

# Introduction à la cryptographie

---

Louiza Khati

4A-Partie 1

# ANSSI

---

- Création juillet 2009
- Acteur majeur de la cyber sécurité en France
- Rôles :
  - Favorise le développement de la cyber sécurité en France
  - Apporte son expertise et son assistance aux administrations et aux industriels
  - Encadre et délivre des « visas de sécurité » (via CCN)
  - Forme les citoyens et entreprises (guides)
  - Etc.



# Laboratoire cryptographie

---

- Favorise la recherche dans ce domaine
- Echange sur les différents sujets auprès des acteurs internationaux
- Participe à la mise en place des bonnes pratiques crypto (guides)
- Apporte son expertise (certifications)

# Ce cours

---

- Introduction à la cryptographie
  - Cryptographie : domaine riche et complexe
- Objectifs :
  - Découvrir la cryptographie
  - Donner des intuitions
  - Connaitre des exemples de constructions
  - Dépend de vous 😊
- Méthodes :
  - Cours + TD



# Ce cours

---

- Ne pas hésitez à poser des questions
  - Notions inconnues/floues
- Si c'est trop lent, trop rapide
- Répondre aux questions
  - Cours plus interactif → plus agréable!
  - Apprentissage plus rapide!
  - Trouver les réponses sur internet n'apporte rien.

# Règles à suivre

---

- Absences :
  - Récupérer le cours et les notes
  - Attention : questionnaires en début de cours généralement
- Retards :
  - Cours à 8h : 15 min de retard tolérées (qrcode à la pause)
  - Les autres cours : 5 minutes de retard tolérées
  - Si j'arrive après cet horaire, ne pas déranger le cours svp.
- Respect :
  - Pas de nourriture, ni de boissons en classe (trop bruyant)
  - Attention au bavardage !
  - Intervenant et camarades de classe.



# Notation

---

- Note CC (contrôle continu)
  - Participation en classe,
  - TD : je fais mon TD moi-même (je pourrais en discuter avec les camarades par la suite)
  - Questionnaire en ligne
    - Tester vos comptes wooclap et savoir utiliser l'application.
- Examen final
  - Examen papier sur l'ensemble du cours (tout le programme de cryptographie)

# Sondage

---



# Cours précédents

---

- Mécanismes symétriques
  - Primitives de chiffrement par bloc (AES, 3DES, Camellia, etc.)
  - Modes opératoires pour le chiffrement (ECB, CBC)
  - Chiffrement symétrique : mode + primitive de chiffrement par bloc
    - Ex : AES-128-CBC, Camellia-ECB, AES-256-CTR, etc...
- Dans ce cours : Intégrité symétrique
  - A base de primitive de chiffrement par bloc
  - A base de fonctions de hachage



# « Message Authentication Code » (MAC)

---

- Chiffrement symétrique « souvent » malléable
  - Vulnérables aux attaques CCA (chiffrés choisis)
  - Modifications contrôlées par l'adversaire dans le chiffré (CBC/CTR)
- Les MACs :
  - Assure qu'une donnée transmise n'a pas été modifiée sur le canal/ donnée stockée
  - Notion : **intégrité/authenticité des messages**  $\neq$  confidentialité
- Confidentialité et intégrité : besoins de sécurité complémentaires



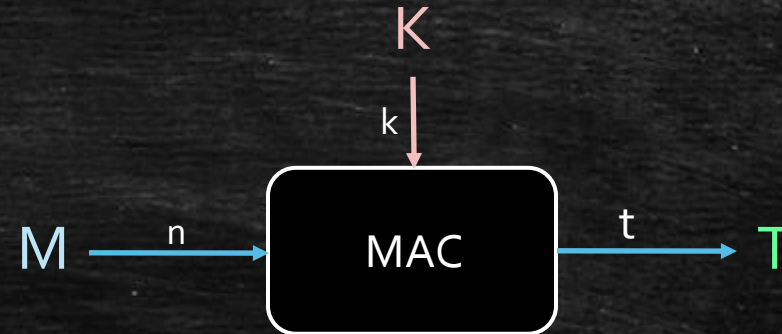
# MAC

---

- «Signature» utilisant de la cryptographie symétrique
  - Garantit **l'intégrité d'une donnée** (avec une clé secrète)
  - Seules les personnes d'un groupe peuvent vérifier la validité d'un MAC
  - Pas de non-répudiation (car clé partagée dans un groupe)
  - Plus rapide qu'une signature (pratique pour les réseaux)
- Utilise une **clé** symétrique
- Peut-être construit avec un chiffrement par bloc ou une fonction de hachage

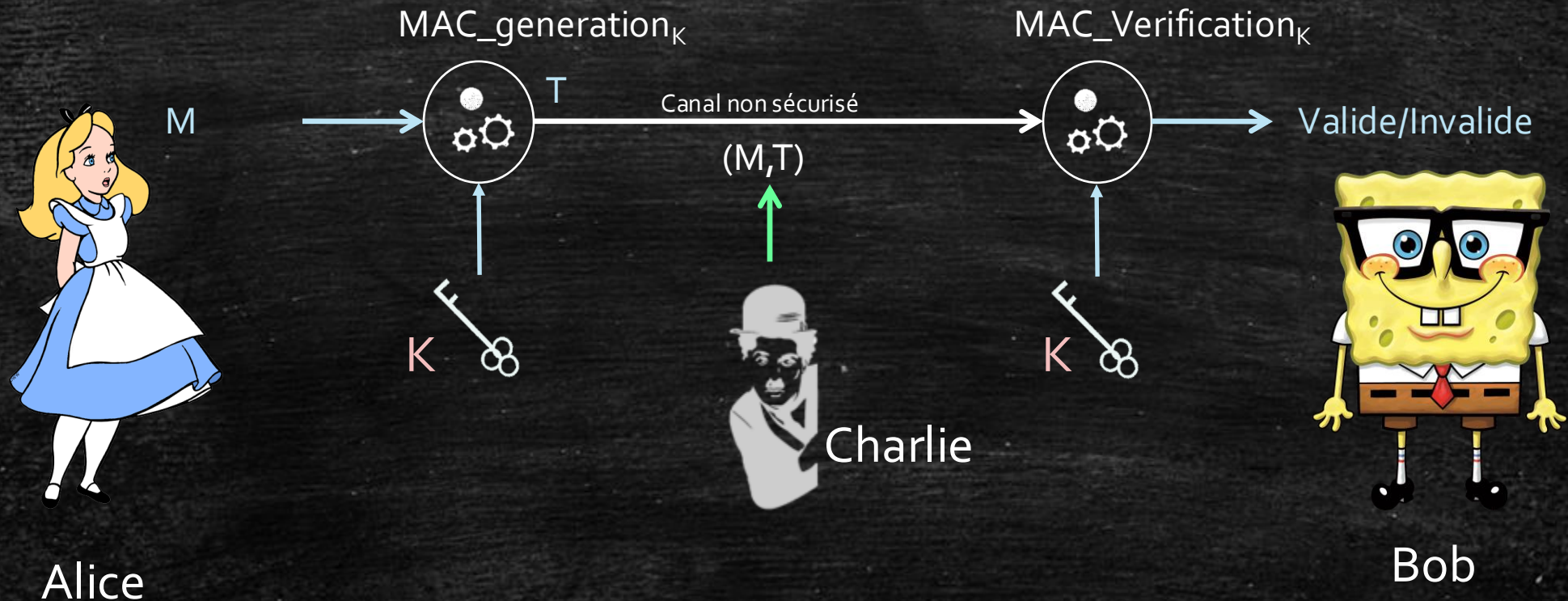
# MAC

- Clé  $K$  de taille  $k > 128$
- Message  $M$  de taille  $n$  quelconque
- Valeur  $T$  :
  - Appelé tag ou MAC
  - Taille constante  $t > 128$  (recommandée)
- MAC :
  - $K \leftarrow \text{keygen}(k)$ ,  $k$  taille de la clé
  - $T \leftarrow \text{MAC\_generation}(K, M)$  déterministe (en général)
  - Valide/Invalide  $\leftarrow \text{Mac\_verification}(K, M, T)$  déterministe





# MAC : authenticité de la donnée

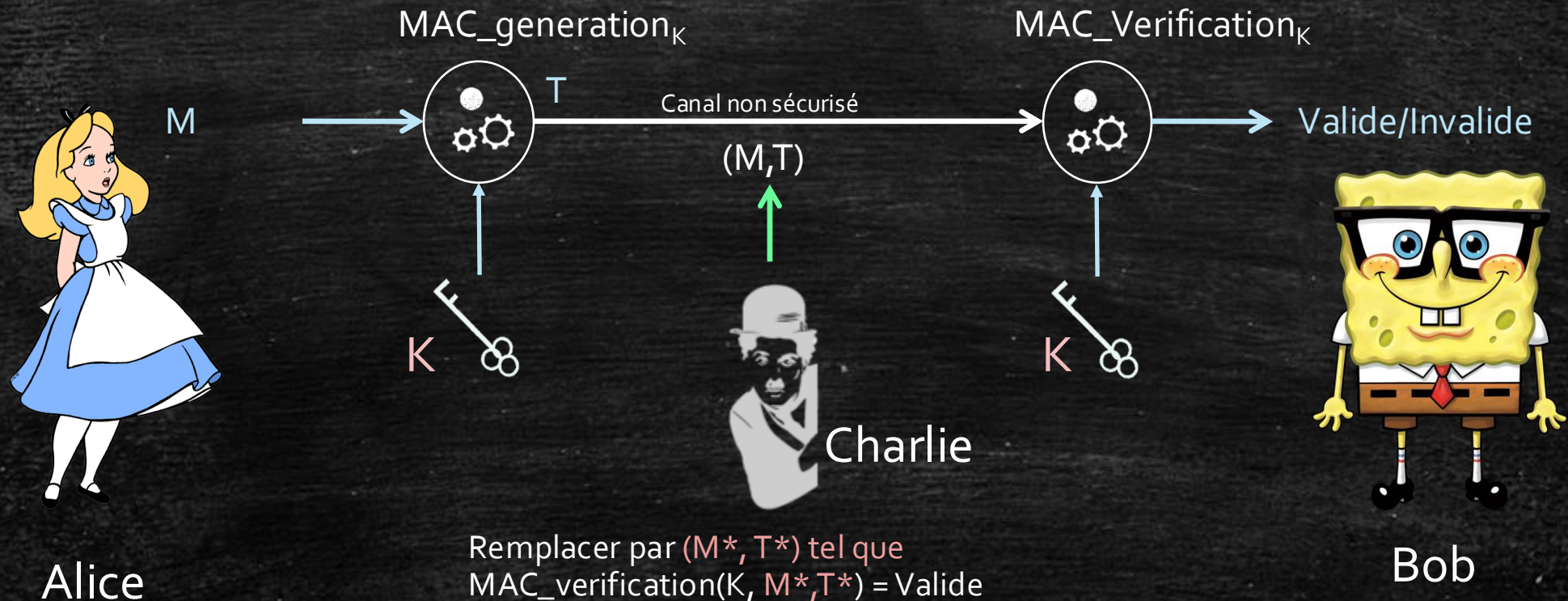


$$\text{MAC\_generation}(K, M) = T$$

$$\text{MAC\_verification}(K, M, T) = \text{Valide/Invalide}$$



# MAC : authenticité de la donnée





# MAC : Modèles d'attaquants

---

- **À messages connus** : l'adversaire a accès à des couples  $(M, T)$  de messages déjà authentifiés (interception de MACs)
- **À messages choisis** : l'adversaire demande le MAC de messages qu'il choisit (accès à un oracle de génération de MACs)
  - Attaque non adaptative : l'ensemble des messages est choisi a priori
  - Attaque adaptative : l'adversaire choisit les messages en fonction des réponses de l'oracle

# MAC : Buts de l'adversaire

---

- Retrouver la clé



# MAC : Buts de l'adversaire

---

- Retrouver la clé
- Forger un MAC pour n'importe quel message

# MAC : Buts de l'adversaire

---

- Retrouver la clé
- Forger un MAC pour n'importe quel message
- Forger un MAC pour un message  $M$  choisi
- Forger un MAC pour un message  $M$  non choisi



# MAC : Buts de l'adversaire


---

- Retrouver la clé
- Forger un MAC pour n'importe quel message
- Forger un MAC pour un message M choisi
- Forger un MAC pour un message M non choisi
- Distinguer un MAC d'une sortie aléatoire


# MAC : Buts de l'adversaire

---

- Retrouver la clé
- Forger un MAC pour n'importe quel message
- Forger un MAC pour un message M choisi
- Forger un MAC pour un message M non choisi
- Distinguer un MAC d'une sortie aléatoire



Attaque de plus en plus simple

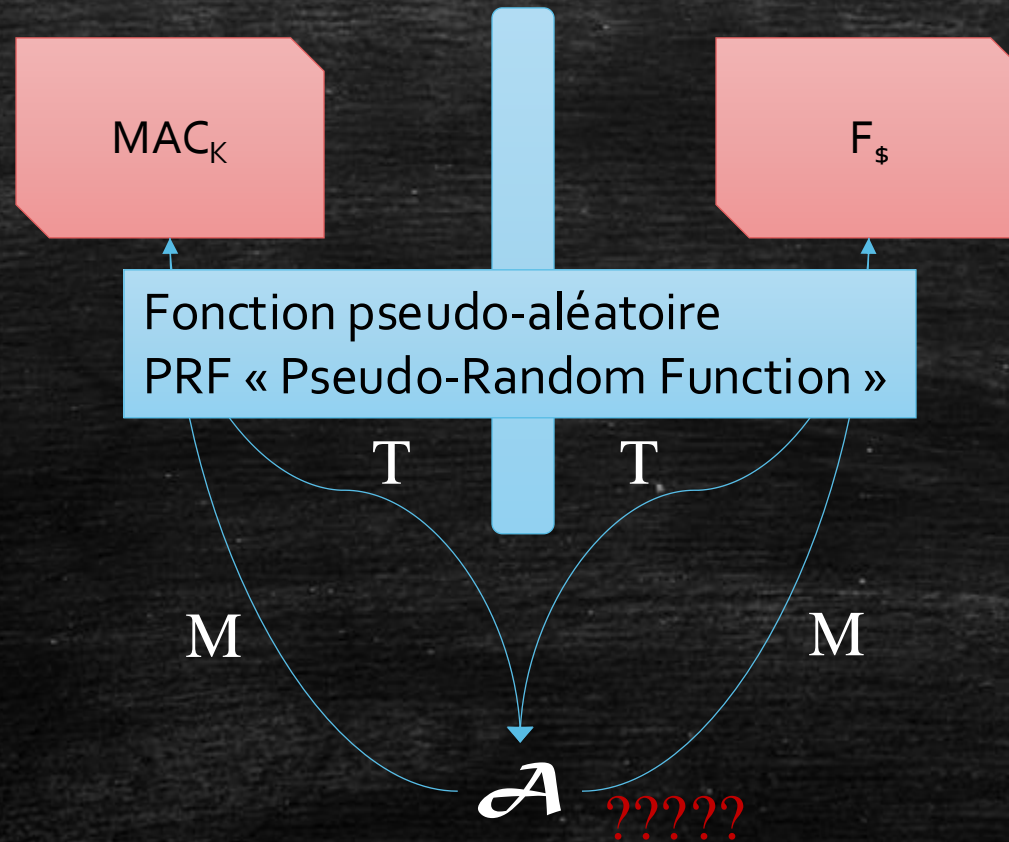


Attaquant de plus en plus fort!

Sécurité maximale



# (MAC : Sécurité)



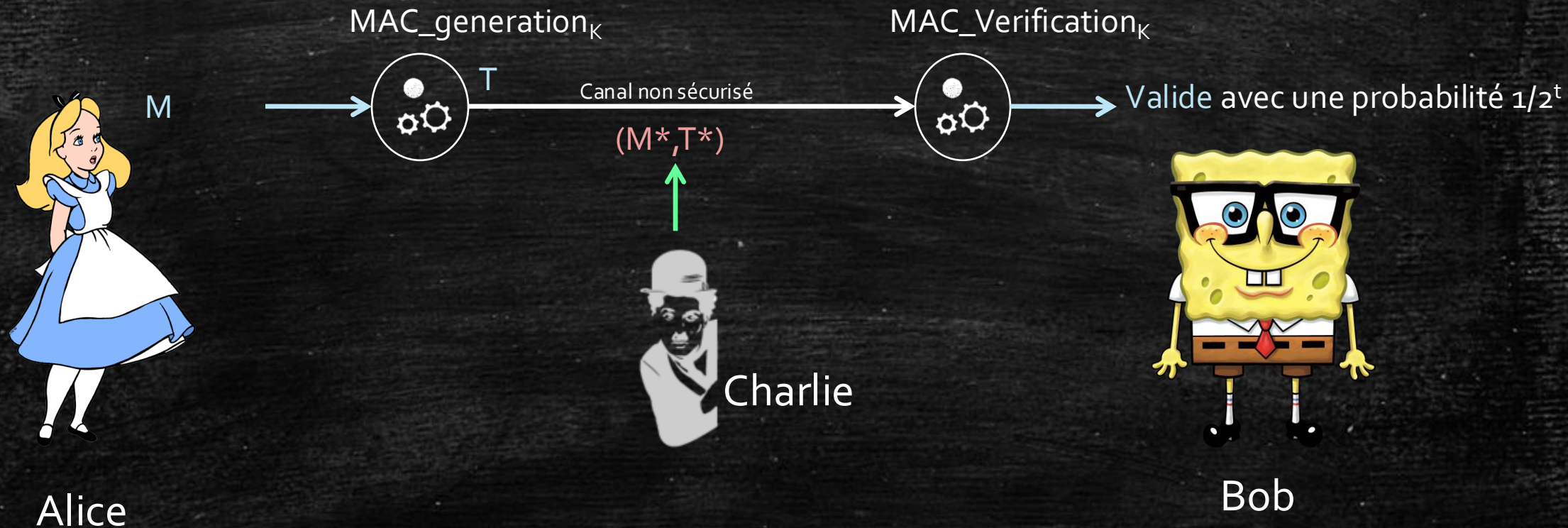
Notion d'indistinguabilité

Sans connaître  $K$ ,  $\mathcal{A}$  peut-il distinguer  $MAC_K$  et  $F_\$$  ?



# MAC : Sécurité

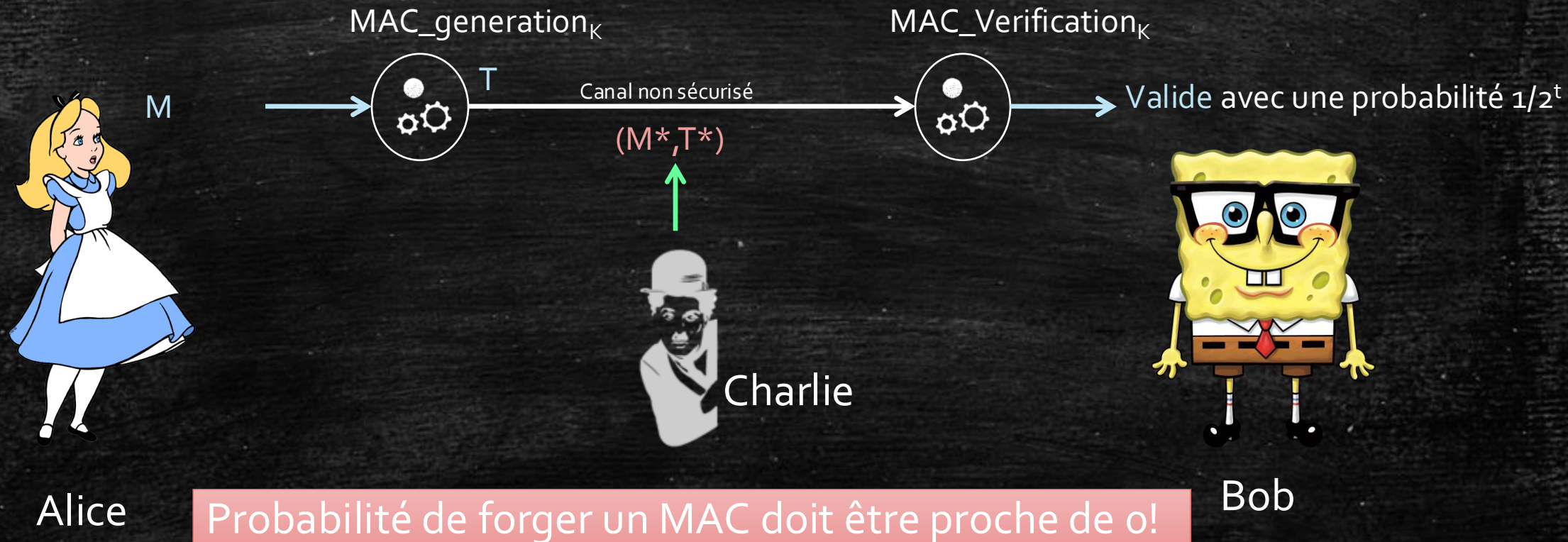
- Probabilité minimale qu'un adversaire produise une contrefaçon :  $1/2^t$





# MAC : Sécurité

- Probabilité minimale qu'un adversaire produise une contrefaçon :  $1/2^t$



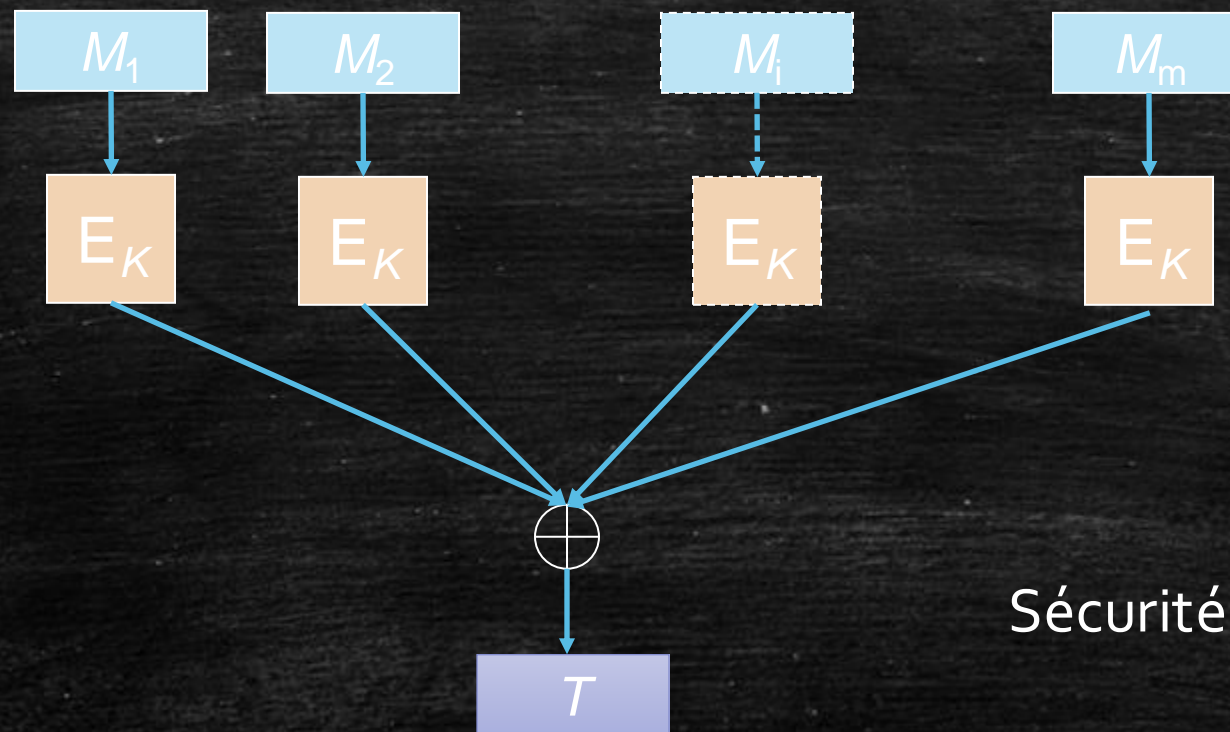
# MACs part 1

---

Basés sur une primitive de chiffrement par bloc

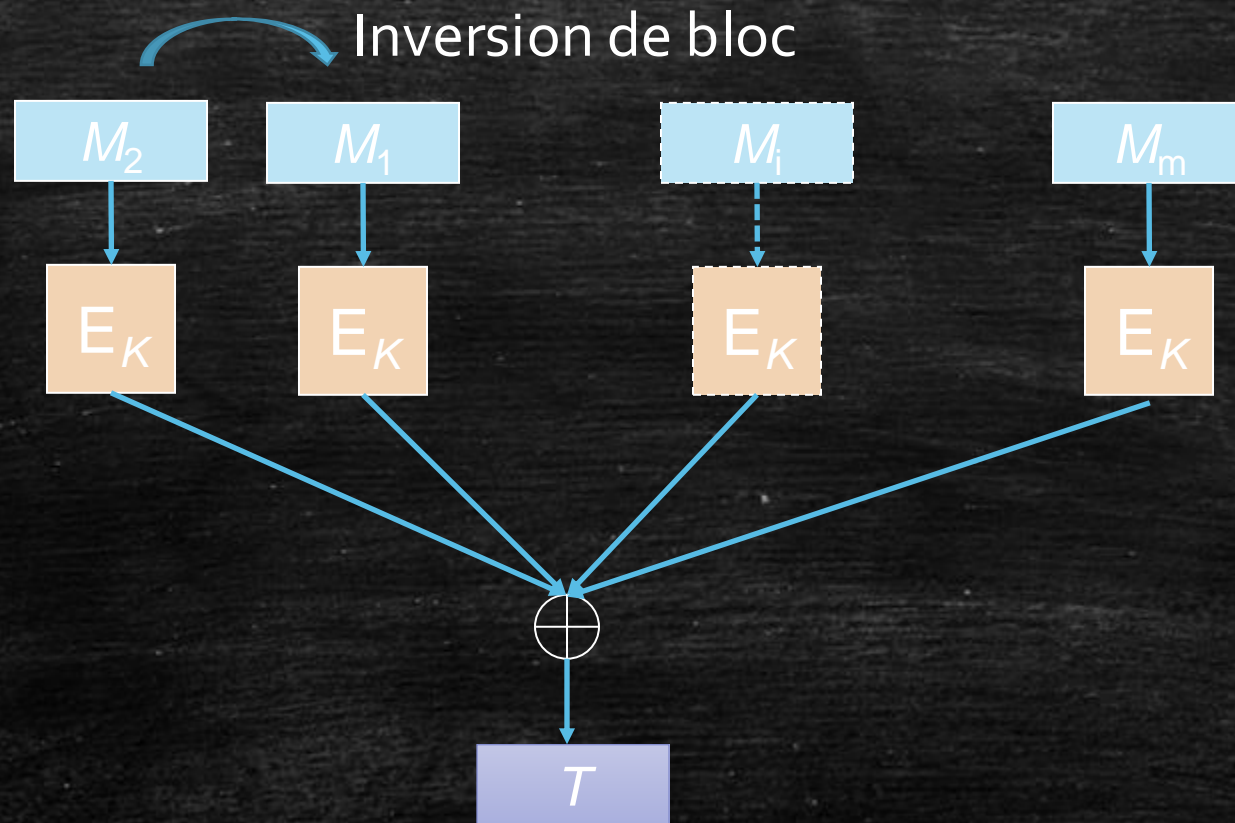


# Un MAC simple



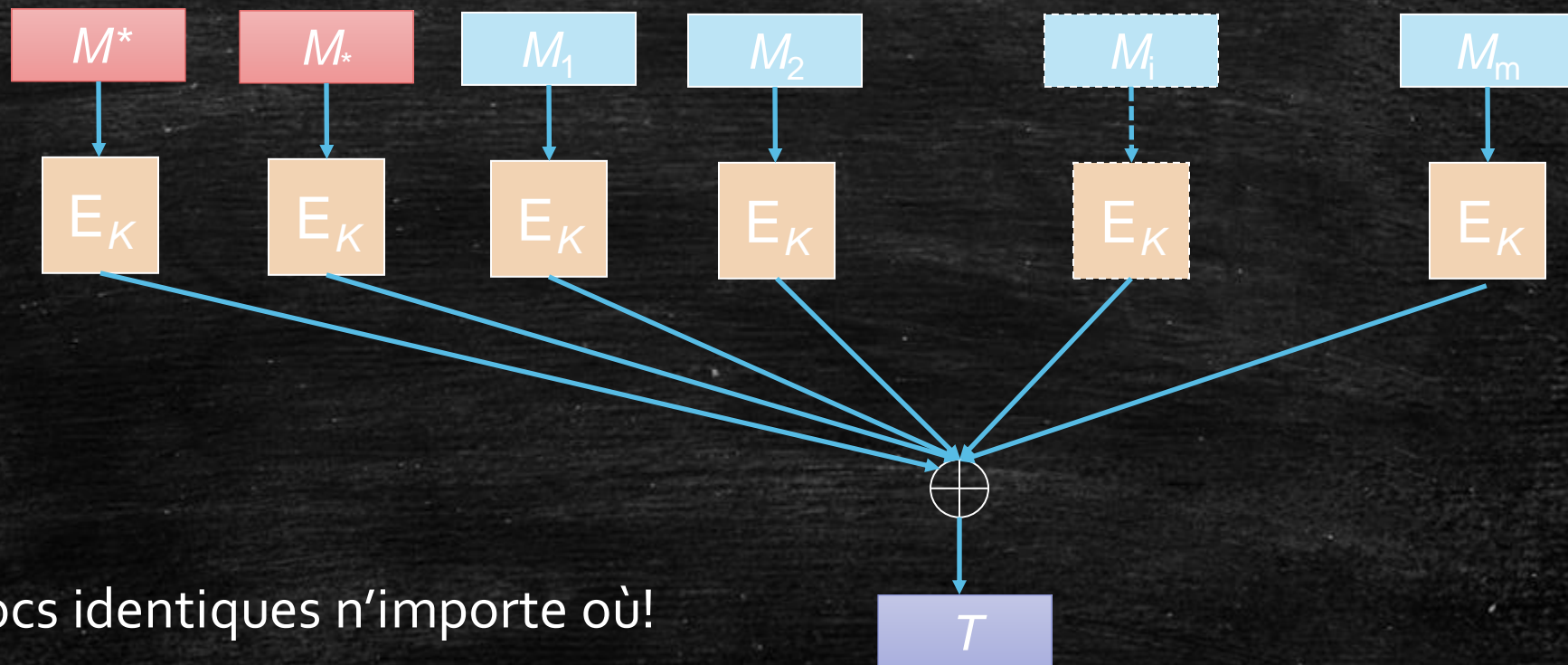
Sécurité de ce MAC ? 

# Un MAC simple



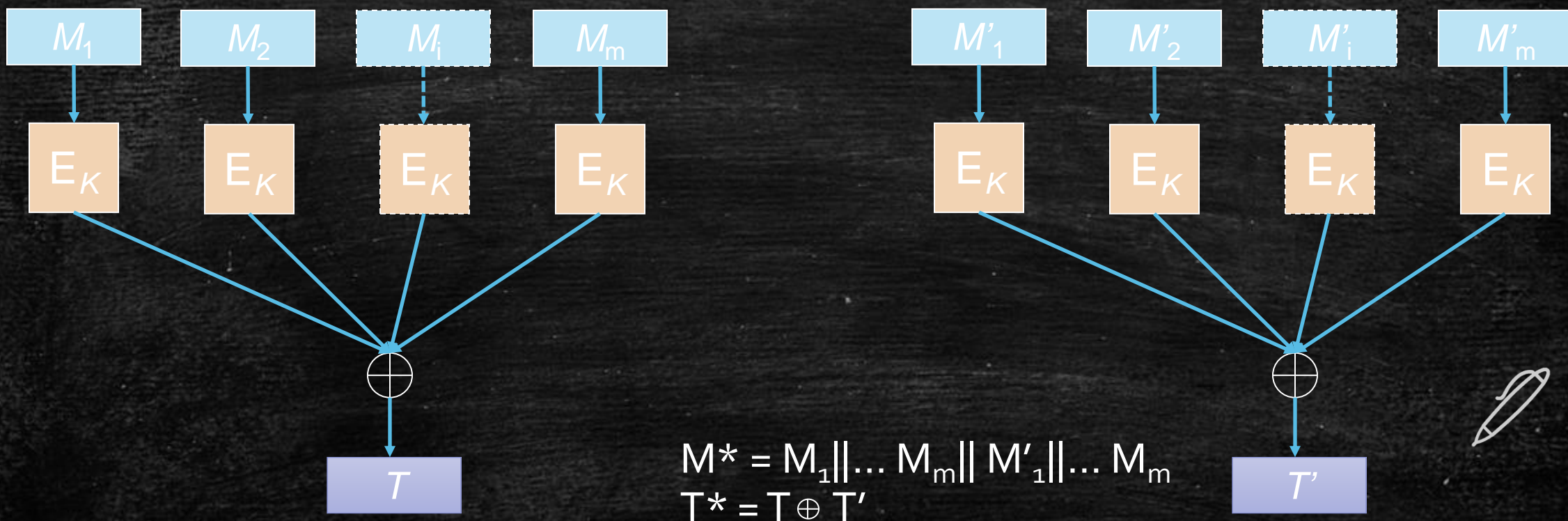


# Un MAC simple



Insertion de blocs identiques n'importe où!

# Un MAC simple





# Un MAC simple

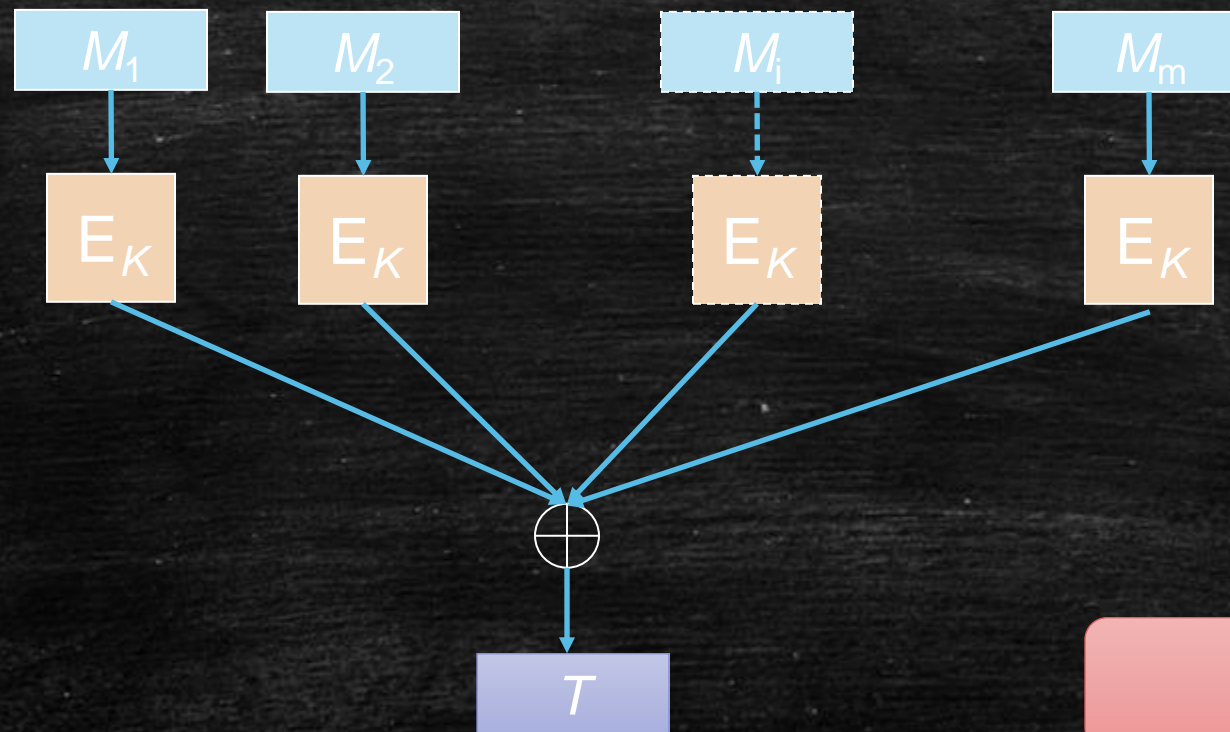


Schéma non sûr!!

# MAC : Exemples

---

- **CBC-MAC**

- Basé sur le chiffrement CBC sans IV

- **EMAC**

- CBC-MAC surchiffré
- Utilisations de deux clés : deux clés dérivées de la même clé maitresse
- Prouvé sûr pour des messages de tailles variables sous des hypothèses raisonnables

- **HMAC**

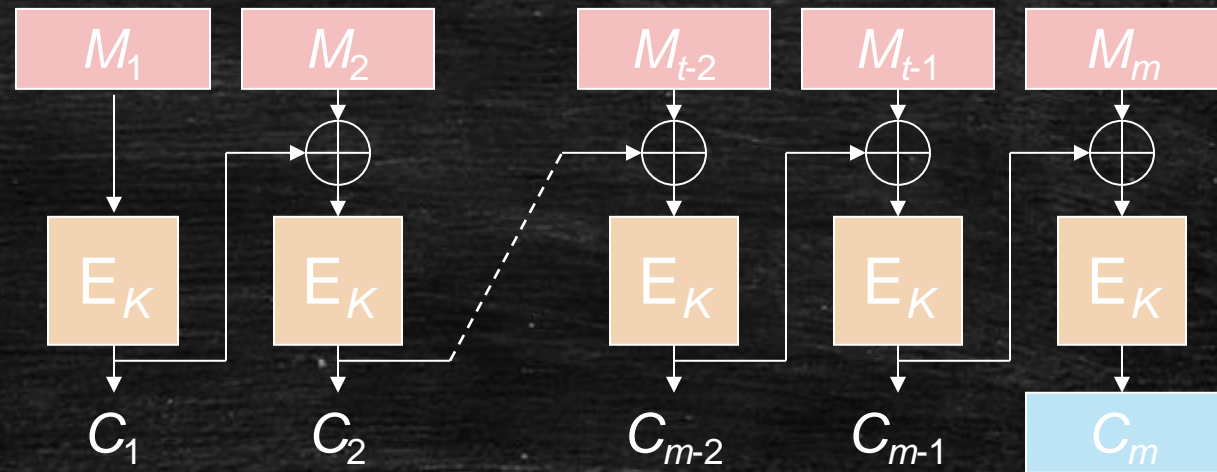
- Très utilisé !
- Utilise une **fonction de hachage**
- Prouvé sûr pour des messages de tailles variables sous des hypothèses raisonnables



# MAC : CBC-MAC

- CBC

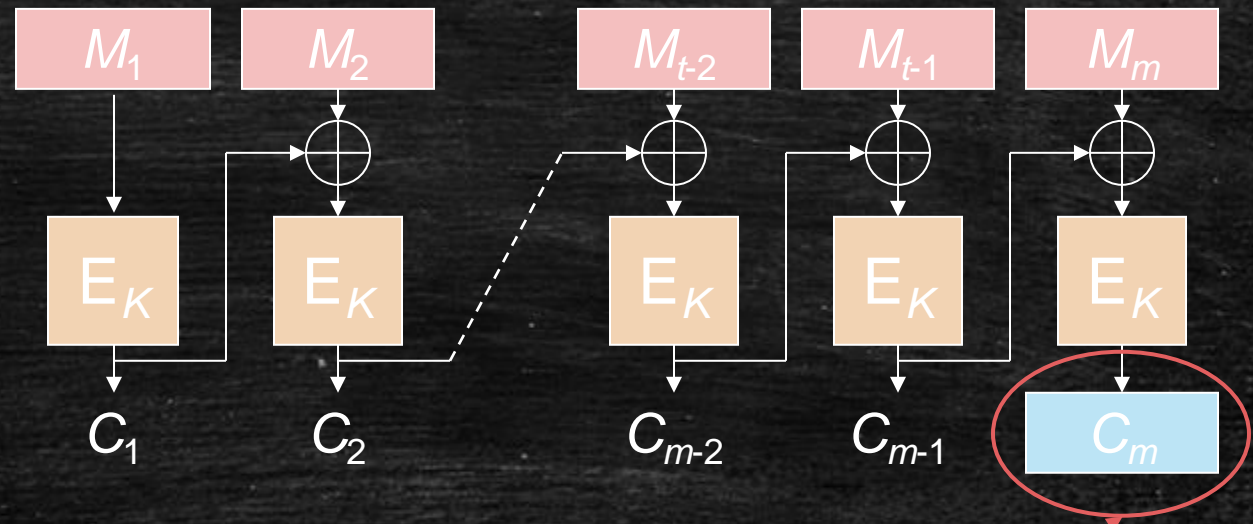
- Pas d'IV
- $T = C_m$
- Valeurs  $C_i$  non publiques
  - $0 < i < m$



# MAC : CBC-MAC

- CBC

- Pas d'IV
- $T = C_m$
- Valeurs  $C_i$  non publiques
  - $0 < i < m$

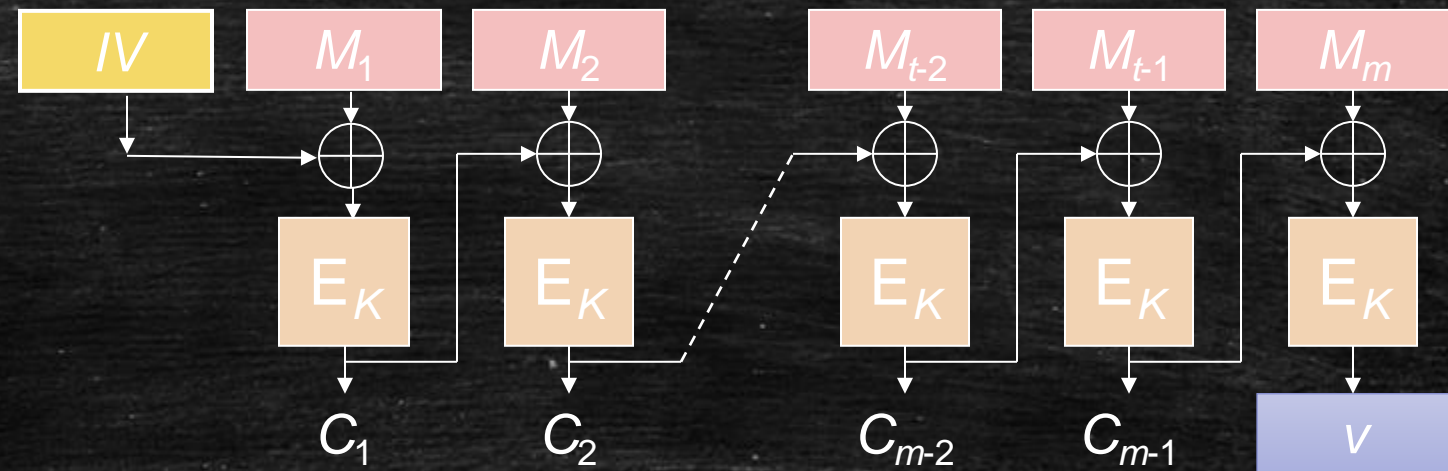


$T = C_m$  et dépend de tous les blocs



# CBC-MAC : IV

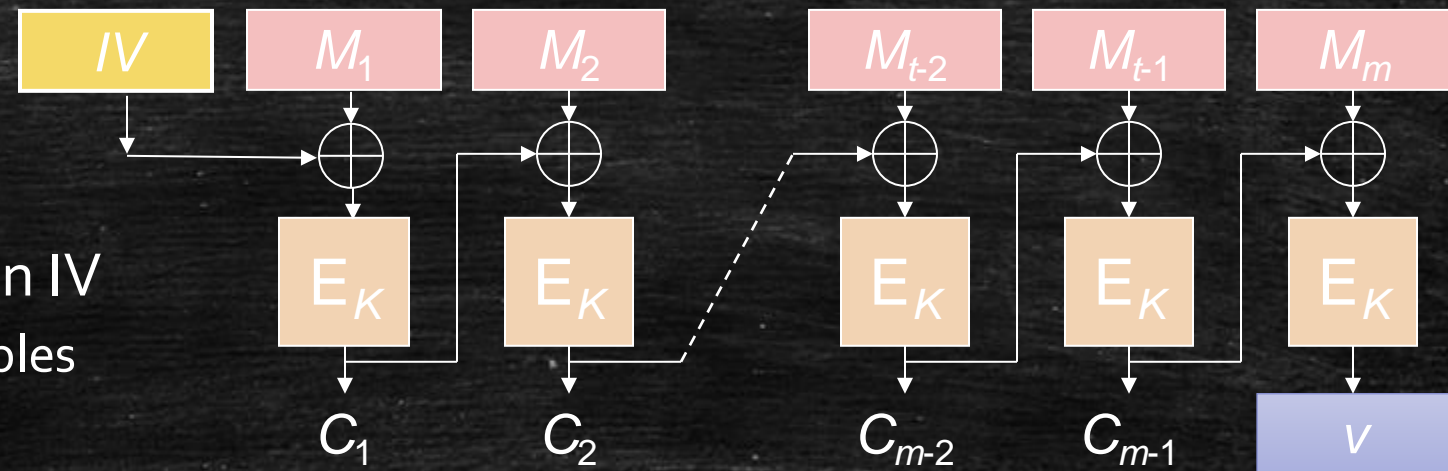
- Si présence d'un IV
  - $T = (IV, v)$



- Possibilité d'utiliser l'IV pour forger
  - Si l'attaquant possède  $(M, (IV, v))$  avec  $M = M_1$  (message composé d'un bloc)
  - Contrefaçon  $(M^*, T^*)$  tel que  $IV' = M_1 + IV + M'_1$ ,  $M^* = M_1$ ,  $T^* = (IV', v)$

# CBC-MAC : Pas d'IV

- Si présence d'un IV
  - Attaques possibles



- Pas d'IV dans le cadre de CBC-MAC!

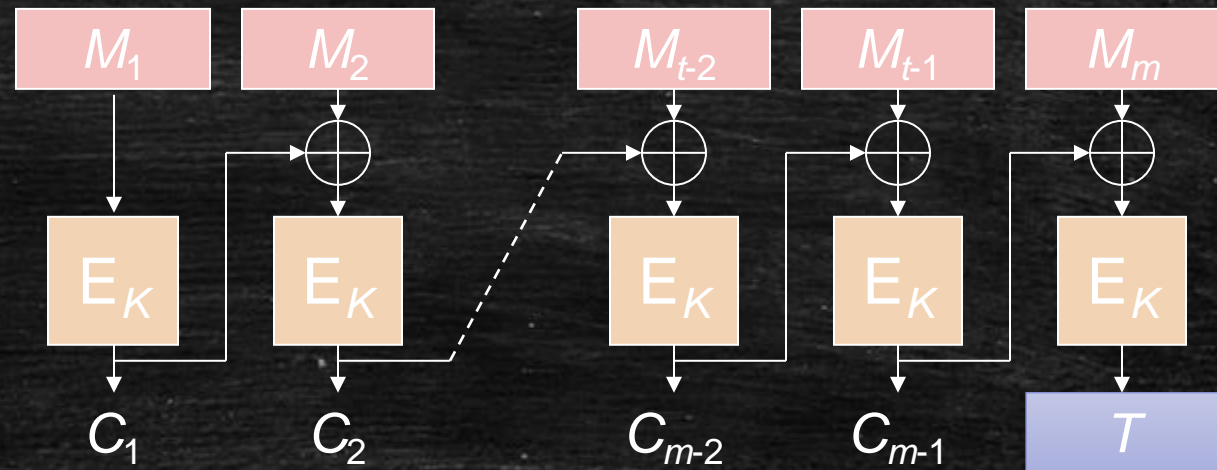


# MAC : CBC-MAC

- CBC

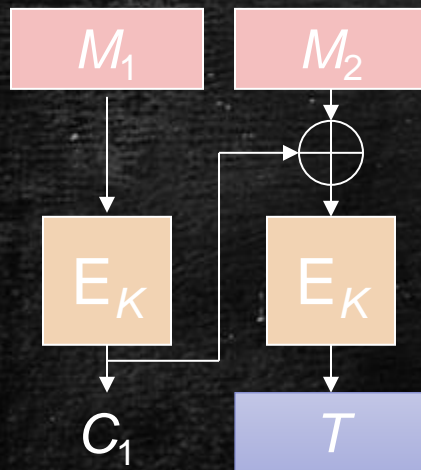
- Pas d'IV
- $T = C_m$
- Valeurs  $C_i$  non publiques
  - $0 < i < m$

Sécurité ?

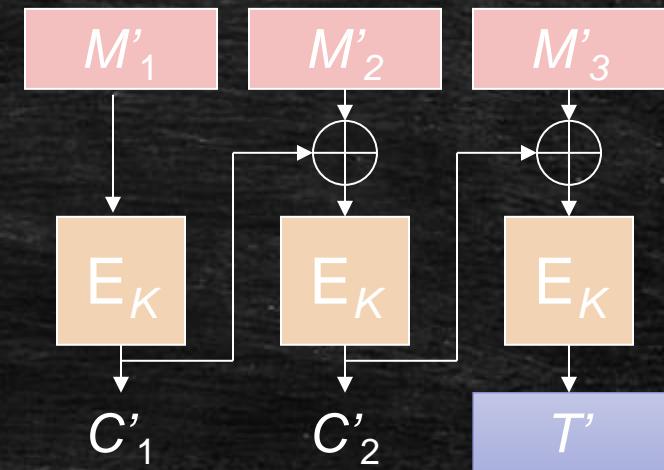


# MAC : CBC-MAC

Message  $M = M_1 \parallel M_2$   
Tag  $T$



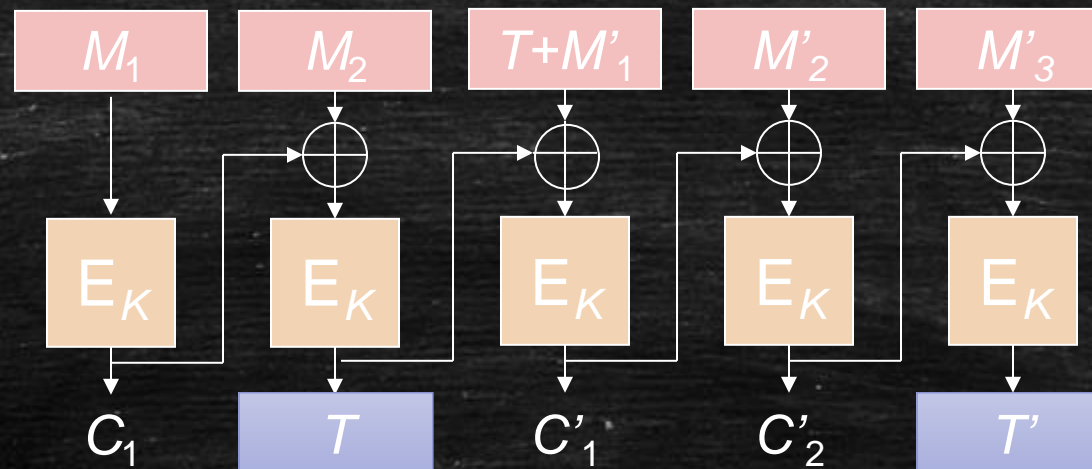
Message  $M' = M'_1 \parallel M'_2 \parallel M'_3$   
Tag  $T'$



Message  $M^* = M_1 \parallel M_2 \parallel T \oplus M'_1 \parallel M'_2 \parallel M'_3$   
Tag  $T^* = T'$



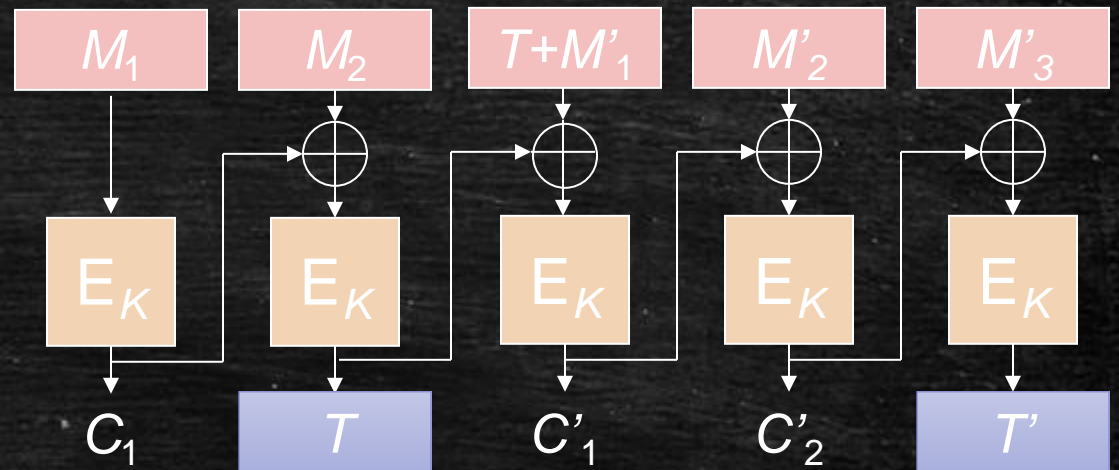
# MAC : CBC-MAC



Message  $M^* = M_1 \parallel M_2 \parallel T+M'_1 \parallel M'_2 \parallel M'_3$   
Tag  $T^* = T'$

# MAC : CBC-MAC

- Pas d'IV
- CBC-MAC sûr pour des messages de taille fixe seulement

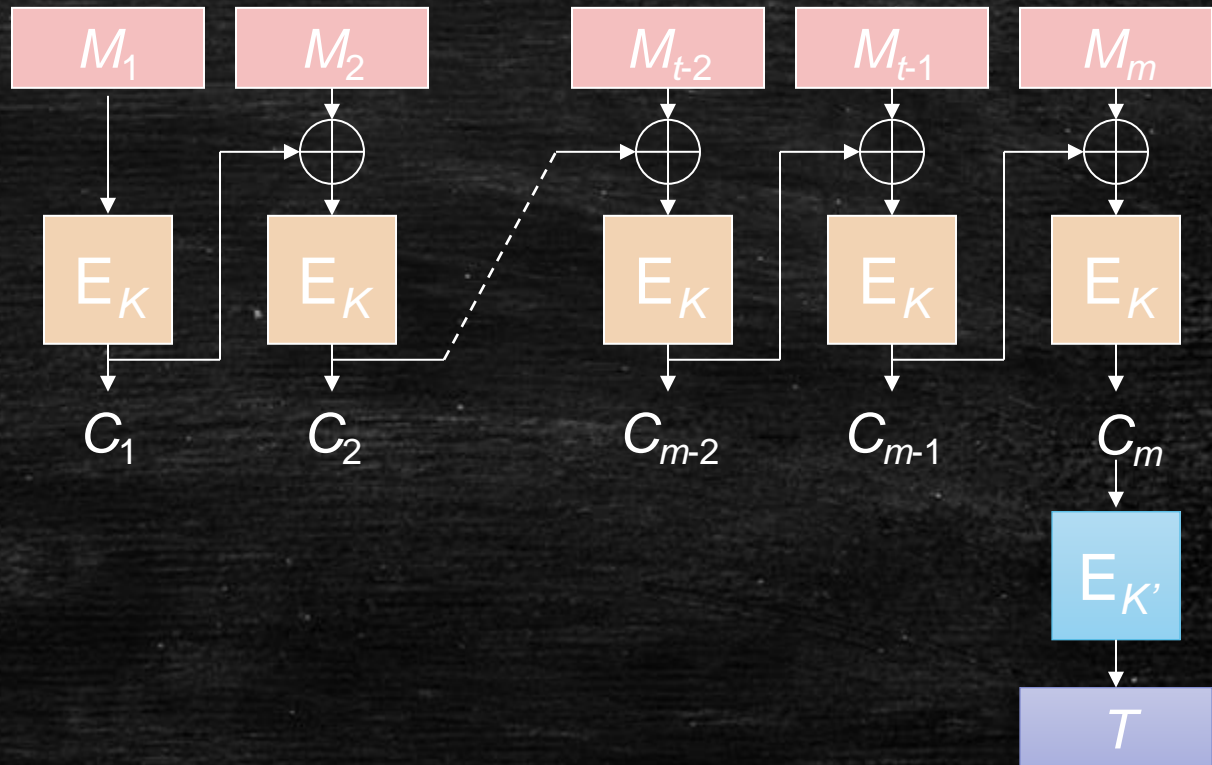


Message  $M^* = M_1 || M_2 || T+M'_1 || M'_2 || M'_3$   
Tag  $T^* = T'$



# MAC : EMAC

- CBC-MAC surchiffré sûr pour des messages de taille variable.
  - Sécurité prouvée :  $q \ll 2^{n/2}$
  - Attaque pour  $q = 2^{n/2}$



# Fonctions de hachage

---



# Fonction de hachage

---

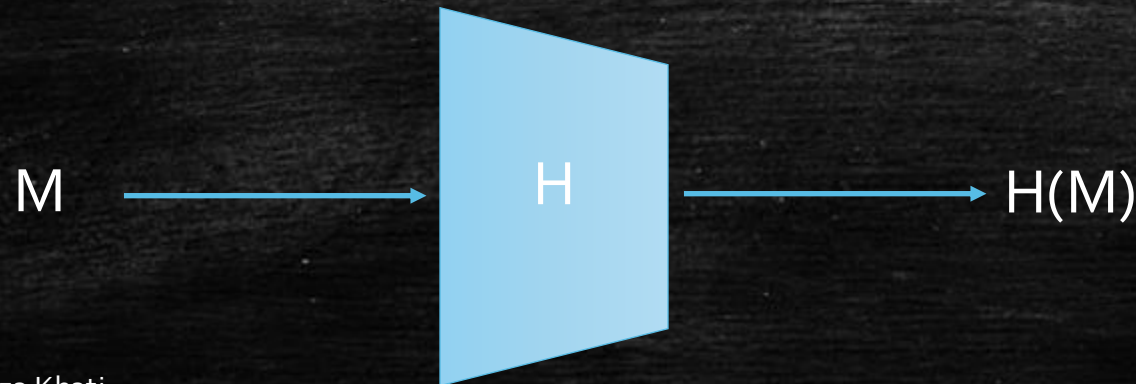
- Une fonction de hachage  $H$  est une fonction :

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

$$M \rightarrow H(M)$$

$M$  est un message de taille quelconque

$H(M)$  est appelé « hash », « haché » ou encore « empreinte » **de taille  $n$** .



# Fonction de hachage : Propriétés

---

- Pas de clé
- Pas d'algorithme inverse
- Rapidité du traitement des données
- Répartition des images sur l'ensemble de sortie
- Compression des données





# Fonction de hachage

---

- Une fonction de hachage cryptographique est une fonction de hachage qui compresse de **manière sécurisée** une entrée de longueur arbitraire et une sortie de taille fixe.



# Fonction de hachage : Utilisations

---

- MACs (dans la suite)
- Stockage de mots de passe
  - Quelle propriété intéressante des fonctions de hachage ?
- Générateur d'aléa
- Signature (dans la suite)
- Dérivation de clé
  - Quelle propriété intéressante des fonctions de hachage ?



# Fonctions de hachage

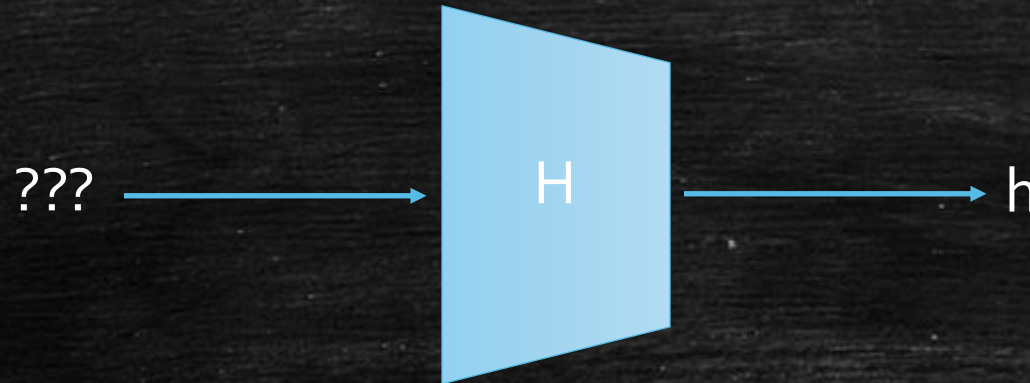
---

- Se comporte comme un « oracle aléatoire »
  - «Comme une fonction aléatoire »
- Pas de clé utilisée --> la fonction est totalement publique
  - Garantir l'intégrité d'une donnée avec une fonction de hachage ? → Pas seule!
- **Propriétés spécifiques aux fonctions de hachage :**
  - Résistance en collisions
  - Résistance en préimages
  - Résistance en seconde préimage

# Propriété de sécurité : préimage

---

- Etant donné  $h \in \{0,1\}^n$ , trouver  $M$  tel que  $H(M) = h$

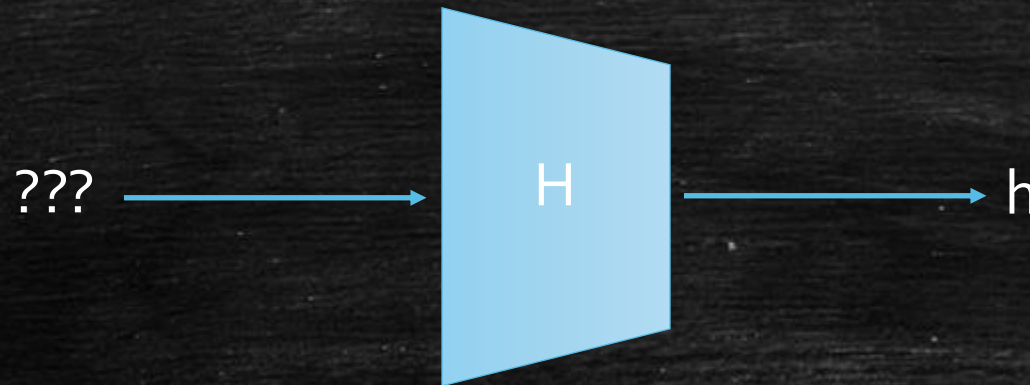




# Propriété de sécurité : préimage

---

- Etant donné  $h \in \{0,1\}^n$ , trouver  $M$  tel que  $H(M) = h$



- Complexité attaque générique : de l'ordre de  $2^n$

# Propriété de sécurité : préimage

---

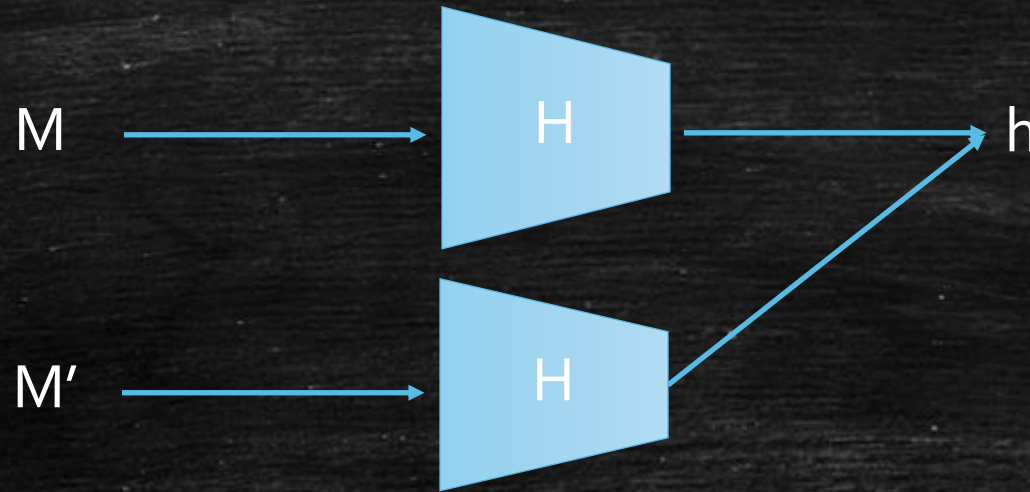
- Entrée  $h$
- Meilleure attaque générique : recherche exhaustive
- Probabilité de trouver une préimage :  $1/2^n$
- Calculer  $H(M)$  pour des messages aléatoires
- Après  $2^n$  messages on s'attend à trouver  $M$  tel que  $H(M) = h$
- Complexité recherche probabiliste :  $O(2^n)$



# Propriété de sécurité : seconde préimage

---

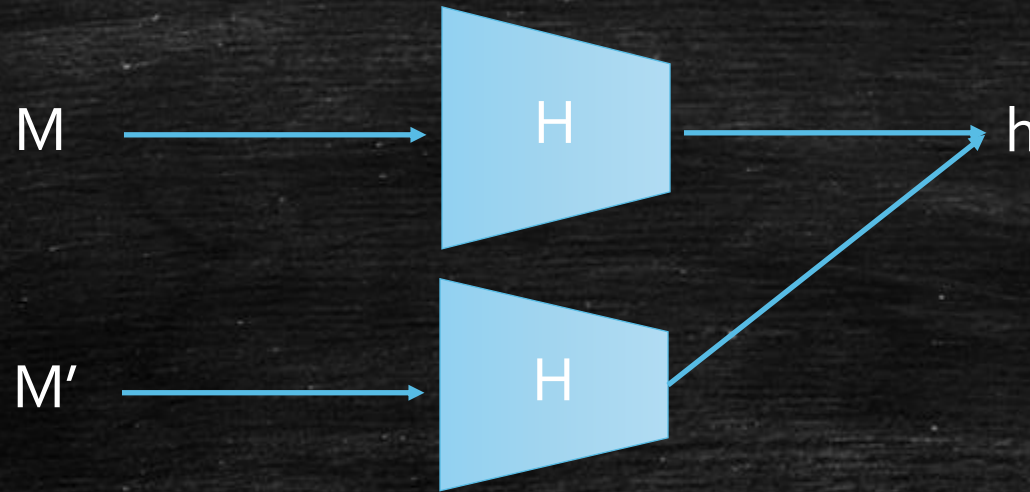
- Etant donné  $M \in \{0,1\}^*$ , trouver  $M' \neq M$  tel que  $H(M) = H(M')$



# Propriété de sécurité : seconde préimage

---

- Etant donné  $M \in \{0,1\}^*$ , trouver  $M' \neq M$  tel que  $H(M) = H(M')$

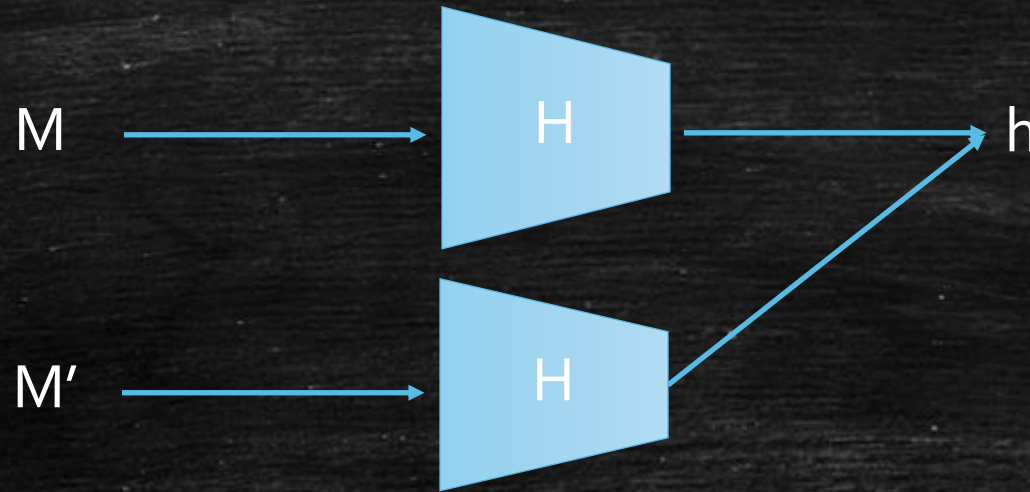


- Complexité attaque générique : de l'ordre de  $2^n$



# Propriété de sécurité : collisions

- Trouver  $M' \neq M$  tel que  $H(M) = H(M')$



- Complexité attaque générique : de l'ordre de  $2^{n/2}$ 
  - **Paradoxe des anniversaires**

# Fonction de hachage : Sécurité

---

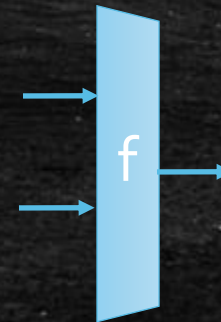
- Cryptanalyse :
  - Trouver une attaque plus efficace qu'une attaque générique
  - $2^{n/2}$  calculs de hachés pour les collisions
  - $2^n$  calculs pour les (secondes) préimages
- En pratique : la résistance aux collisions est la plus difficile à obtenir
- La **taille de la sortie** est déterminante!
  - Recommandation guide crypto ANSSI :  **$n \geq 200$**



# Fonctions de hachage : construction

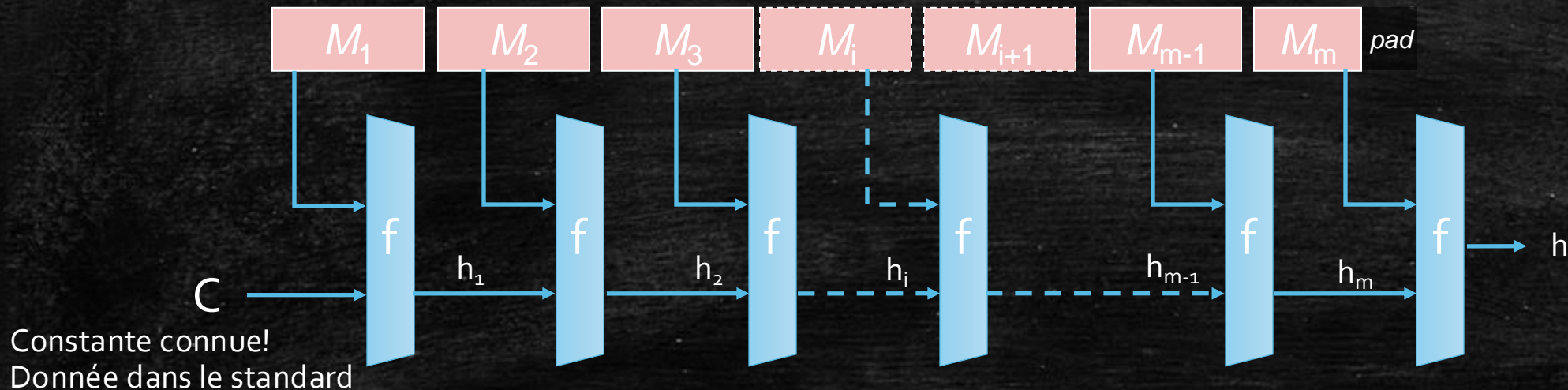
---

- Découpage du message M en bloc de b bits
- Padding du message M (b = taille de bloc)
- Utilisation d'une construction itérative
  - Intégration des blocs de messages
  - Fonction de compression  $f : \{0,1\}^{n_1} \times \{0,1\}^n \rightarrow \{0,1\}^n$
- Algorithme d'**extension de domaine**
  - Ou encore « Mode opératoire »



# Construction : Merkle-Damgard (1979)

- Construction très utilisée
- La fonction  $f$  est résistante en « **pseudo-collision** »
  - Difficile de trouver  $(x,y) \neq (x',y')$  tel que  $f(x,y) = f(x',y')$
  - $\text{Pad}(M)$  : padding classique + encodage de la taille du message





# Fonctions de hachage : constructions

---

- Merkle-Damgard : structure très utilisée
  - MD5, SHA1, RIPEMD-160
  - Famille SHA2
- Nouvelle structure : fonction éponge
  - Nouveau standard SHA3

# Construction éponge « sponge »

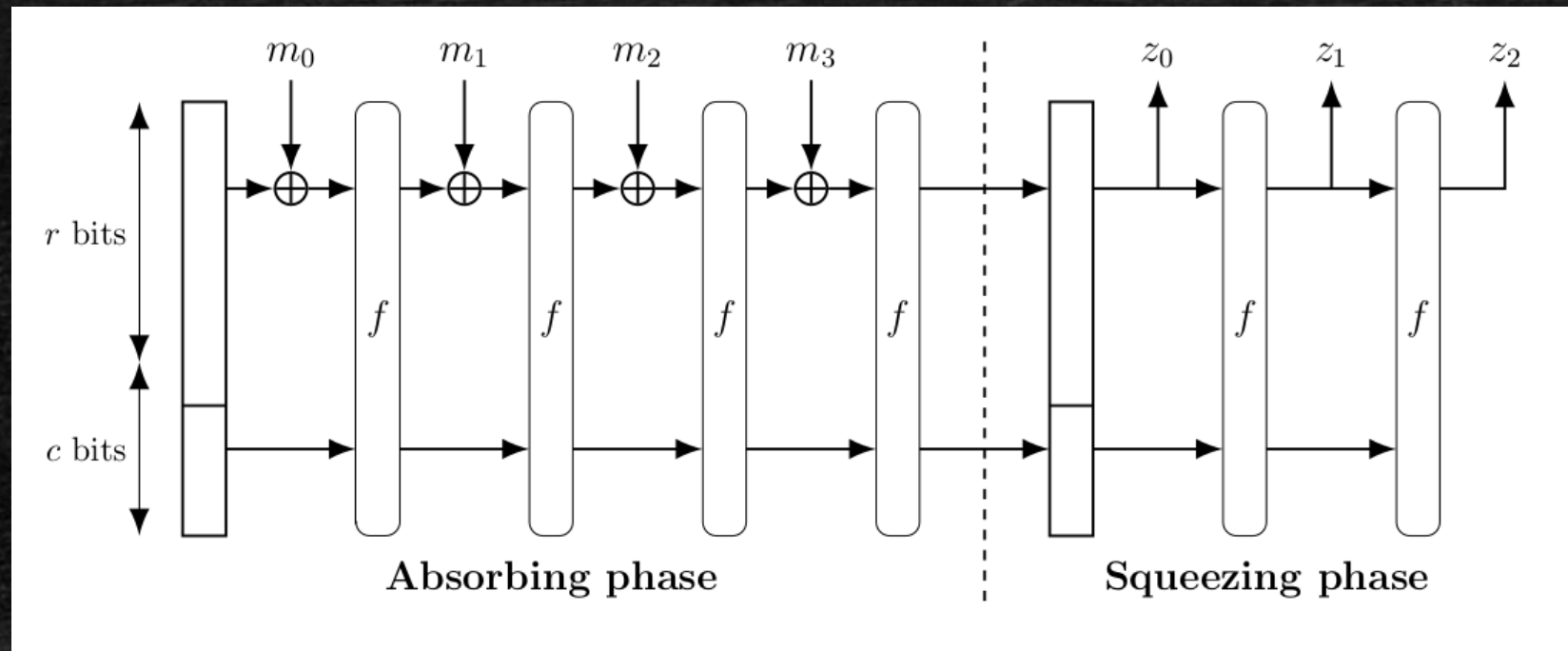
---

- Nouvelle structure utilisée dans SHA<sub>3</sub>



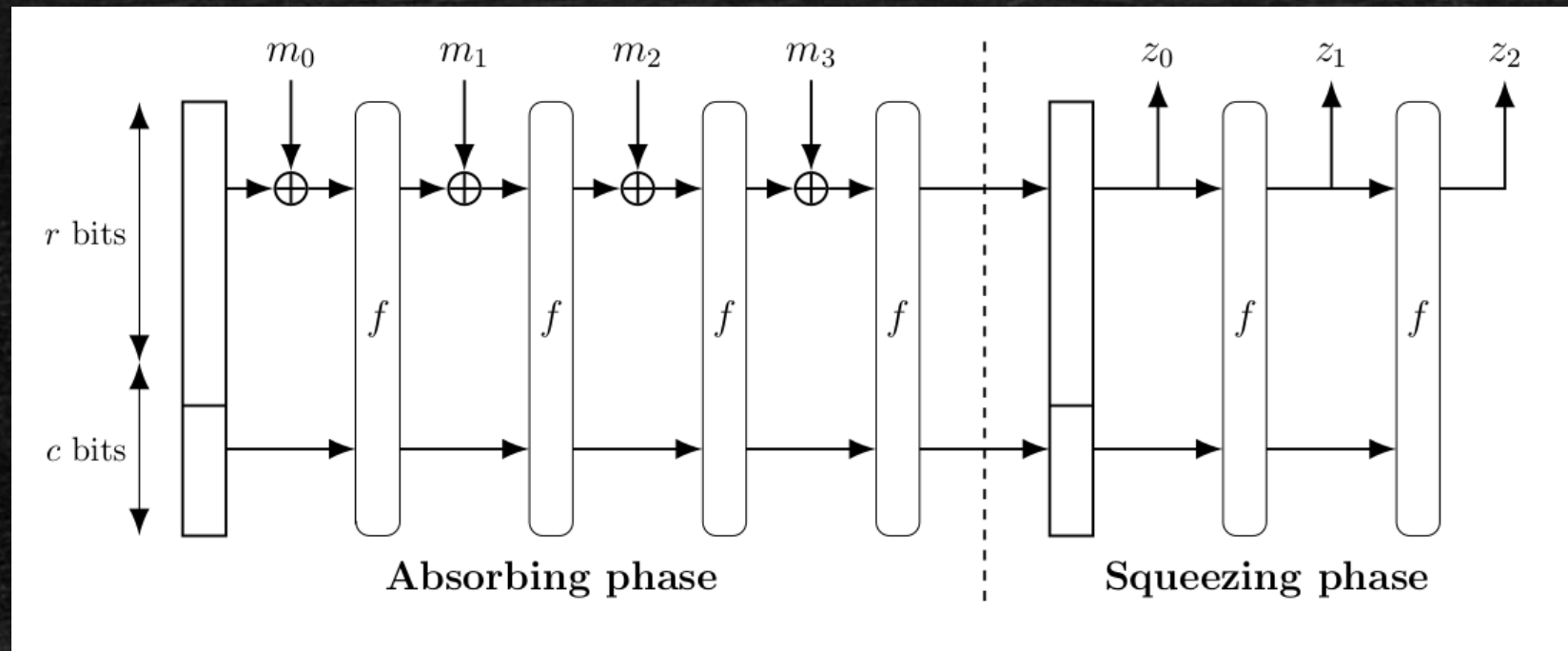
# Fonction éponge

- Permutation  $f$  sur  $r+c$  bits
- Rate sur  $r$  bits et capacité sur  $c$  bits



# Fonction éponge

- Différentes tailles de sorties possibles :  $z_1, z_2, \dots$
- Message découpé en bloc de  $r$  bits --> Performance !





# Fonction éponge

- Sécurité : Taille de la capacité
  - Résistance en collision :  $2^{c/2}$
  - Résistance en seconde-préimage :  $2^{c/2}$
  - Résistance en préimage :  $2^c$
- SHA3 : Keccak
  - Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche and Ronny Van Kee
  - Août 2015 : publication des normes FIPS 180-4 et FIPS 202
  - Normalisation de plusieurs variantes
    - Capacité de 512 bits : SHA3-224, SHA3-256, SHAKE-256 (taille de sortie variable)
    - Capacité de 1024 bits : SHA3-384, SHA3-512, SHAKE-512 (taille de sortie variable)
- Il existe des MACs basés sur la fonction éponge (différent de HMAC)





# Fonctions de hachage : Sécurité

---

- Il est recommandé d'utiliser des fonctions de hachage dont la taille d'empreinte est d'au moins **256 bits** (Guide ANSSI 2021)
- MD5 (128 bits) cassée! --> **ne pas utiliser**
- SHA1 --> **fortement déconseillée**
  - présente des vulnérabilités
  - Empreinte trop petite pour l'état de l'art actuel
- Famille de fonctions de hachage SHA-2 (structure similaire à SHA1)
  - SHA2-224, SHA2-256, SHA2-384, SHA2-512 (**utilisée**)
- Concours SHA3 (fin octobre 2012) par le NIST « au cas où » (**utilisable**)
  - Nouveau standard : Keccak gagnante (fonction éponge)
  - Empreinte de tailles variables 224, 256, 384 et 512 bits par exemple



# Fonctions de hachage : Exemples

Fonction de hachage	Taille empreinte	Remarque
MD5	128	cassé
SHA1	160	NIST (déconseillée)
SHA2 (256, 512)	256, 512	NIST
RIPEMD-*	128, 160	(déconseillée)
Whirlpool	512	
SHA3 (224, ..., 512)	224, ..., 512	NIST



# Fonctions de hachage : Exemples

```
khati@khati-ThinkPad-X280:~$ echo "voici mon message"|sha1sum
024ac31a5cef3979283fa54825d432ab93872dc7 -
khati@khati-ThinkPad-X280:~$ echo "voici mon messagf"|sha1sum
62e1ffec976a358e783f4a049639d02346c61cc2 -
khati@khati-ThinkPad-X280:~$ echo "voici mon messag2"|sha2
sha224sum sha256sum
khati@khati-ThinkPad-X280:~$ echo "voici mon messag2"|sha224sum
eab0fa6eda20f8c9154695fa3cad5e5e2bd29e1ecb57faf7e89174cf -
khati@khati-ThinkPad-X280:~$ echo "voici mon messag2"|sha256sum
9969ff68c9794a83dbce37a3d3e73df6b9275dbca009c337d6fb2734d7113879 -
```

```
>>> len("9969ff68c9794a83dbce37a3d3e73df6b9275dbca009c337d6fb2734d7113879")
64
>>> 64*4
256
>>> len("eab0fa6eda20f8c9154695fa3cad5e5e2bd29e1ecb57faf7e89174cf")
56
>>> 56*4
224
```



# Fonctions de hachage : usages

---

- Ne garantit aucune sécurité concrète seule!
- Utilisée généralement avec **un secret**
  - Avec une clé symétrique (MAC)
  - Avec un mot de passe (stockage de mots de passe)
  - Dériver des clés (secrètes)
- Vérification empreinte logicielle
  - L'**authenticité** de l'empreinte est garantie par le serveur (site web en question)

# MACs part 2

---

Basés sur une fonction de hachage (HMAC)



# HMACs

---

- « Hash-based MACs »
- Construire un MAC avec une fonction de hachage
  - Fonction de hachage : pas de clé
  - MAC : utilise une clé
- Comment introduire la clé ?
- Idées naturelles
  - Clé en tant que préfixe :  $MAC_K(M) = H(K \parallel M)$
  - Clé en tant que suffixe :  $MAC_K(M) = H(M \parallel K)$
  - Clé en tant que préfixe et suffixe :  $MAC_K(M) = H(K \parallel M \parallel K)$

# MAC basé sur une fonction de hachage

---

Clé en tant que préfixe :  $\text{MAC}_K(M) = H(K \parallel M)$

- H basée sur Merkle-Damgard : avec  $t = \text{MAC}_K(M) = H(K \parallel M)$ , on peut calculer le MAC de n'importe quelle extension de M
  - Il suffit d'utiliser t en tant que variable de chainage
  - Adapter le calcul du dernier bloc dans le cas où la taille du message est prise en compte
- Pourrait permettre de **forger des MACs** facilement
  - Dépend de H



# MAC basé sur une fonction de hachage

---

Clé en tant que suffixe :  $MAC_K(M) = H(M || K)$

- Collision  $(M, M')$  sur la fonction  $H$  (paradoxe des anniversaires  $2^{n/2}$ ) « hors ligne »
  - Collision sur  $MAC_K$  : si  $t = MAC_K(M)$  alors  $t = MAC_K(M')$
  - Le tag  $t$  est valable pour  $M'$

Clé en tant que préfixe et suffixe :  $MAC_K(M) = H(K || M || K)$

- Idée de HMAC

# HMAC

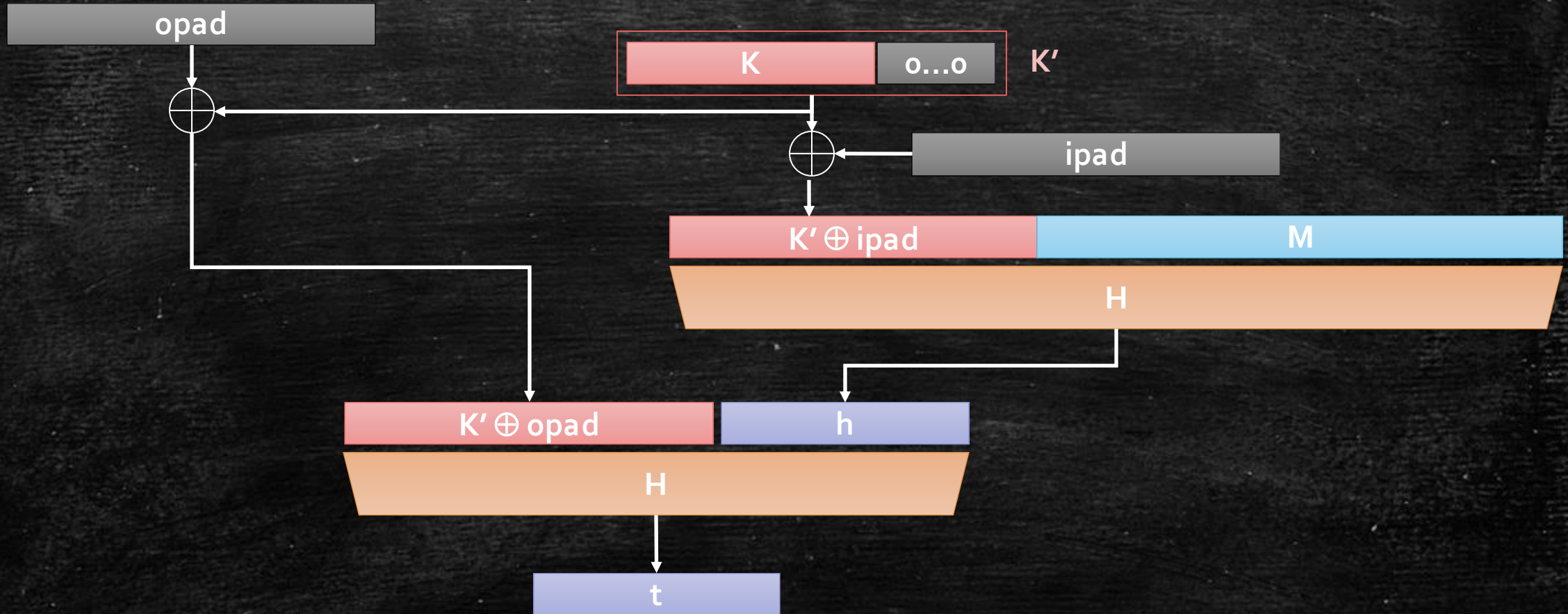
---

$$\text{HMAC}_K(M) = H(K' \oplus \text{opad} || H(K' \oplus \text{ipad} || M))$$

- Taille de bloc  $b$  ( $b=512$  généralement)
- Clé  $K'$  : Clé  $K$  « paddée » avec des '0' (taille  $b$ )
- opad : constante (octets '0x5c' --> taille  $b$ )
- ipad : constante (octets '0x36' --> taille  $b$ )



# HMAC-H



# HMAC : Sécurité

---

- Taille de la clé : recherche exhaustive sur la clé
- La fonction de hachage choisie : non cassée
- La taille de la sortie (taille du MAC  $\geq 128$  bits ) : recherche exhaustive sur le tag/MAC
- Preuve de sécurité (Crypto 2006)
  - HMAC est une bonne PRF (une bonne fonction aléatoire) sous l'hypothèse que la fonction de hachage sous-jacente est « une bonne fonction de hachage ».



# Stockage de mots de passe

---

- Idées ...
  - Stocker le mot de passe en clair dans la base de données (BD)
  - Stocker le chiffré du mot de passe dans la BD
  - Stocker l'empreinte du mot de passe dans la BD :  $h = H(\text{mdp})$
  - Stocker l'empreinte salée et le sel dans la BD :  $h = H(\text{sel}, \text{mdp})$

# MACs : Conclusion

---

- Garantir l'intégrité des données
- Utilise une clé secrète (symétrique)
- Peut utiliser
  - Sur un chiffrement par bloc (taille de clé, taille de bloc)
  - Sur une fonction de hachage (taille de clé, taille de la sortie)
- Sécurité :
  - Brique sous-jacente : chiffrement par bloc, fonction de hachage
  - Taille de la clé, taille du MAC/tag  $t$



# A retenir : MACs

---

- MAC permet de garantir l'intégrité d'une donnée
  - Permet de détecter si le message a été modifié
- MAC : construit à l'aide
  - D'un chiffrement par bloc (Ex : AES-128-CBC-MAC)
  - D'une fonction de hachage (Ex : HMAC-SHA2-256)
- Sécurité :
  - Taille de la clé ( $\geq 128$  bits car clé symétrique)
  - Taille du tag ( $\geq 128$  bits recommandé  $\rightarrow$  recherche exhaustive)
  - Sécurité de la construction + sécurité de la brique de base



# A retenir : MACs

---

- Ne permet pas de garantir la non répudiation du message
  - Clé symétrique = clé partagée
  - → au moins deux personnes partagent la clé !!
- Ne permet pas de garantir la confidentialité des données
  - Différent du chiffrement
- Possibilité de combiner :
  - Chiffrement symétrique + MAC pour garantir confidentialité +intégrité
  - (certaines constructions le font nativement → chiffrements authentifiés)



# A retenir : Fonction de hachage

---

- Propriétés de sécurité
  - Résistance que collisions
  - Résistance à la préimage / second préimage
- Sécurité :
  - Taille de la sortie  $\geq 200$  bits (à cause de l'attaque générique en collision  $2^{n/2}$ )
  - Pas de preuve
  - Confiance : résistance à la cryptanalyse
- Ne s'utilise jamais seule!!!!!!

# Annexes

---



# Modèle de sécurité

---

- Un mécanisme qui assure à la fois la confidentialité et l'intégrité des données
- Garantir la sécurité contre les **attaques à chiffrés choisis**
- Sécurité au sens le plus fort (IND-CCA)

# Chiffrement authentifié

---

- **Notions de sécurité assurées**
  - Confidentialité
  - Intégrité/ authenticité des données (« data authenticity »)
- **Combinaisons naturelles :**
  - « Encrypt-and-MAC » (pas toujours sûr)
  - « MAC-then-Encrypt » (pas toujours sûr)
  - « Encrypt-then-MAC » (composition sûre)
- Algorithmes dédiés en « une passe »



# Chiffrement authentifié

---

- Permet de garantir la confidentialité et l'authenticité des données
- Idée naturelle : utiliser un chiffrement symétrique et un MAC sur la donnée à protéger
  - Encrypt-and-MAC
  - MAC-then-Encrypt
  - Encrypt-then-MAC
- Mode de chiffrement authentifié
  - Modes spécifiques garantissant confidentialité et authenticité des données
  - Plus rapides
  - Compétition CAESAR

# Chiffrement authentifié : CAESAR

---

- Compétition CAESAR (<https://competitions.cr.yp.to/caesar.html>)
  - Début annoncé en 2013
  - Portfolio final annoncé en 2020
- Portfolio
  - Applications lightweight : 1. Ascon 2. ACORN
  - Applications hautes performances : AEGIS-128 et OCB
  - Défense en profondeur : 1. Deoxys-II 2. COLM

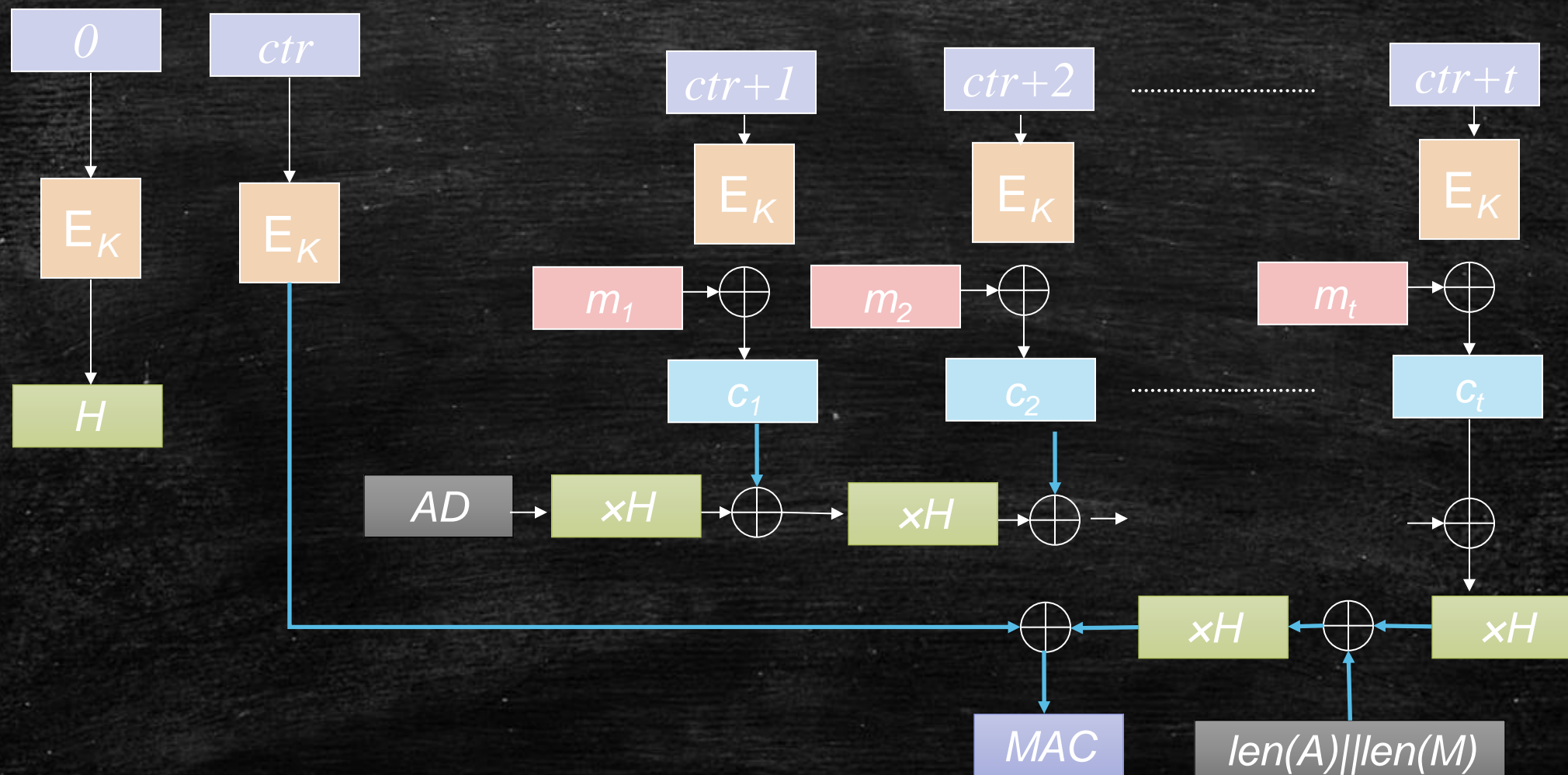


# Chiffrement authentifié

---

- AEAD : Authenticated Encryption with Associated Data
  - Associated Data : Protection en intégrité seulement
  - Message M : protégé en confidentialité et intégrité
- Mode GCM « Galois Counter Mode »
  - Standard NIST SP 800-38D
  - Très utilisé : TLS, chiffrement de fichiers, etc...
  - Multiplication dans le corps  $GF(2^{128})$

# Mode GCM





# Chiffrement authentifié

---

- AEAD : Authenticated Encryption with Associated Data
  - Associated Data : Protection en intégrité seulement
  - Message M : protégé en confidentialité et intégrité
- Mode GCM « Galois Counter Mode »
  - Très utilisé : TLS, chiffrement de fichiers, etc...
  - Multiplication dans le corps  $GF(2^{128})$
- Preuve de sécurité :
  - $E_K$  est une PRP
  - Le **nonce CTR est unique**