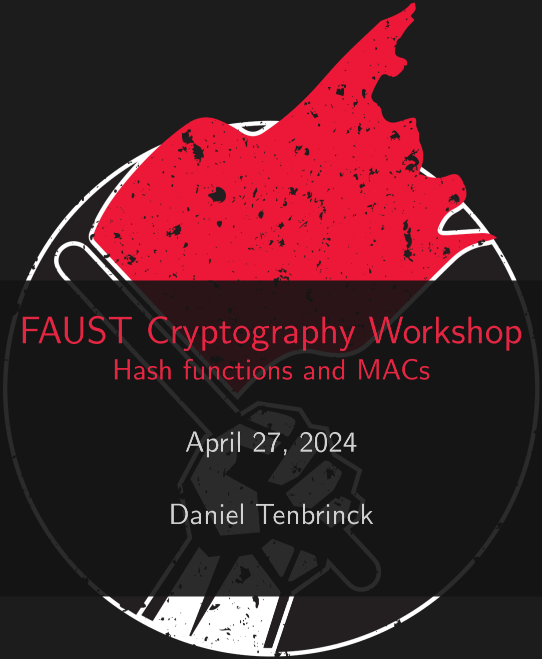


FAUST Cryptography Workshop

Hash functions and MACs

April 27, 2024

Daniel Tenbrinck

Outline

Introduction

Compression and hash functions

Message authentication codes (MACs)

Workshop challenges

Introduction

What is a hash?

The word **hash** is a little bit ambiguous...

What is a hash?

The word **hash** is a little bit ambiguous...

What is a hash?

The word **hash** is a little bit ambiguous...

This presentation is about cryptographic hash functions, *e.g.*, *MD5, SHA256, . . .*

What is a hash?

The word **hash** is a little bit ambiguous...

This presentation is about cryptographic hash functions, *e.g.*, *MD5*, *SHA256*, . . .

Example:

Hash("Your silly string could be here!") → $29c5963522fbf955f9\ldots$

Why do we need cryptographic hash functions?

Hash functions are an important tool for various information security applications, e.g.,

Why do we need cryptographic hash functions?

Hash functions are an important tool for various information security applications, e.g.,

- **data integrity validation**

  → data fingerprinting to check for modifications

  → checksum to detect data corruption

Why do we need cryptographic hash functions?

Hash functions are an important tool for various information security applications, e.g.,

- **data integrity validation**
  → data fingerprinting to check for modifications
  → checksum to detect data corruption
- **authenticity**
  → digital signatures, message authentication codes (MACs)
  → secure password storage

Motivation

Scenario:

Motivation

Scenario:

- Alice wants to send a **plain text message** m to Bob:

$$m := \textit{"Your new bridge is beautiful!"}$$

Motivation

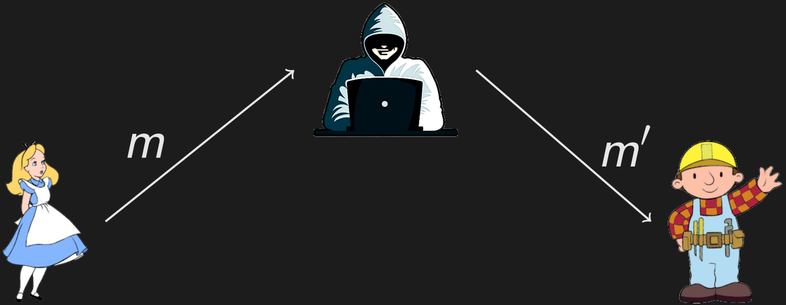
Scenario:

- Alice wants to send a **plain text message** *m* to Bob:

  $m :=$ *"Your new bridge is beautiful!"*

- Hacker performs **MITM attack** and alters the message to:

  $m' :=$ *"Your new bridge is ugly!"*



Compression and hash functions

Compression functions

Mathematical setting:

- let Σ be a finite set of characters encoding our messages
  - $\rightarrow$ *e.g., latin alphabet, hexadecimal encoding*
  - $\rightarrow$ often we assume a binary encoding, i.e., $\Sigma = \{0, 1\}$

Compression functions

Mathematical setting:

- let Σ be a finite set of characters encoding our messages
  $\rightarrow$ *e.g., latin alphabet, hexadecimal encoding*
  $\rightarrow$ often we assume a binary encoding, i.e., $\Sigma = \{0, 1\}$
- we define a **compression function** c as a map:

$$c \colon \Sigma^n \rightarrow \Sigma^k, \qquad n > k.$$

- that means we compress the information of a word $w \in \Sigma^n$ by compressing it to a smaller word $w' =: c(w) \in \Sigma^k$

Compression functions

Mathematical setting:

- let Σ be a finite set of characters encoding our messages
  → *e.g., latin alphabet, hexadecimal encoding*
  → often we assume a binary encoding, i.e., $\Sigma = \{0, 1\}$
- we define a **compression function** c as a map:

$$c\colon \Sigma^n \to \Sigma^k, \qquad n > k.$$

- that means we compress the information of a word $w \in \Sigma^n$ by compressing it to a smaller word $w' =: c(w) \in \Sigma^k$

Example: Binary checksum of words with length 4.

$$"0101" \;\to\; "0", \qquad "1011" \;\to\; "1"$$

Hash functions

Mathematical setting:

- we define a **hash function** h as a map:

$$h\colon \Sigma^* \to \Sigma^k$$

- that means we map words $w \in \Sigma^*$ **of variable size** to a word $w' =: h(w) \in \Sigma^k$ of fixed length

Example: Last byte of a word with variable length

$$"110101" \ \rightarrow \ "1", \qquad "100" \ \rightarrow \ "0"$$

Observations on compression/hash functions

Observation:
It becomes clear that both compression and hash functions are **not injective**, because they map a large set to a smaller set.
This inevitably leads to **collisions**, i.e., different words being mapped to the same value.

Example: Binary checksum of words with length 4.

$$"1101" \rightarrow "1", \qquad "1000" \rightarrow "1"$$

Cryptographic compression/hash functions

To use compression or hash functions for cryptography they have to fulfill certain criteria:

- computing a hash value $h(m)$ from a given message $m \in \Sigma^*$ is **efficient**
- finding collisions is numerically **unfeasible**

  → computing $m, m' \in \Sigma^*$ with $h(m) = h(m')$ impracticable
- generated hash values should be **pseudo-random**

  → small changes should lead to completely different values

Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Idea:

- **partition message** $m \in \Sigma^*$ in $N \in \mathbb{N}$ words, each of size n:

$$m \;=\; m_1 | m_2 | \ldots | m_N$$

Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Idea:

- **partition message** $m \in \Sigma^*$ in $N \in \mathbb{N}$ words, each of size n:

$$m \;=\; m_1|m_2|\ldots|m_N + p$$

- add appropriate padding p at the end
  → e.g., use zeros + binary encoding of message length $|m|$

Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Idea:

- **partition message** $m \in \Sigma^*$ in $N \in \mathbb{N}$ words, each of size n:

$$m \;=\; m_1|m_2|\ldots|m_N + p$$

- add appropriate padding p at the end

  → e.g., use zeros + binary encoding of message length $|m|$

- now compression function c can be applied to every block

Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Idea:

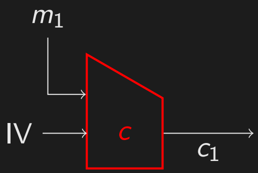
- choose initialization vector (IV), e.g., $\mathrm{IV} := 0^n$
- **successively apply** c to message block $m_i \in \Sigma^n$ combined with last result $c_{i-1} := c(m_{i-1})$, e.g., $c_i = m_i \oplus c_{i-1}$

Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Idea:

- choose initialization vector (IV), e.g., IV $:= 0^n$
- **successively apply** c to message block $m_i \in \Sigma^n$ combined with last result $c_{i-1} := c(m_{i-1})$, e.g., $c_i = m_i \oplus c_{i-1}$

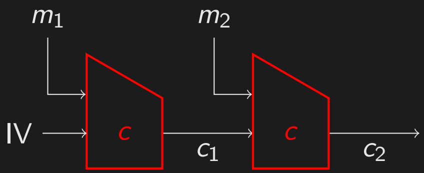


Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Idea:

- choose initialization vector (IV), e.g., IV $:= 0^n$
- **successively apply** c to message block $m_i \in \Sigma^n$ combined with last result $c_{i-1} := c(m_{i-1})$, e.g., $c_i = m_i \oplus c_{i-1}$

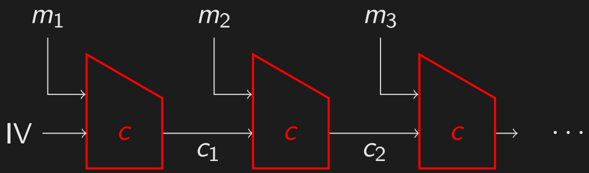


Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Idea:

- choose initialization vector (IV), e.g., $\text{IV} := 0^n$
- **successively apply** c to message block $m_i \in \Sigma^n$ combined with last result $c_{i-1} := c(m_{i-1})$, e.g., $c_i = m_i \oplus c_{i-1}$

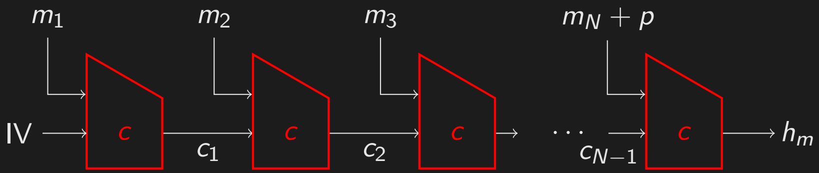


Merkle-Damgård construction

Question: Can we construct a hash function h from a cryptographic compression function $c\colon \Sigma^n \to \Sigma^k, n > k$?

Idea:

- choose initialization vector (IV), e.g., IV $:= 0^n$
- **successively apply** c to message block $m_i \in \Sigma^n$ combined with last result $c_{i-1} := c(m_{i-1})$, e.g., $c_i = m_i \oplus c_{i-1}$
- result of last block c_N defines **output of hash function** for message m, i.e., $h(m) = h_m := c_N$



Message + hash value

Scenario:

Message + hash value

Scenario:

- Alice sends a plain text message m together with its **hash value** $h_m =: h(m)$ to Bob:

$$m := \textit{"Your new bridge is beautiful!"}, \; h_m = e4689a1$$

Message + hash value

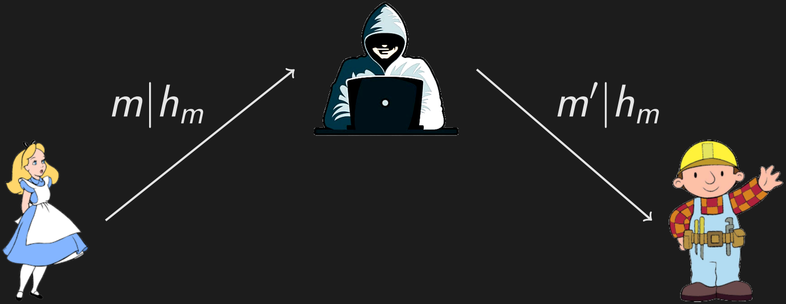
Scenario:

- Alice sends a plain text message m together with its **hash value** $h_m =: h(m)$ to Bob:

  $$m := \text{"}\textit{Your new bridge is beautiful!}\text{"},\ h_m = e4689a1$$

- Hacker performs **MITM attack** and alters the message to:

  $$m' := \text{"}\textit{Your new bridge is ugly!}\text{"}$$

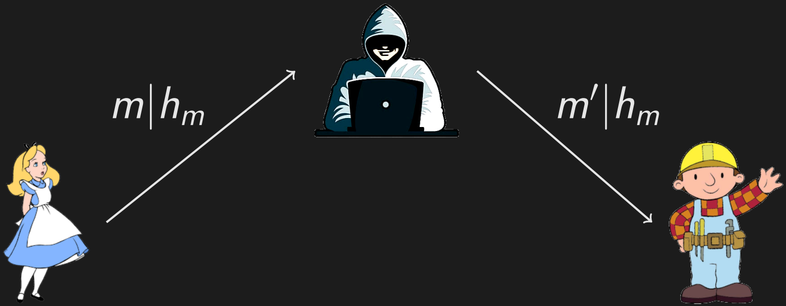


Message + hash value

Scenario:

- Bob receives the message m' and **computes its hash value** $h_{m'} =: h(m')$ as:

$$m' := \text{"}\textit{Your new bridge is ugly!}\text{"},\ h'_m = 54c8b30$$



Message + hash value

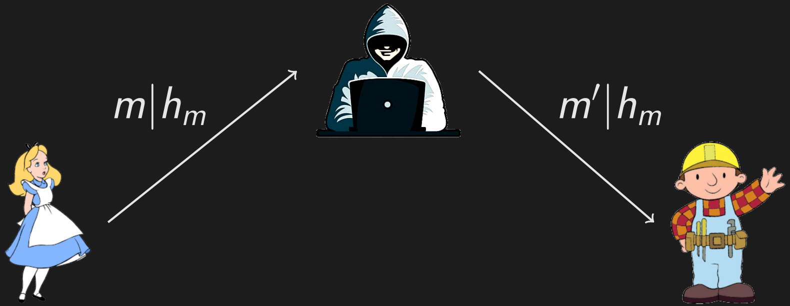
Scenario:

- Bob receives the message m' and **computes its hash value** $h_{m'} =: h(m')$ as:

$$m' := "Your\ new\ bridge\ is\ ugly!",\ h'_m = 54c8b30$$

- Bob realizes the message has been modified because:

$$h_m \ = \ e4689a1 \ \neq \ 54c8b30 \ = \ h_{m'}$$



Message + hash value

Question: Are Alice and Bob now safe from the hacker?

Message + hash value

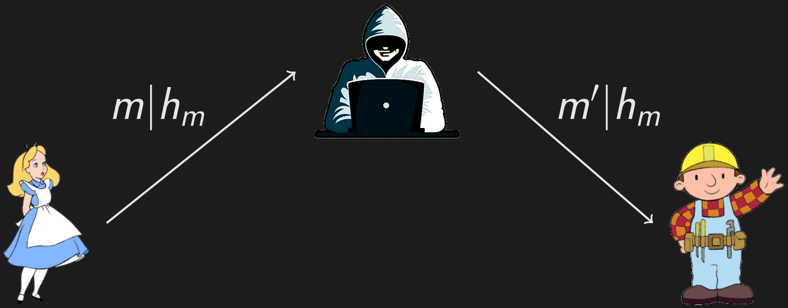
Question: Are Alice and Bob now safe from the hacker?

Answer: No, the hacker can modify the hash value as well.

Message + hash value

Question: Are Alice and Bob now safe from the hacker?

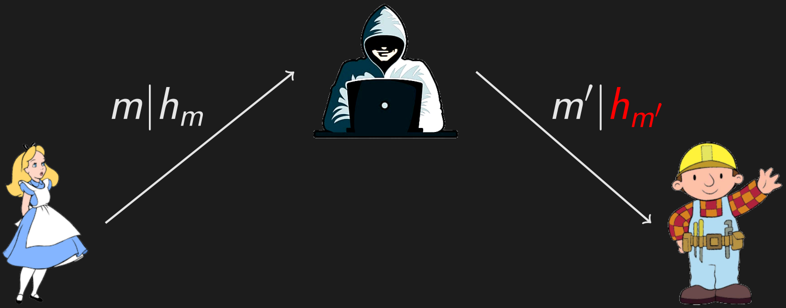
Answer: No, the hacker can modify the hash value as well.



Message + hash value

Question: Are Alice and Bob now safe from the hacker?

Answer: No, the hacker can modify the hash value as well.



Message authentication codes (MACs)

Cryptographic hash functions for authentication

Question:
How can Bob know that the message is really from Alice?

Cryptographic hash functions for authentication

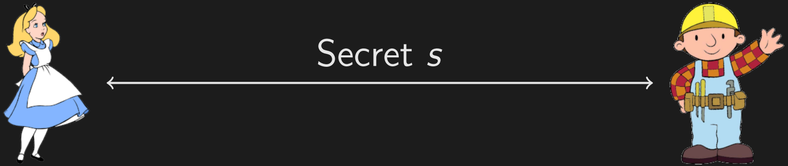
Question:
How can Bob know that the message is really from Alice?

Idea: We need to **add some secret** s to the hash function that only Alice and Bob know.

Cryptographic hash functions for authentication

Question:
How can Bob know that the message is really from Alice?

Idea: We need to **add some secret** s to the hash function that only Alice and Bob know.



Message authentication codes (MACs)

Mathematical setting:

- we define a **parametrized hash function** h^s as a map:

$$h^s\colon S \times \Sigma^* \to \Sigma^k$$

- here S is a key space from which we can choose a secret key s

Message authentication codes (MACs)

Mathematical setting:

- we define a **parametrized hash function** h^s as a map:

$$h^s \colon S \times \Sigma^* \to \Sigma^k$$

- here S is a key space from which we can choose a secret key s
- since the key s is unknown to externals h^s is called message authentication code (MAC)

Message authentication codes (MACs)

Mathematical setting:

- we define a **parametrized hash function** h^s as a map:

$$h^s\colon S \times \Sigma^* \to \Sigma^k$$

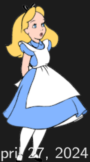
- here S is a key space from which we can choose a secret key s
- since the key s is unknown to externals h^s is called message authentication code (MAC)

Example: Secret key $s \in S$ is prepended to the message $m \in \Sigma^*$ prior to computing a hash value via h, i.e.,

$$h^s(m) \ := \ h(s|m) \ = \ h^s_m$$

Message + MAC

Scenario:



Message + MAC

Scenario:

- Alice sends a plain text message m together with its **MAC** using the secret $s \in S$ as $h_m^s =: h^s(m)$ to Bob:

$$m := \text{"Your new bridge is beautiful!"},\ h_m^s = fa461b$$

Message + MAC

Scenario:

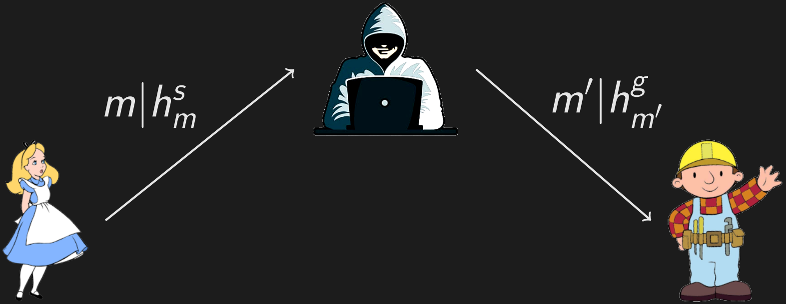
- Alice sends a plain text message m together with its **MAC** using the secret $s \in S$ as $h^s_m =: h^s(m)$ to Bob:
  $m :=$ *"Your new bridge is beautiful!"*, $h^s_m = fa461b$
- Hacker performs **MITM attack** and alters the message to:
  $m' :=$ *"Your new bridge is ugly!"*
- Hacker doesn't know the secret $s \in S$ and guesses $g \in S$ generating the MAC $h^g_{m'} = 40afde$

Message + MAC

Scenario:

- Bob receives the message m' and **computes its MAC** using the secret $s \in S$ as $h^s_{m'} =: h^s(m')$:

$$m' := \text{"}\textit{Your new bridge is ugly!}\text{"},\ h^s_{m'} = 34da47$$



$m|h^s_m$

$m'|h^g_{m'}$

Message + MAC

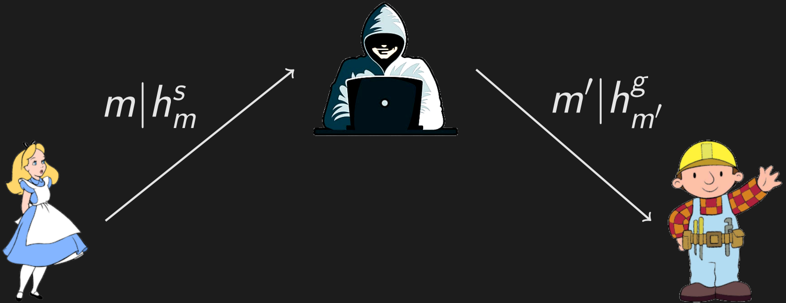
Scenario:

- Bob receives the message m' and **computes its MAC** using the secret $s \in S$ as $h^s_{m'} =: h^s(m')$:

$$m' := \textit{"Your new bridge is ugly!"},\; h^s_{m'} = 34da47$$

- Bob realizes the message was not sent from Alice because:

$$h^s_{m'} \;=\; 34da47 \;\neq\; 40afde \;=\; h^g_{m'}$$



Length extension attack

Question: Are Alice and Bob now safe from the hacker?

Length extension attack

Question: Are Alice and Bob now safe from the hacker?

Answer: Unfortunately not, if:

1. the hash function in the used MAC is based on the Merkle-Damgård construction
2. the secret is prepended to the message

Length extension attack

Question: Are Alice and Bob now safe from the hacker?

Answer: Unfortunately not, if:

1. the hash function in the used MAC is based on the Merkle-Damgård construction
2. the secret is prepended to the message

Then, the hacker can perform a **length extension attack** and forge a message with valid MAC without knowing the secret.

Length extension attack

Assumptions:

- h^s is a MAC that prepends the secret $s \in S$ and has a known Merkle-Damgård hash function and padding p
- $h_m^s := h(s|m|p)$ is MAC for original message m

Length extension attack

Assumptions:

- h^s is a MAC that prepends the secret $s \in S$ and has a known Merkle-Damgård hash function and padding p
- $h^s_m := h(s|m|p)$ is MAC for original message m

Goal: Forge a valid MAC for an extended message $\overline{m} := m|e$

Length extension attack

Assumptions:

- h^s is a MAC that prepends the secret $s \in S$ and has a known Merkle-Damgård hash function and padding p
- $h^s_m := h(s|m|p)$ is MAC for original message m

Goal: Forge a valid MAC for an extended message $\overline{m} := m|e$

Example:

m: `amount=1000&receiver=bob`

$$h^s_m = h(s|m|p) = 7b2f60$$

$\overline{m}$: `amount=1000&receiver=bobby`

Length extension attack

Question: How to forge a valid MAC for the extended message $\overline{m}$?

Length extension attack

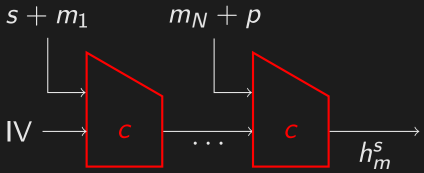
Question: How to forge a valid MAC for the extended message $\overline{m}$?

- s is included in observed MAC h_m^s → use h_m^s as input for compression function c in Merkle-Damgård hash function

Length extension attack

Question: How to forge a valid MAC for the extended message $\overline{m}$?

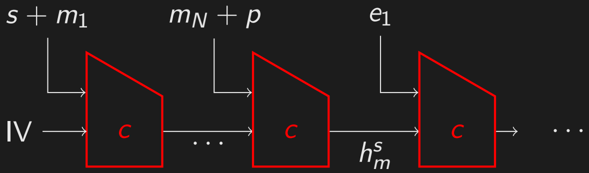
- s is included in observed MAC h_m^s → use h_m^s as input for compression function c in Merkle-Damgård hash function



Length extension attack

Question: How to forge a valid MAC for the extended message $\overline{m}$?

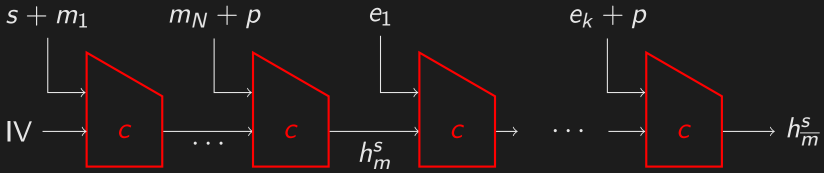
- s is included in observed MAC h_m^s → use h_m^s as input for compression function c in Merkle-Damgård hash function



Length extension attack

Question: How to forge a valid MAC for the extended message $\overline{m}$?

- s is included in observed MAC h_m^s → use h_m^s as input for compression function c in Merkle-Damgård hash function

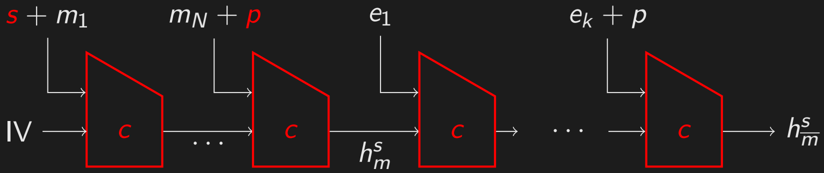


Length extension attack

Question: How to forge a valid MAC for the extended message $\overline{m}$?

- s is included in observed MAC h^s_m → use h^s_m as input for compression function c in Merkle-Damgård hash function
- $h^s_m = h(s|m|p)$, but we don't know length of s and p:

$$h^s_m \; = \; h(s|m_1|m_2|\cdots|m_N|p)$$



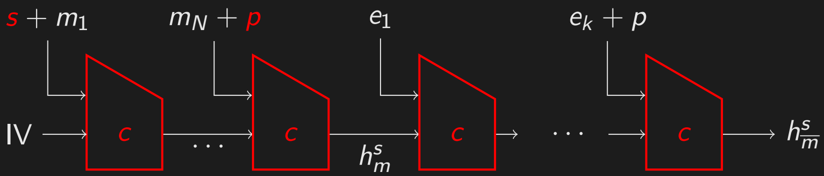
Length extension attack

Question: How to forge a valid MAC for the extended message $\overline{m}$?

- s is included in observed MAC h_m^s → use h_m^s as input for compression function c in Merkle-Damgård hash function
- $h_m^s = h(s|m|p)$, but we don't know length of s and p:

$$h_m^s = h(s|m_1|m_2|\cdots|m_N|p)$$

- use **brute-force** to guess needed padding $\overline{p}$ so that $\overline{m} = m|\overline{p}|e$ generates a valid MAC $h_{\overline{m}}^s$



HMAC

Question: How to protect MACs from length extension attacks?

HMAC

Question: How to protect MACs from length extension attacks?

Idea: Combine secret s and message m via a hash function h in a more sophisticated way to compute a **hash-based message authentication code (HMAC)**.

$$\mathrm{HMAC}(s, m) \;:=\; h\left[s \oplus \mathrm{opad} \,|\, h(s \oplus \mathrm{ipad} \,|\, m)\right]$$

HMAC

Question: How to protect MACs from length extension attacks?

Idea: Combine secret s and message m via a hash function h in a more sophisticated way to compute a **hash-based message authentication code (HMAC)**.

$$\text{HMAC}(s, m) \;:=\; h\left[s \oplus \text{opad} \,|\, h(s \oplus \text{ipad} \,|\, m)\right]$$

- ipad and opad are paddings with two different byte constants, e.g., ipad $= 0x5c \ldots 0x5c$, opad $= 0x36 \ldots 0x36$

HMAC

Question: How to protect MACs from length extension attacks?

Idea: Combine secret s and message m via a hash function h in a more sophisticated way to compute a **hash-based message authentication code (HMAC)**.

$$\mathrm{HMAC}(s, m) \;:=\; h\left[s \oplus \mathrm{opad} \,|\, h(s \oplus \mathrm{ipad} \,|\, m)\right]$$

- ipad and opad are paddings with two different byte constants, e.g., ipad $= 0x5c \dots 0x5c$, opad $= 0x36 \dots 0x36$
- works with any cryptographic hash function h

HMAC

Question: How to protect MACs from length extension attacks?

Idea: Combine secret s and message m via a hash function h in a more sophisticated way to compute a **hash-based message authentication code (HMAC)**.

$$\text{HMAC}(s, m) \ := \ h\left[s \oplus \text{opad} \,|\, h(s \oplus \text{ipad} \,|\, m)\right]$$

- ipad and opad are paddings with two different byte constants, e.g., ipad $= 0x5c \ldots 0x5c$, opad $= 0x36 \ldots 0x36$
- works with any cryptographic hash function h
- proposed in RFC2104

MAC composition

Observations:

- hash functions and MACs are not supposed to hide the content of plain text message m a-priori

MAC composition

Observations:

- hash functions and MACs are not supposed to hide the content of plain text message m a-priori
- different ways to combine MAC h^s with encrypted message e_m:

MAC composition

Observations:

- hash functions and MACs are not supposed to hide the content of plain text message m a-priori
- different ways to combine MAC h^s with encrypted message e_m:

$$\text{Encrypt-and-MAC:} \qquad e(m) \mid h^s(m)$$

MAC composition

Observations:

- hash functions and MACs are not supposed to hide the content of plain text message m a-priori
- different ways to combine MAC h^s with encrypted message e_m:

Encrypt-and-MAC: $e(m) \mid h^s(m)$

MAC-then-Encrypt: $e[\, m \mid h^s(m) \,]$

MAC composition

Observations:

- hash functions and MACs are not supposed to hide the content of plain text message m a-priori
- different ways to combine MAC h^s with encrypted message e_m:

Encrypt-and-MAC:	$e(m) \mid h^s(m)$
MAC-then-Encrypt:	$e[\, m \mid h^s(m)\,]$
Encrypt-then-MAC:	$e(m) \mid h^s(e(m))$

MAC composition

Observations:

- hash functions and MACs are not supposed to hide the content of plain text message m a-priori
- different ways to combine MAC h^s with encrypted message e_m:

Encrypt-and-MAC: $e(m) \mid h^s(m)$

MAC-then-Encrypt: $e[\, m \mid h^s(m)\,]$

Encrypt-then-MAC: $e(m) \mid h^s(e(m))$

- only **last composition** is safe!

MAC composition

Observations:

- hash functions and MACs are not supposed to hide the content of plain text message m a-priori
- different ways to combine MAC h^s with encrypted message e_m:

Encrypt-and-MAC:	$e(m) \mid h^s(m)$
MAC-then-Encrypt:	$e[\, m \mid h^s(m) \,]$
Encrypt-then-MAC:	$e(m) \mid h^s(e(m))$

- only **last composition** is safe!

Example: message m, MAC h^s, ciphertexts $e_1(m), e_2(m)$

- although $e_1(m) \neq e_2(m)$, the MAC $h^s(m)$ is equal!
- allows to correlate message content

Workshop challenges

Hands-on: Hash function challenges

Length extension attack

- authenticate as user `Administrator` to get the flag
- use length extension attack to forge a valid login token
- think about the padding!
- used hash function is SHA256

Hands-on: Hash function challenges

Length extension attack

- authenticate as user `Administrator` to get the flag
- use length extension attack to forge a valid login token
- think about the padding!
- used hash function is SHA256

Unhiding MAC

- all communication is encrypted in this service
- look at the source code!
- server computes HMAC of plaintext, then concatenates with ciphertext
- secret for HMAC is not known
- deduce information from the MACs sent by the server to win flag

How to compute hash values in Python

Example for SHA256:

```
>>> import hashlib

>>> hashlib.sha256(b"Data as bytestring").digest()
b'\x0f\xff\xba\x14\x05$5\xc8\xaf\xed6$=\xd5\xb9w\xda
            \xc1@\xfa\xaf>\xdfL\\\x0b\x0e\xcf\x04\x89VR'

>>> hashlib.sha256(b"Data as bytestring").hexdigest()
'0fffba14052435c8afed36243dd5b977dac140faaf3edf4c5c0b0ecf04895652'
```

Let's goooooo!!!

Get started:

- Hash function challenges:
  `https://workshop.faust.ninja/challenges`
- Presentation slides:
  `https://www.studon.fau.de/crs5693797.html`

If you are stuck: **Ask us any time!**

Resources on cryptographic hash functions

Links to useful websites with more information:

- Merkle-Damgard construction
- Information on SHA-2 hash functions
- Padding in cryptography
- Message authentication code
- HMACs
- Length extension attack
- Stack overflow discussion on MAC composition
- The Cryptographic Doom Principle