# Applications of hash functions

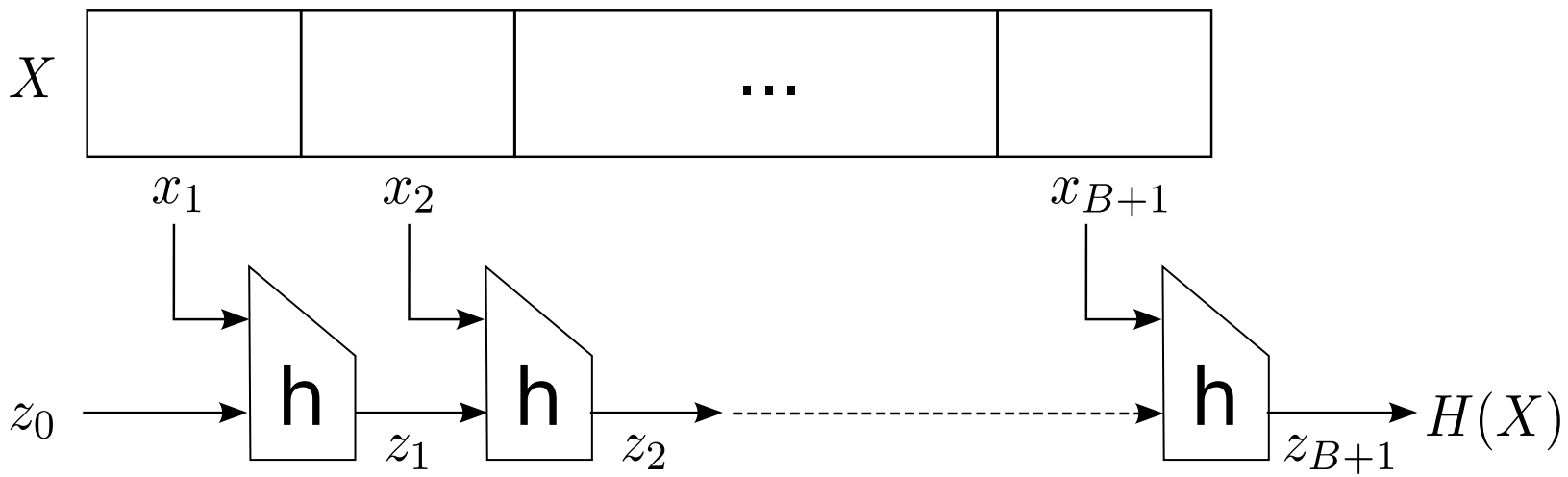
Diego F. Aranha

Institute of Computing
UNICAMP

# Iterated hash functions (Merkle-Damgård)

## Definition

It is a technique that allows constructing a hash function with infinte domain $H : \{0,1\}^* \rightarrow \{0,1\}^m$ through consecutive applications of a **compression function** $h : \{0,1\}^{m+t} \rightarrow \{0,1\}^m$. **Padding** is needed for adding block $x_{B+1}$ to an input $x$ with $B$ blocks.




Important: Collision resistance for $S$ is given by collision resistance for $h$.

# Message Authentication Codes (MACs)

## Construction

A MAC $y$ can be constructed by incorporating a cryptographic key $K$ in an iterated hash function $H$.

# Message Authentication Codes (MACs)

## Construction

A MAC $y$ can be constructed by incorporating a cryptographic key $K$ in an iterated hash function $H$.

First attempt:

- Assign $z_0 = K$ and compute MAC as $y = H_K(x)$.
- Suppose there is no pre- or postprocessing of message $x$.

# Message Authentication Codes (MACs)

## Construction

A MAC $y$ can be constructed by incorporating a cryptographic key $K$ in an iterated hash function $H$.

First attempt:

- Assign $z_0 = K$ and compute MAC as $y = H_K(x)$.
- Suppose there is no pre- or postprocessing of message $x$.
- It is possible to **forge** a MAC $y' = H_K(x||x')$.

# Message Authentication Codes (MACs)

## Construction

A MAC $y$ can be constructed by incorporating a cryptographic key $K$ in an iterated hash function $H$.

First attempt:

- Assign $z_0 = K$ and compute MAC as $y = H_K(x)$.
- Suppose there is no pre- or postprocessing of message $x$.
- It is possible to **forge** a MAC $y' = H_K(x||x')$.

Second attempt:

- Suppose that the message $x$ is now *padded*.
- Define $x' = x||pad(x)||w$, with arbitrary $w$.

# Message Authentication Codes (MACs)

## Construction

A MAC $y$ can be constructed by incorporating a cryptographic key $K$ in an iterated hash function $H$.

First attempt:

- Assign $z_0 = K$ and compute MAC as $y = H_K(x)$.
- Suppose there is no pre- or postprocessing of message $x$.
- It is possible to **forge** a MAC $y' = H_K(x||x')$.

Second attempt:

- Suppose that the message $x$ is now *padded*.
- Define $x' = x||pad(x)||w$, with arbitrary $w$.
- Still possible to **forge** MAC $y' = H_K(x'||pad(x'))$.

# Nested MACs

## Definition

A **nested MAC** is a composition of two cryptographic hash functions with key of the form $h_L(g_K(x))$, where $K, L$ are cryptographic keys and $x \in \mathcal{X}$.

Intuition: Combine short MAC with secure hash function!

# Nested MACs

**Definition**

A **nested MAC** is a composition of two cryptographic hash functions with key of the form $h_L(g_K(x))$, where $K, L$ are cryptographic keys and $x \in \mathcal{X}$.

Intuition: Combine short MAC with secure hash function!

Important: This construction is secure if $h$ is a secure MAC and $g$ is collision-resistant.

# HMAC

## Definition

$$ipad = \texttt{0x3636}\cdots 36$$
$$opad = \texttt{0x5C5C}\cdots 5C$$
$$y' = g_K(x) = h(K||x)$$
$$y = h_L(x) = h(L||y')$$

$$HMAC_K : H((K \oplus opad)||H((K \oplus ipad)||x))$$

Important: Secure if NMAC is secure. Still secure if SHA-1 is not collision resistant anymore?

# HMAC

## Definition

$$ipad = \texttt{0x3636} \cdots 36$$
$$opad = \texttt{0x5C5C} \cdots 5C$$
$$y' = g_K(x) = h(K||x)$$
$$y = h_L(x) = h(L||y')$$

$$HMAC_K : H((K \oplus opad)||H((K \oplus ipad)||x))$$

Important: Secure if pseudo-random function.

Important: Can be combined with a secure block cipher for authenticated encryption!

# Password storage

A classical application of cryptographic hash functions is to store $H(s)$ instead of password $s$!

Problem: Users tend to use the same passwords. Adversary with idle computing power can iterate lots of candidates for $s$ and "invert" $H$ from a large table of common inputs.

Solution: "Salt" cryptographic hashes with random bit strings (*salts*).

# Password storage

A classical application of cryptographic hash functions is to store $H(s)$ instead of password $s$!

Problem: Users tend to use the same passwords. Adversary with idle computing power can iterate lots of candidates for $s$ and "invert" $H$ from a large table of common inputs.

Solution: "Salt" cryptographic hashes with random bit strings (*salts*).

Problem: Compuational power provided by GPUs, FPGAs and ASICs!

Solution: Use sequential key derivation keys (PBKDF2, `bcrypt`) and/or high memory requirements (`scrypt`). If it is possible to store a key $K$ securely even in case of intrusion, compute $HMAC_K$ instead of $H$.