

# Cryptographie

## Authentification symétrique)

Léo COLISSON PALAIS  
Master CSI 2024 – 2025

[leo.colisson-palais@univ-grenoble-alpes.fr](mailto:leo.colisson-palais@univ-grenoble-alpes.fr)  
<https://leo.colisson.me/teaching.html>

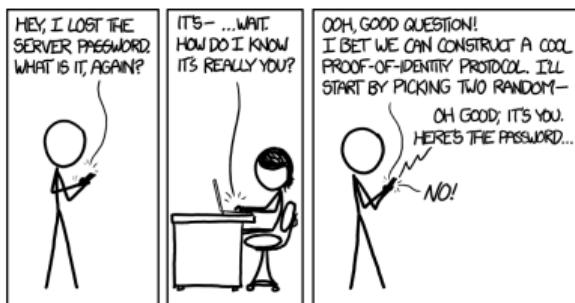
# Authentication: Motivations

# Authentification : Motivations

Authentification/signature = s'assurer que l'on **parle à la bonne personne**

## Motivations:

- Prouver qui l'on est = souvent très important:
  - Accéder à son compte bancaire...: Mot de passe pas pratique/suffisant = authentification à 2 facteurs
  - ... et être sûr que l'on parle bien à sa banque !
  - Ouvrir une voiture/porte d'accès/...
- Stocker des données chez des personnes malicieuses
- "Stateless" serveur : JSON Web Token (JWT)
- Éviter attaque déni de service dans TCP : SYN cookies
- Blockchain = signer une autorisation de transfert d'argent
- Éviter les attaques de "l'homme du milieu" (Man In The Middle, MITM)
- Éviter l'attaque padding oracle  $\Rightarrow$  obtenir sécurité IND-CCA = sécurité contre **adversaires actifs**



# Authentification

Comme chiffrement, 2 grandes familles :



**Clé privée** (symétrique)

= **Code d'authentification de message (MAC)**

= Le vérifieur de la signature doit échanger au préalable une clé privée avec le signataire

**Clé publique** (asymétrique)  
= **signature**

= Le vérifieur de la signature doit connaître la clé publique du signataire

# Authentification

Comme chiffrement, 2 grandes familles :

*ce "coeur" !*

**Clé privée (symétrique)**  
= **Code d'authentification de message (MAC)**

= Le vérifieur de la signature doit échanger au préalable une clé privée avec le signataire

**Clé publique (asymétrique)**  
= **signature**

= Le vérifieur de la signature doit connaître la clé publique du signataire

# Code d'authentification de message (MAC)

## Code d'authentification de message (MAC)

Un code d'authentification de message (=*message authentication code*, **MAC**) pour un espace de message  $\mathcal{M}$  est composé de deux algorithmes :

- $\text{Gen}(1^\lambda)$  qui renvoie une clé secrète  $k$
- $\text{MAC}(k, m)$  est un algorithme déterministe qui prend en entrée une clé  $k$  et un message  $m \in \mathcal{M}$  et renvoie un **tag** (joue le rôle d'une signature)



Comment vérifier si un tag  $t$  authentifie bien le message  $m$  ?

# Code d'authentification de message (MAC)

## Code d'authentification de message (MAC)

Un code d'authentification de message (=*message authentication code*, **MAC**) pour un espace de message  $\mathcal{M}$  est composé de deux algorithmes :

- $\text{Gen}(1^\lambda)$  qui renvoie une clé secrète  $k$
- $\text{MAC}(k, m)$  est un algorithme déterministe qui prend en entrée une clé  $k$  et un message  $m \in \mathcal{M}$  et renvoie un **tag** (joue le rôle d'une signature)



Comment vérifier si un tag  $t$  authentifie bien le message  $m$  ?

MAC est déterministe, donc on calcule  $\text{MAC}(k, m) \stackrel{?}{=} t$  !

# Code d'authentification de message (MAC)

**Disclaimer** : j'aurai souvent tendance à parler de signature au lieu de tag, car c'est moralement la même chose à part clé privée/publique.

# MAC: définitions de sécurité

# Comment formaliser la sécurité ?

Sécurité intuitivement = dur de générer un tag valide sans connaître la clé  $k$ .  
**Comment formaliser cette idée ?**

# Comment formaliser la sécurité ?

**Étape 1:** comment formaliser “dur de trouver X”?

Est-il vrai que:

?

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

- A Non
- B Oui, mais on aurait pu mettre  $\equiv$
- C Oui, et  $\approx$  est le bon symbole

# Comment formaliser la sécurité ?

**Étape 1:** comment formaliser “dur de trouver X”?

Est-il vrai que:

?

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

- A Non
- B Oui, mais on aurait pu mettre  $\equiv$
- C Oui, et  $\approx$  est le bon symbole car il est dur de trouver  $x$ , mais on a une chance négligeable ( $\frac{1}{2^\lambda}$ ) de le trouver

# Comment formaliser la sécurité ?

**Étape 2:** comment formaliser "dur de trouver **un tag valide**"?



Premier essai:

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

Est-ce une bonne idée?

- A Non, car on peut toujours distinguer ces libraries
- B Non, car cette définition n'est pas assez générique
- C Oui

# Comment formaliser la sécurité ?

**Étape 2:** comment formaliser "dur de trouver **un tag valide**"?

Premier essai:

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

$\approx$

$$\begin{array}{|c|} \hline \mathcal{L}_{\text{false}} \\ \hline \text{CHECKTAG}(m, t): \\ \hline \text{return false} \\ \hline \end{array}$$

Est-ce une bonne idée?



- A Non, car on peut toujours distinguer ces libraries
- B Non, car cette définition n'est pas assez générique
  - ✓ Dans la vraie vie, un attaquant va voir passer des tags valides !! Il a donc plus d'information qu'ici. Par exemple,  $\text{MAC}(t, x) := (t, x)$  serait sécurisé avec cette définition, mais en réalité on ne considère pas ceci sécurisé car il suffit de voir une seule "signature" (tag) pour pouvoir signer n'importe quel message !
- C Oui

# Comment formaliser la sécurité ?

Étape 2: comment formaliser "dur de trouver **un tag valide**"?



Deuxième essai:

$\mathcal{L}_{\text{mac-1}}$
$r \leftarrow \text{Gen}(1^\lambda)$
<u>GETTAG(<math>m</math>):</u>
<b>return</b> MAC( $k, m$ )
<u>CHECKTAG(<math>m, t</math>):</u>
<b>return</b> MAC( $k, m$ ) $\stackrel{?}{=} t$

$\mathcal{L}_{\text{mac-1-false}}$
$r \leftarrow \text{Gen}(1^\lambda)$
<u>GETTAG(<math>m</math>):</u>
<b>return</b> MAC( $k, m$ )
<u>CHECKTAG(<math>m, t</math>):</u>
<b>return</b> false

$\approx$

Est-ce une bonne idée?

- A Non, car on peut toujours distinguer ces libraries
- B Non, car cette définition n'est pas assez générique
- C Oui

# Comment formaliser la sécurité ?

**Étape 2:** comment formaliser "dur de trouver **un tag valide**"?



Deuxième essai:

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

Est-ce une bonne idée?

- A Non, car on peut toujours distinguer ces libraries
  - ✓ Essayez de trouver une attaque (exercice Caseine "MAC > MAC > MAC mauvaise definition")
- B Non, car cette définition n'est pas assez générique
- C Oui

# Comment formaliser la sécurité ?

**Étape 2:** comment formaliser "dur de trouver **un tag valide**"?



Deuxième essai:

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

Est-ce une bonne idée?

- A Non, car on peut toujours distinguer ces libraries
  - ✓ Essayez de trouver une attaque (exercice Caseine "MAC > MAC > MAC mauvaise definition")  $\Rightarrow$  Solution: CHECKTAG("hello", GETTAG("hello"))
- B Non, car cette définition n'est pas assez générique
- C Oui

# Comment formaliser la sécurité ?

**Étape 2:** comment formaliser "dur de trouver **un tag valide**"?

Deuxième essai:

$\mathcal{L}_{\text{mac-1-real}}$	$\mathcal{L}_{\text{mac-1-fake}}$	
$r \leftarrow \text{Gen}(1^\lambda)$ <u><math>\text{GETTAG}(m):</math></u> <b>return</b> $\text{MAC}(k, m)$ <u><math>\text{CHECKTAG}(m, t):</math></u> <b>return</b> $\text{MAC}(k, m) \stackrel{?}{=} t$	$r \leftarrow \text{Gen}(1^\lambda)$ <u><math>\text{GETTAG}(m):</math></u> <b>return</b> $\text{MAC}(k, m)$ <u><math>\text{CHECKTAG}(m, t):</math></u> <b>return</b> $\text{false}$	$\approx$ = trop parano ! (1)

Ce n'est pas une attaque si la seule chose que l'on arrive à signer c'est en copiant/collant des signatures existantes ! (mais attention, les replay attacks peuvent poser problème en pratique, mais ce n'est pas une attaque que l'on peut résoudre à ce niveau là)

# Comment formaliser la sécurité ?

Troisième (et dernier) essai: on gagne si on arrive à générer un TAG **jamas vu avant**:

## Définition (EUF-CMA-)

Un MAC ( $\text{Gen}$ ,  $\text{MAC}$ ) est dit **fortement EUF-CMA-sécurisé** (*existentially unforgeable under chosen-message attacks*) si:

$\mathcal{L}_{\text{mac-real}}$	$\mathcal{L}_{\text{mac-fake}}$
$r \leftarrow \text{Gen}(1^\lambda)$ $\text{GETTAG}(m):$ <hr/> <b>return</b> $\text{MAC}(k, m)$ $\text{CHECKTAG}(m, t):$ <hr/> <b>return</b> $\text{MAC}(k, m) \stackrel{?}{=} t$	$r \leftarrow \text{Gen}(1^\lambda)$ $\mathcal{T} := \emptyset$ $\text{GETTAG}(m):$ <hr/> $t := \text{MAC}(k, m)$ $\mathcal{T} := \mathcal{T} \cup \{(m, t)\}$ <b>return</b> $t$ $\text{CHECKTAG}(m, t):$ <hr/> <b>return</b> $(m, t) \in \mathcal{T}$

Note: pour non-fortement EUF-CMA-sécurisé, on remplace simplement  $(m, t) \in \mathcal{T}$  par  $\exists t, (m, t) \in \mathcal{T}$ , c'est à dire pour gagner il faut générer un tag pour un **message différent**.

# Comment formaliser la sécurité ?

Exercice **caséine** (MAC > MAC (quizz) > "MAC sécurité OTP").

Soit  $\mathcal{M} = \{0, 1\}^\lambda$ ,  $\text{Gen}(1^\lambda) := r \xleftarrow{\$} \{0, 1\}^\lambda$ ; **return**  $r$  et  $\text{MAC}(k, m) := k \oplus m$ . Est-ce un MAC sécurisé ? Si oui, prouvez-le, sinon trouvez une attaque.

Rappel de la définition :

?

$\mathcal{L}_{\text{mac-real}}$
$r \leftarrow \text{Gen}(1^\lambda)$
<u><math>\text{GETTAG}(m):</math></u>
<b>return</b> $\text{MAC}(k, m)$
<u><math>\text{CHECKTAG}(m, t):</math></u>
<b>return</b> $\text{MAC}(k, m) \stackrel{?}{=} t$

≈

$\mathcal{L}_{\text{mac-fake}}$
$r \leftarrow \text{Gen}(1^\lambda)$
$\mathcal{T} := \emptyset$
<u><math>\text{GETTAG}(m):</math></u>
$t := \text{MAC}(k, m)$
$\mathcal{T} := \mathcal{T} \cup \{(m, t)\}$
<b>return</b> $t$
<u><math>\text{CHECKTAG}(m, t):</math></u>
<b>return</b> $(m, t) \in \mathcal{T}$

# Comment formaliser la sécurité ?

Universal vs existential forgery:

Pour gagner jeux précédent = il suffit de trouver **un seul message** que l'on sait signer pour gagner = **existential forgery**: *il existe* un message que je sais signer

Dans certaines attaques, on sait même signer **n'importe quel message** = **universal forgery**: je peux signer *tous* les messages !

# Comment construire des MAC

# MAC à partir de PRF

Rappel: Une PRF  $F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  = fonction pseudo-aléatoire.

Si je connais  $F_k(x)$  pour un  $x$  et  $k$  donné, est-ce que je peux trouver  $F_k(x')$  efficacement (avec un avantage non négligeable) lorsque  $x \neq x'$  ?



- A Oui
- B Ça dépend
- C Non

# MAC à partir de PRF

Rappel: Une PRF  $F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  = fonction pseudo-aléatoire.

Si je connais  $F_k(x)$  pour un  $x$  et  $k$  donné, est-ce que je peux trouver  $F_k(x')$  efficacement (avec un avantage non négligeable) lorsque  $x \neq x'$  ?

- A Oui
- B Ça dépend ✓ Mais de quoi ? (Indice: taille)



- C Non

Rappel: Une PRF  $F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  = fonction pseudo-aléatoire.

Si je connais  $F_k(x)$  pour un  $x$  et  $k$  donné, est-ce que je peux trouver  $F_k(x')$  efficacement (avec un avantage non négligeable) lorsque  $x \neq x'$  ?



A Oui

B Ça dépend ✓ Mais de quoi ? (Indice: taille)

- Si  $|\mathcal{Y}| = O(\log \lambda)$ , alors on peut deviner au hasard : probabilité  $\frac{1}{2^{|\mathcal{Y}|}} = \frac{1}{\text{poly}(\lambda)}$  (non négligeable) de deviner  $x$ . Si on peut en plus vérifier si c'est correct, alors on peut juste essayer toutes les possibilités (brute-force).
- Si  $|\mathcal{Y}| = \text{poly}(\lambda)$ , alors le brute-force n'est **pas efficace**:  
⇒ difficile de trouver  $F_k(x)$  !  
⇒ **Bon candidat** pour un MAC !

C Non

# MAC à partir de PRF

Et en effet:

PRF + longue sortie = MAC

Soit  $F: \{0, 1\}^\lambda \times \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^\lambda$  une PRF sécurisé (notez la taille de la sortie). Alors  $\text{MAC}(k, m) := F_k(m)$  et  $\text{Gen}(1^\lambda) := k \xleftarrow{\$} \{0, 1\}^\lambda$ ; **return**  $k$  est fortement EUF-CMA sécurisé, et a pour espace de message  $\mathcal{M} := \{0, 1\}^{\text{in}}$ .

*Idée de la preuve.* Intuitivement, puisque  $F$  est une PRF, connaître  $F_k(m)$  ne donne aucune information sur  $F_k(m')$  pour  $m' \neq m$  car  $F$  est indistinguable d'une fonction où chaque élément est tiré au hasard de manière indépendante. Chaque appel à  $\text{GETTAG}(m)$  nous donne donc un  $F_k(m)$ , mais à la fin il faut deviner  $F_k(m')$  pour un  $m'$  jamais vu: difficile de faire mieux que de deviner aléatoirement: probabilité  $\frac{1}{2^\lambda} = \text{negl}(\lambda)$  de deviner correctement! (preuve complète dans *Joy of cryptography*)

# MAC pour de long messages

## MAC pour de long messages

**Problème :** La méthode PRF fonctionne pour les messages de **taille fixe** in.

# ?

## Comment générer un MAC pour des messages de taille arbitraire ?

# MAC pour de long messages

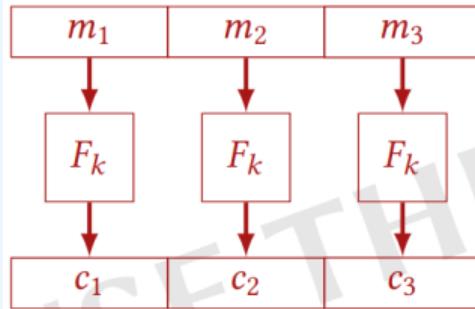
**Problème :** La méthode PRF fonctionne pour les messages de **taille fixe** in.

**?** Comment générer un MAC pour des messages de  
**taille arbitraire** ?

Pour chiffrement, solution = cipher modes (CBC, CTR...). Ici aussi ? (spoiler:  
**pas aussi simple**)

# MAC pour de long messages

?



## Premier essai : ECB-MAC

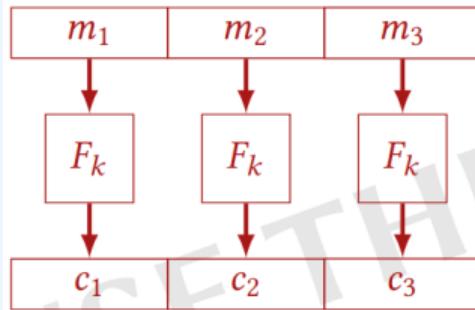
On considère:

```
MAC( $k, m_1 \parallel \dots \parallel m_l$ ):  
return  $F_k(m_1) \parallel \dots \parallel F_k(m_l)$ 
```

Est-ce un MAC sécurisé ?  
(exercice **caséine**)

# MAC pour de long messages

?



## Premier essai : ECB-MAC

On considère:

```
MAC( $k, m_1 \parallel \dots \parallel m_l$ ):  
return  $F_k(m_1) \parallel \dots \parallel F_k(m_l)$ 
```

Est-ce un MAC sécurisé ?

(exercice **caséine**)

✗ Non! (idée: ré-ordonner les blocs)

# MAC pour de long messages

Mode ECB non sécurisé... **pas surprenant !**



# MAC pour de long messages

## Deuxième essai : ECB++MAC

On considère:

?

$\text{MAC}(k, m_0 \| \dots \| m_l):$

**return**  $F_k(0 \| m_0) \| \dots \| F_k(n \| m_l)$

Est-ce un MAC sécurisé ?  
(exercice **caséine**)

# MAC pour de long messages

## Deuxième essai : ECB++MAC

On considère:

?

$\text{MAC}(k, m_0 \| \dots \| m_l):$

**return**  $F_k(0 \| m_0) \| \dots \| F_k(n \| m_l)$

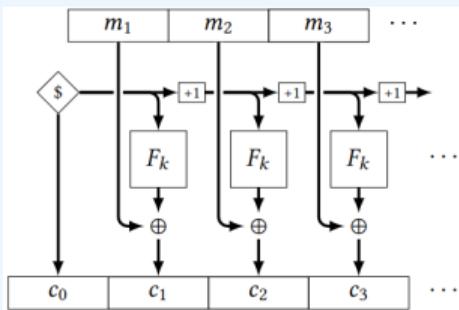
Est-ce un MAC sécurisé ?

(exercice **caséine**)

✗ Non! Idée: mixer les blocs entre plusieurs messages.

# MAC pour de long messages

?



## Troisième essai : CTR-MAC

On considère:

$\text{MAC}(k, m_0 \| \dots \| m_l)$ :

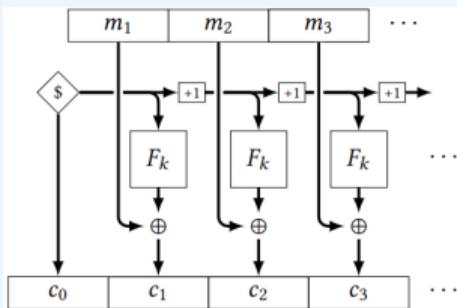
$$c_0 \xleftarrow{\$} \{0, 1\}^\lambda$$

**return**  $c_0 \| F_k(c_0 + 0) \oplus m_0 \| \dots \| F_k(c_0 + n) \oplus m_l$

Est-ce un MAC bien défini ?  
(exercice **caséine**)

# MAC pour de long messages

?



## Troisième essai : CTR-MAC

On considère:

MAC( $k, m_0 \| \dots \| m_l$ ):

$$c_0 \xleftarrow{\$} \{0, 1\}^\lambda$$

**return**  $c_0 \| F_k(c_0 + 0) \oplus m_0 \| \dots \| F_k(c_0 + n) \oplus m_l$

Est-ce un MAC bien défini ?  
(exercice **caséine**)

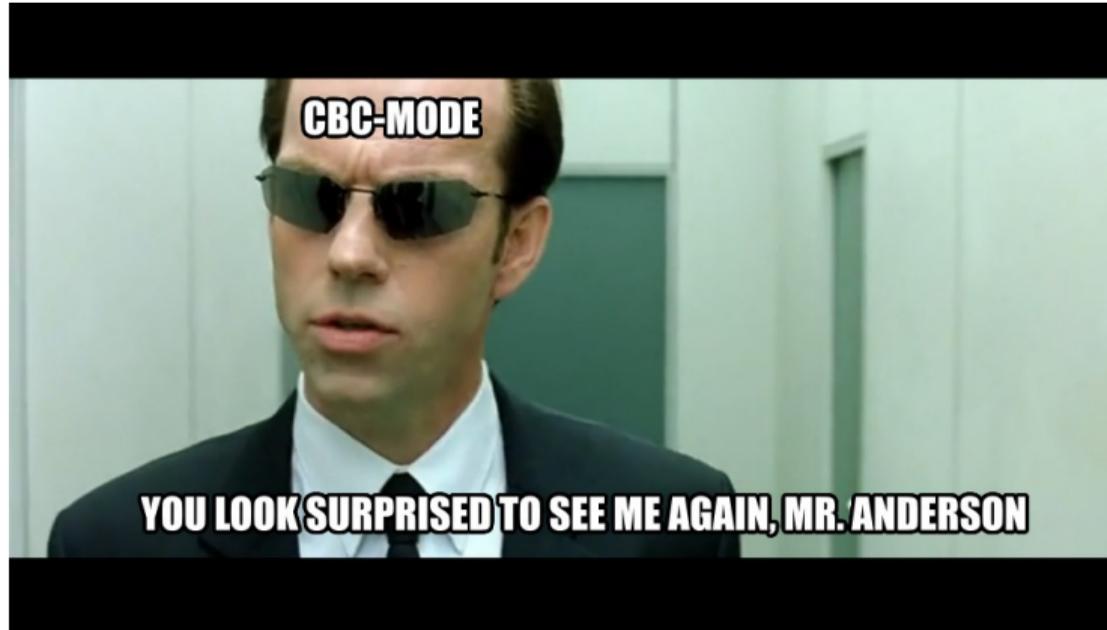
✗ Non, car il n'est pas déterministe !  
Et même s'il l'était (e.g. IV fixe), très simple à casser (même attaque que ECB++-MAC).

# MAC pour de long messages



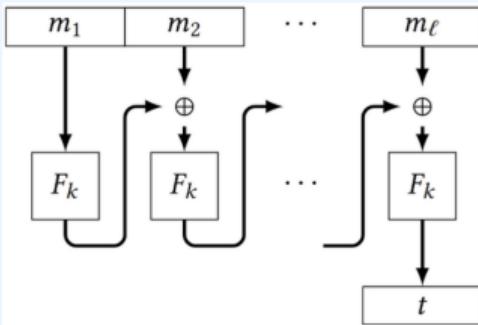
# MAC pour de long messages

Et que donne le **mode CBC**?



# MAC pour de long messages

?



## Quatrième essai : CBC-MAC

On considère:

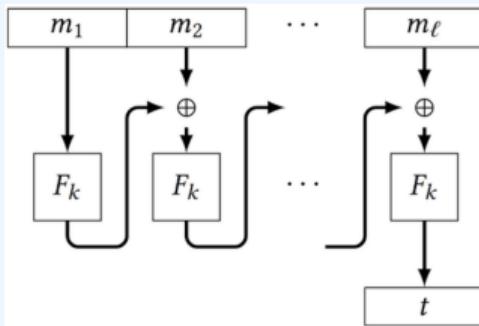
```
MAC( $k, m_1 \| \dots \| m_l$ ):  
   $t := 0^\lambda$   
  for  $i = 1$  to  $l$   
     $t := F_k(m_i \oplus t)$   
  return  $t$ 
```

Est-ce un MAC sécurisé ?

(exercice **caséine** "MAC attaque 4: CBC-MAC")

# MAC pour de long messages

?



## Quatrième essai : CBC-MAC

On considère:

```
MAC( $k, m_1 \| \dots \| m_l$ ):  
   $t := 0^\lambda$   
  for  $i = 1$  to  $l$   
     $t := F_k(m_i \oplus t)$   
  return  $t$ 
```

Est-ce un MAC sécurisé ?

(exercice **caséine** "MAC attaque 4: CBC-MAC")

✓ / ✗ Oui et Non: oui si on ne signe que des messages de même taille, non si on chiffre des messages de taille différente (idée: signer  $m_0$  (tag  $t$ ) et  $t \oplus m_1$ , et le combiner pour avoir un tag de  $m_0 \| m_1$ ).

# MAC pour de long messages

Donc CBC MAC **non sécurisé** car on peut combiner de petits tags pour avoir de gros tags...



# MAC pour de long messages

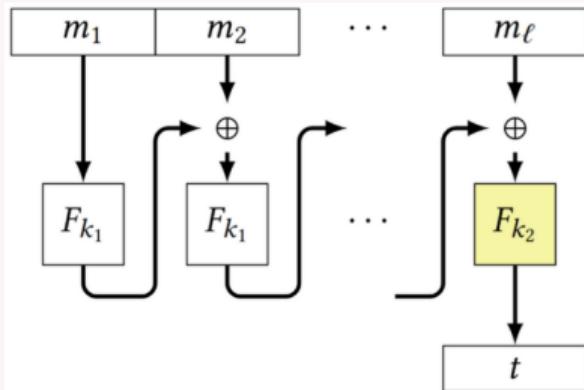
## Solutions:

- ajouter la **taille** du message au début:  
⇒ problème = on doit connaître la taille de message avant le début de la signature, parfois **pas pratique** pour de gros messages (et ajouter la taille à la fin = pas sécurisée)
- utiliser une **autre fonction** à la fin !

# MAC pour de long messages

## Théorème

Soit  $F: \mathcal{K} \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$  une PRF. Le mode ECBC-MAC défini comme:



```
MAC((k1, k2), m1 || ... || ml):  
t := 0λ  
for i = 1 to l - 1  
    t := F_{k1}(m_i ⊕ t)  
return F_{k2}(m_l ⊕ t)
```

est fortement EUF-CMA sécurisé (pour des messages dans  $(\{0, 1\}^\lambda)^*$  avec la construction ci-dessus, et dans  $\{0, 1\}^*$  en utilisant du padding).

# MAC pour de long messages

ECBC-MAC est donc **sécurisé** !



# MAC pour de long messages

ECBC-MAC est sécurisé, mais nécessite **deux clés**: on peut le rendre un peu plus efficace avec **une seule clé** = One-Key CBC-MAC (**OMAC**, or OMAC2), encore un peu amélioré avec OMAC1 (=CMAC), (OMAC étant parfois utilisé pour désigner cette famille).

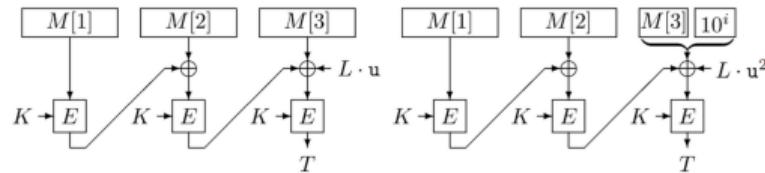


Fig. 2. Illustration of OMAC1. Note that  $L = E_K(0^n)$ .

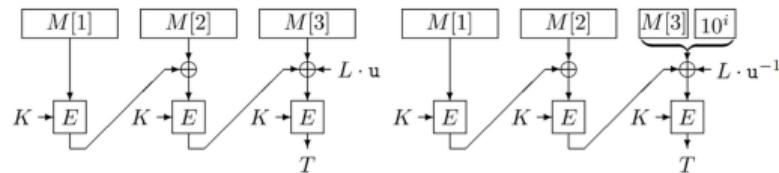


Fig. 3. Illustration of OMAC2.

# MAC à partir de fonctions de hash

# MAC à partir de fonctions de hash

Idées: utiliser des **fonctions de hash** pour faire des MAC ?

- $\text{PrefixMac}_k(m) := H(k\|m)$
- $\text{SuffixMac}_k(m) := H(m\|k)$
- $\text{SandwitchMac}_{k_1\|k_2}(m) := H(k_1\|m\|k_2)$  ("paddées" ?)
- Autre ?



Est-ce vraiment sécurisé ?

# MAC à partir de fonctions de hash

Idées: utiliser des **fonctions de hash** pour faire des MAC ?

- $\text{PrefixMac}_k(m) := H(k\|m)$  ↗ Parfaïs,
- $\text{SuffixMac}_k(m) := H(m\|k)$  ↗ SHA-3 ok  
(faire pour)
- $\text{SandwitchMac}_{k_1\|k_2}(m) := H(k_1\|m\|k_2)$  ("paddées"?)
- Autre ?



Est-ce vraiment sécurisé ?

# MAC à partir de fonctions de hash

Idées: utiliser des **fonctions de hash** pour faire des MAC ?

- $\text{PrefixMac}_k(m) := H(k\|m)$  ↗ Parfois, mais pas toujours !
- $\text{SuffixMac}_k(m) := H(m\|k)$  ↗ SHA-3 ok  
(faire pour)
- $\text{SandwitchMac}_{k_1\|k_2}(m) := H(k_1\|m\|k_2)$  ("paddées" ?)
- Autre ?

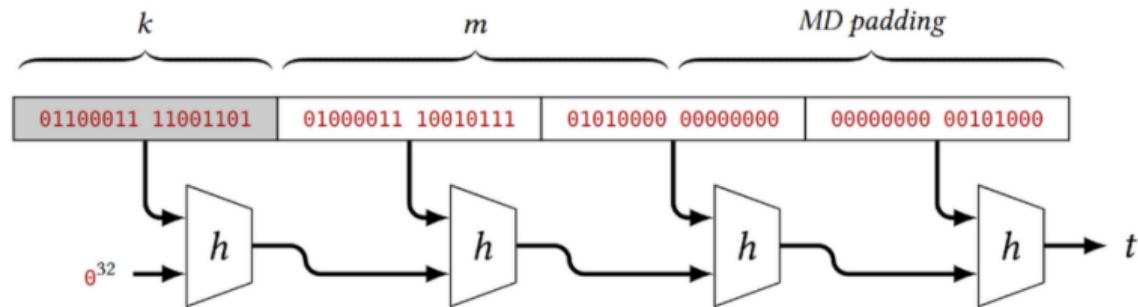
↗ cassé si hash basé sur Merkle-Damgård  
E.g. MD5  
SHA-1  
SHA-2



Est-ce vraiment sécurisé ?

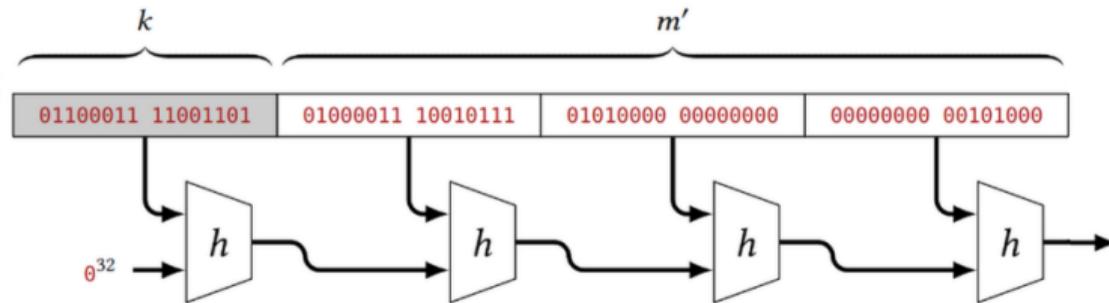
# Attaque par extension de longueur (Length Extension Attack)

Attaque de PrefixMac<sub>k</sub>( $m$ ) :=  $H(k\|m)$  si  $H$  basé sur Merkle-Damgård:



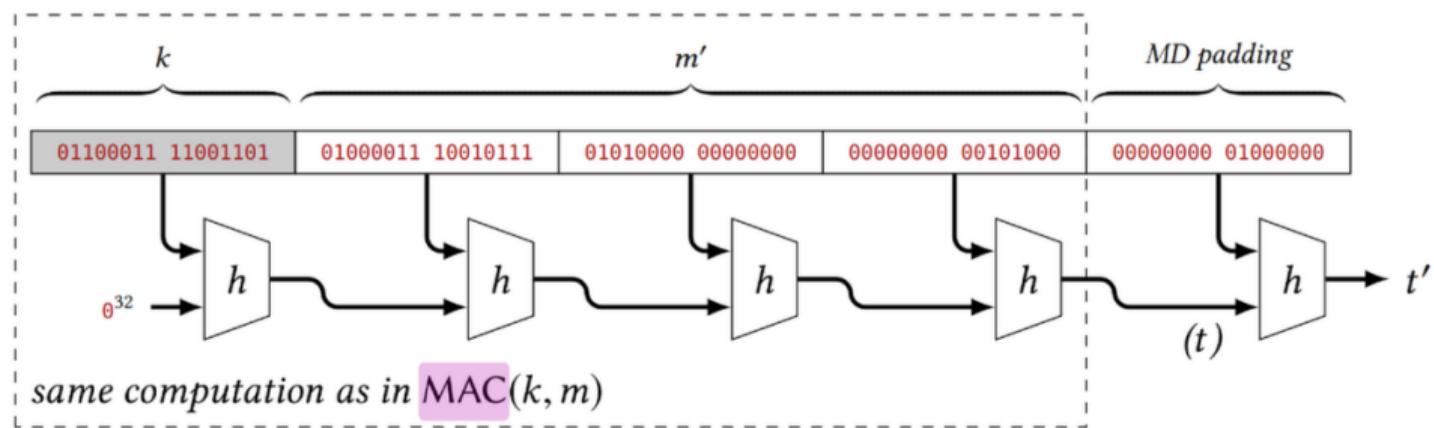
# Attaque par extension de longueur (Length Extension Attack)

Attaque de PrefixMac<sub>k</sub>( $m$ ) :=  $H(k\|m)$  si  $H$  basé sur Merkle-Damgård:



# Attaque par extension de longueur (Length Extension Attack)

Attaque de  $\text{PrefixMac}_k(m) := H(k\|m)$  si  $H$  basé sur Merkle-Damgård:



# Attaque par extension de longueur (Length Extension Attack)



Cette attaque est-elle une forge:

- A universelle
- B existentielle

# Attaque par extension de longueur (Length Extension Attack)

Cette attaque est-elle une forge:



- A universelle
- B existentielle On ne peut signer "que" certains messages, ceux de la forme  $m\|pad_m\|m'$  (ce qui est déjà pas mal...)

# Attaque par extension de longueur (Length Extension Attack)



Implémenter sur caseine l'attaque PrefixMac sur Merkle-Damgård.

TODO

# Attaque par extension de longueur (Length Extension Attack)

Problème : la clé apparaît **avant** + le hash contient **tout l'état interne**  
Solutions ?

- Constructions de hash wide-pipe (on jette une partie de la sortie) ou sponge (cf. cours fonctions de hash): utiliser SHA-3 explicitement prévue pour ça
- Ou **ne pas utiliser PrefixMac**. Mais alors quoi ?

# Attaque par forge de suffix (suffix forgery attack)

Essai 2:  $\text{SuffixMac}_k(m) := H(m\|k)$

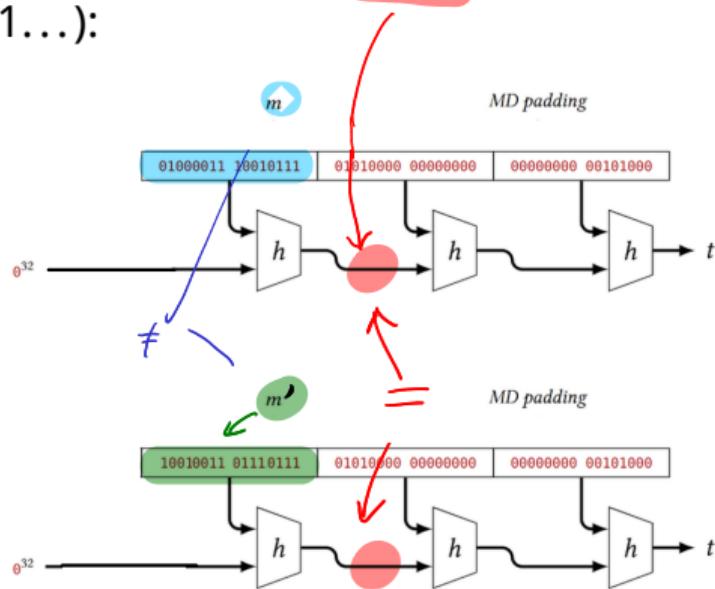
# Attaque par forge de suffix (suffix forgery attack)

Essai 2:  $\text{SuffixMac}_k(m) := H(m\|k)$  si  $H$  basé sur Merkle-Damgård:

# Attaque par forge de suffix (suffix forgery attack)

Essai 2:  $\text{SuffixMac}_k(m) := H(m\|k)$  si  $H$  basé sur Merkle-Damgård:

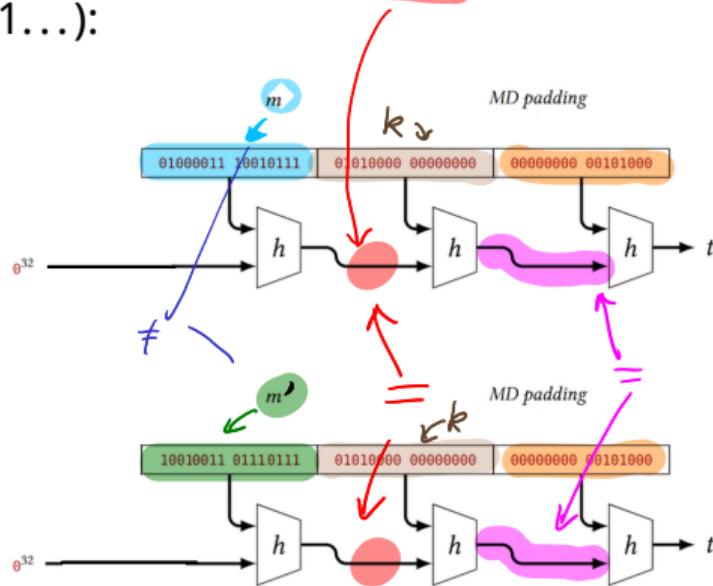
Supposons que l'on connaisse une collision (pas évident, mais connue pour MD5, SHA-0, SHA-1...):



# Attaque par forge de suffix (suffix forgery attack)

Essai 2:  $\text{SuffixMac}_k(m) := H(m\|k)$  si  $H$  basé sur Merkle-Damgård:

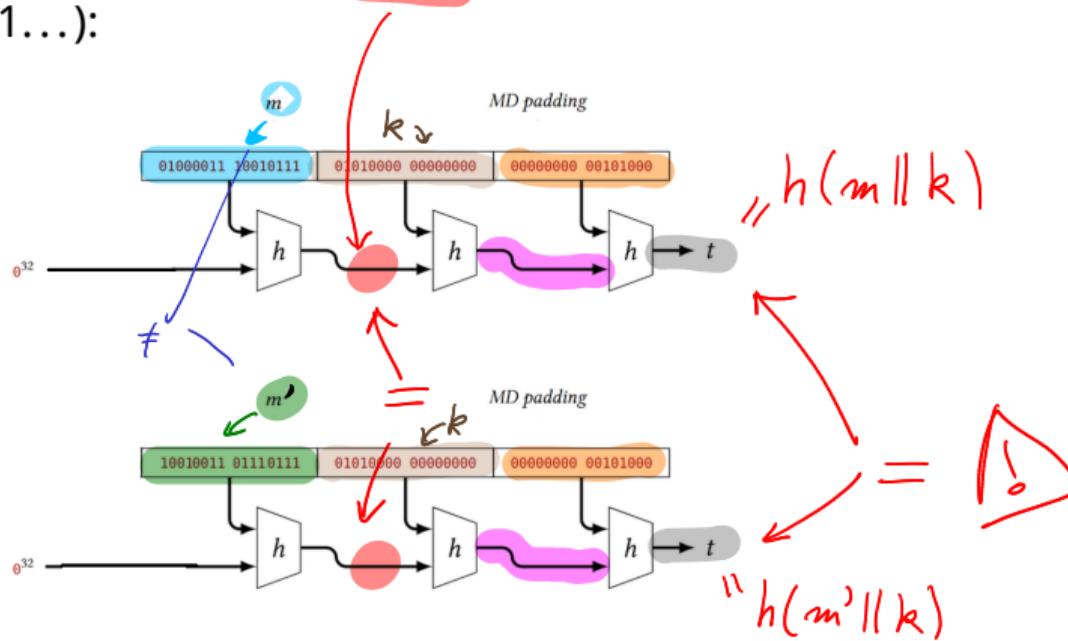
Supposons que l'on connaisse une collision (pas évident, mais connue pour MD5, SHA-0, SHA-1...):



# Attaque par forge de suffix (suffix forgery attack)

Essai 2:  $\text{SuffixMac}_k(m) := H(m\|k)$  si  $H$  basé sur Merkle-Damgård:

Supposons que l'on connaisse une collision (pas évident, mais connue pour MD5, SHA-0, SHA-1...):



# Sécurité y compris si $H$ résistante aux collisions ?

?

Peut-on obtenir un MAC à partir d'une fonction de hachage **vulnérable**, i.e. dont on connaît une collision ?

- Le plus commun (et éprouvé) : **HMAC** (slide suivante)
- Également possible (et prouvé<sup>1</sup>) :  $\text{SandwitchMac}_{k_1\|k_2}(m) := H(k_1\|m\|k_2)$ , mais attention: le message **doit être paddé**<sup>2</sup> pour arriver à une fin de bloc, et chaque clé doit être grande !

(attaque "diviser pour régner" possible sinon, cf <https://crypto.stackexchange.com/questions/15131/>)

- Autre possibilités, e.g. NMAC

---

<sup>1</sup>[https://link.springer.com/chapter/10.1007/978-3-540-73458-1\\_26](https://link.springer.com/chapter/10.1007/978-3-540-73458-1_26)

<sup>2</sup>[https://link.springer.com/chapter/10.1007/3-540-68339-9\\_3](https://link.springer.com/chapter/10.1007/3-540-68339-9_3)

## Définition (HMAC)

HMAC est défini par :

$$\text{HMAC}_k(m) = H\left((k \oplus \text{opad}) \parallel H\left((k \oplus \text{ipad}) \parallel m\right)\right)$$

avec  $\text{ipad} = 0x3636\dots36$  et  $\text{opad} = 0x5c5c\dots5c$  (leur choix est important<sup>a</sup>).

---

<sup>a</sup><https://eprint.iacr.org/2012/684.pdf>

Conclusion: on a besoin d'ipad



pour obtenir un MAC



Coïncidence ? Je ne crois pas...

## Avantages HMAC:

- preuve de sécurité<sup>3</sup>,
- pas besoin de résistance aux collisions,
- fonctionne même si  $H$  basée sur construction de Merkle-Damgård
- et éprouvé par le temps !

---

<sup>3</sup><https://eprint.iacr.org/2006/043.pdf>

# Chiffrement + MAC

## Motivations:

- Motivations pour MAC étaient d'avoir un schéma de chiffrement CCA-secure (attaquants actifs, e.g. padding oracle attack).  
⇒ Comment combiner MAC & Chiffrement pour avoir **sécurité CCA** ?
- Souvent en pratique on veut à la fois chiffrer et authentifier. Peut-on le faire **+ efficacement** que chiffrement + MAC ?

## CCA à partir de MAC et CPA

Soit  $(E.\text{Gen}, E.\text{Enc}, E.\text{Dec}_e)$  un schéma de chiffrement CPA-sécurisé,  $(M.\text{Gen}, M.\text{MAC})$  un MAC (fortement EUF-CMA) sécurisé, alors le schéma de chiffrement suivant "chiffrement-puis-mac" est CCA-sécurisé :

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

$$\mathcal{M} = E.\mathcal{M}$$

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

KeyGen:

$$k_e \leftarrow E.\text{KeyGen}$$

$$k_m \leftarrow M.\text{KeyGen}$$

$$\text{return } (k_e, k_m)$$

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

$$c := E.\text{Enc}(k_e, m)$$

$$t := M.\text{MAC}(k_m, c)$$

return  $(c, t)$

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

if  $t \neq M.\text{MAC}(k_m, c)$ :

return err

return  $E.\text{Dec}(k_e, c)$



Exercice : prouver le théorème précédent.

Exercice simplifié sur **caséine**, avec les jeux pré-remplis. TODO

Typiquement, but d'un canal sécurisé =

- **confidentialité** : le message est caché contre adversaire malicieux
- **authenticité** : tous les messages viennent bien de la personne que l'on souhaite (pas d'ajouts de messages, de modification...)
- **pas de "replay"** : on aimerait éviter les attaques de replay (un adversaire peut envoyer un message déjà vu !)



CCA ne nous protège pas déjà ?

Typiquement, but d'un canal sécurisé =

Il existe des schémas de chiffrement  
(e.g.  $\text{Enc}(k, m) := r \xleftarrow{\$} \{0, 1\}^\lambda; \text{return } E_k(m||r)$ )  
CCA mais où **un attaquant peut envoyer un message aléatoire.**

- **confidentialité** : le message est caché contre adversaire malicieux ✓
- **authenticité** : tous les messages viennent bien de la personne que l'on souhaite (pas d'ajouts de messages, de modification...) ✗
- **pas de "replay"** : on aimerait éviter les attaques de replay (un adversaire peut envoyer un message déjà vu !) ✗

?

CCA ne nous protège pas déjà ?  
⇒ **pas complètement !**

⇒ Besoin d'une meilleure définition:  
**Authenticated Encryption with Associated Data !**  
**(AEAD)**

Limiter replay (ce message est  
le  $n$ -ième message envoyé  
dans ce “contexte” (=session)  $d$ )

Éviter replay = on introduit un “associated data”/**contexte  $d$**  (e.g. ID session & numéro message, hash de tout l'historique de conversation...) identifiant **la connexion actuelle**, et on change le chiffrement et déchiffrement.

$$\text{Enc}(k, d, m) \quad \text{Dec}(k, d, c)$$

⇒ Objectif = **impossible pour un adversaire de générer un chiffré  $(c, d)$  qui n'ait pas déjà été vu**

# AEAD

Note: pour simplifier (et augmenter) la sécurité, on demande **en plus** que le chiffrement soit indistinguable d'un élément aléatoire dans l'espace des chiffrés  $\mathcal{C}$ :

## AEAD

Soit  $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$  un chiffrement. On dit que  $\Sigma$  est un **AEAD\$** sécurisé si :

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

# Construction AEAD

# Construction AEAD

Plusieurs approches possibles :

- combiner chiffrement + MAC : simple, mais moins efficace
- chiffrements “3-en-1” AEAD : plus complexe, mais plus efficace

# Construction AEAD : chiffrement-puis-mac (version AEAD)

Première méthode chiffrement-puis-mac (version AEAD):

Chiffrement + MAC

Soit  $(\text{Gen}_e, \text{Enc}, \text{Dec})$  un chiffrement CPA-sécurisé (resp. CPA\$-sécurisé, i.e. le chiffré est indistinguable d'aléa), et  $(\text{Gen}_m, \text{MAC})$  un MAC sécurisé, alors la construction ci-dessous est un **AEAD sécurisé** (resp. AEAD\$) :

**Gen**( $1^\lambda$ ):

$k_e \leftarrow \text{Gen}_e(1^\lambda)$   
 $k_m \leftarrow \text{Gen}_m(1^\lambda)$   
**return**  $(k_e, k_m)$

**Enc**(( $k_e, k_m$ ),  $d, m$ ):

$c \leftarrow \text{Enc}_{k_e}(m)$   
 $t := \text{MAC}(k_m, d \| c)$   
**return**  $(c, t)$

**Dec**(( $k_e, k_m$ ),  $d, (c, t)$ ):

**if**  $t \neq \text{MAC}(k_m, d \| c)$   
    **return** **err**  
**return**  $\text{Dec}_{k_e}(c)$

*Idée preuve* : Similaire à la preuve que “chiffrement-puis-mac” est CCA-sécurisé.

## Construction AEAD : inefficacité chiffrement + MAC

Si on instancie chiffrement + MAC avec le chiffrement CBC et le MAC CBC-MAC, on appelle le block-cipher **2× par bloc** !

## Construction AEAD : inefficacité chiffrement + MAC

Si on instancie chiffrement + MAC avec le chiffrement CBC et le MAC CBC-MAC, on appelle le block-cipher **2× par bloc** !

⇒ **inefficace** ! (on fait **2× quasiment le même travail**)

# Construction AEAD : inefficacité chiffrement + MAC

Si on instancie chiffrement + MAC avec le chiffrement CBC et le MAC CBC-MAC, on appelle le block-cipher **2× par bloc** !

⇒ **inefficace** ! (on fait **2× quasiment le même travail**)



Peut-on faire mieux ?

# Construction AEAD : inefficacité chiffrement + MAC

Si on instancie chiffrement + MAC avec le chiffrement CBC et le MAC CBC-MAC, on appelle le block-cipher **2× par bloc** !

⇒ **inefficace** ! (on fait **2× quasiment le même travail**)

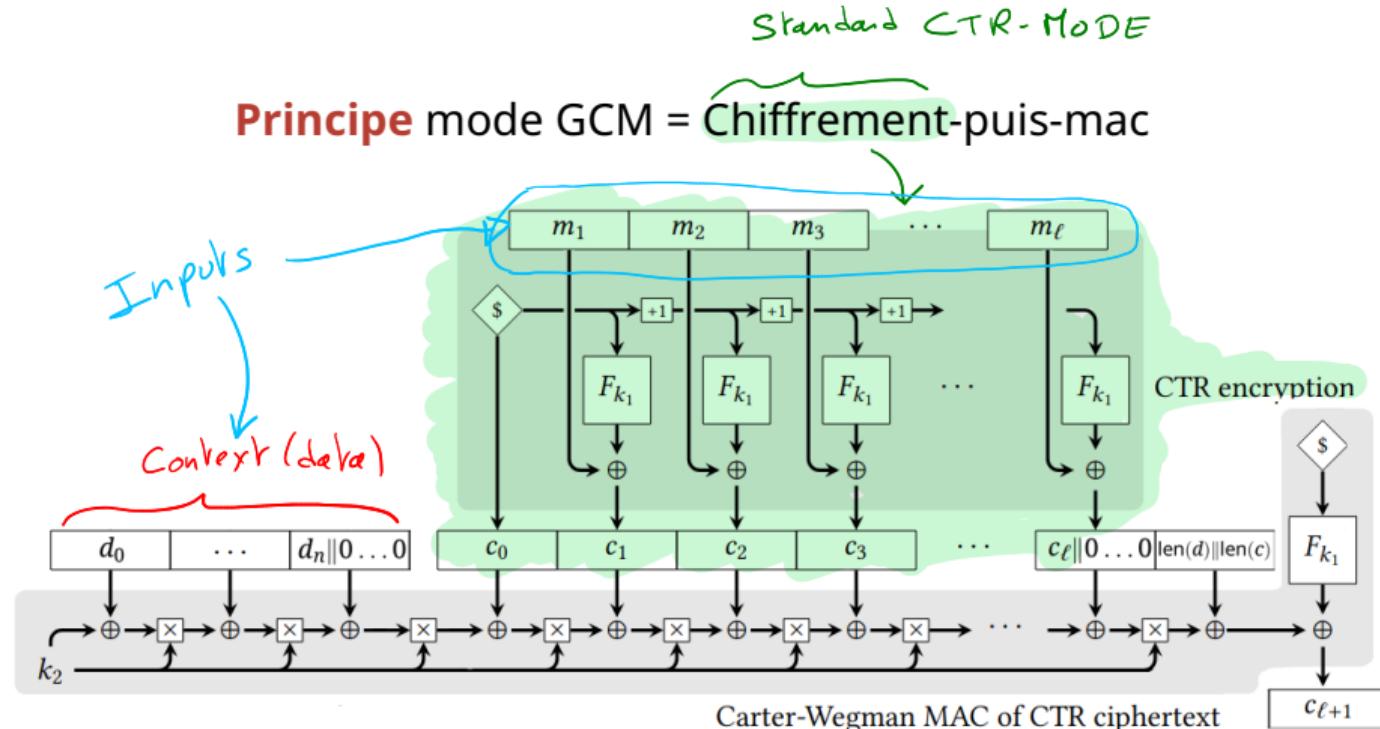


Peut-on faire mieux ?

✓ **Oui : Mode GCM !**

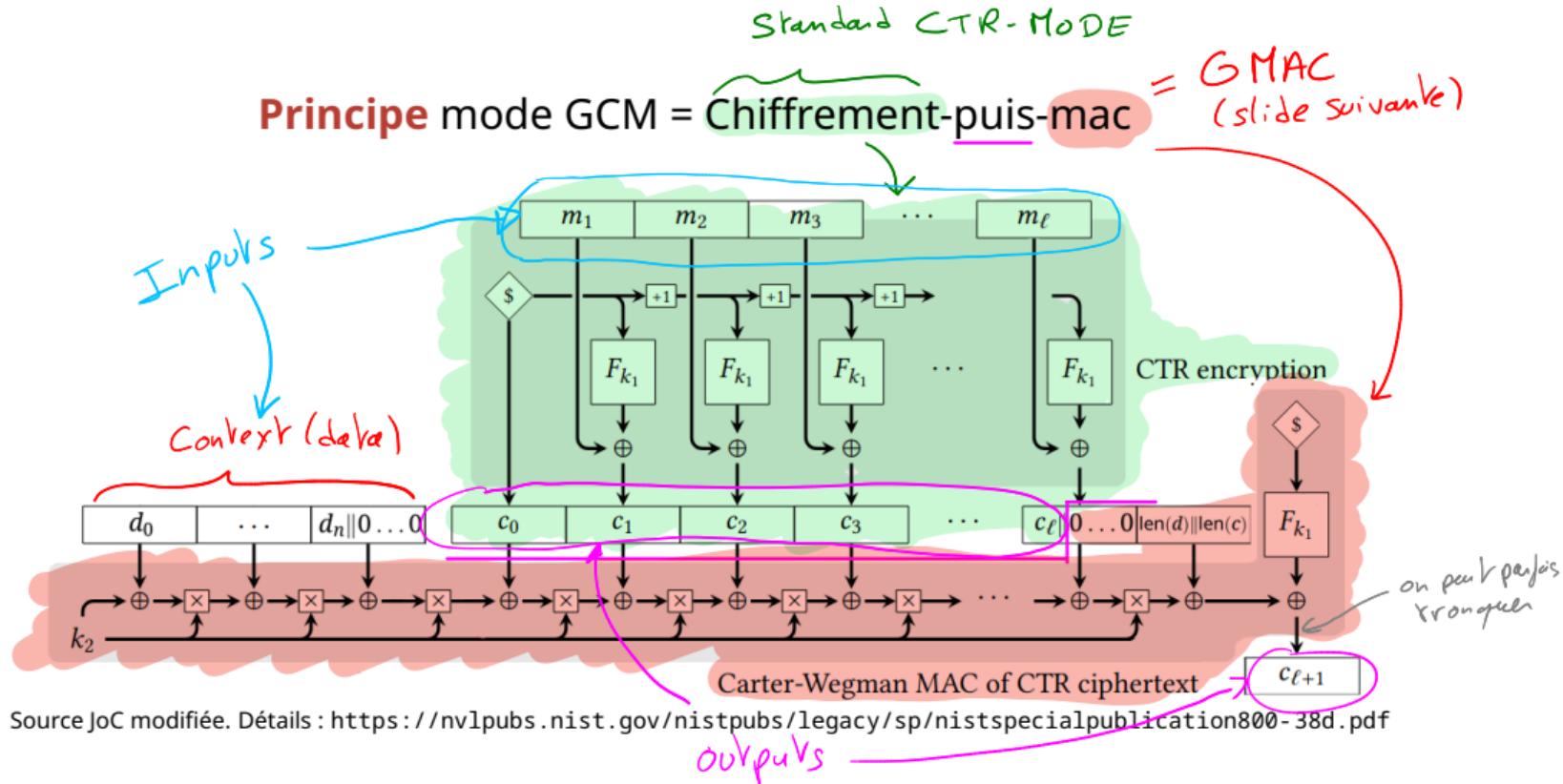
# Mode GCM

# Mode GCM



Source JoC modifiée. Détails : <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf>

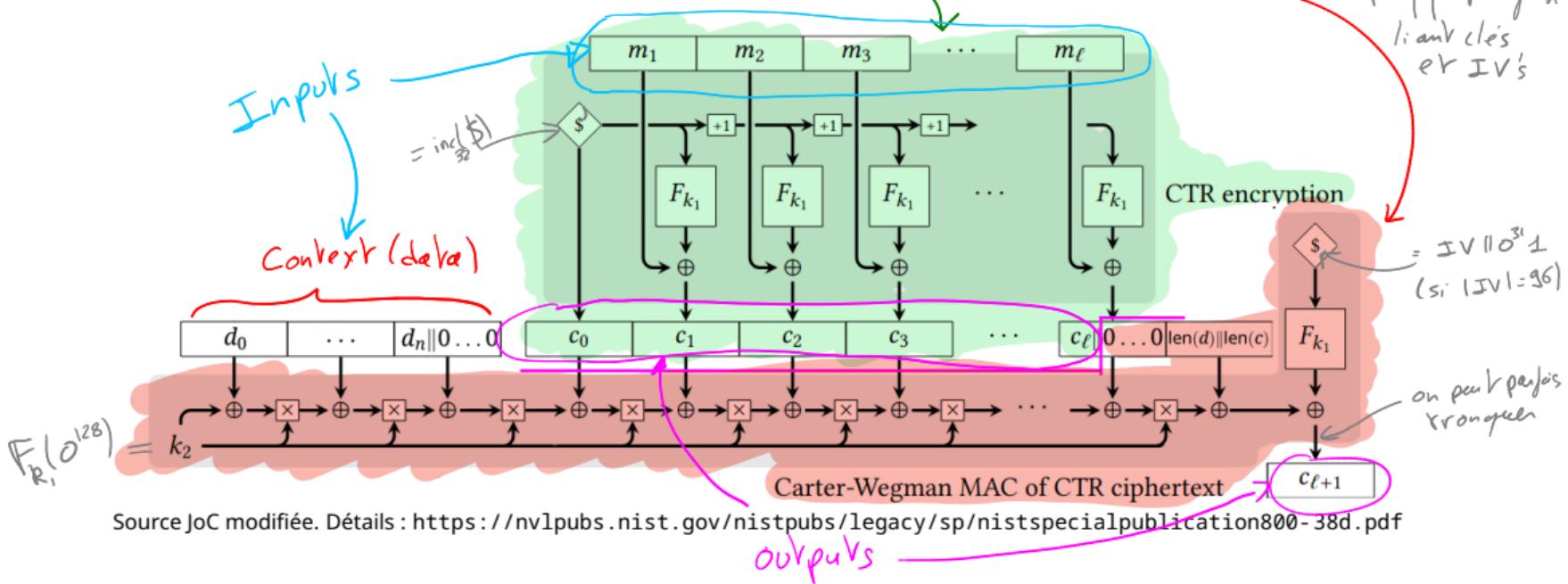
# Mode GCM



Mode GCM

Standard CTR-Mode

= GMAC  
(slide suivante)



Source JoC modifiée. Détails : <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf>

**GMAC** (Carter-Wegman MAC) : construction de MAC qui n'utilise **qu'un appel au cipher**, sinon que des multiplications !  
⇒ beaucoup **plus efficace** !

Construction GMAC = 2 étapes :

simple évaluation d'un polynôme  $\sum_{i=0}^l c_{l-i}s^i$  (s = salt)  
sur un corps fini où les calculs sont efficaces

- ① On utilise une fonction de **hash universelle** = très efficace, mais **très peu sécurisée** :  
(= Résistante aux collisions ssi on ne connaît pas le salt + 1 seul essai !)
- ② On applique un pseudo-OTP à la fin sur le résultat (donc 1 seul appel au block-cipher !) pour **booster** la sécurité en "cachant" la sortie de la fonction, et donc son sel  $s = k_2$  (si révélé, on peut signer tout ce que l'on veut !)  
⇒ c'est une PRF et donc un MAC !



On vient de construire une PRF... mais avec le block-cipher utilisé  
on avait déjà une PRF. Quel avantage ?



On vient de construire une PRF... mais avec le block-cipher utilisé  
on avait déjà une PRF. Quel avantage ?

⇒ Ici on a fabriqué une PRF pour des entrées **de taille non-bornée** ! (block-cipher = taille fixe)

La fonction universelle calcule seulement

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

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

Comment le faire **éfficacement** ?

La fonction universelle calcule seulement

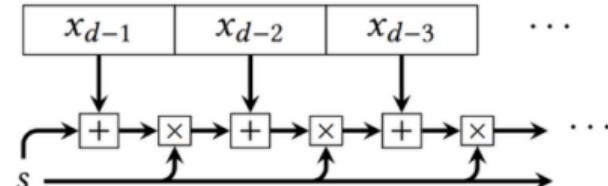
$$\sum_{i=0}^l x_{l-i} s^l$$

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

Comment le faire **éfficacement** ?

⇒ **Méthode de Ruffini-Horner**

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



?

Une **multiplication** entre des listes de bits...?!?





## Une **multiplication** entre des listes de bits...?!?

⇒ Messages interprétés comme des éléments de  $\mathbb{F}_{2^{128}}$ :

- $+$  = XOR bit à bit
- $\times$  = chaque élément  $a = a_0 \dots a_{127} \in \{0, 1\}^{127}$  vu comme polynôme  $a_0 + a_1X + a_2X^2 \dots a_{127}X^{127} \in \mathbb{Z}_2[X]$ , on multiplie alors les polynômes, et on simplifie (pour garder 128 bits) en considérant que  $X^{128} = 1 + X + X^2 + X^7$  (i.e. modulo un polynôme irréductible).



Combien vaut  $110\dots01 \times 1010\dots0$  ?

# En pratique

# Et en pratique ?

## En pratique:

- CBC-MAC est utilisé dans le mode AEAD CCM, lui même utilisé dans IEEE 802.11i, IPsec, TLS 1.2 & 1.3 (désactivé dans 1.3 par défaut dans openssl), Bluetooth Low Energy (4.0).
- OMAC est utilisé dans le mode AEAD EAX (remplaçant CCM)
- AEAD **GCM très adopté** (efficacité), utilisé dans 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: utilisé dans IPsec, TLS, JWT Json Web Tokens (RFC 7519)...
- Poly1305 (=hash qui peut être utilisé en MAC) utilisé dans le AEAD ChaCha20-Poly1305, lui même utilisé dans IPsec, SSH, (D)TLS 1.2 & 1.3, WireGuard, S/MIME 4.0, OTRv4... Très rapide en software, remplace souvent GCM quand il n'y a pas d'instructions hardware

# Conclusion

# Conclusion

- **MAC permet de “signer” (=tagger) un message** si on partage une clé privée avec le destinataire
- On peut **formaliser la sécurité** avec un jeu visant à forger de nouveaux tags ( $\approx$  signatures)  $\Rightarrow$  (fortement) EUF-CMA sécurisé
- On peut **construire des MAC** sécurisés à partir de:
  - block-ciphers (attention, très différent du chiffrement !)
  - fonctions de hash, mais attention attaques si mal fait (length extension attack...)
- Chiffrement-puis-MAC permet d'avoir **sécurité CCA**
- Mais CCA n'est pas suffisant (attaques replay etc)  
 $\Rightarrow$  on définit **AEAD** (encore + sécurisé) en introduisant la notion de *contexte*
- Chiffrement-puis-MAC est AEAD sécurisé... mais on peut gagner en **efficacité** avec le **mode GCM**, très utilisé