# Cryptography

# Symmetric authentication


Léo Colisson Palais
Master CySec UGA

leo.colisson-palais@univ-grenoble-alpes.fr

https://leo.colisson.me/teaching/

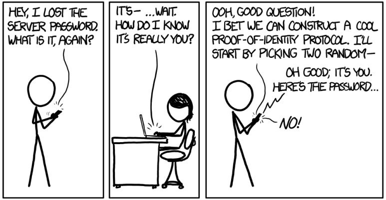
# Authentication: Motivations

# Authentication: Motivations

Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:

- Proving who you are is often very important:

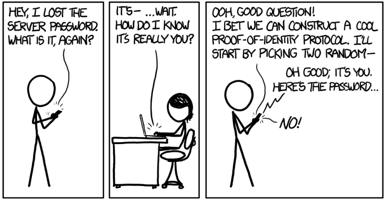


# Authentication: Motivations

Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:

- Proving who you are is often very important:
  - Accessing your bank account. . . : passwords are not practical/sufficient = two-factor authentication

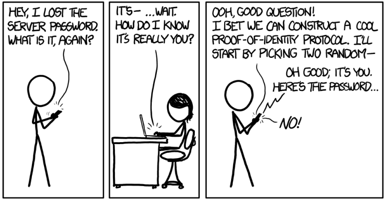


# Authentication: Motivations

Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:

- Proving who you are is often very important:
  - Accessing your bank account. . . : passwords are not practical/sufficient = two-factor authentication
  - . . . and being sure that you are really talking to your bank!

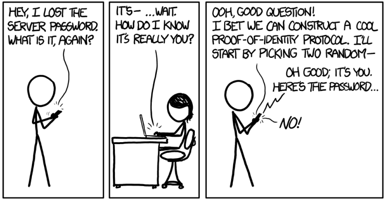


# Authentication: Motivations

Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:

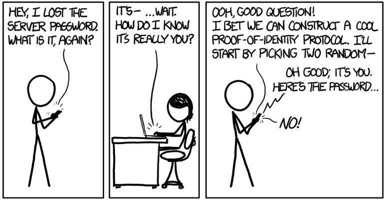
- Proving who you are is often very important:
  - Accessing your bank account. . . : passwords are not practical/sufficient = two-factor authentication
  - . . . and being sure that you are really talking to your bank!
  - Opening a car/door/access gate/. . .



Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:

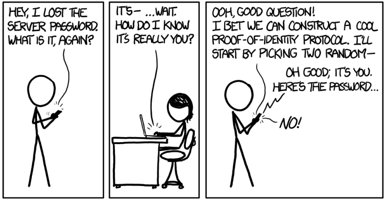
- Proving who you are is often very important:
  - Accessing your bank account. . . : passwords are not practical/sufficient = two-factor authentication
  - . . . and being sure that you are really talking to your bank!
  - Opening a car/door/access gate/. . .
- Storing data with malicious parties



# Authentication: Motivations

Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:

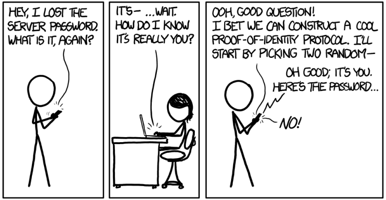



- Proving who you are is often very important:
  - Accessing your bank account. . . : passwords are not practical/sufficient = two-factor authentication
  - . . . and being sure that you are really talking to your bank!
  - Opening a car/door/access gate/. . .
- Storing data with malicious parties
- “Stateless” server: JSON Web Token (JWT)

# Authentication: Motivations

Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:

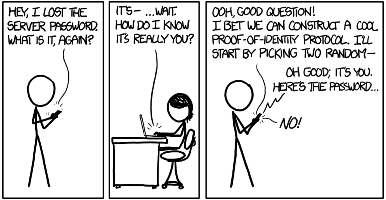



- Proving who you are is often very important:
  - Accessing your bank account. . .: passwords are not practical/sufficient = two-factor authentication
  - . . . and being sure that you are really talking to your bank!
  - Opening a car/door/access gate/. . .
- Storing data with malicious parties
- "Stateless" server: JSON Web Token (JWT)
- Avoid denial-of-service attacks in TCP: SYN cookies

# Authentication: Motivations

Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:

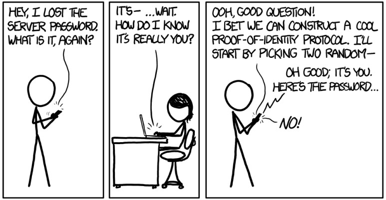



- Proving who you are is often very important:
  - Accessing your bank account. . . : passwords are not practical/sufficient = two-factor authentication
  - . . . and being sure that you are really talking to your bank!
  - Opening a car/door/access gate/. . .
- Storing data with malicious parties
- "Stateless" server: JSON Web Token (JWT)
- Avoid denial-of-service attacks in TCP: SYN cookies
- Blockchain = signing an authorization to transfer money

# Authentication: Motivations

Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:

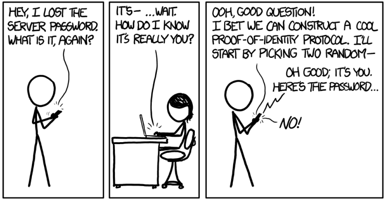



- Proving who you are is often very important:
  - Accessing your bank account. . . : passwords are not practical/sufficient = two-factor authentication
  - . . . and being sure that you are really talking to your bank!
  - Opening a car/door/access gate/. . .
- Storing data with malicious parties
- “Stateless” server: JSON Web Token (JWT)
- Avoid denial-of-service attacks in TCP: SYN cookies
- Blockchain = signing an authorization to transfer money
- Avoid “man-in-the-middle” attacks (MITM)

# Authentication: Motivations

Authentication/signature = ensuring that we **are talking to the right person**

**Motivations**:




- Proving who you are is often very important:
  - Accessing your bank account. . . : passwords are not practical/sufficient = two-factor authentication
  - . . . and being sure that you are really talking to your bank!
  - Opening a car/door/access gate/. . .
- Storing data with malicious parties
- “Stateless” server: JSON Web Token (JWT)
- Avoid denial-of-service attacks in TCP: SYN cookies
- Blockchain = signing an authorization to transfer money
- Avoid “man-in-the-middle” attacks (MITM)
- Avoid padding oracle attacks $\Rightarrow$ achieve IND-CCA security = security against **active adversaries**

# Authentication

Like encryption, two main families:

**Private key** (symmetric)
= **Message Authentication Code (MAC)**

The verifier of the signature must first share a private key with the signer

**Public key** (asymmetric)
= **signature**

The verifier of the signature must know the signer's public key

# Authentication

Focus of the course

Like encryption, two main families:

**Private key** (symmetric)
= **Message Authentication Code (MAC)**

The verifier of the signature must first share a private key with the signer

**Public key** (asymmetric)
= **signature**

The verifier of the signature must know the signer's public key

# Message Authentication Code (MAC)

## Message Authentication Code (MAC)

A message authentication code (*MAC*) for a message space $\mathcal{M}$ consists of two algorithms:

- $\mathsf{Gen}(1^\lambda)$, **which outputs a secret key** $k$
- $\mathsf{MAC}(k, m)$, **a deterministic algorithm that takes as input a key** $k$ **and a message** $m \in \mathcal{M}$ **and returns a tag (acting as a signature)**

? How can we verify whether a tag $t$ really authenticates the message $m$?

# Message Authentication Code (MAC)

**Message Authentication Code (MAC)**

A message authentication code (*MAC*) for a message space $\mathcal{M}$ consists of two algorithms:

- $\mathsf{Gen}(1^\lambda)$, **which outputs a secret key** $k$
- $\mathsf{MAC}(k, m)$, **a deterministic algorithm that takes as input a key** $k$ **and a message** $m \in \mathcal{M}$ **and returns a tag (acting as a signature)**

? How can we verify whether a tag $t$ really authenticates the message $m$?

MAC is deterministic, so compute $\mathsf{MAC}(k, m) \stackrel{?}{=} t$!

# Message Authentication Code (MAC)

**Disclaimer** : I will often tend to talk about a signature instead of a tag, because it is morally the same thing except for the private/public key distinction.

# MAC: security definitions

# How to formalize security?

Intuitively, security means it is hard to generate a valid tag without knowing the key $k$.

**How can we formalize this idea?**

# How to formalize security?

**Step 1**: how to formalize "**hard to find** X"?

Is the following true:

$$\begin{array}{|l|}\hline \mathcal{L}_{\text{guess-r}} \\ \hline r \leftarrow \mathsf{Gen}(1^\lambda) \\ \underline{\text{GUESS}(x):} \\ \quad \textbf{return } x \stackrel{?}{=} x \\ \hline \end{array} \approx \begin{array}{|l|}\hline \mathcal{L}_{\text{guess-false}} \\ \hline \underline{\text{GUESS}(x):} \\ \quad \textbf{return } \texttt{false} \\ \hline \end{array}$$

?

- A No
- B Yes, but we could have used $\equiv$
- C Yes, and $\approx$ is the right symbol

# How to formalize security?

**Step 1**: how to formalize "**hard to find** X"?

Is the following true:

$$\begin{array}{|l|}\hline \mathcal{L}_{\text{guess-r}} \\ \hline r \leftarrow \text{Gen}(1^\lambda) \\ \underline{\text{GUESS}(x)\text{:}} \\ \quad \textbf{return } x \stackrel{?}{=} x \\ \hline \end{array} \approx \begin{array}{|l|}\hline \mathcal{L}_{\text{guess-false}} \\ \hline \underline{\text{GUESS}(x)\text{:}} \\ \quad \textbf{return } \texttt{false} \\ \hline \end{array}$$

?

- (A) No
- (B) Yes, but we could have used $\equiv$
- (C) Yes, and $\approx$ is the right symbol ✓ because **it is hard to find** $x$, but there is a negligible chance ($\frac{1}{2^\lambda}$) to find it

# How to formalize security?

**Step 2**: how to formalize "hard to find **a valid tag**"?

?

First attempt:

$$\begin{array}{|l|}\hline \mathcal{L}_{\text{mac-1}} \\ \hline r \leftarrow \text{Gen}(1^\lambda) \\ \underline{\text{CHECKTAG}(m,t):} \\ \quad \textbf{return } \text{MAC}(k,m) \stackrel{?}{=} t \\ \hline \end{array} \approx \begin{array}{|l|}\hline \mathcal{L}_{\text{false}} \\ \hline \underline{\text{CHECKTAG}(m,t):} \\ \quad \textbf{return } \texttt{false} \\ \hline \end{array}$$

Is this a good idea?

- (A) No, because one can always distinguish these libraries
- (B) No, because this definition is not generic enough
- (C) Yes

# How to formalize security?

**Step 2**: how to formalize "hard to find **a valid tag**"?

First attempt:

$$\begin{array}{|l|}\hline \mathcal{L}_{\text{mac-1}} \\ \hline r \leftarrow \mathsf{Gen}(1^\lambda) \\ \underline{\mathsf{CHECKTAG}(m,t)\text{:}} \\ \quad \textbf{return } \mathsf{MAC}(k,m) \stackrel{?}{=} t \\ \hline \end{array} \approx \begin{array}{|l|}\hline \mathcal{L}_{\text{false}} \\ \hline \underline{\mathsf{CHECKTAG}(m,t)\text{:}} \\ \quad \textbf{return } \texttt{false} \\ \hline \end{array}$$

?

Is this a good idea?

- (A) No, because one can always distinguish these libraries
- (B) No, because this definition is not generic enough

  ✓ In real life, an attacker will see valid tags!! They therefore have more information than here. For example, $\mathsf{MAC}(t,x) := (t,x)$ would be secure under this definition, but in reality this is not considered secure because seeing a single "signature" (tag) would allow signing any message!
- (C) Yes

# How to formalize security?

**Step 2**: how to formalize "hard to find **a valid tag**"?

?

Second attempt:

| $\mathcal{L}_{\text{mac-1}}$ |
|---|
| $r \leftarrow \mathsf{Gen}(1^\lambda)$ |
| $\underline{\mathsf{GETTAG}(m):}$ |
|   **return** $\mathsf{MAC}(k, m)$ |
| $\underline{\mathsf{CHECKTAG}(m, t):}$ |
|   **return** $\mathsf{MAC}(k, m) \stackrel{?}{=} t$ |

$\approx$

| $\mathcal{L}_{\text{mac-1-false}}$ |
|---|
| $r \leftarrow \mathsf{Gen}(1^\lambda)$ |
| $\underline{\mathsf{GETTAG}(m):}$ |
|   **return** $\mathsf{MAC}(k, m)$ |
| $\underline{\mathsf{CHECKTAG}(m, t):}$ |
|   **return** `false` |

Is this a good idea?

- (A) No, because one can always distinguish these libraries
- (B) No, because this definition is not generic enough
- (C) Yes

# How to formalize security?

**Step 2**: how to formalize "hard to find **a valid tag**"?

?

Second attempt:

| $\mathcal{L}_{\text{mac-1}}$ |
|---|
| $r \leftarrow \text{Gen}(1^\lambda)$ |
| $\underline{\text{GETTAG}(m)\text{:}}$ |
| $\quad$ **return** $\text{MAC}(k, m)$ |
| $\underline{\text{CHECKTAG}(m, t)\text{:}}$ |
| $\quad$ **return** $\text{MAC}(k, m) \stackrel{?}{=} t$ |

$\approx$

| $\mathcal{L}_{\text{mac-1-false}}$ |
|---|
| $r \leftarrow \text{Gen}(1^\lambda)$ |
| $\underline{\text{GETTAG}(m)\text{:}}$ |
| $\quad$ **return** $\text{MAC}(k, m)$ |
| $\underline{\text{CHECKTAG}(m, t)\text{:}}$ |
| $\quad$ **return** `false` |

Is this a good idea?

- (A) No, because one can always distinguish these libraries
  Yes Try to find an attack (exercise Caseine "MAC > MAC > MAC bad definition")
- (B) No, because this definition is not generic enough
- (C) Yes

# How to formalize security?

**Step 2**: how to formalize "hard to find **a valid tag**"?

Second attempt:

| $\mathcal{L}_{\text{mac-1}}$ | | $\mathcal{L}_{\text{mac-1-false}}$ |
|---|---|---|
| $r \leftarrow \mathsf{Gen}(1^\lambda)$<br>$\underline{\text{GETTAG}(m):}$<br>  $\textbf{return } \mathsf{MAC}(k, m)$<br>$\underline{\text{CHECKTAG}(m, t):}$<br>  $\textbf{return } \mathsf{MAC}(k, m) \stackrel{?}{=} t$ | $\approx$ | $r \leftarrow \mathsf{Gen}(1^\lambda)$<br>$\underline{\text{GETTAG}(m):}$<br>  $\textbf{return } \mathsf{MAC}(k, m)$<br>$\underline{\text{CHECKTAG}(m, t):}$<br>  $\textbf{return}$ `false` |

?

Is this a good idea?

- (A) No, because one can always distinguish these libraries
  Yes Try to find an attack (exercise Caseine "MAC > MAC > MAC bad definition") $\Rightarrow$ Solution: CHECKTAG("hello", GETTAG("hello"))
- (B) No, because this definition is not generic enough
- (C) Yes

# How to formalize security?

**Step 2**: how to formalize "hard to find **a valid tag**"?
Second attempt:

$$\begin{array}{|l|}\hline \mathcal{L}_{\text{mac-1-real}} \\ \hline r \leftarrow \mathsf{Gen}(1^\lambda) \\ \underline{\mathsf{GETTAG}(m):} \\ \quad \textbf{return } \mathsf{MAC}(k,m) \\ \underline{\mathsf{CHECKTAG}(m,t):} \\ \quad \textbf{return } \mathsf{MAC}(k,m) \stackrel{?}{=} t \\ \hline \end{array} \approx \begin{array}{|l|}\hline \mathcal{L}_{\text{mac-1-fake}} \\ \hline r \leftarrow \mathsf{Gen}(1^\lambda) \\ \underline{\mathsf{GETTAG}(m):} \\ \quad \textbf{return } \mathsf{MAC}(k,m) \\ \underline{\mathsf{CHECKTAG}(m,t):} \\ \quad \textbf{return } \texttt{false} \\ \hline \end{array} = \text{Too paranoid} \tag{1}$$

It is not an attack if the only thing one can "sign" is by copying/pasting existing signatures! (but beware, replay attacks can be problematic in practice, though they cannot be solved at this level)

# How to formalize security?

Third (and final) attempt: we win if we manage to generate a TAG **never seen before**:

## Definition (EUF-CMA-)

A MAC $(\mathsf{Gen}, \mathsf{MAC})$ is said to be **strongly EUF-CMA-secure** (existentially unforgeable under chosen-message attacks) if:

$$
\boxed{\begin{array}{l}
\multicolumn{1}{c}{\mathcal{L}_{\text{mac-real}}} \\
\hline
r \leftarrow \mathsf{Gen}(1^\lambda) \\
\underline{\text{GETTAG}(m):} \\
\quad \textbf{return } \mathsf{MAC}(k, m) \\
\underline{\text{CHECKTAG}(m, t):} \\
\quad \textbf{return } \mathsf{MAC}(k, m) \stackrel{?}{=} t
\end{array}}
\approx
\boxed{\begin{array}{l}
\multicolumn{1}{c}{\mathcal{L}_{\text{mac-fake}}} \\
\hline
r \leftarrow \mathsf{Gen}(1^\lambda) \\
\mathcal{T} := \emptyset \\
\underline{\text{GETTAG}(m):} \\
\quad t := \mathsf{MAC}(k, m) \\
\quad \mathcal{T} := \mathcal{T} \cup \{(m, t)\} \\
\quad \textbf{return } t \\
\underline{\text{CHECKTAG}(m, t):} \\
\quad \textbf{return } (m, t) \in \mathcal{T}
\end{array}}
$$

Note: for non-strong EUF-CMA security, we simply replace $(m, t) \in \mathcal{T}$ by $\exists t, (m, t) \in \mathcal{T}$, i.e. to win one must generate a tag for a **different message**.

# How to formalize security?

?

Caseine exercise (MAC > MAC (quiz) > "MAC OTP security"). Let $\mathcal{M} = \{\mathtt{0},\mathtt{1}\}^\lambda$, $\mathsf{Gen}(1^\lambda) := r \xleftarrow{\$} \{\mathtt{0},\mathtt{1}\}^\lambda; \textbf{return } r$ and $\mathsf{MAC}(k,m) := k \oplus m$. Is this a secure MAC? If yes, prove it; otherwise, find an attack. Reminder of the definition:

| $\mathcal{L}_{\text{mac-real}}$ |
|---|
| $r \leftarrow \mathsf{Gen}(1^\lambda)$ |
| $\underline{\mathsf{GETTAG}(m)}$: |
| $\quad \textbf{return } \mathsf{MAC}(k,m)$ |
| $\underline{\mathsf{CHECKTAG}(m,t)}$: |
| $\quad \textbf{return } \mathsf{MAC}(k,m) \stackrel{?}{=} t$ |

$\approx$

| $\mathcal{L}_{\text{mac-fake}}$ |
|---|
| $r \leftarrow \mathsf{Gen}(1^\lambda)$ |
| $\mathcal{T} := \emptyset$ |
| $\underline{\mathsf{GETTAG}(m)}$: |
| $\quad t := \mathsf{MAC}(k,m)$ |
| $\quad \mathcal{T} := \mathcal{T} \cup \{(m,t)\}$ |
| $\quad \textbf{return } t$ |
| $\underline{\mathsf{CHECKTAG}(m,t)}$: |
| $\quad \textbf{return } (m,t) \in \mathcal{T}$ |

# How to formalize security?

Universal vs existential forgery:
To win the previous game you only need to find **a single message** that you can sign = **existential forgery**: *there exists* a message that I can sign

In some attacks, you can even sign **any message** = **universal forgery**: I can sign *all* messages!

# How to build MAC

# MAC from a PRF

Reminder: A PRF $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ = pseudorandom function.

**?** If I know $F_k(x)$ for some given $x$ and $k$, can I find $F_k(x')$ efficiently (with non-negligible advantage) where $x \neq x'$?

- (A) Yes
- (B) It depends
- (C) No

# MAC from a PRF

Reminder: A PRF $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ = pseudorandom function.

?

If I know $F_k(x)$ for some given $x$ and $k$, can I find $F_k(x')$ efficiently (with non-negligible advantage) where $x \neq x'$?

- (A) Yes
- (B) It depends ✓ But on what? (Hint: size)
- (C) No

# MAC from a PRF

Reminder: A PRF $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ = pseudorandom function.

**?**

If I know $F_k(x)$ for some given $x$ and $k$, can I find $F_k(x')$ efficiently (with non-negligible advantage) where $x \neq x'$?

- (A) Yes
- (B) It depends ✓ But on what? (Hint: size)
  - If $|\mathcal{Y}| = O(\log \lambda)$, then one can guess at random: probability $\frac{1}{2^{|\mathcal{Y}|}} = \frac{1}{\mathsf{poly}(\lambda)}$ (non-negligible) to guess $x$. If one can also verify it's correct, then one can just try all possibilities (brute-force).
  - If $|\mathcal{Y}| = \mathsf{poly}(\lambda)$, then brute-force is **not efficient**:
    $\Rightarrow$ hard to find $F_k(x)$!
    $\Rightarrow$ **Good candidate** for a MAC!
- (C) No

# MAC from a PRF

And indeed:

**PRFs with long outputs are MAC**

Let $F$ be a secure PRF with input length in and output length $\lambda$. Then the scheme $\mathsf{MAC}(k,m) := F_k(m)$ and $\mathsf{Gen}(1^\lambda) := r \xleftarrow{\$} \{0,1\}^\lambda; \textbf{return } r$ is a strong EUF-CMA secure MAC for the message space $\mathcal{M} := \{0,1\}^{\mathsf{in}}$.

*Idea of the proof.* Intuitively, since $F$ is a PRF, knowing $F_k(m)$ gives no information about $F_k(m')$ for $m' \neq m$, because $F$ is indistinguishable from a function where each output is independently random. Each call to $\textsc{GetTag}(m)$ gives us $F_k(m)$, but in the end one must guess $F_k(m')$ for a never-before-seen $m'$: hard to do better than random guessing, with probability $\frac{1}{2^\lambda} = \mathsf{negl}(\lambda)$ to guess correctly! (full proof in *Joy of Cryptography*)

# MAC for arbitrarily long messages

# MAC for long messages

**Problem** : The PRF method works for **fixed-length** messages in.

? How to generate a MAC for **arbitrary-length** messages?

# MAC for long messages

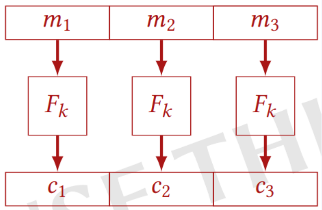
**Problem** : The PRF method works for **fixed-length** messages in.

**?** How to generate a MAC for **arbitrary-length** messages?

For encryption, the solution = cipher modes (CBC, CTR...). Here too? (spoiler: **not that simple**)

# MAC for long messages

?



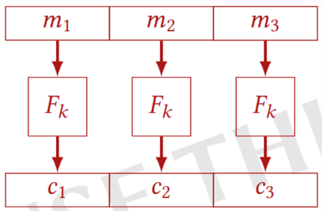

**First attempt: ECB-MAC**

We consider:

$$\boxed{\begin{array}{l}\underline{\text{MAC}(k, m_1 \| \ldots \| m_l):} \\ \quad \textbf{return } F_k(m_1) \| \ldots \| F_k(m_l)\end{array}}$$

Is this a secure MAC?
(Caseine exercise)

# MAC for long messages



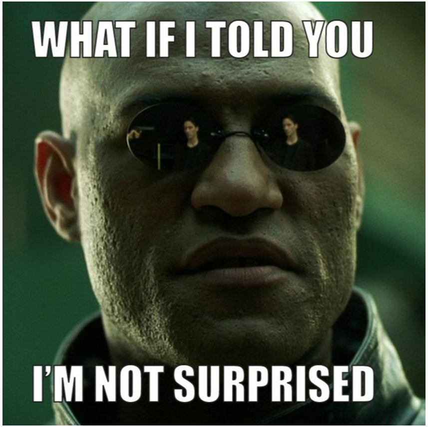

**First attempt: ECB-MAC**

We consider:

$$\boxed{\begin{array}{l} \underline{\text{MAC}(k, m_1 \| \ldots \| m_l):} \\ \quad \textbf{return } F_k(m_1) \| \ldots \| F_k(m_l) \end{array}}$$

Is this a secure MAC?
(Caseine exercise)
No! (idea: reorder the blocks)

ECB mode not secure... **no big news!**



# MAC for long messages

**Second attempt: ECB++MAC**

We consider:

$$\underline{\mathsf{MAC}(k, m_0 \| \ldots \| m_l):} \\ \quad \textbf{return } F_k(\boxed{0} \| m_0) \| \ldots \| F_k(\boxed{n} \| m_l)$$

Is this a secure MAC?
(Caseine exercise)

# MAC for long messages

?

**Second attempt: ECB++MAC**

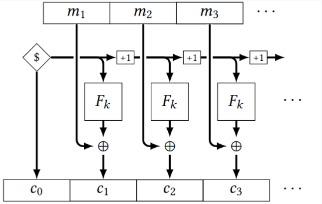
We consider:

$$\boxed{\begin{array}{l}\underline{\mathsf{MAC}(k, m_0 \| \ldots \| m_l):} \\ \quad \mathbf{return}\ F_k(\,0\,\|m_0)\| \ldots \|F_k(\,n\,\|m_l)\end{array}}$$

Is this a secure MAC?
(Caseine exercise)
No! Idea: mix the blocks across several messages

# MAC pour de long messages

?



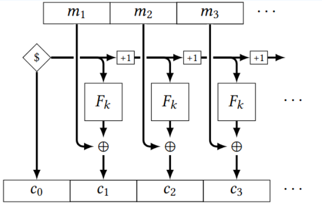

## Third attempt: CTR-MAC

We consider:

$$\boxed{\begin{array}{l} \underline{\mathsf{MAC}(k, m_0 \| \ldots \| m_l):} \\ \quad c_0 \xleftarrow{\$} \{0, 1\}^\lambda \\ \quad \textbf{return } c_0 \| F_k(c_0 + 0) \oplus m_0 \| \ldots \\ \qquad \ldots \| F_k(c_0 + n) \oplus m_l \end{array}}$$

Is this a well-defined MAC?
(Caseine exercise)

# MAC pour de long messages




**Third attempt: CTR-MAC**

We consider:

$$\boxed{\begin{array}{l}\underline{\mathsf{MAC}(k, m_0 \| \dots \| m_l):} \\ \quad c_0 \xleftarrow{\$} \{0, 1\}^\lambda \\ \quad \textbf{return } c_0 \| F_k(c_0 + 0) \oplus m_0 \| \dots \\ \qquad \dots \| F_k(c_0 + n) \oplus m_l \end{array}}$$

Is this a well-defined MAC?

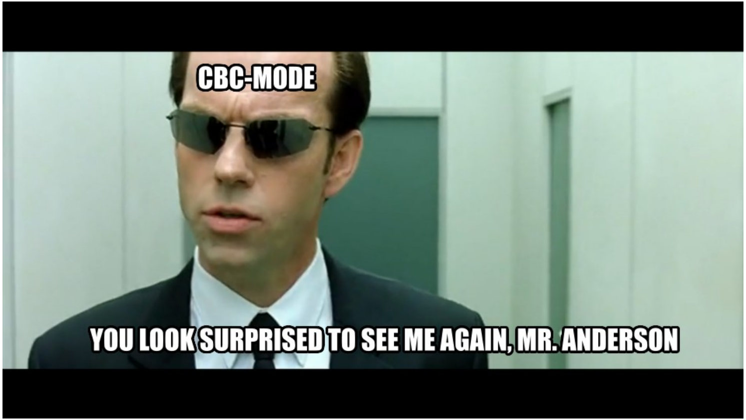
(Caseine exercise)

No, because it is not deterministic! And even if it were (e.g., fixed IV), very easy to break (same attack as ECB++-MAC).

# MAC for long messages

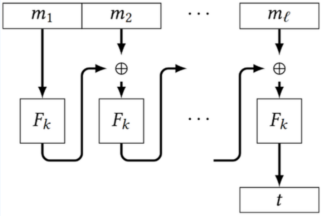
# MAC for long messages

And what about **CBC mode**?



# MAC pour de long messages

?




**Fourth try: CBC-MAC**

We consider:

$\text{MAC}(k, m_1 \| \ldots \| m_l)$:

$t := 0^\lambda$

**for** $i = 1$ to $l$

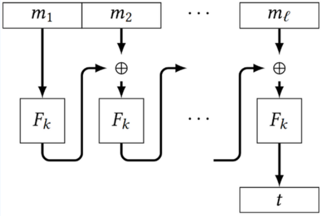
$t := F_k(m_i \oplus t)$

**return** $t$

Is this a secure MAC?

(Caseine exercise “MAC attack 4: CBC-MAC”)

?




**Fourth try: CBC-MAC**

We consider:

$\mathsf{MAC}(k, m_1 \| \ldots \| m_l)$:

> $t := 0^\lambda$
>
> **for** $i = 1$ to $l$
>
> $\quad t := F_k(m_i \oplus t)$
>
> **return** $t$

Is this a secure MAC?

(Caseine exercise “MAC attack 4: CBC-MAC”)

✓ / ✗ Yes and No: yes if only signing messages of the same length, no if signing messages of different lengths (idea: sign $m_0$ (tag $t$) and $t \oplus m_1$, then combine to get a tag for $m_0 \| m_1$).

# MAC for long messages

So CBC-MAC is **not secure** because one can combine small tags to obtain large tags...

# MAC for long messages

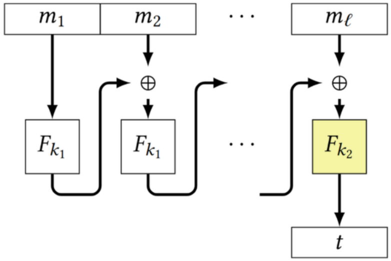
**Solutions**:

- Add the **length** of the message at the beginning:
  $\Rightarrow$ Problem = one must know the message length before starting the signature, sometimes **not practical** for large messages (and adding the length at the end = not secure)
- Use a **different function** at the end!

# MAC for long messages

**Theorem**

Let $F\colon \mathcal{K} \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a PRF. The ECBC-MAC mode defined as:

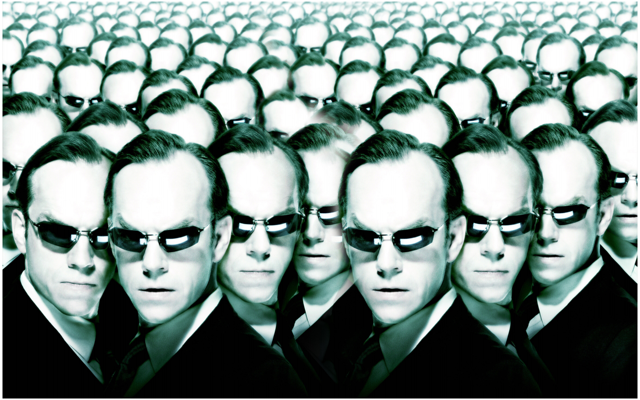



MAC$((k_1,k_2), m_1\|\ldots\|m_l)$:
- $t := 0^\lambda$
- **for** $i = 1$ to $l-1$
  - $t := F_{k_1}(m_i \oplus t)$
- **return** $F_{k_2}(m_l \oplus t)$

is strongly EUF-CMA secure (for messages in $(\{0,1\}^\lambda)^*$ with the above construction, and in $\{0,1\}^*$ using padding).

# MAC for long messages

ECBC-MAC is thus **secure**!



# MAC pour de long messages

ECBC-MAC is secure, but requires **two keys**: it can be made a bit more efficient with **a single key** = One-Key CBC-MAC (**OMAC**, or OMAC2), further slightly improved with OMAC1 (=CMAC), (OMAC is sometimes used to refer to this family).

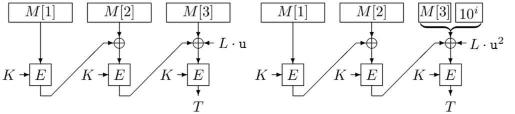



**Fig. 2.** Illustration of OMAC1. Note that $L = E_K(0^n)$.

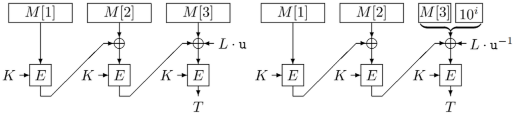



**Fig. 3.** Illustration of OMAC2.

https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/omac/omac-ad.pdf

# MAC from hash functions

# MAC from hash functions

Ideas: use **hash functions** to build MACs?

- $\mathsf{PrefixMac}_k(m) := H(k \| m)$
- $\mathsf{SuffixMac}_k(m) := H(m \| k)$
- $\mathsf{SandwitchMac}_{k_1 \| k_2}(m) := H(k_1 \| m \| k_2)$ ("padded" ?)
- Other ?

**?** Is it really secure

# MAC from hash functions

Ideas: use **hash functions** to build MACs?

- $\text{PrefixMac}_k(m) := H(k \| m)$ → sometimes ok : e.g. SHA-3 (designed this way)
- $\text{SuffixMac}_k(m) := H(m \| k)$
- $\text{SandwitchMac}_{k_1 \| k_2}(m) := H(k_1 \| m \| k_2)$ (“padded” ?)
- Other ?

**?** Is it really secure

# MAC from hash functions

Ideas: use **hash functions** to build MACs?

- $\text{PrefixMac}_k(m) := H(k \| m)$ → sometimes ok: e.g. SHA-3 (designed this way)
- $\text{SuffixMac}_k(m) := H(m \| k)$
- $\text{SandwitchMac}_{k_1 \| k_2}(m) := H(k_1 \| m \| k_2)$ ("padded" ?)
- Other ?

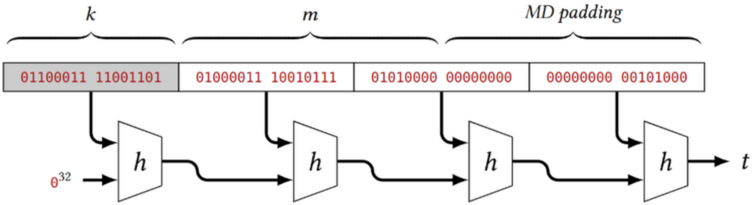
⚠ Sometimes broken when used with Merkle-Damgård ⇝ length extension attack

Eg. • MD5
• SHA-1
• SHA-2

**?** Is it really secure

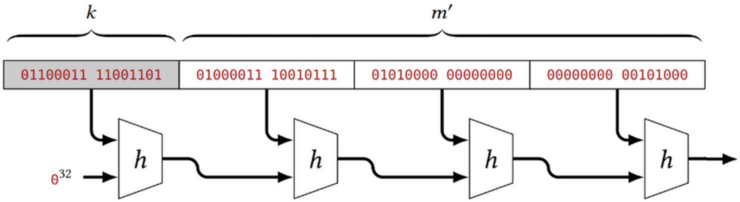
# Length Extension Attack

Attack on $\text{PrefixMac}_k(m) := H(k\|m)$ if $H$ is based on Merkle-Damgård:



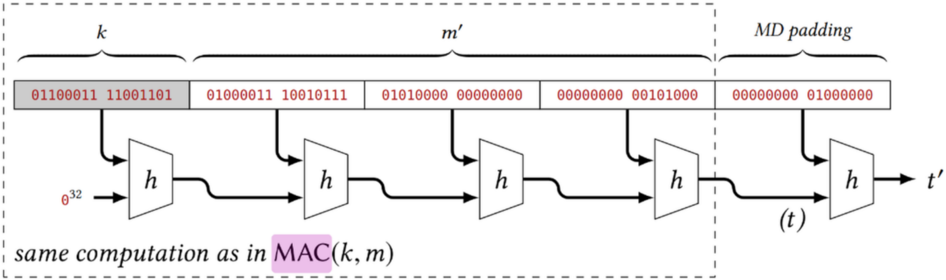
# Length Extension Attack

Attack on PrefixMac$_k(m) := H(k \| m)$ if $H$ is based on Merkle-Damgård:



# Length Extension Attack

Attack on PrefixMac$_k(m) := H(k\|m)$ if $H$ is based on Merkle-Damgård:



# Length Extension Attack

?

Is this attack a forgery:

- (A) universal
- (B) existential

# Length Extension Attack

?

Is this attack a forgery:

- Ⓐ universal
- Ⓑ existential ✓ One can only sign certain messages, those of the form $m \| pad_m \| m'$ (which is already pretty useful…)

# Length Extension Attack

Problem: the key appears **before** + the hash contains **the entire internal state**

Solutions?

- Wide-pipe hash constructions (discard part of the output) or sponge constructions (see hash functions lecture): use SHA-3 which is explicitly designed for this
- Or **do not use** PrefixMac. But then what?

# Suffix forgery attack

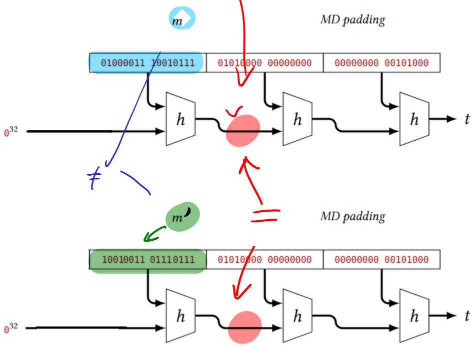
Attempt 2: $\mathsf{SuffixMac}_k(m) := H(m\|k)$

# Suffix forgery attack

Attempt 2: $\text{SuffixMac}_k(m) := H(m \| k)$ if $H$ is based on Merkle-Damgård:

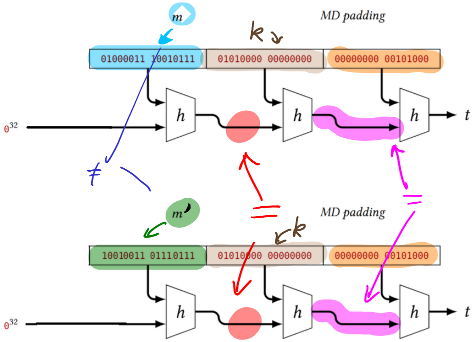
# Suffix forgery attack

Attempt 2: $\text{SuffixMac}_k(m) := H(m\|k)$ if $H$ is based on Merkle-Damgård:
Suppose we know a collision (not obvious, but known for MD5, SHA-0, SHA-1...):



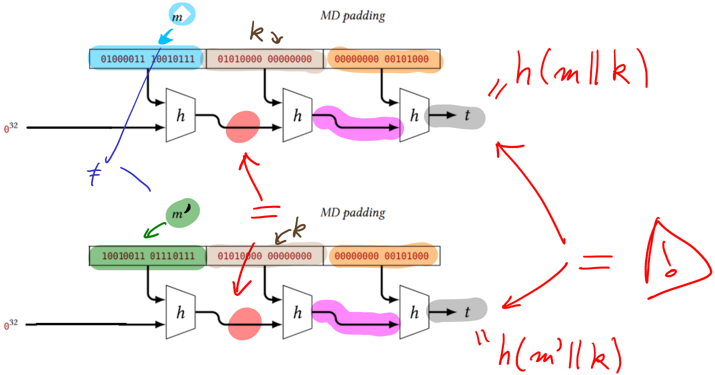
# Suffix forgery attack

Attempt 2: $\text{SuffixMac}_k(m) := H(m\|k)$ if $H$ is based on Merkle-Damgård:
Suppose we know a collision (not obvious, but known for MD5, SHA-0, SHA-1...):



# Suffix forgery attack

Attempt 2: $\text{SuffixMac}_k(m) := H(m \| k)$ if $H$ is based on Merkle-Damgård: Suppose we know a collision (not obvious, but known for MD5, SHA-0, SHA-1...):



# Security even if $H$ is not collision-resistant?

**?** Can we obtain a MAC from a **vulnerable** hash function, i.e. one for which a collision is known?

- The most common (and battle-tested): **HMAC** (next slide)
- Also possible (and proven[1]): $\text{SandwitchMac}_{k_1\|k_2}(m) := H(k_1\|m\|k_2)$, but beware: the message **must be padded**[2] to a block boundary, and each key must be large!
- Other possibilities, e.g. NMAC

[1] https://link.springer.com/chapter/10.1007/978-3-540-73458-1_26
[2] https://link.springer.com/chapter/10.1007/3-540-68339-9_3

**Definition (HMAC)**

HMAC is defined as:

$$\mathrm{HMAC}_k(m) = H\Big((k \oplus \text{opad}) \mathbin{||} H\big((k \oplus \text{ipad}) \mathbin{||} m\big)\Big)$$

where ipad $=$ `0x3636...36` and opad $=$ `0x5c5c...5c` (their choice is important[a]).

[a] `https://eprint.iacr.org/2012/684.pdf`

Conclusion: we need ipad  to get a MAC . Coincidence? I don't think so…

**Advantages of HMAC:**

- provable security[3],
- does not require collision resistance,
- works even if $H$ is based on the Merkle–Damgård construction,
- and has stood the test of time!

[3] `https://eprint.iacr.org/2006/043.pdf`

# Encryption + MAC

# Encryption + MAC

**Motivations:**

- The motivation for MACs was to have a CCA-secure encryption scheme (active attackers, e.g., padding oracle attack).
  $\Rightarrow$ How to combine MAC & Encryption to achieve **CCA security**?
- Often in practice we want both encryption and authentication. Can we do it **more efficiently** than encryption + MAC?

## CCA from MAC and CPA

Let $(E.\mathsf{Gen}, E.\mathsf{Enc}, E.\mathsf{Dec}_e)$ be a CPA-secure encryption scheme, and $(M.\mathsf{Gen}, M.\mathsf{MAC})$ a (strongly EUF-CMA) secure MAC. Then the following "encrypt-then-MAC" scheme is CCA-secure:

$$\mathcal{K} = E.\mathcal{K} \times M.\mathcal{K}$$
$$\mathcal{M} = E.\mathcal{M}$$
$$C = E.C \times M.\mathcal{T}$$

KeyGen:

$k_e \leftarrow E.\mathsf{KeyGen}$
$k_m \leftarrow M.\mathsf{KeyGen}$
return $(k_e, k_m)$

$\mathsf{Enc}((k_e, k_m), m)$:

$c := E.\mathsf{Enc}(k_e, m)$
$t := M.\mathsf{MAC}(k_m, c)$
return $(c, t)$

$\mathsf{Dec}((k_e, k_m), (c, t))$:

if $t \neq M.\mathsf{MAC}(k_m, c)$:
  return `err`
return $E.\mathsf{Dec}(k_e, c)$

# CCA Security

?

Exercise: prove the previous theorem.
Simplified exercise for **caséine**, with pre-filled games to order (MAC section, activity "CCA from MAC and CPA").

# CCA Security

Typically, the goal of a secure channel =

- **confidentiality** : the message is hidden against a malicious adversary
- **authenticity** : all messages truly come from the intended sender (no message insertion or modification...)
- **no "replay"** : we want to prevent replay attacks (an adversary could resend a previously seen message)!

**?** Doesn't CCA already protect us?

# CCA Security

There exist encryption schemes
(e.g. $\mathsf{Enc}(k,m) := r \xleftarrow{\$} \{0,1\}^\lambda; \textbf{return } E_k(m\|r)$)
that are CCA but where **an attacker can send an arbitrary message**.

Typically, the goal of a sec

- **confidentiality** : the message is hidden against a malicious adversary ✓
- **authenticity** : all messages truly come from the intended sender (no message insertion or modification...) ✗
- **no "replay"** : we want to prevent replay attacks (an adversary could resend a previously seen message)! ✗

**?**

Doesn't CCA already protect us?
$\Rightarrow$ **not completely!**

$\Rightarrow$ **Need a better definition:**
**Authenticated Encryption with Associated Data !**
**(AEAD)**

**Limit replay (this message is the $n$-th message sent in this "context" (=session) $d$)**

# AEAD

Preventing replay = introduce “associated data”/**context** $d$ (e.g. session ID & message number, hash of entire conversation history…) identifying **the current connection**, and modify encryption and decryption accordingly.

$$\mathsf{Enc}(k, \boxed{d}, m) \qquad \mathsf{Dec}(k, \boxed{d}, c)$$

$\Rightarrow$ Goal = **it is impossible for an adversary to generate a ciphertext** $(c, d)$ **that has not already been seen**

Note: to simplify (and strengthen) security, we additionally require that the encryption be indistinguishable from a random element in the ciphertext space $\mathcal{C}$:

## AEAD

Let $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$ be an encryption scheme. We say that $\Sigma$ has indistinguishable security against **Associated Encryption and Associated Data (AEAD)** if:

| $\mathcal{L}^{\Sigma}_{\text{aead\$-real}}$ |
|---|
| $k \leftarrow \text{Gen}(1^\lambda)$ |
| $\mathcal{S} := \emptyset$ |
| $\underline{\text{CTXT}(d, m)}$: |
|   $c \leftarrow \text{Enc}(k, d, m)$ |
|   $\mathcal{S} := \mathcal{S} \cup \{(d, c)\}$ |
|   **return** $c$ |
| $\underline{\text{DECRYPT}(c \in \mathcal{C})}$: |
|   if $(d, c) \in \mathcal{S}$ return err |
|   return $\text{Dec}(k, d, c)$ |

$\approx$

| $\mathcal{L}^{\Sigma}_{\text{aead\$-real}}$ |
|---|
| $\underline{\text{CTXT}(d, m)}$: |
|   $c \xleftarrow{\$} \mathcal{C}$ |
|   return $c$ |
| $\underline{\text{DECRYPT}(c \in \mathcal{C})}$: |
|   **return** err |

# AEAD construction

# AEAD Construction

Several approaches are possible:

- combine encryption + MAC: simple, but less efficient
- “3-in-1” AEAD ciphers: more complex, but more efficient

# AEAD Construction: Encrypt-then-MAC (AEAD version)

**First method** chiffrement-puis-mac (version AEAD):

**Encryption + MAC = AEAD**

Let $(\mathsf{Gen}_e, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure encryption scheme (resp. CPA$-secure, i.e., ciphertext is indistinguishable from random), and $(\mathsf{Gen}_m, \mathsf{MAC})$ a secure MAC, then the construction below is a **secure AEAD** (resp. AEAD$):

$$\begin{array}{l} \underline{\mathsf{Gen}(1^\lambda):} \\ \quad k_e \leftarrow \mathsf{Gen}_e(1^\lambda) \\ \quad k_m \leftarrow \mathsf{Gen}_m(1^\lambda) \\ \quad \textbf{return } (k_e, k_m) \end{array}$$

$$\begin{array}{l} \underline{\mathsf{Enc}((k_e, k_m), d, m):} \\ \quad c \leftarrow \mathsf{Enc}_{k_e}(m) \\ \quad t := \mathsf{MAC}(k_m, d \| c) \\ \quad \textbf{return } (c, t) \end{array}$$

$$\begin{array}{l} \underline{\mathsf{Dec}((k_e, k_m), d, (c, t)):} \\ \quad \textbf{if } t \neq \mathsf{MAC}(k_m, d \| c) \\ \quad\quad \textbf{return } \texttt{err} \\ \quad \textbf{return } \mathsf{Dec}_{k_e}(c) \end{array}$$

*Proof idea:* Similar to the proof that "encrypt-then-MAC" is CCA-secure.

# AEAD Construction: inefficiency of encryption + MAC

If we instantiate encryption + MAC with CBC encryption and CBC-MAC, we call the block cipher $2\times$ **per block**!

# AEAD Construction: inefficiency of encryption + MAC

If we instantiate encryption + MAC with CBC encryption and CBC-MAC, we call the block cipher $2\times$ **per block**!

$\Rightarrow$ **inefficient** ! (we do $2\times$ **almost the same work**)

# AEAD Construction: inefficiency of encryption + MAC

If we instantiate encryption + MAC with CBC encryption and CBC-MAC, we call the block cipher $2\times$ **per block**!

$\Rightarrow$ **inefficient** ! (we do $2\times$ **almost the same work**)

**?** Can we do better?

# AEAD Construction: inefficiency of encryption + MAC

If we instantiate encryption + MAC with CBC encryption and CBC-MAC, we call the block cipher $2\times$ **per block**!

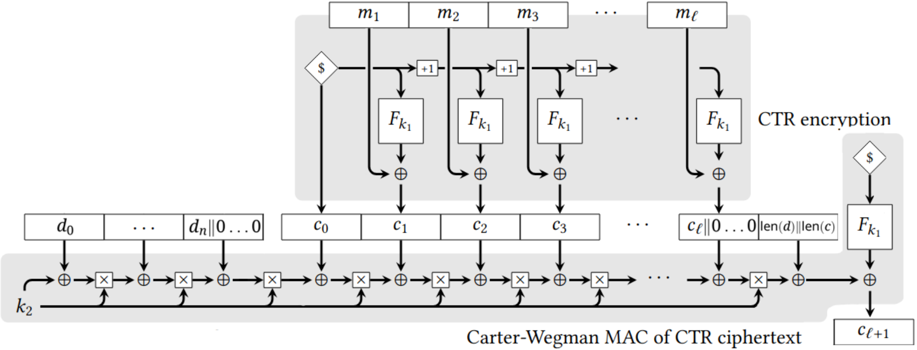
$\Rightarrow$ **inefficient** ! (we do $2\times$ **almost the same work**)

**?** Can we do better?

✓ **Yes: GCM mode!**

# GCM mode

**Principle** GCM mode = encrypt-then-MAC




Source JoC modified. Details: `https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf`

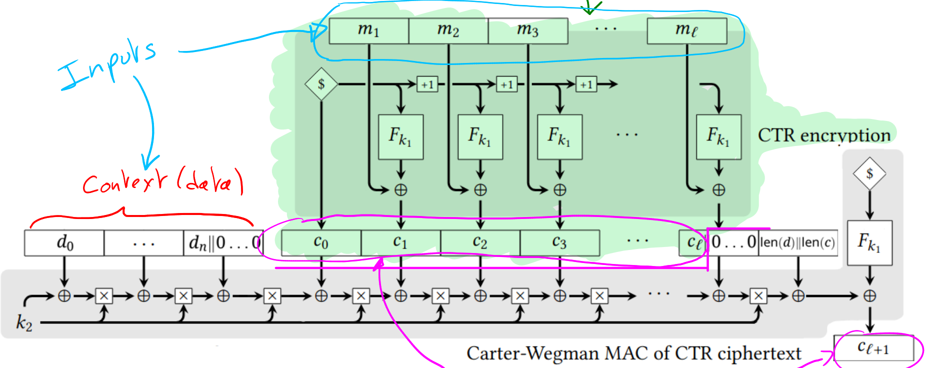
# GCM mode

Standard CTR-MODE

**Principle** GCM mode = encrypt-then-MAC

Inputs

Context (data)




Source JoC modified. Details: `https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf`

outputs

# GCM mode

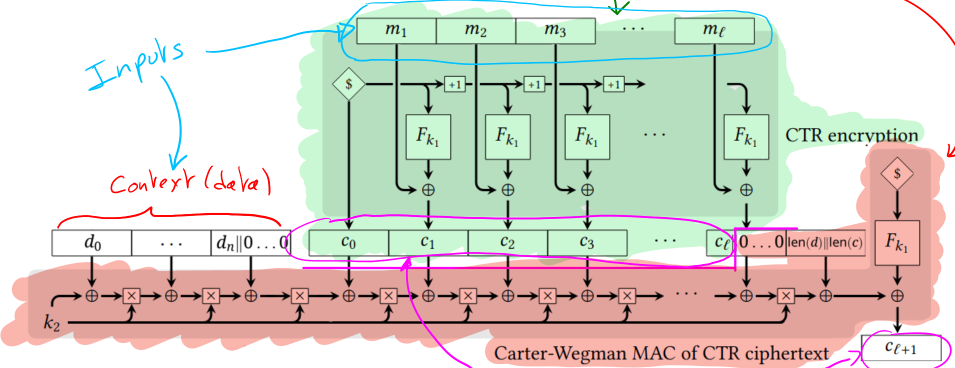
Standard CTR-MODE

= GMAC (slide suivante)

**Principle** GCM mode = encrypt-then-MAC

Inputs

Context (data)




Source JoC modified. Details: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf

outputs

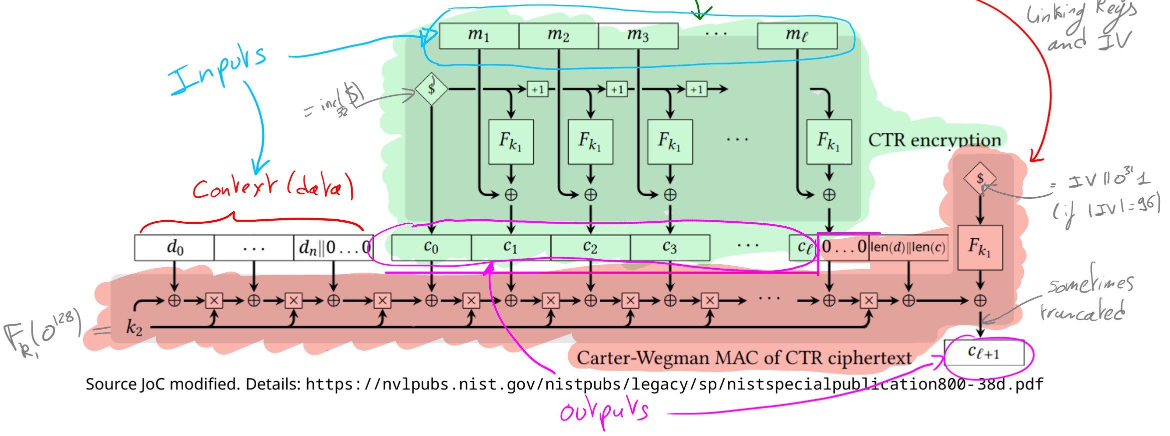
# GCM mode

**Principle** GCM mode = encrypt-then-MAC

Standard CTR-MODE

= GMAC (slide suivante)

+ some specificities linking keys and IV




Source JoC modified. Details: `https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf`

# GMAC

**GMAC** (Carter-Wegman MAC) : MAC construction that uses only **one cipher call**, otherwise only multiplications!
$\Rightarrow$ much **more efficient** !

GMAC construction = 2 steps:

très efficace

1. Use a **universal hash** function = very efficientsimple evaluation of a polynomial $\sum_{i=0}^{l} c_{l-i} s^l$ ($s$ = salt) over a finite field where operations are efficient, but **very insecure** :
(= collision-resistant if the salt is unknown + 1 single attempt)
2. Apply a pseudo-OTP at the end on the result (thus only 1 block-cipher call!) to **boost** security by “hiding” the function output, and thus its salt $s = k_2$ (if revealed, one can sign anything)
$\Rightarrow$ it is a PRF and thus a MAC

# GMAC

?

We just built a PRF... but with the block-cipher used we already had a PRF. What is the advantage?

?

We just built a PRF... but with the block-cipher used we already had a PRF. What is the advantage?

$\Rightarrow$ Here we built a PRF for **unbounded-size** inputs! (block-cipher = fixed size)

# GMAC

The universal function only computes

$$\sum_{i=0}^{l} x_{l-i} s^l$$

($s$ = salt, $x = d_{\text{0-padded}} \| c_{\text{0-padded}} \| \text{len}(d) \| \text{len}(c)$)

How to do it **efficiently**?

# GMAC

The universal function only computes

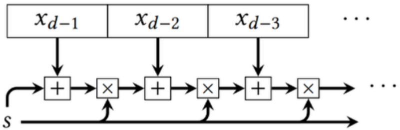
$$\sum_{i=0}^{l} x_{l-i} s^{l}$$

($s$ = salt, $x = d_{\text{0-padded}} \| c_{\text{0-padded}} \| \text{len}(d) \| \text{len}(c)$)

How to do it **efficiently**?

$\Rightarrow$ **Ruffini-Horner method**

$$\sum_{i=0}^{l} x_{l-i} s^{l} = \ldots (s \cdot (s \cdot (s + x_{l-1}) + x_{l-2}) + x_{l-3}) \ldots$$



? A **multiplication** between bitstrings. . . ?!?

? A **multiplication** between bitstrings…?!?

$\Rightarrow$ Messages interpreted as elements of $\mathbb{F}_{2^{128}}$:

- $+$ = bitwise XOR
- $\times$: each element $a = a_0 \dots a_{127} \in \{0, 1\}^{128}$ seen as a polynomial $a_0 + a_1 X + a_2 X^2 \dots + a_{127} X^{127} \in \mathbb{Z}_2[X]$, multiply polynomials, then reduce (keep 128 bits) modulo $X^{128} + X^7 + X^2 + X + 1$

? What is $\texttt{110...01} \times \texttt{1010...0}$?

# In practice

# Et en pratique ?

**En pratique**:

- CBC-MAC is used in AEAD CCM mode, itself used in IEEE 802.11i, IPsec, TLS 1.2 & 1.3 (disabled by default in 1.3 in openssl), Bluetooth Low Energy (4.0).
- OMAC is used in AEAD EAX mode (replacement for CCM)
- AEAD **GCM widely adopted** (efficient), used in IEEE 802.1AE (MACsec), Ethernet security, WPA3-Enterprise Wifi security protocol, IEEE 802.11ad, ANSI (INCITS) Fibre Channel Security Protocols (FC-SP), IEEE P1619.1 tape storage, IETF IPsec standards, SSH, TLS 1.2 and TLS 1.3, OpenVPN…
- HMAC: used in IPsec, TLS, JWT JSON Web Tokens (RFC 7519)…
- Poly1305 (hash usable as MAC) used in AEAD ChaCha20-Poly1305, itself used in IPsec, SSH, (D)TLS 1.2 & 1.3, WireGuard, S/MIME 4.0, OTRv4…Very fast in software, often replaces GCM when no hardware instructions available

# Conclusion

# Conclusion

- **MAC allows to “sign” (=tag) a message** si on partage une clé privée avec le destinataire
- **Security can be formalized** avec un jeu visant à forger de nouveaux tags ($\approx$ signatures) $\Rightarrow$ (fortement) EUF-CMA sécurisé
- **Secure MACs can be constructed from**:
  - block-ciphers (care: very different from encryption!)
  - hash functions, but beware attacks if misused (length extension attack...)
- Encrypt-then-MAC provides CCA security
- But CCA is not sufficient (replay attacks etc.)
  $\Rightarrow$ **define AEAD (even more secure) by introducing context**
- Encrypt-then-MAC is AEAD-secure...but can be made more efficient with GCM mode, widely used