Cryptographie

Authentification symétrique

Léo Colisson Palais
Master CSI 2024 – 2025

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

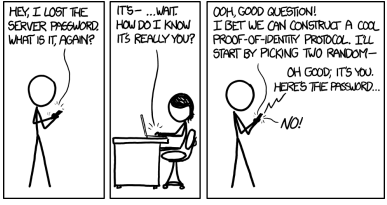
https://leo.colisson.me/teaching.html

Authentification: 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 connaitre la clé publique du signataire

Authentification

Comme chiffrement, 2 grandes familles :

ce cours !

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 connaitre 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 :

- $\mathrm{Gen}(1^\lambda)$ qui renvoit une clé secrète k
- $\mathrm{MAC}(k, m)$ 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 :

- $\mathrm{Gen}(1^\lambda)$ qui renvoit une clé secrète k
- $\mathrm{MAC}(k, m)$ 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 $\mathrm{MAC}(k, m) \stackrel{?}{=} t$!

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 connaitre la clé k.
Comment formaliser cette idée ?

Comment formaliser la sécurité ?

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

Est-il vrai que:

?

$\mathcal{L}_{\text{guess-r}}$

$r \leftarrow \text{Gen}(1^\lambda)$

GUESS(x):

return $x \stackrel{?}{=} x$

$\approx$

$\mathcal{L}_{\text{guess-false}}$

GUESS(x):

return `false`

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:

$\mathcal{L}_{\text{guess-r}}$			$\mathcal{L}_{\text{guess-false}}$
$r \leftarrow \text{Gen}(1^\lambda)$ $\underline{\text{GUESS}(x):}$ **return** $x \stackrel{?}{=} x$		$\approx$	$\underline{\text{GUESS}(x):}$ **return** `false`

?

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:

$\mathcal{L}_{\text{mac-1}}$

$r \leftarrow \text{Gen}(1^\lambda)$

CHECKTAG(m, t):

return $\text{MAC}(k, m) \stackrel{?}{=} t$

$\approx$

$\mathcal{L}_{\text{false}}$

CHECKTAG(m, t):

return `false`

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:

$\mathcal{L}_{\text{mac-1}}$

$r \leftarrow \text{Gen}(1^\lambda)$

CHECKTAG(m, t):

return MAC$(k, m) \stackrel{?}{=} t$

$\approx$

$\mathcal{L}_{\text{false}}$

CHECKTAG(m, t):

return `false`

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, 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:

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

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:

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

?

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:

$\mathcal{L}_{\text{mac-1}}$

$r \leftarrow \text{Gen}(1^\lambda)$

GETTAG(m):

return MAC(k, m)

CHECKTAG(m, t):

return MAC$(k, m) \stackrel{?}{=} t$

$\approx$

$\mathcal{L}_{\text{mac-1-false}}$

$r \leftarrow \text{Gen}(1^\lambda)$

GETTAG(m):

return MAC(k, m)

CHECKTAG(m, t):

return `false`

?

Est-ce une bonne idée?

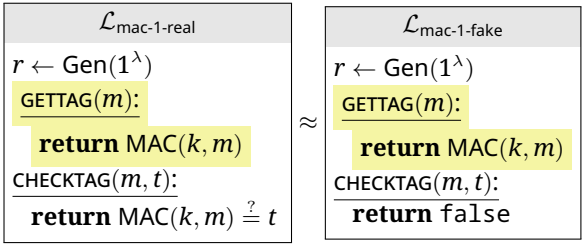
A. Non, car on peut toujours distinguer ces libraries
   ✓ Essayez de trouver une attaque (exercice Caseine “MAC > MAC > MAC mauvaise definition”) ⇒ Solution: CHECKTAG(`"hello"`, GETTAG(`"hello"`))

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:



$=$ 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 généner un TAG **jamais vu avant**:

Définition (EUF-CMA-)

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

$\mathcal{L}_{\text{mac-real}}$

$r \leftarrow \mathsf{Gen}(1^\lambda)$

$\underline{\textsc{GetTag}(m)}$:

return $\mathsf{MAC}(k, m)$

$\underline{\textsc{CheckTag}(m, t)}$:

return $\mathsf{MAC}(k, m) \stackrel{?}{=} t$

$\approx$

$\mathcal{L}_{\text{mac-fake}}$

$r \leftarrow \mathsf{Gen}(1^\lambda)$

$\mathcal{T} := \emptyset$

$\textsc{GetTag}(m)$:

$t := \mathsf{MAC}(k, m)$

$\mathcal{T} := \mathcal{T} \cup \{(m, t)\}$

return t

$\underline{\textsc{CheckTag}(m, t)}$:

return $(m, t) \in \mathcal{T}$

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

?

$\mathcal{L}_{\text{mac-real}}$

$r \leftarrow \mathsf{Gen}(1^\lambda)$
$\underline{\mathsf{GETTAG}(m):}$
  return $\mathsf{MAC}(k,m)$
$\underline{\mathsf{CHECKTAG}(m,t):}$
  return $\mathsf{MAC}(k,m) \stackrel{?}{=} t$

$\approx$

$\mathcal{L}_{\text{mac-fake}}$

$r \leftarrow \mathsf{Gen}(1^\lambda)$
$\mathcal{T} := \emptyset$
$\underline{\mathsf{GETTAG}(m):}$
  $t := \mathsf{MAC}(k,m)$
  $\mathcal{T} := \mathcal{T} \cup \{(m,t)\}$
  return t
$\underline{\mathsf{CHECKTAG}(m,t):}$
  return $(m,t) \in \mathcal{T}$

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 $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ = 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 $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ = 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

MAC à partir de PRF

Rappel: Une PRF $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ = 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**:
  $\Rightarrow$ difficile de trouver $F_k(x)$!
  $\Rightarrow$ **Bon candidat** pour un MAC !

C. Non

MAC à partir de PRF

Et en effet:

PRF + longue sortie = MAC

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

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 $\textsc{gettag}(m)$ 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 $\frac{1}{2^\lambda} = \mathsf{negl}(\lambda)$ 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

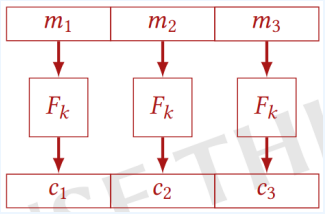
Problème : La méthode PRF fonctionne pour les messages de **taille fixe** in.

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

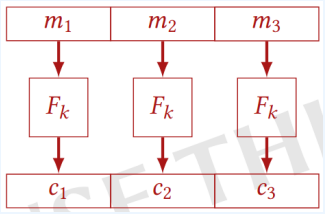
On considère:

$$\frac{\text{MAC}(k, m_1 \| \ldots \| m_l):}{\textbf{return } F_k(m_1) \| \ldots \| F_k(m_l)}$$

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

MAC pour de long messages

?



Premier essai : ECB-MAC

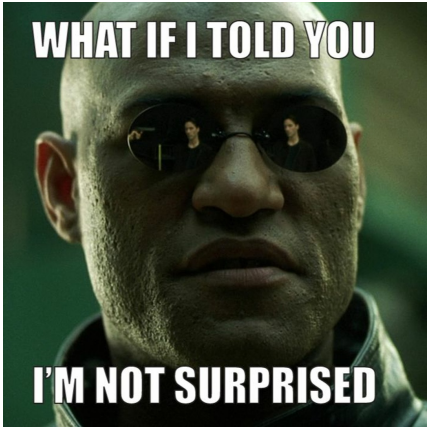
On considère:

$$\frac{\text{MAC}(k, m_1 \| \ldots \| m_l):}{\textbf{return } F_k(m_1) \| \ldots \| F_k(m_l)}$$

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:

?

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

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

MAC pour de long messages

Deuxième essai : ECB++MAC

On considère:

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

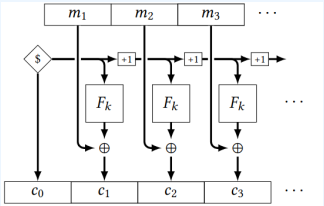
Est-ce un MAC sécurisé ?

(exercice **caséine**)

 Non! Idée: mixer les blocs entre plusieurs messages.

MAC pour de long messages

?



Troisième essai : CTR-MAC

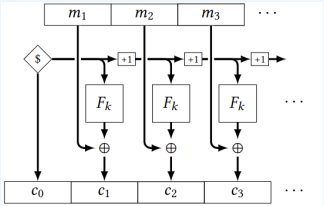
On considère:

$\underline{\text{MAC}(k, m_0\| \ldots \|m_l):}$
$c_0 \xleftarrow{\$} \{0,1\}^\lambda$
return $c_0 \| F_k(c_0+0) \oplus m_0 \| \ldots$
$\ldots \| F_k(c_0+n) \oplus m_l$

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

MAC pour de long messages

?



Troisième essai : CTR-MAC

On considère:

$\underline{\text{MAC}(k, m_0\| \ldots \|m_l):}$
$c_0 \xleftarrow{\$} \{0,1\}^\lambda$
return $c_0 \| F_k(c_0+0) \oplus m_0 \| \ldots$
$\ldots \| F_k(c_0+n) \oplus m_l$

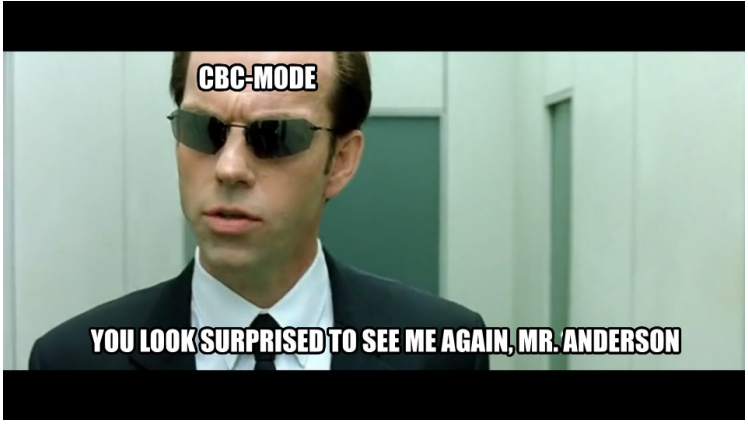
Est-ce un MAC bien défini ?
(exercice **caséine**)

❌ 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

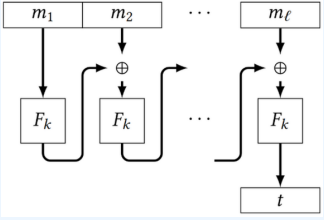
MAC pour de long messages

Et que donne le **mode CBC**?



MAC pour de long messages

?



Quatrième essai : CBC-MAC

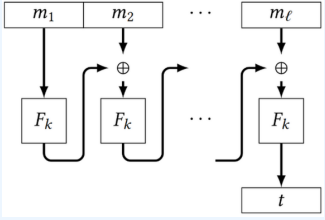
On considère:

$\mathrm{MAC}(k, m_1 \| \ldots \| m_l)$:
  $t := 0^\lambda$
  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:

$\mathrm{MAC}(k, m_1 \| \ldots \| m_l)$:
$t := 0^\lambda$
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

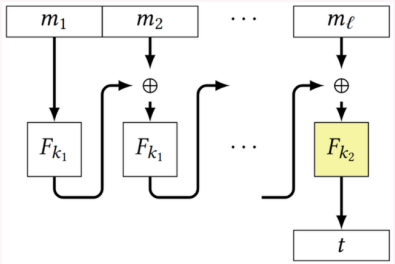
Solutions:

- ajouter la **taille** du message au début:
  $\Rightarrow$ 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 $F\colon \mathcal{K} \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ une PRF. Le mode ECBC-MAC défini comme:

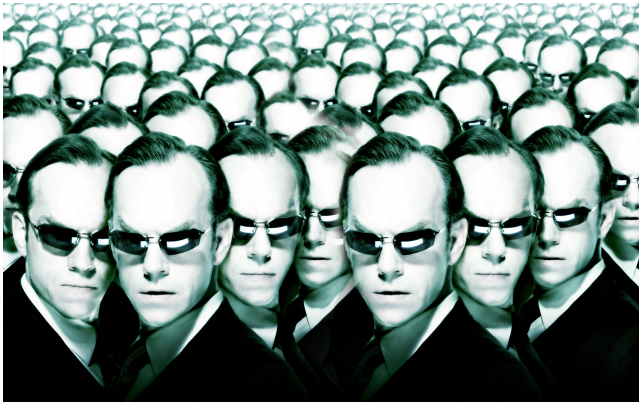


$\mathsf{MAC}((k_1,k_2), m_1 \| \ldots \| m_l)$:
$t \coloneqq 0^\lambda$
for $i = 1$ to $l-1$
$t \coloneqq F_{k_1}(m_i \oplus t)$
return $F_{k_2}(m_l \oplus t)$

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).

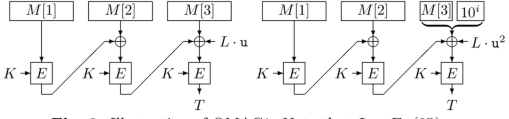


Fig. 2. Illustration of OMAC1. Note that $L = E_K(0^n)$.

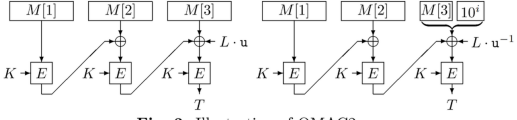


Fig. 3. Illustration of OMAC2.

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

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 ?

- $\text{PrefixMac}_k(m) := H(k \| m)$ → Parfois, ↳ SHA-3 ok (pair pour)
- $\text{SuffixMac}_k(m) := H(m \| k)$
- $\text{SandwitchMac}_{k_1 \| k_2}(m) := H(k_1 \| m \| k_2)$ (“paddées” ?)
- Autre ?

Est-ce vraiment sécurisé ?

MAC à partir de fonctions de hash

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

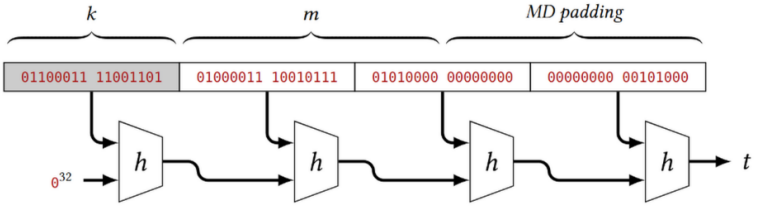
- $\text{PrefixMac}_k(m) := H(k \| m)$
- $\text{SuffixMac}_k(m) := H(m \| k)$
- $\text{SandwitchMac}_{k_1 \| k_2}(m) := H(k_1 \| m \| k_2)$ (“paddées” ?)
- Autre ?

Parfois, mais pas toujours
↳ SHA-3 ok (fait pour)
↳ cassé si hash basé sur Merkle-Damgård
E.g. MD5, SHA-1, SHA-2

? Est-ce vraiment sécurisé ?

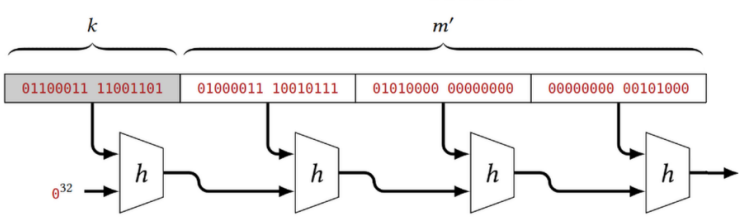
Attaque par extension de longueur (Length Extension Attack)

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



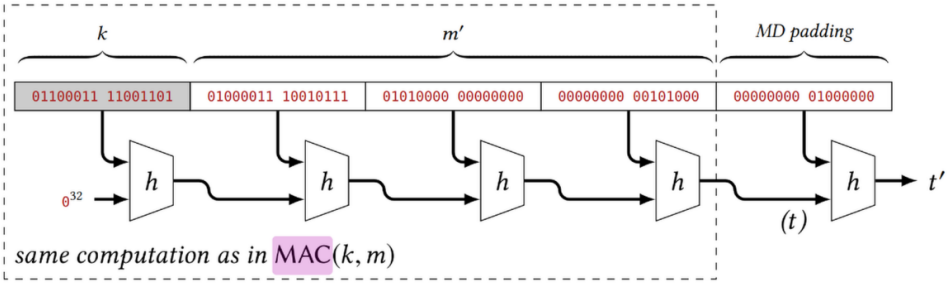
Attaque par extension de longueur (Length Extension Attack)

Attaque de $\text{PrefixMac}_k(m) \coloneqq H(k \| m)$ 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)

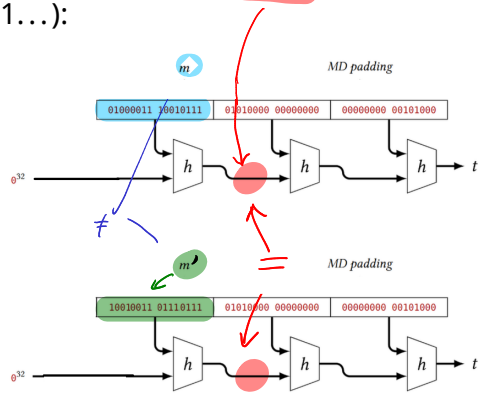
Essai 2: $\text{SuffixMac}_k(m) \coloneqq H(m \| k)$

Attaque par forge de suffix (suffix forgery attack)

Essai 2: SuffixMac$_k(m) \coloneqq H(m\|k)$ si H basé sur Merkle-Damgård:

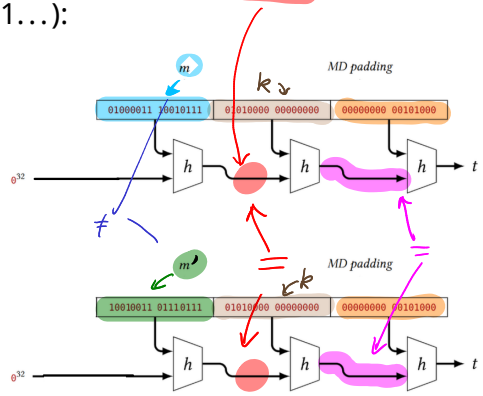
Attaque par forge de suffix (suffix forgery attack)

Essai 2: SuffixMac$_k(m) \coloneqq H(m\|k)$ si H basé sur Merkle-Damgård:
Supposons que l'on connaisse une collision (pas évident, mais connue pour MD5, SHA-0, SHA-1…):



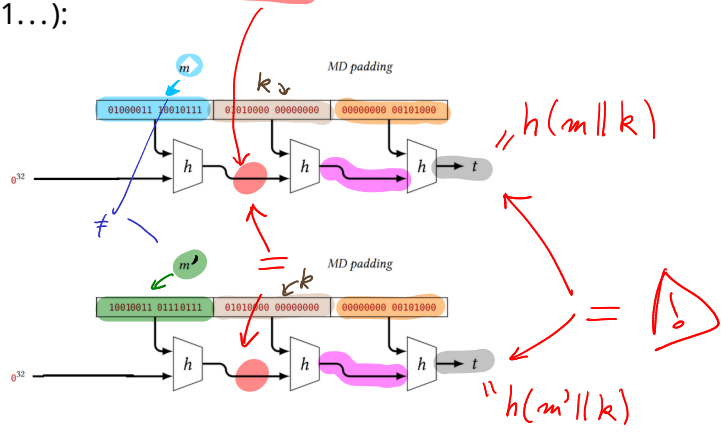
Attaque par forge de suffix (suffix forgery attack)

Essai 2: $\text{SuffixMac}_k(m) \coloneqq H(m \| k)$ 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: $\text{SuffixMac}_k(m) \coloneqq H(m \| k)$ 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[1]) : SandwitchMac$_{k_1 \| k_2}(m) \coloneqq H(k_1 \| m \| k_2)$, mais attention: le message **doit être paddé**[2] 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

[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`

Définition (HMAC)

HMAC est défini par :

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

avec ipad $=$ `0x3636...36` et opad $=$ `0x5c5c...5c` (leur choix est important[a]).

[a] 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[3],
- pas besoin de résistance aux collisions,
- fonctionne même si H basée sur construction de Merkle-Damgård
- et éprouvé par le temps !

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

Chiffrement + MAC

Chiffrement + MAC

Motivations:

- Motivations pour MAC était d'avoir un schéma de chiffrement CCA-secure (attaquants actifs, e.g. padding oracle attack).
  $\Rightarrow$ 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 ?

Sécurité CCA

CCA à partir de MAC et CPA

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

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

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

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

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

Sécurité CCA

? Exercice : prouver le théorème précédent.
Exercice simplifié sur **caséine**, avec les jeux pré-remplis à ordonner (section MAC, activité “CCA à partir de MAC et CPA”).

Sécurité CCA

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 ?

Sécurité CCA

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

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 ?

$\Rightarrow$ **pas complètement !**

AEAD

$\Rightarrow$ **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)

AEAD

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.

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

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

$\mathcal{L}^{\Sigma}_{\text{aead\$-real}}$

$k \leftarrow \mathsf{Gen}(1^{\lambda})$
$\mathcal{S} := \emptyset$
$\mathsf{CTXT}(d, m)$:
  $c \leftarrow \mathsf{Enc}(k, d, m)$
  $\mathcal{S} := \mathcal{S} \cup \{(d, c)\}$
  return c
$\mathsf{DECRYPT}(c \in \mathcal{C})$:
  if $(d, c) \in \mathcal{S}$ return `err`
  return $\mathsf{Dec}(k, d, c)$

$\approx$

$\mathcal{L}^{\Sigma}_{\text{aead\$-real}}$

$\mathsf{CTXT}(d, m)$:
  $c \xleftarrow{\$} \mathcal{C}$
  return c
$\mathsf{DECRYPT}(c \in \mathcal{C})$:
  return `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 $(\mathsf{Gen_e}, \mathsf{Enc}, \mathsf{Dec})$ un chiffrement CPA-sécurisé (resp. CPA$-sécurisé, i.e. le chiffré est indistinguable d'aléa), et $(\mathsf{Gen_m}, \mathsf{MAC})$ un MAC sécurisé, alors la construction ci-dessous est un **AEAD sécurisé** (resp. AEAD$) :

$\underline{\mathsf{Gen}(1^\lambda)}$:
$k_\mathsf{e} \leftarrow \mathsf{Gen_e}(1^\lambda)$
$k_\mathsf{m} \leftarrow \mathsf{Gen_m}(1^\lambda)$
return $(k_\mathsf{e}, k_\mathsf{m})$

$\underline{\mathsf{Enc}((k_\mathsf{e}, k_\mathsf{m}), d, m)}$:
$c \leftarrow \mathsf{Enc}_{k_\mathsf{e}}(m)$
$t := \mathsf{MAC}(k_\mathsf{m}, d \| c)$
return (c, t)

$\underline{\mathsf{Dec}((k_\mathsf{e}, k_\mathsf{m}), d, (c, t))}$:
if $t \neq \mathsf{MAC}(k_\mathsf{m}, d \| c)$
return `err`
return $\mathsf{Dec}_{k_\mathsf{e}}(c)$

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 $2\times$ **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 $2\times$ **par bloc** !

$\Rightarrow$ **inefficace** ! (on fait $2\times$ **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 $2\times$ **par bloc** !

$\Rightarrow$ **inefficace** ! (on fait $2\times$ **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 $2\times$ **par bloc** !

$\Rightarrow$ **inefficace** ! (on fait $2\times$ **quasiment le même travail**)

Peut-on faire mieux ?

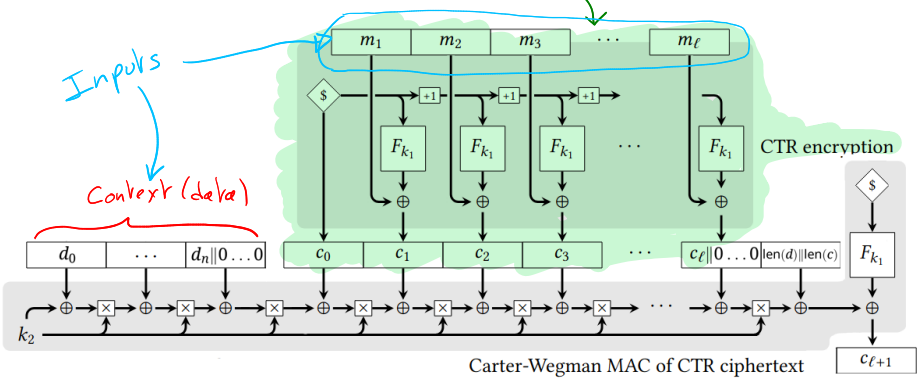
✔ **Oui : Mode GCM !**

Mode GCM

Mode GCM

Standard CTR-MODE

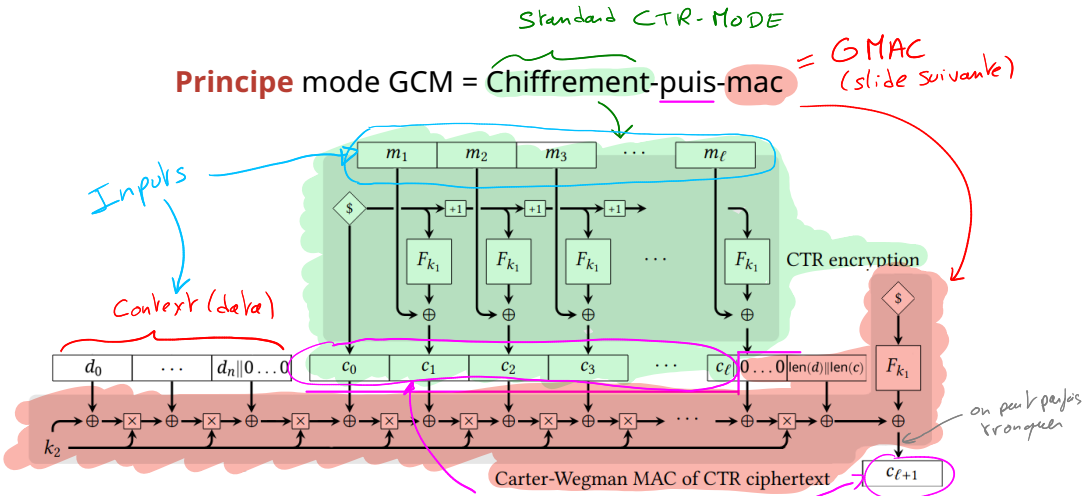
Principe mode GCM = Chiffrement-puis-mac



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

Mode GCM

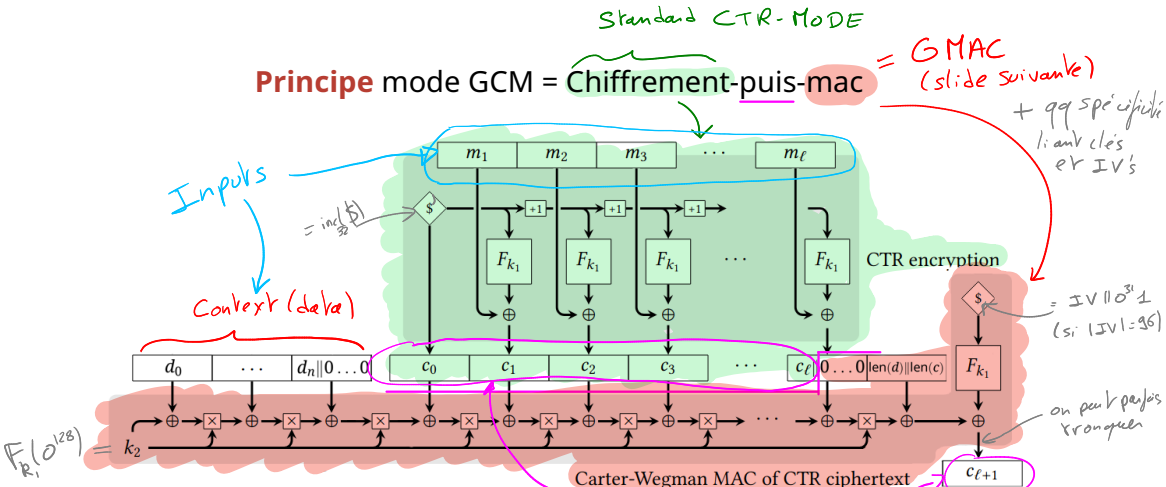
Principe mode GCM = Chiffrement-puis-mac



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

Mode GCM

Principe mode GCM = Chiffrement-puis-mac



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

GMAC

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

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

Construction GMAC = 2 étapes :

1. On utilise une fonction de **hash universelle** = très efficace, mais **très peu sécurisée** :
   (= Résistante aux collisions ssi on ne connait 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 !)
   $\Rightarrow$ c'est une PRF et donc un MAC !

GMAC

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

GMAC

?

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

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

GMAC

La fonction universelle calcule seulement

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

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

Comment le faire **efficacement** ?

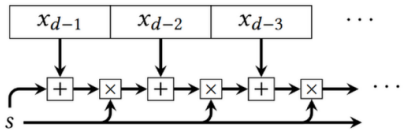
La fonction universelle calcule seulement

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

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

Comment le faire **efficacement** ?
$\Rightarrow$ **Méthode de Ruffini-Horner**

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



?

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

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

$\Rightarrow$ 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 $a_0 + a_1 X + a_2 X^2 \dots a_{127} X^{127} \in \mathbb{Z}_2[X]$, 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