

Chapitre 3 : Intégrité des Messages

MAC, Hachage et Authenticated Encryption

Cours de Sécurité Informatique - Niveau Universitaire
Partie 1 : Cryptographie à Clé Secrète

12 janvier 2026

Table des matières

1 Introduction

Problème : Le chiffrement seul (chapitres 1-2) garantit la **confidentialité** mais pas l'**intégrité**.
Un attaquant peut modifier les ciphertexts !

Objectifs :

- Détecter toute modification des messages
- Authentifier la source du message
- Combiner confidentialité + intégrité (Authenticated Encryption)

Outils :

1. Message Authentication Codes (MAC)
2. Fonctions de hachage résistantes aux collisions
3. Authenticated Encryption with Associated Data (AEAD)

2 Message Authentication Codes (MAC)

2.1 Définition

Message Authentication Code (MAC)

Un MAC est un tuple d'algorithmes $(\text{Tag}, \text{Vrfy})$:

- KeyGen : génère une clé secrète $k \xleftarrow{\$}$
- $\text{Tag}_k(m)$: produit un tag $t \in \mathcal{T}$ pour le message m
- $\text{Vrfy}_k(m, t)$: retourne 1 (valide) ou 0 (invalid)

Correction : $\text{Vrfy}_k(m, \text{Tag}_k(m)) = 1$ pour tous k, m .

2.2 Modèle de sécurité : UF-CMA

Unforgeability under Chosen Message Attack

Algorithm 1 Jeu UF-CMA

Challenger génère $k \xleftarrow{\$}$
interroge $\text{Tag}_k(\cdot)$ sur m_1, \dots, m_q (obtient t_1, \dots, t_q)
produit (m^*, t^*) avec $m^* \notin \{m_1, \dots, m_q\}$
gagne si $\text{Vrfy}_k(m^*, t^*) = 1$

Sécurité : $\Pr[\text{gagne}] \leq \epsilon$ négligeable.

2.3 Constructions

2.3.1 CBC-MAC

Principe : Utiliser un block cipher en mode CBC, ne garder que le dernier bloc comme tag.

Propriétés :

- Déterministe (pas d'IV aléatoire comme en chiffrement)
- Sécurisé (UF-CMA) pour messages de **longueur fixe**
- Ne nécessite qu'une seule clé k

Algorithm 2 CBC-MAC

Require: Message $m = m_1 || m_2 || \dots || m_\ell$ (chaque m_i de n bits), clé k

$t_0 \leftarrow 0^n$ // IV = vecteur nul
for $i = 1$ to ℓ **do**
 $t_i \leftarrow E_k(t_{i-1} \oplus m_i)$
end for
return t_ℓ (dernier bloc)

DANGER : CBC-MAC basique est INSÉCURISÉ pour messages de longueur variable !

Attaque : Si on connaît $t = \text{CBC-MAC}_k(m)$ pour un message m de 1 bloc, alors pour un message $m' = m || (m \oplus t)$ de 2 blocs :

$$\text{CBC-MAC}_k(m') = E_k(E_k(m) \oplus (m \oplus t)) = E_k(t \oplus m \oplus t) = E_k(m) = t$$

L'attaquant peut forger un tag pour m' sans connaître la clé !

Solutions :

1. **CBC-MAC longueur fixe :** Spécifier ℓ dans la définition du schéma. Sécurisé mais inflexible.
2. **Encoder la longueur :** Préfixer le message par sa longueur ℓ . Sécurisé mais nécessite de connaître ℓ à l'avance.
3. **CMAC (Cipher-based MAC, NIST SP 800-38B) :** Utiliser deux clés dérivées k_1, k_2 et traiter le dernier bloc différemment.
4. **OMAC (One-Key MAC) :** Variante de CMAC avec une seule clé et dérivation via multiplication dans $\text{GF}(2^n)$.

CMAC Construction (standard actuel) :

- Dériver deux sous-clés : $k_1 = E_k(0^n) \cdot x$ et $k_2 = k_1 \cdot x$ dans $\text{GF}(2^n)$
- CBC-MAC classique pour les $\ell - 1$ premiers blocs
- Dernier bloc :
 - Si bloc complet : $m_\ell \oplus k_1$
 - Si bloc incomplet : $\text{pad}(m_\ell) \oplus k_2$

Sécurité de CMAC : UF-CMA sécurisé pour messages de longueur variable (prouvé)

Usage : Standard NIST, utilisé dans AES-CCM

2.3.2 HMAC (Hash-based MAC)

HMAC Construction

Soit H une fonction de hachage (ex : SHA-256). On définit :

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

où :

- $\text{ipad} = 0x36$ répété (inner padding)
- $\text{opad} = 0x5C$ répété (outer padding)
- k est ajusté à la taille de bloc de H

Sécurité : HMAC est UF-CMA sécurisé si H est résistante aux collisions.

Standards : HMAC-SHA256, HMAC-SHA384, HMAC-SHA512

3 Fonctions de hachage cryptographiques

3.1 Définition et propriétés

Fonction de hachage cryptographique

$H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ doit satisfaire :

1. **Résistance aux préimages** : Difficile de trouver m tel que $H(m) = h$ (étant donné h)
2. **Résistance aux secondes préimages** : Difficile de trouver $m' \neq m$ tel que $H(m') = H(m)$ (étant donné m)
3. **Résistance aux collisions** : Difficile de trouver $m_1 \neq m_2$ tel que $H(m_1) = H(m_2)$

3.2 Paradoxe des anniversaires

Question : Combien de personnes faut-il pour que deux aient la même date d'anniversaire avec probabilité $> 50\%$?

Réponse : 23 personnes ! (contre-intuitif)

Généralisation : Pour une fonction de hachage $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$, trouver une collision nécessite $\approx \sqrt{2^n} = 2^{n/2}$ évaluations.

Conséquence : SHA-256 ($n = 256$) résiste à 2^{128} essais.

3.3 Construction : Merkle-Damgård

Principe : Construire une fonction de hachage à longueur variable à partir d'une fonction de compression à longueur fixe.

3.3.1 Fonction de compression

Fonction de compression

$h : \{0, 1\}^n \times \{0, 1\}^b \rightarrow \{0, 1\}^n$ prend un **chaining value** de n bits et un **bloc de message** de b bits, produit un nouveau chaining value de n bits.

Exemple : Pour SHA-256, $n = 256$ bits, $b = 512$ bits.

3.3.2 Construction Merkle-Damgård

Composants clés :

1. **IV (Initialization Vector)** : Constante publique fixée dans le standard (ex : SHA-256 utilise les 32 premiers bits des racines carrées des 8 premiers nombres premiers)
2. **Padding** :
 - Toujours ajouter au moins 1 bit (le bit '1')
 - Encoder la longueur du message original en fin (MD-strengthening)
 - Assure que $\text{pad}(m_1) \neq \text{pad}(m_2)$ si $m_1 \neq m_2$
3. **Fonction de compression h** : Construction spécifique à chaque algorithme (Davies-Meyer pour SHA-2, éponge pour SHA-3)

Algorithm 3 Merkle-Damgård Hash

Require: Message m de longueur arbitraire

```
// Étape 1 : Padding
 $m' \leftarrow m \parallel 1 \parallel 0^k \parallel \langle |m| \rangle_{64}$  //  $k$  tel que longueur totale  $\equiv 0 \pmod{b}$ 
// Étape 2 : Découpage en blocs
Découper  $m'$  en blocs  $m_1, m_2, \dots, m_t$  de  $b$  bits chacun
// Étape 3 : Itération
 $H_0 \leftarrow IV$  // Valeur d'initialisation fixée (ex : constantes pour SHA-256)
for  $i = 1$  to  $t$  do
     $H_i \leftarrow h(H_{i-1}, m_i)$  // Fonction de compression
end for
return  $H_t$ 
```

Théorème 3.1 (Merkle-Damgård). *Si la fonction de compression h est résistante aux collisions, alors la fonction de hachage H construite par Merkle-Damgård est résistante aux collisions.*

Idée de la preuve. Par contraposée. Supposons qu'on trouve une collision pour H : $H(m) = H(m')$ avec $m \neq m'$.

Soient m_1, \dots, m_t et $m'_1, \dots, m'_{t'}$ les blocs après padding.

Si $t = t'$ et $m_t \neq m'_t$: alors $h(H_{t-1}, m_t) = h(H'_{t-1}, m'_t)$ est une collision pour h .

Si $t \neq t'$ ou les derniers blocs sont égaux mais les chaining values diffèrent : on remonte itérativement pour trouver une collision pour h .

Donc collision sur $H \Rightarrow$ collision sur h . Par contraposée : h résistant aux collisions $\Rightarrow H$ résistant aux collisions. \square \square

3.3.3 Davies-Meyer (utilisé dans SHA-256)

Une construction populaire de fonction de compression :

$$h(H_{i-1}, m_i) = E_{m_i}(H_{i-1}) \oplus H_{i-1}$$

où E_k est un block cipher (clé = bloc de message, plaintext = chaining value).

3.3.4 Limitations de Merkle-Damgård

— **Length extension attack** : Si on connaît $H(m)$ (mais pas m), on peut calculer $H(m \parallel \text{suffix})$ pour n'importe quel suffix !

Conséquence : $H(k \parallel m)$ n'est PAS un MAC sécurisé (où k est une clé secrète)

— **Séquentiel** : Impossible de paralléliser le calcul

— **Pas de résistance aux collisions pour états internes** : Collision sur H_i ne correspond pas nécessairement à collision sur $H(m)$

Évolution : SHA-3 utilise la construction **éponge** (sponge), différente de Merkle-Damgård, qui n'est pas vulnérable aux length extension attacks.

3.4 Fonctions de hachage modernes

Fonction	Sortie	Sécurité	Statut
MD5	128 bits	CASSÉ	Obsolète
SHA-1	160 bits	CASSÉ (2017)	Déprécié
SHA-256	256 bits	Sécurisé	Standard
SHA-3	Variable	Sécurisé	Standard (2015)
BLAKE2	Variable	Sécurisé	Rapide

4 Authenticated Encryption

4.1 Principe

Combiner chiffrement (confidentialité) et MAC (intégrité/authenticité) en un seul schéma.

4.2 Compositions naïves

Trois approches possibles :

1. **Encrypt-and-MAC** : $c =_k(m)$, $t = \text{MAC}_{k'}(m)$
2. **MAC-then-Encrypt** : $t = \text{MAC}_{k'}(m)$, $c =_k(m||t)$
3. **Encrypt-then-MAC** : $c =_k(m)$, $t = \text{MAC}_{k'}(c)$

Seul Encrypt-then-MAC est génériquement sécurisé !

- Encrypt-and-MAC : Le tag peut fuiter de l'information sur m
- MAC-then-Encrypt : Vulnérable aux padding oracle attacks
- **Encrypt-then-MAC** : Prouvé sécurisé (CCA-secure)

4.3 AEAD : Authenticated Encryption with Associated Data

AEAD

API :

- $_k(m, \text{AD})$: Chiffre m , authentifie m et les données associées AD
- $_k(c, \text{AD})$: Déchiffre et vérifie, retourne m ou \perp (erreur)

Associated Data (AD) : Données non chiffrées mais authentifiées (ex : en-têtes réseau)

4.4 Standards AEAD

4.4.1 AES-GCM (Galois/Counter Mode)

Le plus utilisé en pratique

- Chiffrement : AES en mode CTR
- Authentification : GMAC (basé sur multiplication dans $\text{GF}(2^{128})$)
- Très rapide avec accélération matérielle (instructions AES-NI, PCLMULQDQ)
- Utilisé dans TLS 1.3, IPsec, SSH

4.4.2 ChaCha20-Poly1305

Alternative moderne

- Chiffrement : ChaCha20 stream cipher
- Authentification : Poly1305 MAC
- Plus rapide qu'AES-GCM sans accélération matérielle
- Utilisé dans TLS 1.3, WireGuard VPN

4.4.3 AES-CCM (Counter with CBC-MAC)

Construction :

- Authentification : CBC-MAC (CMAC)
- Chiffrement : AES en mode CTR
- Composition : MAC-then-Encrypt (calcule MAC du plaintext, puis chiffre tout)

Propriétés :

- Prouvé sécurisé (CCA-secure)
- Nécessite deux passes sur les données (MAC puis chiffrement) \Rightarrow plus lent que GCM
- Taille de tag configurable (4-16 octets)

Usage : WPA2 (Wi-Fi), Bluetooth LE, IEEE 802.15.4 (Zigbee), IPsec

4.4.4 AES-OCB (Offset Codebook Mode)

Construction : Mode AEAD avec une seule passe (chiffrement et authentification simultanés).

Principe :

- Utilise des offsets pseudo-aléatoires dérivés du nonce
- Chaque bloc : $c_i = E_k(m_i \oplus \text{Offset}_i) \oplus \text{Offset}_i$
- Tag : XOR de tous les états internes

Avantages :

- **Très rapide :** Une seule passe, parallélisable
- Optimal en nombre d'appels à E_k (approche le minimum théorique)
- Sécurité prouvée

Inconvénient :

- **Brevets** (jusqu'en 2021) \Rightarrow adoption limitée
- Licence gratuite pour logiciels open-source depuis 2013

Usage : Rare en pratique (à cause de l'historique de brevets), mais techniquement supérieur

4.4.5 ASCON

Contexte : Vainqueur de la compétition CAESAR (2014-2019) pour AEAD lightweight.

Construction :

- Basé sur la construction **éponge** (comme SHA-3)
- Permutation cryptographique : 320 bits d'état, 12 rounds
- Très efficace en hardware (petite surface de silicium)

Variantes :

- **ASCON-128** : Clé 128 bits, nonce 128 bits, tag 128 bits
- **ASCON-128a** : Optimisé pour vitesse (8 rounds au lieu de 12)
- **ASCON-80pq** : Clé 160 bits (résistance quantique accrue)

Propriétés :

- Conçu pour IoT, RFID, systèmes embarqués
- Sécurité prouvée dans le modèle standard
- Résistant aux side-channel attacks (design simple, sans lookups)
- Standardisé par NIST (2023) pour lightweight cryptography

Performance :

- Hardware : ~ 3000 GE (gate equivalents) - très compact
- Software : Compétitif avec ChaCha20-Poly1305 sur microcontrôleurs

Usage : Futur standard pour IoT et systèmes embarqués contraints

5 Attaques pratiques

5.1 Padding Oracle Attack

Scénario : CBC + MAC-then-Encrypt avec messages d'erreur différents pour "padding invalide" vs "MAC invalide".

Résultat : L'attaquant peut déchiffrer complètement sans connaître la clé !

Exemples historiques :

- ASP.NET (2010)
- OpenSSL (2003)
- Variantes : POODLE (SSL 3.0), Lucky 13 (TLS)

dans GCM]Nonce reuse

dans GCM

Catastrophe : Réutiliser un nonce en AES-GCM permet de :

- Retrouver la clé d'authentification
- Forger des tags arbitraires
- Casser complètement la confidentialité

Moralité : Les nonces doivent être STRICTEMENT uniques !

6 Notebooks pratiques

- `03_demo_mac.ipynb` : *HMAC – SHA256*
- `03_demo_hash_collisions.ipynb` : *Paradoxes anniversaires, attaque sur MD5*
- `03_demo_aead.ipynb` : *AES – GCM et ChaCha20 – Poly1305*
- `03_exercices.ipynb` : *Exercices guidés*

7 Conclusion

Points clés :

- Chiffrement seul NE SUFFIT PAS : il faut aussi l'intégrité
- MAC : authentification avec clé partagée (HMAC standard)
- Hash : empreintes, résistance aux collisions (SHA-256, SHA-3)
- AEAD : combinaison sécurisée (AES-GCM, ChaCha20-Poly1305)
- Encrypt-then-MAC : seule composition générique sûre
- Nonce management : critique pour la sécurité

Le chapitre suivant introduira la **cryptographie à clé publique**, qui résout le problème de la distribution des clés.