COMP8760 Lecture

Hash Functions

Sanjay Bhattacherjee

University of Kent

qctxre

Outline

Study Material for Hash Functions

Book 1	*Cryptography Made Simple*
Author	Nigel P. Smart.
	Link to eBook

Section 1.4.2 Birthday Paradox
Section 1.5 Big Numbers

Chapter 14 Hash Functions
Section 14.1 Collision Resistance
Section 14.2 Padding
Section 14.3 The Merkle–Damgård construction
Section 14.4.3 SHA-256

Book Sections

Part 1 Preliminaries

- Birthday Paradox (Section 1.4.2)
- Big Numbers (Section 1.5)

Part 2 Cryptographic Hash Function

- The three security properties (Section 14.1)
- Padding (Section 14.2)
- The Merkle–Damgård construction (Section 14.3)
- SHA-256 (Section 14.4.3)

Outline

As people keep entering a room

What is the probability that none of them have the same birthday?

As people keep entering a room

What is the probability that none of them have the same birthday?

Empty Room

As people keep entering a room

What is the probability that none of them have the same birthday?

Empty Room

As people keep entering a room

What is the probability that none of them have the same birthday?

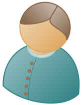
Empty Room $\frac{365}{365}$

As people keep entering a room

What is the probability that none of them have the same birthday?

Empty Room

$\frac{365}{365}$

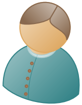


As people keep entering a room

What is the probability that none of them have the same birthday?

Empty Room

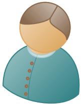
$\frac{365}{365}$



$\frac{365-1}{365}$

As people keep entering a room

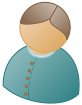
What is the probability that none of them have the same birthday?


Empty Room $\frac{365}{365}$ $\frac{365-1}{365}$

As people keep entering a room

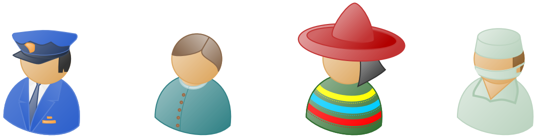
What is the probability that none of them have the same birthday?

Empty Room $\frac{365}{365}$ $\frac{365-1}{365}$ $\frac{365-2}{365}$

As people keep entering a room

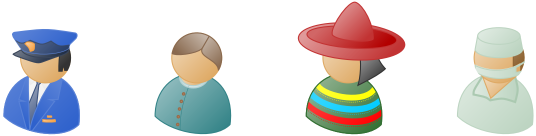
What is the probability that none of them have the same birthday?



Empty Room $\frac{365}{365}$ $\frac{365-1}{365}$ $\frac{365-2}{365}$

As people keep entering a room

What is the probability that none of them have the same birthday?



Empty Room $\frac{365}{365}$ $\frac{365-1}{365}$ $\frac{365-2}{365}$ $\frac{365-3}{365}$

As people keep entering a room

What is the probability that none of them have the same birthday?

Empty Room $\frac{365}{365}$ $\frac{365-1}{365}$ $\frac{365-2}{365}$ $\frac{365-3}{365}$

As people keep entering a room

What is the probability that none of them have the same birthday?

Empty Room $\frac{365}{365}$ $\frac{365-1}{365}$ $\frac{365-2}{365}$ $\frac{365-3}{365}$ $\frac{365-4}{365}$

As people keep entering a room

What is the probability that none of them have the same birthday?

Empty Room $\frac{365}{365}$ $\frac{365-1}{365}$ $\frac{365-2}{365}$ $\frac{365-3}{365}$ $\frac{365-4}{365}$

$$\Pr[\text{None have same birthday}] = \frac{m\cdot(m-1)\cdot(m-2)\cdots(m-k+1)}{m^k} = \frac{{}^{m}P_k}{m^k}$$

$$\Pr[\text{At least one pair collides}] = 1 - \frac{m\cdot(m-1)\cdot(m-2)\cdots(m-k+1)}{m^k} = 1 - \frac{{}^{m}P_k}{m^k}$$

Birthday Paradox: Probability of collision ≈ 1 for $k \ll 365$

What is the probability that none of them have the same birthday?

$\frac{365}{365}$ $\frac{365-1}{365}$ $\frac{365-2}{365}$ $\frac{365-3}{365}$ $\frac{365-4}{365}$

- With 23 people in a room, the probability of a collision is
$$1 - \frac{{}^{365}P_{23}}{365^{23}} = 0.507.$$
- With 30 people in a room, the probability of a collision is
$$1 - \frac{{}^{365}P_{30}}{365^{30}} = 0.706.$$
- With 100 people in a room, the probability of a collision is
$$1 - \frac{{}^{365}P_{100}}{365^{100}} > 0.999.$$

Birthday Bound

The probability of collision in choosing n elements from m is at most

$$\frac{n^2}{2 \cdot m}.$$

Outline

Computational Difficulty

How big can a number be
before it is impossible for someone
to perform that many (classical) computer operations.

- $2^{10} \approx 10^3$.
- 3GHz processor can execute $3,000,000,000$ instructions per second.
- 3GHz octa-core processor can execute 24×10^9 instructions per second.
- Let $\mathcal{C}$ be an advanced classical computer that performs $10^{12} \approx 2^{40}$ instructions per second.
  - For 2^{64} operations:
    $\mathcal{C}$ would take $\frac{2^{64}}{2^{40}}$ seconds or 194 days;
    or one day for 194 $\mathcal{C}$s.
  - For 2^{80} operations ($> 10^{23}$):
    $\mathcal{C}$ would take $\frac{2^{80}}{2^{40}}$ seconds or 34,900 years;
    or 2 years for 15,000 $\mathcal{C}$s.
    Still imaginable!
  - For 2^{128} operations ($> 10^{37}$):
    $\mathcal{C}$ would take $\frac{2^{128}}{2^{40}}$ seconds or $9 \cdot 10^{18}$ years.
    Unimaginable!

Outline

qctxre

Cryptographic Hash Function

Hash Function

A function that
takes arbitrary length bit strings as input and
produces a fixed-length bit string as output;

The output is often called digest, hashcode or hash value.

Cryptographic Hash Function

Hash Function

A function that
takes arbitrary length bit strings as input and
produces a fixed-length bit string as output;

The output is often called digest, hashcode or hash value.

Cryptographic Hash Function

... is a hash function that is one-way.

In other words,
given $H : D \to C$, and $y \in C$,
it is computationally infeasible to find any value $x \in D$ such that

$$y = H(x).$$

Hash Function $H()$

Hash Function $H()$

- Long input, fixed-length output (hash / digest)

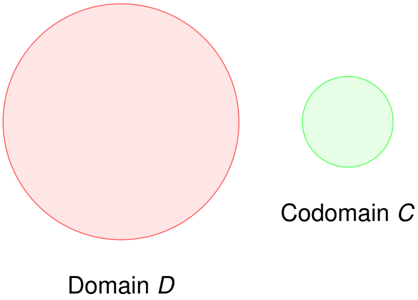
Hash Function $H()$

- Long input, fixed-length output (hash / digest)
- Easy to compute

Hash Function $H()$

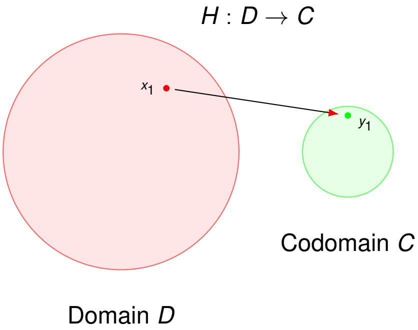
- Long input, fixed-length output (hash / digest)
- Easy to compute
- Hard to invert

$H : D \to C$



Hash Function $H()$

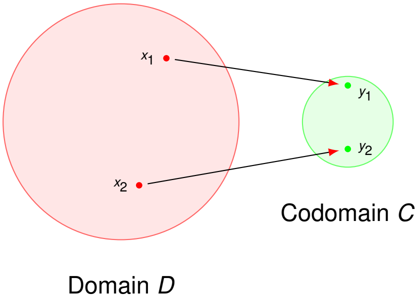
- Long input, fixed-length output (hash / digest)
- Easy to compute
- Hard to invert



Hash Function $H()$

- Long input, fixed-length output (hash / digest)
- Easy to compute
- Hard to invert

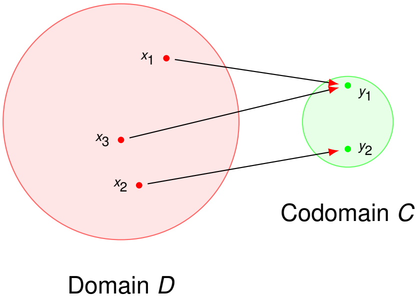
$$H : D \rightarrow C$$



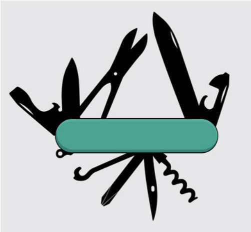
Hash Function $H()$

- Long input, fixed-length output (hash / digest)
- Easy to compute
- Hard to invert

$H : D \to C$



Applications



- Integrity checks: if the hashes do not match, the message must have been altered
- Authentication: through cryptographic signatures
- Key derivation functions: to generate keys from various inputs
- Proof-of-work: blockchains, mitigating spamming
- Password: stored in place of the plaintext password
- Indexing for search: in databases, other file systems
- Perceptual hashing: proximity search for text/image/video
- et cetera ...

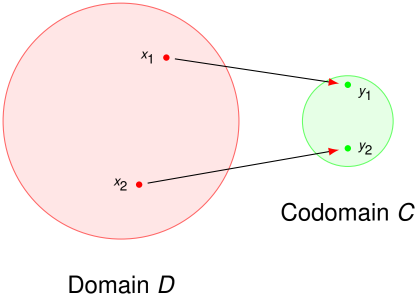
Outline

qctxre

1. Preimage resistance

For a hash function H which produces outputs of t bits, finding preimages should require $O(2^t)$ time.

$$H : D \rightarrow C$$



Preimage Resistance $\equiv$ One-Way Function

Challenger

Adversary

Advantage of Adversary = Pr[Adversary wins the game]

Preimage Resistance $\equiv$ One-Way Function

Challenger

Adversary

H $\longrightarrow$

Advantage of Adversary = Pr[Adversary wins the game]

Preimage Resistance $\equiv$ One-Way Function

Challenger

Adversary

H $\longrightarrow$

$m \leftarrow D$

Advantage of Adversary = Pr[Adversary wins the game]

Preimage Resistance $\equiv$ One-Way Function

Challenger | Adversary

$H \longrightarrow$

$m \leftarrow D$

$h \leftarrow H(m) \longrightarrow$

Advantage of Adversary = Pr[Adversary wins the game]

Preimage Resistance $\equiv$ One-Way Function

Challenger | Adversary

$H \longrightarrow$

$m \leftarrow D$

$h \leftarrow H(m) \longrightarrow$

$m' \longleftarrow$

Advantage of Adversary = Pr[Adversary wins the game]

Preimage Resistance $\equiv$ One-Way Function

Challenger | Adversary

$H \longrightarrow$

$m \leftarrow D$

$h \leftarrow H(m) \longrightarrow$

$m' \longleftarrow$

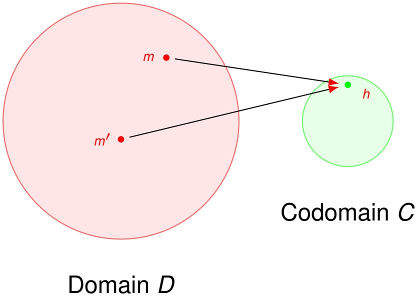
Win if $H(m') = h$

Advantage of Adversary = Pr[Adversary wins the game]

2. Second-preimage resistance

Given m, it should be hard to find an $m' \neq m$ with $H(m') = H(m)$.

$H : D \to C$



Second-Preimage Resistance

Challenger

Adversary

Advantage of Adversary = Pr[Adversary wins the game]

Second-Preimage Resistance

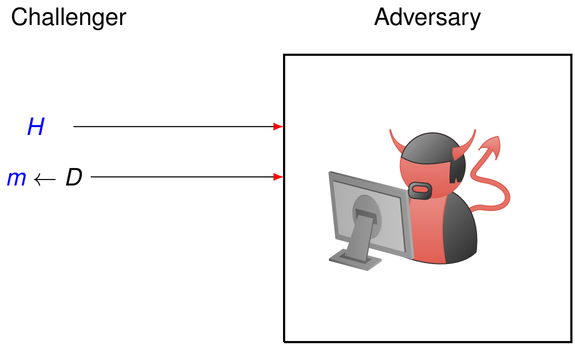
Challenger

Adversary

H ⟶

Advantage of Adversary = Pr[Adversary wins the game]

Second-Preimage Resistance



Advantage of Adversary = Pr[Adversary wins the game]

Second-Preimage Resistance

Challenger | Adversary

$H \longrightarrow$

$m \leftarrow D \longrightarrow$

$m' \longleftarrow$

Advantage of Adversary = Pr[Adversary wins the game]

Second-Preimage Resistance

Challenger | Adversary

H $\longrightarrow$

$m \leftarrow D$ $\longrightarrow$

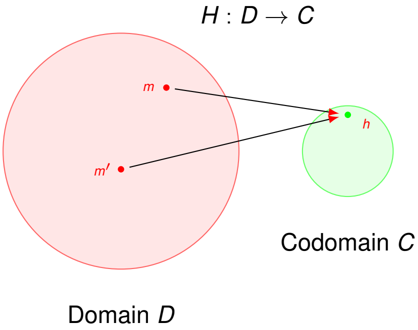
m' $\longleftarrow$

Win if $H(m) = H(m')$
and $m \neq m'$

Advantage of Adversary = Pr[Adversary wins the game]

3. Collision resistance

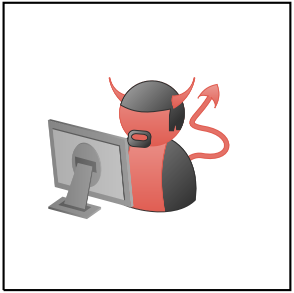
It should be hard to find a pair $m' \neq m$ with $H(m') = H(m)$.



Collision Resistance

Challenger

Adversary



Advantage of Adversary = Pr[Adversary wins the game]

Collision Resistance

Challenger

Adversary

H $\longrightarrow$

Advantage of Adversary = Pr[Adversary wins the game]

Collision Resistance

Challenger | Adversary

$H \longrightarrow$

$m, m' \longleftarrow$

Advantage of Adversary = Pr[Adversary wins the game]

Collision Resistance

Challenger | Adversary

$H \longrightarrow$

$m, m' \longleftarrow$

Win if $H(m) = H(m')$
and $m \neq m'$

Advantage of Adversary = Pr[Adversary wins the game]

Assumption: Collision Resistance by Human Ignorance

$$H : D \to C$$

Assumption: Collision Resistance by Human Ignorance

$$H : D \to C$$

A function H is said to be collision resistant (by human ignorance) or `HI-CR` secure

Assumption: Collision Resistance by Human Ignorance

$$H : D \to C$$

A function H is said to be collision resistant (by human ignorance)
or $\mathtt{HI\text{-}CR}$ secure
if it is believed to be infeasible to find a collision.

Assumption: Collision Resistance by Human Ignorance

$$H : D \to C$$

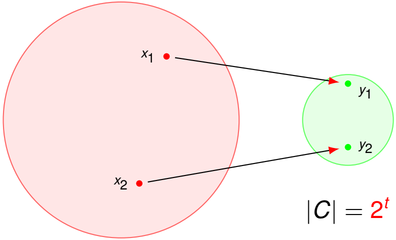
A function H is said to be collision resistant (by human ignorance)
or $\texttt{HI-CR}$ secure
if it is believed to be infeasible to find a collision.

In other words, it is infeasible to find two elements $m, m' \in D$ such that

$$H(m) = H(m').$$

How many queries to find a preimage?

$$H : D \to C$$

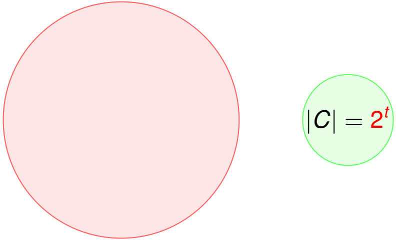


For a “well-designed hash function”, the preimage of an element would be one of 2^t elements of D.
So, the number of queries would be approximately

$$q_{\text{finding preimage}} = 2^t.$$

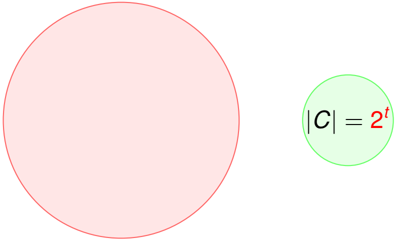
How many queries to find a collision?

$$H : D \to C$$



How many queries to find a collision?

$$H : D \to C$$

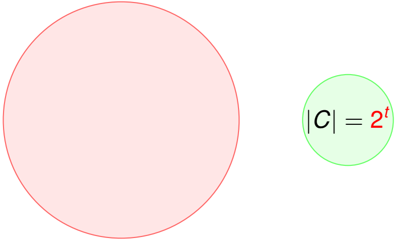


By the birthday bound, after q queries, the probability of collision is at most

$$\frac{q^2}{2 \cdot 2^t}$$

How many queries to find a collision?

$$H : D \to C$$



By the birthday bound, after q queries, the probability of collision is at most

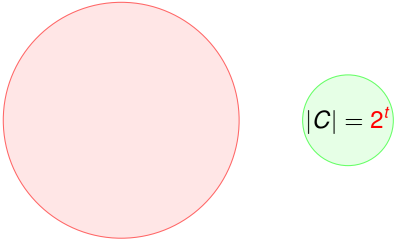
$$\frac{q^2}{2 \cdot 2^t}$$

So, a collision will almost certainly be found after

$$q_{\text{finding collision}} = \sqrt{2^{t+1}} \text{ queries.}$$

How many queries to find a collision?

$$H : D \to C$$



By the birthday bound, after q queries, the probability of collision is at most

$$\frac{q^2}{2 \cdot 2^t}$$

So, a collision will almost certainly be found after

$$q_{\text{finding collision}} = \sqrt{2^{t+1}} \text{ queries.}$$

So, if we require 128-bit security of a hash function, we would want the output to be of size at least 256 bits.

Collision Resistance -> Preimage Resistance (Reduction)

Lemma 14.2

If a function H is preimage resistant for every element of the range of H, it will be collision resistant and second-preimage resistant.

Collision Resistance -> Preimage Resistance (Reduction)

Lemma 14.2

If a function H is preimage resistant for every element of the range of H, it will be collision resistant and second-preimage resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a preimage.

Collision Resistance -> Preimage Resistance (Reduction)

Lemma 14.2

If a function H is preimage resistant for every element of the range of H, it will be collision resistant and second-preimage resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a preimage.
We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

Collision Resistance -> Preimage Resistance (Reduction)

Lemma 14.2

If a function H is preimage resistant for every element of the range of H, it will be collision resistant and second-preimage resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a preimage. We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{preimage}$ be an oracle that provides a preimage for any element of C

Collision Resistance -> Preimage Resistance (Reduction)

Lemma 14.2

If a function H is preimage resistant for every element of the range of H, it will be collision resistant and second-preimage resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a preimage.
We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{preimage}$ be an oracle that provides a preimage for any element of C
- Choose a random $x \in D$ and find $h = H(x)$

Collision Resistance -> Preimage Resistance (Reduction)

Lemma 14.2

If a function H is preimage resistant for every element of the range of H, it will be collision resistant and second-preimage resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a preimage. We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{preimage}$ be an oracle that provides a preimage for any element of C
- Choose a random $x \in D$ and find $h = H(x)$
- Find $x' \leftarrow \mathcal{O}_{preimage}(h)$

Collision Resistance -> Preimage Resistance (Reduction)

Lemma 14.2

If a function H is preimage resistant for every element of the range of H, it will be collision resistant and second-preimage resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a preimage. We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{preimage}$ be an oracle that provides a preimage for any element of C
- Choose a random $x \in D$ and find $h = H(x)$
- Find $x' \leftarrow \mathcal{O}_{preimage}(h)$

Since D is a huge set with many preimages of h, hence it is highly likely that $x \neq x'$.

Collision Resistance -> Preimage Resistance (Reduction)

Lemma 14.2

If a function H is preimage resistant for every element of the range of H, it will be collision resistant and second-preimage resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a preimage. We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{preimage}$ be an oracle that provides a preimage for any element of C
- Choose a random $x \in D$ and find $h = H(x)$
- Find $x' \leftarrow \mathcal{O}_{preimage}(h)$

Since D is a huge set with many preimages of h, hence it is highly likely that $x \neq x'$. Thus, $\mathcal{A}_{collision}$ has found a collision using $\mathcal{O}_{preimage}$!

If there is an efficient $\mathcal{O}_{preimage}$, then there is an efficient $\mathcal{A}_{collision}$.
By contrapositivity, since there is no efficient $\mathcal{A}_{collision}$, hence there is no efficient $\mathcal{O}_{preimage}$.

Collision Resistance -> Preimage Resistance (Reduction)

Lemma 14.2

If a function H is preimage resistant for every element of the range of H, it will be collision resistant and second-preimage resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a preimage. We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{preimage}$ be an oracle that provides a preimage for any element of C
- Choose a random $x \in D$ and find $h = H(x)$
- Find $x' \leftarrow \mathcal{O}_{preimage}(h)$

Since D is a huge set with many preimages of h, hence it is highly likely that $x \neq x'$. Thus, $\mathcal{A}_{collision}$ has found a collision using $\mathcal{O}_{preimage}$!

If there is an efficient $\mathcal{O}_{preimage}$, then there is an efficient $\mathcal{A}_{collision}$.
By contrapositivity, since there is no efficient $\mathcal{A}_{collision}$, hence there is no efficient $\mathcal{O}_{preimage}$.

Similarly for second-preimage resistance.

Hence, Preimage Resistance is a weaker assumption
than Collision Resistance or Second-Preimage Resistance.

Collision Resistance -> Second-Preimage Resistance (Reduction)

Lemma 14.3

If a function H is second-preimage resistant, then it is also collision resistant.

qctxre

Collision Resistance -> Second-Preimage Resistance (Reduction)

Lemma 14.3

If a function H is second-preimage resistant, then it is also collision resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a second-preimage.

Collision Resistance -> Second-Preimage Resistance (Reduction)

Lemma 14.3

If a function H is second-preimage resistant, then it is also collision resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a second-preimage.
We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

Collision Resistance -> Second-Preimage Resistance (Reduction)

Lemma 14.3

If a function H is second-preimage resistant, then it is also collision resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a second-preimage.
We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{second-preimage}$ be an oracle that given an input $x \in D$, provides a second-preimage $x' \in D$.

Collision Resistance -> Second-Preimage Resistance (Reduction)

Lemma 14.3

If a function H is second-preimage resistant, then it is also collision resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a second-preimage.
We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{second-preimage}$ be an oracle that given an input $x \in D$, provides a second-preimage $x' \in D$.
- Choose a random $x \in D$

Collision Resistance -> Second-Preimage Resistance (Reduction)

Lemma 14.3

If a function H is second-preimage resistant, then it is also collision resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a second-preimage.
We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{second-preimage}$ be an oracle that given an input $x \in D$, provides a second-preimage $x' \in D$.
- Choose a random $x \in D$
- $x' \leftarrow \mathcal{O}_{second-preimage}(x)$

Thus, $\mathcal{A}_{collision}$ has found a collision using $\mathcal{O}_{second-preimage}$!

Collision Resistance -> Second-Preimage Resistance (Reduction)

Lemma 14.3

If a function H is second-preimage resistant, then it is also collision resistant.

Proof

We prove this by reducing the problem of finding a collision to that of finding a second-preimage.
We construct an adversary $\mathcal{A}_{collision}$ for finding a collision.

- Let $\mathcal{O}_{second-preimage}$ be an oracle that given an input $x \in D$, provides a second-preimage $x' \in D$.
- Choose a random $x \in D$
- $x' \leftarrow \mathcal{O}_{second-preimage}(x)$

Thus, $\mathcal{A}_{collision}$ has found a collision using $\mathcal{O}_{second-preimage}$!

If there is an efficient $\mathcal{O}_{second-preimage}$, then there is an efficient $\mathcal{A}_{collision}$.
By contrapositivity, since there is no efficient $\mathcal{A}_{collision}$, hence there is no efficient $\mathcal{O}_{second-preimage}$.

Hence, Second-Preimage Resistance is a weaker assumption than Collision Resistance.

Outline

Outline

- Hash Functions: Preliminaries
  - Birthday Bound
  - “Big” Numbers
- Cryptographic Hash Function
  - Hash Function: Security Properties
- SHA-256
  - Padding
  - The Merkle–Damgård construction
  - SHA-256

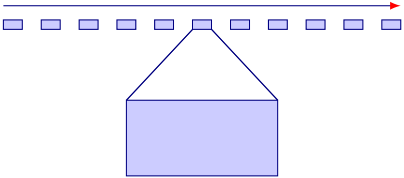
Padding

What if the entire message does not fit in one plaintext?

Padding

What if the entire message does not fit in one plaintext?

- Split the message into smaller plaintext blocks and encrypt each block.

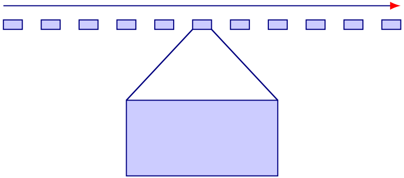


One plaintext block

Padding

What if the entire message does not fit in one plaintext?

- Split the message into smaller plaintext blocks and encrypt each block.



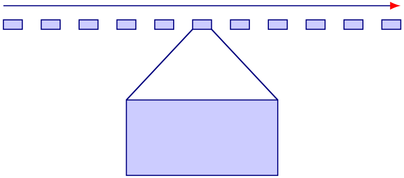
One plaintext block

- What if the number of bits in the message is not a multiple of the number of bits in one plaintext block?

Padding

What if the entire message does not fit in one plaintext?

- Split the message into smaller plaintext blocks and encrypt each block.



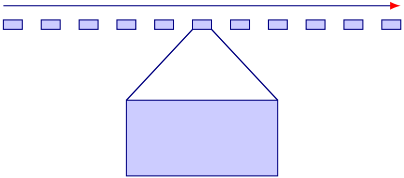
One plaintext block

- What if the number of bits in the message is not a multiple of the number of bits in one plaintext block?
  - The block size b = number of bits in one block

Padding

What if the entire message does not fit in one plaintext?

- Split the message into smaller plaintext blocks and encrypt each block.



One plaintext block

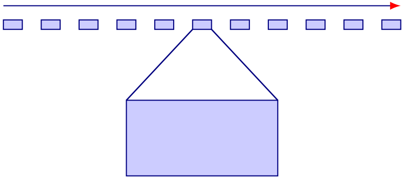
- What if the number of bits in the message is not a multiple of the number of bits in one plaintext block?
    - The block size b = number of bits in one block
    - Add some redundant bits to the message so that.

$$|m| = k \cdot b.$$

Padding

What if the entire message does not fit in one plaintext?

- Split the message into smaller plaintext blocks and encrypt each block.



One plaintext block

- What if the number of bits in the message is not a multiple of the number of bits in one plaintext block?
  - The block size b = number of bits in one block
  - Add some redundant bits to the message so that.

$$|m| = k \cdot b.$$

  - The manner of adding bits should allow the decryption algorithm to separate the portion of the original plaintext from the padded part of the plaintext block.

Types of Padding

m	$pad_0(\|m\|, b)$

Also written as: $\underbrace{m || pad_0(|m|, b)}_{\text{Concatenation}}$

Types of Padding

Also written as: $\underbrace{m||pad_0(|m|, b)}_{\text{Concatenation}}$

Method 0:

Let $v \leftarrow b - \underbrace{|m| \pmod b}_{\text{message bits in last block}}$. Then, the padded message is

$$m||\underbrace{pad_0(|m|, b)}_{\text{padding bits}} = m||\underbrace{0 \ldots 0}_{v \text{ bits}}.$$

Types of Padding

m	$pad_1(\lvert m\rvert, b)$

Also written as: $\underbrace{m \| pad_1(|m|, b)}_{\text{Concatenation}}$

Method 1:

Let $v \leftarrow b - \underbrace{(|m| + 1) \pmod{b}}_{\text{message bits } +1 \text{ in last block}}$. Then, the padded message is

$$m \| \underbrace{pad_1(|m|, b)}_{\text{padding bits}} = m \| 1 \underbrace{0 \ldots 0}_{v \text{ bits}}.$$

qctxre

University of Kent Institute of Cyber Security for Society (iCSS)

Types of Padding

m	$pad_2(\|m\|, b)$

Also written as: $\underbrace{m || pad_2(|m|, b)}_{\text{Concatenation}}$

Method 2:

Let $v \leftarrow b - \underbrace{(|m| + 64 + 1) \pmod b}_{\text{message bits } +65 \text{ in last block}}$. Then, the padded message is

$$m || \underbrace{pad_2(|m|, b)}_{\text{padding bits}} = m || 1 \underbrace{0 \ldots 0}_{v \text{ bits}} || \underbrace{|m|}_{64 \text{ bits}} .$$

Types of Padding

Also written as: $\underbrace{m || pad_3(|m|, b)}_{\text{Concatenation}}$

Method 3:

Let $v \leftarrow b - \underbrace{(|m| + 64) \pmod b}_{\text{message bits } +64 \text{ in last block}}$. Then, the padded message is

$$m || \underbrace{pad_3(|m|, b)}_{\text{padding bits}} = m || \underbrace{0 \ldots 0}_{v \text{ bits}} || \underbrace{|m|}_{64 \text{ bits}} .$$

Types of Padding

m	$pad_4(\|m\|, b)$

Also written as: $\underbrace{m \| pad_4(|m|, b)}_{\text{Concatenation}}$

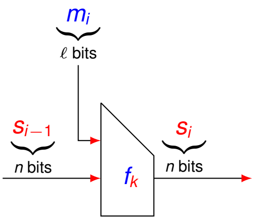
Method 4:

Let $v \leftarrow b - \underbrace{(|m| + 2) \pmod{b}}_{\text{message bits } +2 \text{ in last block}}$. Then, the padded message is

$$m \| \underbrace{pad_4(|m|, b)}_{\text{padding bits}} = m \| 1 \underbrace{0 \ldots 0}_{v \text{ bits}} 1.$$

Outline

The Merkle–Damgård construction



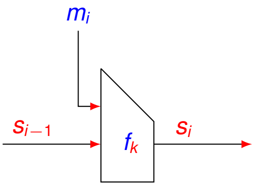
The Building Block: A Compression Function

- A compression function is a hash function taking inputs of a fixed length:

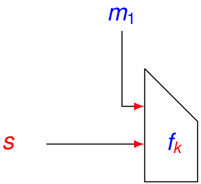
$$f_k : \{0,1\}^{\ell+n} \to \{0,1\}^n.$$

- Map $(\ell + n)$-bit inputs to n-bit outputs.
- When applied iteratively, the variable s_i works as an internal state.

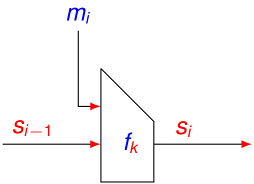
The Merkle–Damgård construction



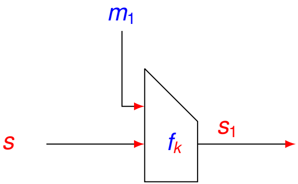
The Building Block: A Compression Function



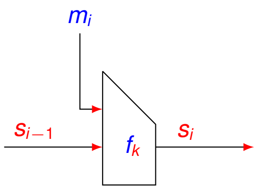
The Merkle–Damgård construction



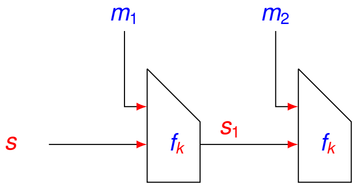
The Building Block: A Compression Function



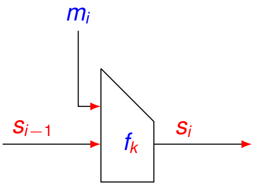
The Merkle–Damgård construction



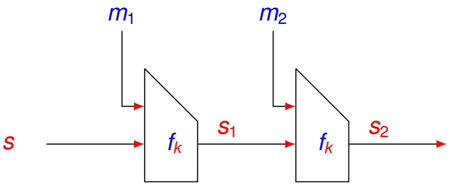
The Building Block: A Compression Function



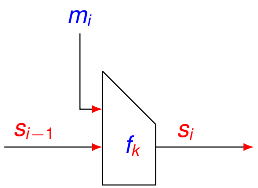
The Merkle–Damgård construction



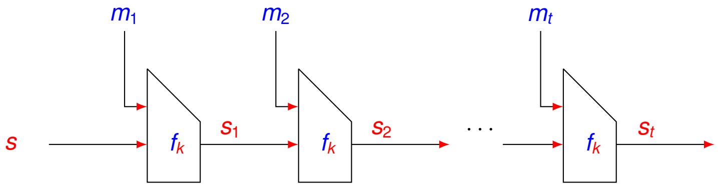
The Building Block: A Compression Function



The Merkle–Damgård construction



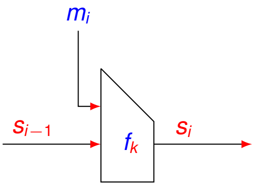
The Building Block: A Compression Function



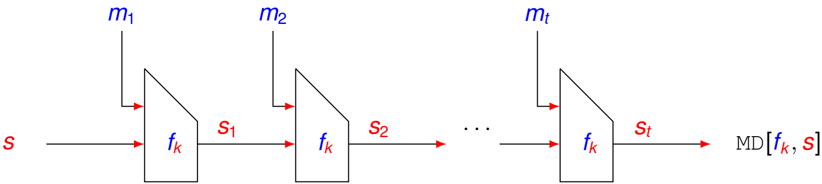
qctxre

University of Kent Institute of Cyber Security for Society (iCSS)

The Merkle–Damgård construction



The Building Block: A Compression Function



The Merkle–Damgård construction

The Merkle–Damgård construction

The Algorithm

Merkle–Damgård construction

The Merkle–Damgård construction

The Algorithm

Merkle–Damgård construction

Step 1: Padding: $m || pad_i(|m|, \ell)$

The Merkle–Damgård construction

The Algorithm

Merkle–Damgård construction

Step 1: Padding: $m || pad_i(|m|, \ell)$

Step 2: Split $\underbrace{m || pad_i(|m|, \ell)}_{\text{padded message}}$ into $\underbrace{m_1 || m_2 || \cdots || m_t}_{t \text{ blocks}}$, each of length ℓ bits.

The Merkle–Damgård construction

The Algorithm

Merkle–Damgård construction

Step 1: Padding: $m||pad_i(|m|, \ell)$

Step 2: Split $\underbrace{m||pad_i(|m|, \ell)}_{\text{padded message}}$ into $\underbrace{m_1||m_2||\cdots||m_t}_{t \text{ blocks}}$, each of length ℓ bits.

Step 3: $s_0 \leftarrow s$

The Algorithm

Merkle–Damgård construction

Step 1: Padding: $m||pad_i(|m|, \ell)$

Step 2: Split $\underbrace{m||pad_i(|m|, \ell)}_{\text{padded message}}$ into $\underbrace{m_1||m_2||\cdots||m_t}_{t \text{ blocks}}$, each of length ℓ bits.

Step 3: $s_0 \leftarrow s$

Step 4: For $i = 1$ to t do:

$f(s_{i-1}||m_i) \rightarrow s_i$

The Merkle–Damgård construction

The Algorithm

Merkle–Damgård construction

Step 1: Padding: $m||pad_i(|m|, \ell)$

Step 2: Split $\underbrace{m||pad_i(|m|, \ell)}_{\text{padded message}}$ into $\underbrace{m_1||m_2||\cdots||m_t}_{t\text{ blocks}}$, each of length ℓ bits.

Step 3: $s_0 \leftarrow s$

Step 4: For $i = 1$ to t do:
$f(s_{i-1}||m_i) \to s_i$

Step 5: return s_t

The Merkle–Damgård construction

Padding Scheme determines Collision Resistance

Let $n = \ell = 4$. Consider the messages

$$m_1 = 0b0 \text{ and } m_2 = 0b00.$$

qctxre

University of Kent | Institute of Cyber Security for Society (iCSS)

The Merkle–Damgård construction

Padding Scheme determines Collision Resistance

Let $n = \ell = 4$. Consider the messages

$$m_1 = 0b0 \text{ and } m_2 = 0b00.$$

When padded using Method 0, we get the same message:

$$0b000000.$$

Padding Scheme determines Collision Resistance

Let $n = \ell = 4$. Consider the messages

$$m_1 = 0b0 \text{ and } m_2 = 0b00.$$

When padded using Method 0, we get the same message:

$$0b000000.$$

So, the output of $f(0b000000)$ will of course be equal in both cases, resulting in a collision!

The Merkle–Damgård construction

Padding Scheme determines Collision Resistance

Let $n = \ell = 4$. Consider the messages

$$m_1 = 0b0 \text{ and } m_2 = 0b00.$$

When padded using Method 0, we get the same message:

$$0b000000.$$

So, the output of $f(0b000000)$ will of course be equal in both cases, resulting in a collision!

Theorem 14.4

Let

$$H_{k,s}(m) = \mathrm{MD}[f_{k,s}](m)$$

denote the keyed hash function constructed using the Merkle–Damgård method from the keyed compression function $f_k(x)$ and Padding Method 2.
If $f_k(x)$ is collision resistant, then $H_{k,s}$ is collision resistant as well.

Common Hash functions

Commonly used Hash functions

Common Hash functions

Commonly used Hash functions

- MD-5, RIPEMD-160, SHA-1 and SHA-2

Common Hash functions

Commonly used Hash functions

- MD-5, RIPEMD-160, SHA-1 and SHA-2
  - All are based on the Merkle–Damgård construction
  - All use a fixed (i.e. unkeyed) compression function f

Common Hash functions

Commonly used Hash functions

- MD-5, RIPEMD-160, SHA-1 and SHA-2
  - All are based on the Merkle–Damgård construction
  - All use a fixed (i.e. unkeyed) compression function f
- MD-5: outputs 128-bit hashes

Common Hash functions

Commonly used Hash functions

- MD-5, RIPEMD-160, SHA-1 and SHA-2
  - All are based on the Merkle–Damgård construction
  - All use a fixed (i.e. unkeyed) compression function f
- MD-5: outputs 128-bit hashes
- RIPEMD-160 and SHA-1: output 160-bit hashes

Common Hash functions

Commonly used Hash functions

- MD-5, RIPEMD-160, SHA-1 and SHA-2
  - All are based on the Merkle–Damgård construction
  - All use a fixed (i.e. unkeyed) compression function f
- MD-5: outputs 128-bit hashes
- RIPEMD-160 and SHA-1: output 160-bit hashes
- SHA-2: three algorithms

Common Hash functions

Commonly used Hash functions

- MD-5, RIPEMD-160, SHA-1 and SHA-2
  - All are based on the Merkle–Damgård construction
  - All use a fixed (i.e. unkeyed) compression function f
- MD-5: outputs 128-bit hashes
- RIPEMD-160 and SHA-1: output 160-bit hashes
- SHA-2: three algorithms
  - SHA-256: outputs 256-bit hashes
  - SHA-384: outputs 384-bit hashes
  - SHA-512: outputs 512-bit hashes

Outline

SHA-256

qctxre

University of Kent | Institute of Cyber Security for Society (iCSS)

SHA-256

- Input: message m of length $0 \leq |m| < 2^{64}$ to be divided into t blocks

$$\underbrace{\underbrace{m_1}_{\text{512 bits}},\ \underbrace{m_2}_{\text{512 bits}},\ldots,\ \underbrace{m_t}_{\text{512 bits}}}_{\leq 2^{64} \approx 16 \times 10^{24} \text{ bits}}$$

each of length $\ell = 512$ bits.

SHA-256

- Input: message m of length $0 \leq |m| < 2^{64}$ to be divided into t blocks

$$\underbrace{\underbrace{m_1}_{\text{512 bits}}, \underbrace{m_2}_{\text{512 bits}}, \ldots, \underbrace{m_t}_{\text{512 bits}}}_{\leq 2^{64} \approx 16 \times 10^{24} \text{ bits}}$$

each of length $\ell = 512$ bits.
For each m_i, the $\ell = 512$ bits are further sub-divided into 16 words

$$\underbrace{\underbrace{X_1}_{\text{32 bits}}, \underbrace{X_2}_{\text{32 bits}}, \underbrace{X_2}_{\text{32 bits}}, \ldots, \underbrace{X_{16}}_{\text{32 bits}},}_{\text{512 bits}}$$

each of size 32 bits.

SHA-256

- Input: message m of length $0 \leq |m| < 2^{64}$ to be divided into t blocks

$$\underbrace{\underbrace{m_1}_{\text{512 bits}}, \underbrace{m_2}_{\text{512 bits}}, \ldots, \underbrace{m_t}_{\text{512 bits}}}_{\leq 2^{64} \approx 16 \times 10^{24} \text{ bits}}$$

  each of length $\ell = 512$ bits.
  For each m_i, the $\ell = 512$ bits are further sub-divided into 16 words

$$\underbrace{\underbrace{X_1}_{\text{32 bits}}, \underbrace{X_2}_{\text{32 bits}}, \underbrace{X_2}_{\text{32 bits}}, \ldots, \underbrace{X_{16}}_{\text{32 bits}},}_{\text{512 bits}}$$

  each of size 32 bits.
- Padding: Method 2 (to make sure $|m|$ is a multiple of $\ell = 512$)

SHA-256

- Input: message m of length $0 \le |m| < 2^{64}$ to be divided into t blocks

$$\underbrace{\underbrace{m_1}_{512\text{ bits}},\ \underbrace{m_2}_{512\text{ bits}},\ \ldots,\ \underbrace{m_t}_{512\text{ bits}}}_{\le 2^{64} \approx 16 \times 10^{24}\text{ bits}}$$

  each of length $\ell = 512$ bits.
  For each m_i, the $\ell = 512$ bits are further sub-divided into 16 words

$$\underbrace{\underbrace{X_1}_{32\text{ bits}},\ \underbrace{X_2}_{32\text{ bits}},\ \underbrace{X_2}_{32\text{ bits}},\ \ldots,\ \underbrace{X_{16}}_{32\text{ bits}},}_{512\text{ bits}}$$

  each of size 32 bits.
- Padding: Method 2 (to make sure $|m|$ is a multiple of $\ell = 512$)
- Output: hash of length $n = 256$ bits

SHA-256

- Input: message m of length $0 \leq |m| < 2^{64}$ to be divided into t blocks

$$\underbrace{\underbrace{m_1}_{\text{512 bits}}, \underbrace{m_2}_{\text{512 bits}}, \ldots, \underbrace{m_t}_{\text{512 bits}}}_{\leq 2^{64} \approx 16 \times 10^{24} \text{ bits}}$$

  each of length $\ell = 512$ bits.
  For each m_i, the $\ell = 512$ bits are further sub-divided into 16 words

$$\underbrace{\underbrace{X_1}_{\text{32 bits}}, \underbrace{X_2}_{\text{32 bits}}, \underbrace{X_2}_{\text{32 bits}}, \ldots, \underbrace{X_{16}}_{\text{32 bits}},}_{\text{512 bits}}$$

  each of size 32 bits.
- Padding: Method 2 (to make sure $|m|$ is a multiple of $\ell = 512$)
- Output: hash of length $n = 256$ bits
- Compression function

$$f : \{0,1\}^{\ell+n} \to \{0,1\}^n$$

  (64 rounds, each of one step)

SHA-256: 32-bit word functions

For 32-bit inputs u, v, w, x, we define the six functions:

Here, $\ggg$ denotes right rotate and $\gg$ denotes right shift

$$f'(u, v, w) \quad = (u \wedge v) \oplus ((\neg u) \wedge w),$$

qctxre

University of Kent | Institute of Cyber Security for Society (iCSS)

SHA-256: 32-bit word functions

For 32-bit inputs u, v, w, x, we define the six functions:

Here, $\ggg$ denotes right rotate and $\gg$ denotes right shift

$$f'(u, v, w) = (u \wedge v) \oplus ((\neg u) \wedge w),$$

$$g'(u, v, w) = (u \wedge v) \oplus (u \wedge w) \oplus (v \wedge w),$$

qctxre

SHA-256: 32-bit word functions

For 32-bit inputs u, v, w, x, we define the six functions:

Here, $\ggg$ denotes right rotate and $\gg$ denotes right shift

$$f'(u, v, w) = (u \wedge v) \oplus ((\neg u) \wedge w),$$

$$g'(u, v, w) = (u \wedge v) \oplus (u \wedge w) \oplus (v \wedge w),$$

$$\textstyle\sum_0(x) = (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22),$$

qctxre

SHA-256: 32-bit word functions

For 32-bit inputs u, v, w, x, we define the six functions:

Here, $\ggg$ denotes right rotate and $\gg$ denotes right shift

$$
\begin{aligned}
f'(u, v, w) &= (u \wedge v) \oplus ((\neg u) \wedge w), \\
g'(u, v, w) &= (u \wedge v) \oplus (u \wedge w) \oplus (v \wedge w), \\
\textstyle\sum_0(x) &= (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22), \\
\textstyle\sum_1(x) &= (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25),
\end{aligned}
$$

SHA-256: 32-bit word functions

For 32-bit inputs u, v, w, x, we define the six functions:

Here, $\ggg$ denotes right rotate and $\gg$ denotes right shift

$$
\begin{aligned}
f'(u, v, w) &= (u \wedge v) \oplus ((\neg u) \wedge w), \\
g'(u, v, w) &= (u \wedge v) \oplus (u \wedge w) \oplus (v \wedge w), \\
\textstyle\sum_0(x) &= (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22), \\
\textstyle\sum_1(x) &= (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25), \\
\sigma_0(x) &= (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3), \text{ and}
\end{aligned}
$$

qctxre

SHA-256: 32-bit word functions

For 32-bit inputs u, v, w, x, we define the six functions:

Here, $\ggg$ denotes right rotate and $\gg$ denotes right shift

$$
\begin{aligned}
f'(u, v, w) &= (u \wedge v) \oplus ((\neg u) \wedge w), \\
g'(u, v, w) &= (u \wedge v) \oplus (u \wedge w) \oplus (v \wedge w), \\
\Sigma_0(x) &= (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22), \\
\Sigma_1(x) &= (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25), \\
\sigma_0(x) &= (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3), \text{ and} \\
\sigma_1(x) &= (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10).
\end{aligned}
$$

SHA-256: 32-bit word constants $K_0, \ldots, K_{63}$

We define 64 constant words: $K_0, \ldots, K_{63}$.
Each K_i is the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers.

K_0 first 32 bits of the fractional part of $\sqrt[3]{2}$,

SHA-256: 32-bit word constants $K_0, \ldots, K_{63}$

We define 64 constant words: $K_0, \ldots, K_{63}$.
Each K_i is the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers.

K_0	first 32 bits of the fractional part of $\sqrt[3]{2}$,
K_1	first 32 bits of the fractional part of $\sqrt[3]{3}$,

SHA-256: 32-bit word constants $K_0, \ldots, K_{63}$

We define 64 constant words: $K_0, \ldots, K_{63}$.
Each K_i is the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers.

K_0	first 32 bits of the fractional part of $\sqrt[3]{2}$,
K_1	first 32 bits of the fractional part of $\sqrt[3]{3}$,
K_2	first 32 bits of the fractional part of $\sqrt[3]{5}$,

qctxre

SHA-256: 32-bit word constants $K_0, \ldots, K_{63}$

We define 64 constant words: $K_0, \ldots, K_{63}$.
Each K_i is the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers.

$$
\begin{array}{ll}
K_0 & \text{first 32 bits of the fractional part of } \sqrt[3]{2}, \\
K_1 & \text{first 32 bits of the fractional part of } \sqrt[3]{3}, \\
K_2 & \text{first 32 bits of the fractional part of } \sqrt[3]{5}, \\
\vdots & \qquad\qquad\qquad \vdots
\end{array}
$$

SHA-256: 32-bit word constants $K_0, \ldots, K_{63}$

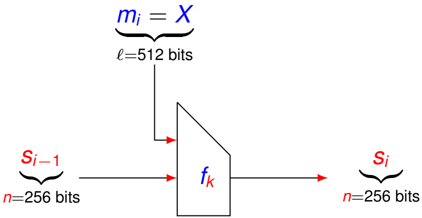
We define 64 constant words: $K_0, \ldots, K_{63}$.
Each K_i is the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers.

K_0	first 32 bits of the fractional part of $\sqrt[3]{2}$,
K_1	first 32 bits of the fractional part of $\sqrt[3]{3}$,
K_2	first 32 bits of the fractional part of $\sqrt[3]{5}$,
$\vdots$	$\vdots$
K_{63}	first 32 bits of the fractional part of $\sqrt[3]{p_{63}}$

qctxre

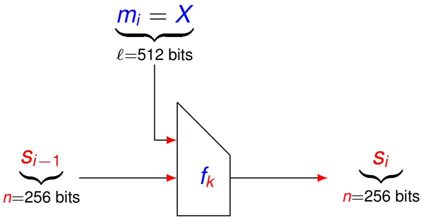
University of Kent Institute of Cyber Security for Society (iCSS)

SHA-256: Internal State s_i



The Building Block: A Compression Function $f : \{0,1\}^{\ell+n} \rightarrow \{0,1\}^n$

SHA-256: Internal State s_i



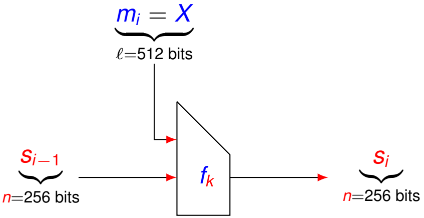
The Building Block: A Compression Function $f : \{0,1\}^{\ell+n} \to \{0,1\}^n$

The internal state s_i is a set of 8 values, each of size 32 bits.

$$H = \underbrace{(\underbrace{H_1}_{32 \text{ bits}}, \underbrace{H_2}_{32 \text{ bits}}, \underbrace{H_3}_{32 \text{ bits}}, \underbrace{H_4}_{32 \text{ bits}}, \underbrace{H_5}_{32 \text{ bits}}, \underbrace{H_6}_{32 \text{ bits}}, \underbrace{H_7}_{32 \text{ bits}}, \underbrace{H_8}_{32 \text{ bits}})}_{256 \text{ bits}}$$

This also corresponds to the 256 bits of input key s_0 / output s_t

SHA-256: Initial State s_0



$$H = \underbrace{(\underbrace{H_1}_{32 \text{ bits}}, \underbrace{H_2}_{32 \text{ bits}}, \underbrace{H_3}_{32 \text{ bits}}, \underbrace{H_4}_{32 \text{ bits}}, \underbrace{H_5}_{32 \text{ bits}}, \underbrace{H_6}_{32 \text{ bits}}, \underbrace{H_7}_{32 \text{ bits}}, \underbrace{H_8}_{32 \text{ bits}})}_{256 \text{ bits}}$$

For the initial state s_0, we assign:

$$
\begin{array}{ll}
H_1 \leftarrow 0x6A09E667, & H_2 \leftarrow 0xBB67AE85, \\
H_3 \leftarrow 0x3C6EF372, & H_4 \leftarrow 0xA54FF53A, \\
H_5 \leftarrow 0x510E527F, & H_6 \leftarrow 0x9B05688C, \\
H_7 \leftarrow 0x1F83D9AB, \text{ and} & H_8 \leftarrow 0x5BE0CD19.
\end{array}
$$

SHA-256: The Compression Function

SHA-256 compression function $f(X||s_{i-1})$

Initialisation: $(Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7, Y_8) \leftarrow (H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$

Expansion: for $j = 17$ to 64 do

$$X_i \leftarrow (\sigma_1(X_{i-2}) + X_{i-7} + \sigma_0(X_{i-15}) + X_{i-16})$$

Rounds: for $j = 1$ to 64 do

$$t_1 \leftarrow (Y_8 + \sum_1(Y_5) + f'(Y_5, Y_6, Y_7) + K_i + X_i)$$
$$t_2 \leftarrow (\sum_0(Y_1) + g'(Y_1, Y_2, Y_3))$$
$$Y_1 \leftarrow (t_1 + t_2)$$
$$Y_2, Y_3, Y_4 \leftarrow Y_1, Y_2, Y_3$$
$$Y_5 \leftarrow Y_4 + t_1$$
$$Y_6, Y_7, Y_8 \leftarrow Y_5, Y_6, Y_7$$

for $k = 1$ to 8

$$H_k \leftarrow Y_k + H_k$$

$$s_i \leftarrow (H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$$

Popular Hash Functions $H(n)$

Check this link: `https://www.browserling.com/tools/all-hashes`

Check the following SHA-256 hashes:

SHA256("WISDOM IS NOWHERE") =
"7db96cca6d0160496161297d522425ffab681f25c6bd57c85bf976a8b7d7372d"

SHA256("WISDOM IS NOW HERE") =
"eae0bc219689024e359b23a1a3e81be946423be9a7f2332c430f7a17c1a315e7".

Note: Each character in a hash is the
hexadecimal encoding of 4 binary digits.

Hex	Binary	Hex	Binary	Hex	Binary	Hex	Binary
0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

Note: A "slight" change in the input, completely changes the hash value!

Very useful for implementing integrity checks!

qctxre

Thank you for your kind attention!

Telephone Directory

Easy to search by name (sorted, hence binary search);
Hard to search by number (unsorted, hence exhaustive search)

Budi Arief	07932731720
Adam Back	07896572341
David Barnes	07723812346
Mark Batty	07983184516
Laura Bocchi	07988762343
Howard Bowman	07981722451
Janet Carter	07986662211
David Chadwick	07766923565
Jacqueline Chetty	07898234553
Olaf Chitil	07799012748
Theodosios Dimitrakos	07978343434
Sally Fincher	07898347543
Virginia Franqueira	07711524511
Radu Grigore	07776351939
Marek Grzes	07928672498
Julio Hernandez-Castro	07717210073
Tim Hopkins	07792378464
Ozgur Kafali	07989634247
Stefan Kahrs	07798127475
Stephen Kell	07987234122
Andy King	07898374335
Julien Lange	07912685674
Rogerio de Lemos	07973542352
Shujun Li	07713467929
Stefan Marr	07987641123
Jason Nurse	07777712981
Dominic Orchard	07981279642
Carlos Perez Delgado	07772129875
Simon Thompson	07981237412
Charles Xavier	07927862449

qctxre

Telephone Directory

Easy to search by name (sorted, hence binary search);
Hard to search by number (unsorted, hence exhaustive search)

Budi Arief	07932731720
Adam Back	07896572341
David Barnes	07723812346
Mark Batty	07983184516
Laura Bocchi	07988762343
Howard Bowman	07981722451
Janet Carter	07986662211
David Chadwick	07766923565
Jacqueline Chetty	07898234553
Olaf Chitil	07799012748
Theodosios Dimitrakos	07978343434
Sally Fincher	07898347543
Virginia Franqueira	07711524511
Radu Grigore	07776351939
Marek Grzes	07928672498
Julio Hernandez-Castro	07717210073
Tim Hopkins	07792378464
Ozgur Kafali	07989634247
Stefan Kahrs	07798127475
Stephen Kell	07987234122
Andy King	07898374335
Julien Lange	07912685674
Rogerio de Lemos	07973542352
Shujun Li	07713467929
Stefan Marr	07987641123
Jason Nurse	07777712981
Dominic Orchard	07981279642
Carlos Perez Delgado	07772129875
Simon Thompson	07981237412
Charles Xavier	07927862449

Phone number of Jason Nurse?

Telephone Directory

Easy to search by name (sorted, hence binary search);
Hard to search by number (unsorted, hence exhaustive search)

Budi Arief	07932731720
Adam Back	07896572341
David Barnes	07723812346
Mark Batty	07983184516
Laura Bocchi	07988762343
Howard Bowman	07981722451
Janet Carter	07986662211
David Chadwick	07766923565
Jacqueline Chetty	07898234553
Olaf Chitil	07799012748
Theodosios Dimitrakos	07978343434
Sally Fincher	07898347543
Virginia Franqueira	07711524511
Radu Grigore	07776351939
Marek Grzes	07928672498
Julio Hernandez-Castro	07717210073
Tim Hopkins	07792378464
Ozgur Kafali	07989634247
Stefan Kahrs	07798127475
Stephen Kell	07987234122
Andy King	07898374335
Julien Lange	07912685674
Rogerio de Lemos	07973542352
Shujun Li	07713467929
Stefan Marr	07987641123
Jason Nurse	07777712981
Dominic Orchard	07981279642
Carlos Perez Delgado	07772129875
Simon Thompson	07981237412
Charles Xavier	07927862449

Phone number of Jason Nurse?

qctxre

University of Kent
Institute of Cyber Security for Society (iCSS)

Telephone Directory

Easy to search by name (sorted, hence binary search);

Hard to search by number (unsorted, hence exhaustive search)

Budi Arief	07932731720
Adam Back	07896572341
David Barnes	07723812346
Mark Batty	07983184516
Laura Bocchi	07988762343
Howard Bowman	07981722451
Janet Carter	07986662211
David Chadwick	07766923565
Jacqueline Chetty	07898234553
Olaf Chitil	07799012748
Theodosios Dimitrakos	07978343434
Sally Fincher	07898347543
Virginia Franqueira	07711524511
Radu Grigore	07776351939
Marek Grzes	07928672498
Julio Hernandez-Castro	07717210073
Tim Hopkins	07792378464
Ozgur Kafali	07989634247
Stefan Kahrs	07798127475
Stephen Kell	07987234122
Andy King	07898374335
Julien Lange	07912685674
Rogerio de Lemos	07973542352
Shujun Li	07713467929
Stefan Marr	07987641123
Jason Nurse	07777712981
Dominic Orchard	07981279642
Carlos Perez Delgado	07772129875
Simon Thompson	07981237412
Charles Xavier	07927862449

Phone number of Jason Nurse? Easy!

qctxre

Telephone Directory

Easy to search by name (sorted, hence binary search);
Hard to search by number (unsorted, hence exhaustive search)

Budi Arief	07932731720
Adam Back	07896572341
David Barnes	07723812346
Mark Batty	07983184516
Laura Bocchi	07988762343
Howard Bowman	07981722451
Janet Carter	07986662211
David Chadwick	07766923565
Jacqueline Chetty	07898234553
Olaf Chitil	07799012748
Theodosios Dimitrakos	07978343434
Sally Fincher	07898347543
Virginia Franqueira	07711524511
Radu Grigore	07776351939
Marek Grzes	07928672498
Julio Hernandez-Castro	07717210073
Tim Hopkins	07792378464
Ozgur Kafali	07989634247
Stefan Kahrs	07798127475
Stephen Kell	07987234122
Andy King	07898374335
Julien Lange	07912685674
Rogerio de Lemos	07973542352
Shujun Li	07713467929
Stefan Marr	07987641123
Jason Nurse	07777712981
Dominic Orchard	07981279642
Carlos Perez Delgado	07772129875
Simon Thompson	07981237412
Charles Xavier	07927862449

Phone number of Jason Nurse? Easy!

Where is 07973542352?

qctxre

Telephone Directory

Easy to search by name (sorted, hence binary search);
Hard to search by number (unsorted, hence exhaustive search)

Budi Arief	07932731720
Adam Back	07896572341
David Barnes	07723812346
Mark Batty	07983184516
Laura Bocchi	07988762343
Howard Bowman	07981722451
Janet Carter	07986662211
David Chadwick	07766923565
Jacqueline Chetty	07898234553
Olaf Chitil	07799012748
Theodosios Dimitrakos	07978343434
Sally Fincher	07898347543
Virginia Franqueira	07711524511
Radu Grigore	07776351939
Marek Grzes	07928672498
Julio Hernandez-Castro	07717210073
Tim Hopkins	07792378464
Ozgur Kafali	07989634247
Stefan Kahrs	07798127475
Stephen Kell	07987234122
Andy King	07898374335
Julien Lange	07912685674
Rogerio de Lemos	07973542352
Shujun Li	07713467929
Stefan Marr	07987641123
Jason Nurse	07777712981
Dominic Orchard	07981279642
Carlos Perez Delgado	07772129875
Simon Thompson	07981237412
Charles Xavier	07927862449

Phone number of Jason Nurse? Easy!

Where is 07973542352?

qctxre

University of Kent Institute of Cyber Security for Society (iCSS)

Telephone Directory

Easy to search by name (sorted, hence binary search);
Hard to search by number (unsorted, hence exhaustive search)

Budi Arief	07932731720
Adam Back	07896572341
David Barnes	07723812346
Mark Batty	07983184516
Laura Bocchi	07988762343
Howard Bowman	07981722451
Janet Carter	07986662211
David Chadwick	07766923565
Jacqueline Chetty	07898234553
Olaf Chitil	07799012748
Theodosios Dimitrakos	07978343434
Sally Fincher	07898347543
Virginia Franqueira	07711524511
Radu Grigore	07776351939
Marek Grzes	07928672498
Julio Hernandez-Castro	07717210073
Tim Hopkins	07792378464
Ozgur Kafali	07989634247
Stefan Kahrs	07798127475
Stephen Kell	07987234122
Andy King	07898374335
Julien Lange	07912685674
Rogerio de Lemos	07973542352
Shujun Li	07713467929
Stefan Marr	07987641123
Jason Nurse	07777712981
Dominic Orchard	07981279642
Carlos Perez Delgado	07772129875
Simon Thompson	07981237412
Charles Xavier	07927862449

Phone number of Jason Nurse? Easy!

Where is 07973542352? Hard!