HASH FUNCTIONS

Hash functions

- MD: MD4, MD5, MD6
- SHA2: SHA1, SHA224, SHA256, SHA384, SHA512
- SHA3: SHA3-224, SHA3-256, SHA3-384, SHA3-512

Their primary purpose is collision-resistant data compression, but they have many other purposes and properties as well ... A hash function is often treated like a magic wand ...

Some uses:
- Certificates: How you know www.snapchat.com really is Snapchat
- Bitcoin
- Data authentication with HMAC: TLS, ...

SHA = “Secure Hash Algorithm” 😊
A collision for a function \( h : D \rightarrow \{0, 1\}^n \) is a pair \( x_1, x_2 \in D \) of points such that

\[
\begin{align*}
 & h(x_1) = h(x_2), \\
 & x_1 \neq x_2.
\end{align*}
\]

If \(|D| > 2^n\) then the pigeonhole principle tells us that there must exist a collision for \( h \).
Collisions

A collision for a function \( h: D \to \{0, 1\}^n \) is a pair \( x_1, x_2 \in D \) of points such that

- \( h(x_1) = h(x_2) \), and
- \( x_1 \neq x_2 \).

If \( |D| > 2^n \) then the pigeonhole principle tells us that there must exist a collision for \( h \).

We want that even though collisions exist, they are hard to find.

Collision-resistance of a function family

The formalism considers a family \( H: \text{Keys} \times D \to R \) of functions, meaning for each \( K \in \text{Keys} \) we have a function \( H_K: D \to R \) defined by \( H_K(x) = H(K, x) \).

Game \( \text{CR}_H \)

<table>
<thead>
<tr>
<th>procedure Initialize ( K \leftarrow^$ \text{Keys} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return ( K )</td>
</tr>
</tbody>
</table>

procedure \( \text{Finalize}(x_1, x_2) \)

If \( x_1 = x_2 \) then return false
If \( x_1 \notin D \) or \( x_2 \notin D \) then return false
Return \( (H_K(x_1) = H_K(x_2)) \)

Let

\[
\text{Adv}_H^A(A) = \Pr \left[ \text{CR}_H^A \Rightarrow \text{true} \right].
\]

Collision-resistance

A collision for a function \( h: D \to \{0, 1\}^n \) is a pair \( x_1, x_2 \in D \) of points such that

- \( h(x_1) = h(x_2) \), and
- \( x_1 \neq x_2 \).

If \( |D| > 2^n \) then the pigeonhole principle tells us that there must exist a collision for \( h \).

We want that even though collisions exist, they are hard to find.

Let

\[
\text{Adv}_H^A(A) = \Pr \left[ \text{CR}_H^A \Rightarrow \text{true} \right].
\]
Example

Let $N = 2^{256}$ and define

$$H: \{1, \ldots, N\} \times \{0, 1, 2, \ldots\} \rightarrow \{0, 1, \ldots, N - 1\}$$

by

$$H(K, x) = (x \mod K).$$

Q: Is $H$ collision resistant?

A: NO!

Why? $(x + K) \mod K = x \mod K$

Example

Let $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher.
Let $H: \{0, 1\}^k \times \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$ be defined by

$$\text{Alg } H(K, x[1]x[2])$$

$$y \leftarrow E_K(E_K(x[1]) \oplus x[2]); \text{ Return } y$$

Let’s show that $H$ is not collision-resistant by giving an efficient adversary $A$ such that $\text{Adv}^{cr}_H(A) = 1$.

Example

Let $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher.
Let $H: \{0, 1\}^k \times \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$ be defined by

$$\text{Alg } H(K, x[1]x[2])$$

$$y \leftarrow E_K(E_K(x[1]) \oplus x[2]); \text{ Return } y$$

Let’s show that $H$ is not collision-resistant by giving an efficient adversary $A$ such that $\text{Adv}^{cr}_H(A) = 1$.

Idea: Pick $x_1 = x_2[1]x_2[2]$ and $x_2 = x_2[1]x_2[2]$ so that

$$E_K(x_1[1]) \oplus x_1[2] = E_K(x_2[1]) \oplus x_2[2]$$
Example

**Alg** $H(K, x[1]x[2])$

$y \leftarrow E_K(E_K(x[1]) \oplus x[2])$; Return $y$

**Idea:** Pick $x_1 = x_1[1]x_1[2]$ and $x_2 = x_2[1]x_2[2]$ so that

$E_K(x_1[1]) \oplus x_1[2] = E_K(x_2[1]) \oplus x_2[2]$

**adversary** $A(K)$

$x_1 \leftarrow 0^n1^n; x_2[2] \leftarrow 0^n; x_2[1] \leftarrow E_K^{-1}(E_K(x_1[1]) \oplus x_1[2] \oplus x_2[2])$

return $x_1, x_2$

Then $Adv^{CT}_H(A) = 1$ and $A$ is efficient, so $H$ is not CR.

Note how we used the fact that $A$ knows $K$ and the fact that $E$ is a blockcipher!

---

**Exercise**

Let $E: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ be a blockcipher. Let $D$ be the set of all strings whose length is a positive multiple of $l$.

Define the hash function $H: \{0, 1\}^k \times D \rightarrow \{0, 1\}^l$ as follows:

**Alg** $H(K, M)$


$C[0] \leftarrow 0^l$

For $i = 1, \ldots, n$

$B[i] \leftarrow E(K, C[i - 1] \oplus M[i]); C[i] \leftarrow E(K, B[i] \oplus M[i])$

Return $C[n]$

Show that $H$ is not CR by giving an efficient adversary $A$ such that $Adv^{CT}_H(A) = 1$.

Keyless hash functions

We say that $H$: $\text{Keys} \times D \rightarrow R$ is **keyless** if $\text{Keys} = \{\varepsilon\}$ consists of just one key, the empty string.

In this case we write $H(x)$ in place of $H(\varepsilon, x)$ or $H_e(x)$.

Practical hash functions like the MD, SHA2 and SHA3 series are keyless.

---

**SHA256**

The hash function SHA256: $\{0, 1\}^{<2^{64}} \rightarrow \{0, 1\}^{256}$ is **keyless**, with

- Inputs being strings $X$ of any length strictly less than $2^{64}$
- Outputs always having length 256.

**Alg** $\text{SHA256}(X)$  // $|X| < 2^{64}$

$M \leftarrow \text{shapad}(X)  // |M| \text{ mod } 512 = 0$

$M^{(1)}M^{(2)} \ldots M^{(n)} \leftarrow M  // \text{Break } M \text{ into } 512 \text{ bit blocks}$

$H^0_0 \leftarrow 6a09e6677 \ ; \ H^0_1 \leftarrow \text{bb67ae85} \ ; \ \ldots ; \ H^{(0)}_l \leftarrow 5be0cd19$

$H^{(0)}_0 \leftarrow H^{(0)}_1 \ ; \ H^{(0)}_2 \ ; \ \ldots ; \ H^{(0)}_l  // |H^{(0)}_l| = 32, |H^{(0)}_0| = 256$

For $i = 1, \ldots, n$ do $H^{(i)} \leftarrow \text{sha256}(M^{(i)} \| H^{(i-1)})$

Return $H^{(n)}$

sha256: $\{0, 1\}^{512+256} \rightarrow \{0, 1\}^{256}$ is the **compression function**.
Padding, and initialization vector $H^{(0)}$

\[ \text{Alg} \; \text{shapad}(X) \quad // \quad |X| < 2^{64} \]
\[ d \leftarrow (447 - |X|) \mod 512 \quad // \quad \text{Chosen to make } |M| \text{ a multiple of 512} \]
Let $\ell$ be the 64-bit binary representation of $|M|$ \[ M \leftarrow X \| 1 \| 0^d \| \ell \quad // \quad |M| \text{ is a multiple of 512} \]
return $M$ \[ \]

The 32-bit word $H^{(0)}$ was obtained by taking the first 32 bits of the fractional part of the square root of the $j$-th prime number ($0 \leq j \leq 7$).

Compression function sha256

Compression function sha256: $\{0, 1\}^{512+256} \rightarrow \{0, 1\}^{256}$ takes a 512 + 256 = 768 bit input and returns a 256-bit output.

\[ \text{Alg} \; \text{sha256}(x) \quad // \quad X = 512, v = 256 \]
\[ w \leftarrow E^{\text{sha256}}(x, v) \]
\[ w_0 \cdots w_7 \leftarrow w \quad // \quad \text{Break } w \text{ into 32-bit words} \]
\[ v_0 \cdots v_7 \leftarrow v \quad // \quad \text{Break } v \text{ into 32-bit words} \]
For $j = 0, \ldots, 7$ do $h_j \leftarrow w_j + v_j$ \[ h \leftarrow h_0 \ldots h_7 \quad // \quad |h| = 256 \]
Return $h$

Here and on next slide, “+” denotes addition modulo $2^{32}$. $E^{\text{sha256}}: \{0, 1\}^{512} \times \{0, 1\}^{256} \rightarrow \{0, 1\}^{256}$ is a block cipher with 512-bit keys and 256-bit blocks.

Block cipher $E^{\text{sha256}}$

\[ \text{Alg} \; E^{\text{sha256}}(x, v) \quad // \quad x \text{ is a 512-bit key, } v \text{ is a 256-bit input} \]
\[ x_0 \cdots x_{15} \leftarrow x \quad // \quad \text{Break } x \text{ into 32-bit words} \]
For $t = 0, \ldots, 15$ do $W_t \leftarrow x_t$ \[ \]
For $t = 16, \ldots, 63$ do $W_t \leftarrow \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$ \[ v_0 \cdots v_7 \leftarrow v \quad // \quad \text{Break } v \text{ into 32-bit words} \]
For $j = 0, \ldots, 7$ do $S_j \leftarrow v_j$ \[ \]
For $t = 0, \ldots, 63$ do $\quad // \quad 64$ rounds
\[ T_1 \leftarrow S_7 + \gamma_1(S_4) + Ch(S_4, S_5, S_6) + C_t + W_t \]
\[ T_2 \leftarrow \gamma_0(S_0) + Maj(S_0, S_1, S_2) \]
\[ S_7 \leftarrow S_6 \quad ; \quad S_6 \leftarrow S_5 \quad ; \quad S_5 \leftarrow S_4 \quad ; \quad S_4 \leftarrow S_3 + T_1 \]
\[ S_3 \leftarrow S_2 \quad ; \quad S_2 \leftarrow S_1 \quad ; \quad S_1 \leftarrow S_0 \quad ; \quad S_0 \leftarrow T_1 + T_2 \]
\[ S \leftarrow S_0 \cdots S_7 \]
Return $S$ \quad // \quad 256-bit output

Internals of block cipher $E^{\text{sha256}}$

On the previous slide:
- $\sigma_0, \sigma_1, \gamma_0, \gamma_1, Ch, Maj$ are functions not detailed here.
- $C_t = 428a2f98$, $C_0 = 71374491$, $\ldots$, $C_{63} = c67178f2$ are constants, where $C_i$ is the first 32 bits of the fractional part of the cube root of the $i$-th prime.
**SHA256 hash calculator**

http://www.xorbin.com/tools/sha256-hash-calculator

SHA-256 produces a 256-bit (32-byte) hash value.

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSE 107 is way too easy!</td>
</tr>
</tbody>
</table>

**SHA-256 hash**

7263ee436ed0568b95f70b588465f49923ab2c39677a9c862d536a427988db96f

Calculate SHA256 hash

---

**Usage of hash functions**

Uses include hashing the data before signing in creation of certificates, data authentication with HMAC, key-derivation, Bitcoin, ...

These will have to wait, so we illustrate another use, the hashing of passwords.

---

**Authentication via passwords**

- Client $A$ has a password $PW$ that is also stored by server $B$
- $A$ authenticates itself by sending $PW$ to $B$ over a secure channel (TLS)

$$A^{PW} \rightarrow PW \rightarrow B^{PW}$$

**Problem:** The password will be found by an attacker who compromises the server.

These types of server compromises are common and often in the news: Yahoo, Equifax, ...

---

**Hashed passwords**

- Client $A$ has a password $PW$ and server stores $\overline{PW} = H(PW)$.
- $A$ sends $PW$ to $B$ (over a secure channel) and $B$ checks that $H(PW) = \overline{PW}$

$$A^{PW} \rightarrow PW \rightarrow B^{\overline{PW}}$$

Server compromise results in attacker getting $\overline{PW}$ which should not reveal $PW$ as long as $H$ is one-way, which is a consequence of collision-resistance.

But we will revisit this when we consider dictionary attacks!

This is how client authentication is done on the Internet, for example login to gmail.com.
Birthday collision-finding attack

Let \( H : \{0, 1\}^k \times D \rightarrow \{0, 1\}^n \) be a family of functions with \(|D| > 2^n\). The \( q \)-trial birthday attack is the following adversary \( A_q \) for game \( \text{CR}_H \):

adversary \( A_q(K) \)

for \( i = 1, \ldots, q \) do \( x_i \leftarrow D ; y_i \leftarrow H_K(x_i) \)
if \( \exists i, j \ (i \neq j \text{ and } y_i = y_j \text{ and } x_i \neq x_j) \) then return \( x_i, x_j \)
else return \( \perp \)

Interestingly, the analysis of this via the birthday problem is not trivial, but it shows that

\[
\text{Adv}^*_H(A_q) \geq 0.3 \cdot \frac{q(q-1)}{2^n}.
\]

So a collision can usually be found in about \( q = \sqrt{2^n} \) trials.

Birthday attack times

<table>
<thead>
<tr>
<th>Function</th>
<th>( n )</th>
<th>( T_B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD4</td>
<td>128</td>
<td>( 2^{64} )</td>
</tr>
<tr>
<td>MD5</td>
<td>128</td>
<td>( 2^{64} )</td>
</tr>
<tr>
<td>SHA1</td>
<td>160</td>
<td>( 2^{80} )</td>
</tr>
<tr>
<td>SHA256</td>
<td>256</td>
<td>( 2^{128} )</td>
</tr>
<tr>
<td>SHA512</td>
<td>512</td>
<td>( 2^{256} )</td>
</tr>
<tr>
<td>SHA3-256</td>
<td>256</td>
<td>( 2^{128} )</td>
</tr>
<tr>
<td>SHA3-512</td>
<td>512</td>
<td>( 2^{256} )</td>
</tr>
</tbody>
</table>

\( T_B \) is the number of trials to find collisions via a birthday attack.

Design of hash functions aims to make the birthday attack the best collision-finding attack, meaning it is desired that there be no attack succeeding in time much less than \( T_B \).

Compression functions

A compression function is a family \( h : \{0, 1\}^k \times \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n \) of functions whose inputs are of a fixed size \( b+n \), where \( b \) is called the block size.

E.g. \( b = 512 \) and \( n = 256 \), in which case

\[
h : \{0, 1\}^k \times \{0, 1\}^{768} \rightarrow \{0, 1\}^{256}
\]

The MD transform

Let \( h : \{0, 1\}^k \times \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n \) be a compression function with block length \( b \). Let \( D \) be the set of all strings of at most \( 2^b - 1 \) blocks.

The MD transform builds from \( h \) a family of functions

\[
H : \{0, 1\}^k \times D \rightarrow \{0, 1\}^n
\]

such that: \( \text{If } h \text{ is CR, then so is } H \)

The problem of hashing long inputs has been reduced to the problem of hashing fixed-length inputs.

There is no need to try to attack \( H \). You won’t find a weakness in it unless \( h \) has one. That is, \( H \) is guaranteed to be secure assuming \( h \) is secure.

For this reason, MD is the design used in many hash functions, including the MD and SHA2 series. SHA3 uses a different paradigm.
MD setup

**Given:** Compression function $h: \{0,1\}^k \times \{0,1\}^{b+n} \rightarrow \{0,1\}^n$.
**Build:** Hash function $H: \{0,1\}^k \times D \rightarrow \{0,1\}^n$.

Since $M \in D$, its length $\ell = |M|$ is a multiple of the block length $b$. We let $\|M\|_b = |M|/b$ be the number of $b$-bit blocks in $M$, and parse as

$$M[1] \ldots M[\ell] \leftarrow M.$$ 

Let $\langle \ell \rangle$ denote the $b$-bit binary representation of $\ell \in \{0, \ldots, 2^b - 1\}$.

MD preserves CR

**Theorem:** Let $h: \{0,1\}^k \times \{0,1\}^{b+n} \rightarrow \{0,1\}^n$ be a family of functions and let $H: \{0,1\}^k \times D \rightarrow \{0,1\}^n$ be obtained from $h$ via the MD transform. Given a cr-adversary $A_H$ we can build a cr-adversary $A_h$ such that

$$\text{Adv}_H^H(A_H) \leq \text{Adv}_h^H(A_h)$$

and the running time of $A_h$ is that of $A_H$ plus the time for computing $h$ on the outputs of $A_H$.

**Implication:**

$$h \text{ CR } \Rightarrow \text{ Adv}_h^H(A_h) \text{ small} \Rightarrow \text{ Adv}_H^H(A_H) \text{ small} \Rightarrow H \text{ CR}$$

MD transform

**Given:** Compression function $h: \{0,1\}^k \times \{0,1\}^{b+n} \rightarrow \{0,1\}^n$.

**Build:** Hash function $H: \{0,1\}^k \times D \rightarrow \{0,1\}^n$.

Algorithm $H_K(M)$

$m \leftarrow \|M\|_b$; $M[m+1] \leftarrow \langle m \rangle$; $V[0] \leftarrow 0^n$

For $i = 1, \ldots, m + 1$ do $v[i] \leftarrow h_K(M[i]|V[i-1])$

Return $V[m+1]$

How are compression functions designed?

Let $E: \{0,1\}^b \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher. Let us define keyless compression function $h: \{0,1\}^{b+n} \rightarrow \{0,1\}^n$ by

$$h(x|v) = E_x(v).$$

**Question:** Is $h$ collision resistant?
Let $E : \{0,1\}^b \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher. Let us define keyless compression function $h : \{0,1\}^{b+n} \to \{0,1\}^n$ by

$$h(x\|v) = E_x(v) .$$

**Question:** Is $h$ collision resistant?

We seek an adversary that outputs distinct $x_1\|v_1$, $x_2\|v_2$ satisfying

$$E_{x_1}(v_1) = E_{x_2}(v_2) .$$

**Answer:** NO, $h$ is NOT collision-resistant, because the following adversary $A$ has $\text{Adv}_h^\text{cr}(A) = 1$:

1. adversary $A$
   - $x_1 \leftarrow 0^b$ ; $x_2 \leftarrow 1^b$ ; $v_1 \leftarrow 0^n$ ; $y \leftarrow E_{x_1}(v_1)$ ; $v_2 \leftarrow E_{x_2}^{-1}(y)$
2. Return $x_1\|v_1$, $x_2\|v_2$
The Davies-Meyer method

Let $E : \{0,1\}^b \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher. Let us define keyless compression function $h : \{0,1\}^{b+n} \rightarrow \{0,1\}^n$ by

$$h(x||v) = E_x(v) \oplus v.$$  

This is called the Davies-Meyer method and is used in the MD and SHA2 series of hash functions, modulo that the $\oplus$ may be replaced by addition.

In particular the compression function sha256 of SHA256 is underlain in this way by the block cipher $E^{sha256} : \{0,1\}^{512} \times \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ that we saw earlier, with the $\oplus$ being replaced by component-wise addition modulo $2^{32}$.

Cryptanalytic attacks

So far we have looked at attacks that do not attempt to exploit the structure of $h$.

Can we get better attacks if we do exploit the structure?

Ideally not, but hash functions have fallen short!

Cryptanalytic attacks against hash functions

<table>
<thead>
<tr>
<th>Year</th>
<th>Against</th>
<th>Time</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993, 1996</td>
<td>md5</td>
<td>$2^{16}$</td>
<td>[dBBo,Do]</td>
</tr>
<tr>
<td>2004</td>
<td>MD5</td>
<td>1 hour</td>
<td>[WaFeLaYu]</td>
</tr>
<tr>
<td>2005, 2006</td>
<td>MD5</td>
<td>1 minute</td>
<td>[LeWadW,KI]</td>
</tr>
<tr>
<td>2005</td>
<td>SHA1</td>
<td>$2^{69}$</td>
<td>[WaYiYu]</td>
</tr>
<tr>
<td>2017</td>
<td>SHA1</td>
<td>$2^{63.1}$</td>
<td>[SBKAM]</td>
</tr>
</tbody>
</table>

Collisions found in compression function md5 of MD5 did not yield collisions for MD5, but collisions for MD5 are now easy.

https://shattered.io/.

2017: Google, Microsoft and Mozilla browsers stop accepting SHA1-based certificates.

The SHA256 and SHA512 hash functions are still viewed as secure, meaning the best known attack is the birthday attack.

SHA1 collision

Here are two PDF files that display different content, yet have the same SHA-1 digest.
Flame exploited an MD5 attack

Cryptographer job-performance evaluation

Why don’t cryptographers build secure hash functions?

Assess their job performance in light of attacks by selecting a grade below:

A – Cryptographers are doing super well
B – They are OK
C – They suck
F – Just fire them all and give the job to AI

Cryptographers’ tightrope

Why don’t cryptographers build secure hash functions?
Cryptographers’ tightrope

Why don’t cryptographers build secure hash functions?

Cryptographers seem perfectly capable of building secure hash functions. The difficulty is that they strive for VERY HIGH SPEED.

SHA256 can run at 3.5 cycles/byte (eBACS: 2018 Intel Core i3-8121U, https://bench.cr.yp.to/results-hash.html) or 0.6 ns per byte, and hardware will make it even faster.

It is AMAZING that one gets ANY security at such low cost.

If you allow cryptographers a 10x slowdown, they can up rounds by 10x and designs seem almost impossible to break.

SHA3

National Institute for Standards and Technology (NIST) held a world-wide competition to develop a new hash function standard.

Contest webpage: http://csrc.nist.gov/groups/ST/hash/index.html

Requested parameters:

- Design: Family of functions with 224, 256, 384, 512 bit output sizes
- Security: CR, one-wayness, near-collision resistance, others...
- Efficiency: as fast or faster than SHA2-256

SHA3: The Sponge construction

Submissions: 64

Round 1: 51


SHA3: 1: Keccak

SHA3: The Sponge construction

\[
\begin{align*}
M & \xrightarrow{\text{pad}} r & \xrightarrow{f} & c & \xrightarrow{f} & \cdots \xrightarrow{f} Z \xrightarrow{\text{Trunc}_d}
\end{align*}
\]

\[f: \{0,1\}^{r+c} \to \{0,1\}^{r+c} \text{ is a (public, invertible!) permutation.}
\]

d is the number of output bits, and \(c = 2d\).

SHA3 does not use the MD paradigm used by the MD and SHA2 series.

Shake\((M, d)\) — Extendable-output function, returning any given number \(d\) of bits.