H(M) = H(M') = h(u')$, but $u \neq u'$.
Hence, $(u, u')$ is a colliding pair for $h$.

Case 2: $h(m_1) = h(m_1')$ and $h(m_2) = h(m_2')$.
Since $M \neq M'$, then $m_1 \neq m_1'$ or $m_2 \neq m_2'$ (or both).
Whichever pair is unequal is a colliding pair for $h$.

In each case, we have found a colliding pair for $h$.

Hence, $H$ not strong collision-free $\Rightarrow$ $h$ not strong collision-free.
Equivalently, $h$ strong collision-free $\Rightarrow$ $H$ strong collision-free.

# A general chaining method

Let $h$ : $r$-bits $\rightarrow$ $t$-bits be a hash function, where $r \geq t + 2$.
(In the above example, $r = 256$ and $t = 128$.)
Define $H(m)$ for $m$ of arbitrary length.

- Divide $m$ after appropriate padding into blocks $m_1 m_2 \ldots m_k$, each of length $r - t - 1$.
- Compute a sequence of $t$-bit states:

$$\begin{aligned} s_1 &= h(0^t 0 m_1) \\ s_2 &= h(s_1 1 m_2) \\ &\vdots \\ s_k &= h(s_{k-1} 1 m_k). \end{aligned}$$

Then $H(m) = s_k$.

# Chaining construction gives strong collision-free hash

**Theorem**

*Let $h$ be a strong collision-free hash function. Then the hash function $H$ constructed from $h$ by chaining is also strong collision-free.*

# Correctness proof

Assume $H$ has a colliding pair $(m, m')$.
We find a colliding pair for $h$.

- Let $m = m_1 m_2 \dots m_k$ give state sequence $s_1, \dots, s_k$.
- Let $m' = m'_1 m'_2 \dots m'_{k'}$ give state sequence $s'_1, \dots, s'_{k'}$.

Assume without loss of generality that $k \leq k'$.

Because $m$ and $m'$ collide under $H$, we have $s_k = s'_{k'}$.
Let $r$ be the largest value for which $s_{k-r} = s'_{k'-r}$.

Let $i = k - r$, the index of the first such equal pair $s_i = s'_{k'-k+i}$.

We proceed by cases.

(continued...)

# Correctness proof (case 1)

**Case 1:** $i = 1$ and $k = k'$.

Then $s_j = s'_j$ for all $j = 1, \ldots, k$.

Because $m \neq m'$, there must be some $\ell$ such that $m_\ell \neq m'_\ell$.

If $\ell = 1$, then $(0^t 0 m_1, 0^t 0 m'_1)$ is a colliding pair for $h$.

If $\ell > 1$, then $(s_{\ell-1} 1 m_\ell,\ s'_{\ell-1} 1 m'_\ell)$ is a colliding pair for $h$.

(continued...)

# Correctness proof (case 2)

**Case 2:** $i = 1$ and $k < k'$.

Let $u = k' - k + 1$.

Then $s_1 = s'_u$.

Since $u > 1$ we have that

$$h(0^t 0 m_1) = s_1 = s'_u = h(s'_{u-1} 1 m'_u),$$

so $(0^t 0 m_1,\ s'_{u-1} 1 m'_u)$ is a colliding pair for $h$.

Note that this is true even if $0^t = s'_{u-1}$ and $m_1 = m'_u$, a possibility that we have not ruled out.

(continued...)

# Correctness proof (case 3)

**Case 3:** $i > 1$.

Then $u = k' - k + i > 1$.

By choice of $i$, we have $s_i = s'_u$, but $s_{i-1} \neq s'_{u-1}$.

Hence,

$$h(s_{i-1}1m_i) = s_i = s'_u = h(s'_{u-1}1m'_u),$$

so $(s_{i-1}1m_i,\ s'_{u-1}1m'_u)$ is a colliding pair for $h$.

(continued...)

# Correctness proof (conclusion)

In each case, we found a colliding pair for $h$.

The contradicts the assumption that $h$ is strong collision-free.

Hence, $H$ is also strong collision-free.