Introduction à la Cryptographie

4. Fonctions de hachage

Cécile Pierrot, Chargée de Recherche INRIA Nancy
cecile.pierrot@inria.fr

Supports de E. Thomé

Telecom Nancy, 2A ISS – 2021

Plan

Fonctions de hachage

Attaques simples et réductions

Constructions de fonctions de hachage

Hachage actuel : SHA2 et SHA3

MACs

Qu'est-ce que le hachage ?

Le hachage est un procédé connu de programmation.

Principe d'une table de hachage, pour stocker des données arbitraires.

Données arbitraires = chaînes de bits : $\{0,1\}^*$.

- On dispose d'une fonction de hachage $F : \{0,1\}^* \to \{0,1\}^t$.
- La longueur t est fixe.
- Les données sont stockées dans 2^t listes indépendantes au total : $L_0, \ldots, L_{2^t-1}$.
- La donnée x est stockée dans la liste $L_{F(x)}$.

En général, dans ce contexte, on n'est pas en train de parler de fonction de hachage cryptographique.

Desiderata

Pour l'application informatique ≪table de hachage≫, on souhaite que les valeurs prises par $F(x)$ soient bien réparties.

Choix d'implémentation pour une fonction de hachage :

- Rapide à calculer.
- Espace des valeurs prises $= [0, 2^t - 1]$.

Idée commune : on ne va pas rencontrer par hasard la situation $F(x) = F(x')$ (collision).

Hachage cryptographique

Le terme de fonction de hachage est aussi utilisé dans le contexte cryptographique.

- Les exemples d'applications sont multiples.
- Le hasard devient autre chose : un attaquant.

Point commun avec ce qui précède : fonction de $\{0,1\}^* \to \{0,1\}^t$ avec t fixe.

Exemples d'usage :

- Contrôle d'intégrité.
- Stockage de mots de passe.
- Signature numérique.
- Preuve de travail
- Brique de base pour l'authentification symétrique (MAC).

Contrôle d'intégrité

Il existe des logiciels de contrôle d'intégrité :

- But : vérifier d'éventuelles altérations de fichiers
  - Haché de chaque fichier d'origine dans une table de référence
  - Vérification périodique de chaque fichier par rapport à son haché dans la table
- Attention, la table ne doit pas être elle-même sujette aux manipulations.
- Exemple : tripwire.

Preuve de travail

But : dissuader les attaques par déni de service ou le spam.

- Exige un calcul complexe effectué par l'ordinateur : lent et coûteux.
- Ce calcul = travail doit être asymétrique :
  - lent et coûteux (mais réalisable) pour pour le demandeur
  - facile à vérifier pour le fournisseur de service.
- Idée : inversion d'une fonction de hachage. Le demandeur doit trouver un message à contrainte donnée donc l'empreinte finie par un certains nombres de bits à 0. Le nbr de bits donne la difficulté.
- Exemple : Bitcoin.

Stockage de mots de passe

Situation : serveur où on se connecte avec un mot de passe.

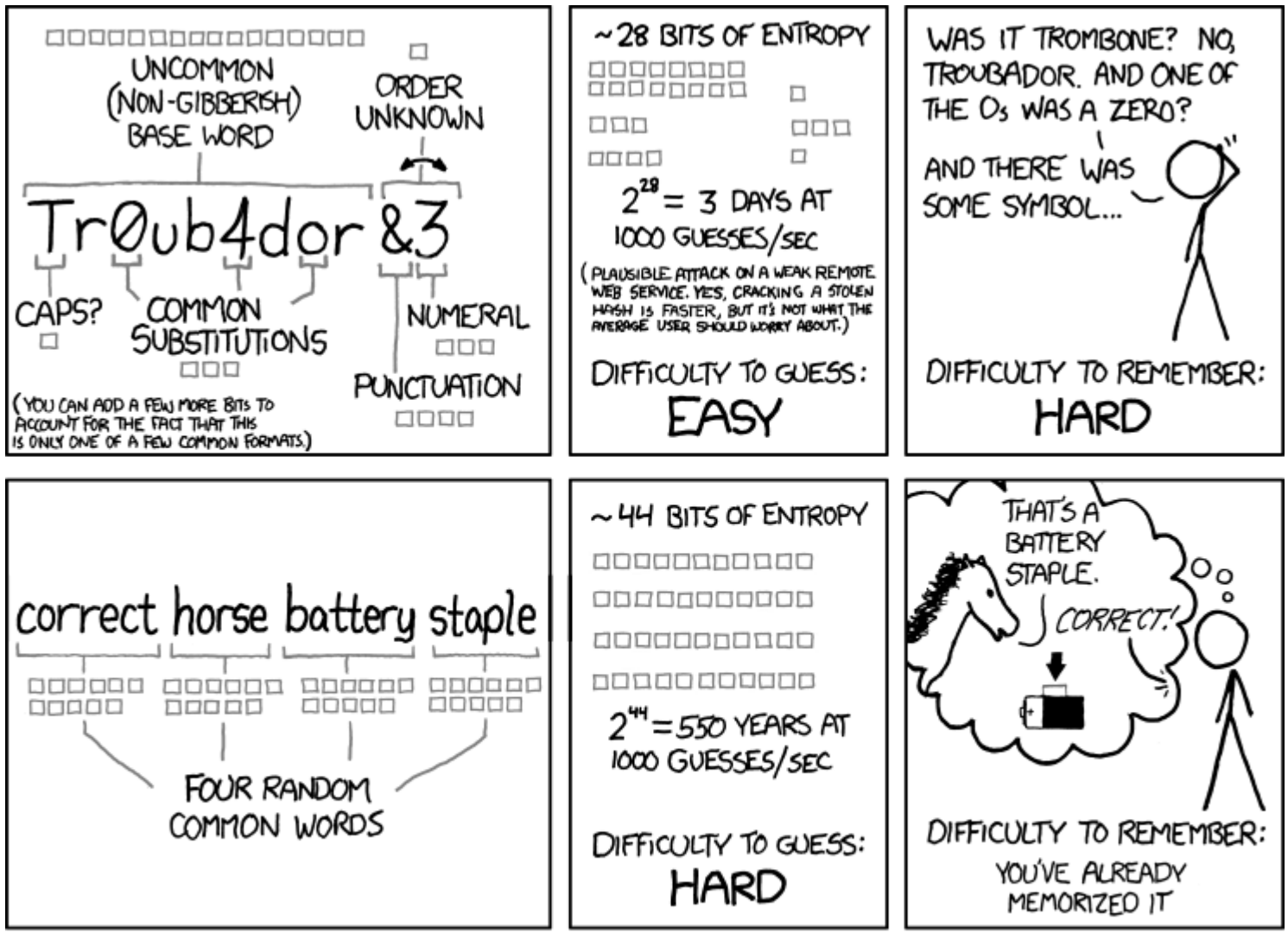
Si les mots de passe sont stockés en clair sur le serveur :

- Attaque du serveur = tous les mots de passe sniffés.
- (Mais : si M. Tartempion perd son mot de passe, il peut le retrouver).

Alternative : stocker le haché $F(\text{login}\|\text{mot de passe})$.

- Retrouver le mdp à partir du haché nécessite d'inverser F.

Interlude : Sécurité des mots de passe



Recommandations NIST

- Longueur minimale : 8 caractères
- Pas de borne supérieure, ou au moins 64 caractères possibles
- Ne pas tronquer les mots de passe
- Pas de règle de composition (mais une liste noire !)
- Si un MdP est rejeté, expliquer pourquoi, et donner des conseils
- Pas de « question/phrase secrète »
- Pas d'obligation de changer le mot de passe régulièrement (seulement si fuite/attaque $\rightarrow$ changement obligatoire !)
- Ralentir ou borner le nombre d'essais
- Permettre le copier coller (gestionnaire des mots de passe !)
- Stockage sécurisé (hachage avec sel)
- Utiliser un canal sécurisé (HTTPS, SSH, . . .)

Hachage cryptographique : propriétés voulues

On demande à une fonction de hachage cryptographique de remplir les conditions suivantes.

- Résistance au calcul de préimage.
  Étant donné y, difficile de trouver x tel que $F(x) = y$.
- Résistance au calcul de seconde préimage.
  Étant donné x, difficile de trouver $x' \neq x$ tel que $F(x) = F(x')$.
- Résistance aux collisions.
  Difficile de trouver x et x' tels que $F(x) = F(x')$.

Ces propriétés entretiennent des liens.

Résistance au calcul de préimage

Par exemple, pour le stockage de mot de passe, on a :

$$h = F(\text{login}\|\text{mot de passe}).$$

- Trouver un antécédent de $h \Rightarrow$ retrouver le mot de passe.

Résistance au calcul de préimage

Par exemple, pour le stockage de mot de passe, on a :

$$h = F(\text{login}\|\text{mot de passe}).$$

- Trouver un antécédent de $h \Rightarrow$ retrouver le mot de passe.
- Il existe des fonctions de hachage spéciales pour le stockage de mot de passe, qui sont particulièrement lentes (et donc qui ralentissent une éventuelle attaque) : bcrypt, scrypt ou Argon2.

Résistance au calcul de préimage

Par exemple, pour le stockage de mot de passe, on a :

$$h = F(\text{login} \| \text{mot de passe}).$$

- Trouver un antécédent de $h \Rightarrow$ retrouver le mot de passe.
- Il existe des fonctions de hachage spéciales pour le stockage de mot de passe, qui sont particulièrement lentes (et donc qui ralentissent une éventuelle attaque) : bcrypt, scrypt ou Argon2.

- Un terme équivalent : F doit être une fonction à sens unique (one-way function).

Collisions

Application « pile ou face par téléphone ».

Alice et Bob jouent à pile ou face par téléphone.

- Convention : pile = chaines avec une majorité de 0.
- Alice trouve x et x' tels que $F(x) = F(x')$, et $x \to$ pile, $x' \to$ face.
- Alice envoie $F(x)$ à Bob.
- Bob choisit pile ou face.
- Alice révèle x ou x' selon ce qui l'arrange.

Plan

De l'impossibilité

Qu'entend-on par la difficulté de trouver une collision ?

- $\{0,1\}^*$ est un ensemble infini.
- $\{0,1\}^t$ est un ensemble fini.

Donc il existe des collisions.

On impose une notion de difficulté calculatoire. On présume ne pas connaître d'implantation d'algorithme qui en un temps T résout le problème fixé avec probabilité $> \epsilon$.

Notion de sécurité souhaitée :

- On veut $T/\epsilon \geq 2^k$ pour un paramètre de sécurité k.
- Valeurs communes $k \geq 128$ de nos jours.

Attaques triviales

Recherche de préimage pour $y \in [0, 2^t[$.

- Algorithme 1 : essayer une valeur. $\Pr(\text{succès}) = 2^{-t}$.
- Algorithme 2 : essayer 2^t valeurs. $\Pr(\text{succès}) = ?$.

Attaques triviales

Recherche de préimage pour $y \in [0, 2^t[$.

- Algorithme 1 : essayer une valeur. $\Pr(\text{succès}) = 2^{-t}$.
- Algorithme 2 : essayer 2^t valeurs. $\Pr(\text{succès}) = \ ?$.
  - Hypothèse : $F(x)$ valeur aléatoire dans $[0, 2^t[$.
  - Probabilité que 2^t tirages évitent la valeur cible : $(1 - \frac{1}{2^t})^{2^t} \approx \frac{1}{e}$.
  - Proba de succès $\approx 1 - 1/e$ après 2^t essais.

Si le paramètre de sécurité est k, la résistance au calcul de préimage exige au minimum $t \geq k$.

Proposition

Pour qu'une préimage ne soit pas calculable en temps 2^{128}, il faut hacher sur 128 bits au moins.

Attaques triviales (2)

Recherche de collisions.

- Algorithme : essayer 2^x valeurs au hasard, calculer les hachés, et voir s'il y a une collision.
- En temps 2^x, mais quelle est la probabilité de succès π ?

Attaques triviales (2)

Recherche de collisions.

- Algorithme : essayer 2^x valeurs au hasard, calculer les hachés, et voir s'il y a une collision.
- En temps 2^x, mais quelle est la probabilité de succès π ?

C'est une application du paradoxe des anniversaires.

Paradoxe des anniversaires

Cas général n tirages parmi m valeurs (ici $n = 2^x$, $m = 2^t$).

Soit $1 - \pi = \Pr(\text{échec}) = \Pr(\text{aucune collision})$.

$$1 - \pi = 1 \times (1 - \frac{1}{m}) \times (1 - \frac{2}{m}) \times (1 - \frac{3}{m}) \times \ldots \times (1 - \frac{n-1}{m}),$$

$$\cdots 1 - \pi \leq \exp(-n^2/2m),$$

d'où la probabilité de succès $\pi \geq 1 - \exp(-n^2/2m)$.

Combien de personnes dans une salle pour avoir plus d'1 chance sur 2 que 2 d'entre-elles aient le même anniversaire ? $\pi = 0.5$ et $m = 365$ donnent $n = 23$, ce qui est contre-intuitif (en général).

Paradoxe des anniversaires

- La probabilité d'avoir des collisions s'approche de 1 quand $n/\sqrt{m}$ dépasse quelques unités.
- Pour $n = \sqrt{m}$ on a déjà des fortes chances de trouver une collision. Pour notre application cela nous donne $2^x = \sqrt{(2^t)} = 2^{t/2}$: il suffit de regarder des documents de $x = t/2$ bits.
- Contexte des fonctions de hachage : Pour avoir $T/\epsilon > 2^k$, on doit donc prendre $t \geq 2k$.

Proposition

Pour qu'il soit impossible de trouver des collisions en temps 2^{128}, il faut au moins hacher sur 256 bits.

Réductions entre propriétés

Quelle est la propriété la plus forte ? = Celle qui est le plus difficile à obtenir pour une fonction de hachage, mais la plus facile à réaliser pour un attaquant ?

1. Résistance au calcul de préimage ?
2. Résistance au calcul de seconde préimage ?
3. Résistance au calcul de collisions ?

Réductions entre propriétés

Quelle est la propriété la plus forte ? = Celle qui est le plus difficile à obtenir pour une fonction de hachage, mais la plus facile à réaliser pour un attaquant ?

1. Résistance au calcul de préimage : NON.
2. Résistance au calcul de seconde préimage : NON.
3. Résistance au calcul de collisions.

Proposition

Une fonction résistante au calcul de collisions est résistante au calcul de préimage.

Démonstration :

Pouvoir calculer une préimage $\Rightarrow$ pouvoir calculer une seconde préimage $\Rightarrow$ pouvoir calculer des collisions.

Plan

Comment faire

Principe général : ne pas prendre une bête fonction pour une table de hachage (qui ne sera peut être pas du tout cryptographique).

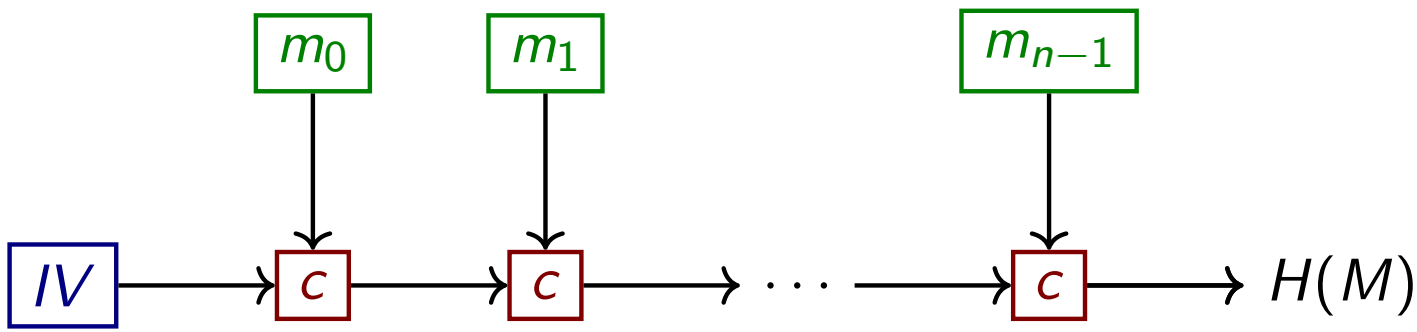
Il existe des constructions standard, largement étudiées.

- Construction de Merkle-Damgård.
- Fonctions de hachage à partir de chiffrement par blocs.
- Éponges.

Merkle-Damgård

- Brique de base : fonction de compression $c : \{0,1\}^{b+v} \to \{0,1\}^v$.
- Valeur initiale IV de v bits.
- Message de $n \times b$ bits : $M = (m_0, \ldots, m_{n-1})$.
- Haché $H(M) = h_n$, où :

$$h_0 = \mathrm{IV}, \quad h_1 = c(h_0 \| m_0), \quad \ldots, \quad h_n = c(h_{n-1} \| m_{n-1}).$$



Théorème (admis)

Si c est résistante aux collisions alors H aussi.

Attention avec Merkle-Damgård

La construction Merkle-Damgård a ses défauts :

- En réalité, les messages ont une longueur qui n'est pas multiple de quelque chose. Nécessité d'un padding $M \to \text{Pad}(M)$.
  Une idée qui ne marche pas :
  -
  - Compléter par zéro ou plusieurs 0 jusqu'à un multiple de b bits.
  -

Attention avec Merkle-Damgård

La construction Merkle-Damgård a ses défauts :

- En réalité, les messages ont une longueur qui n'est pas multiple de quelque chose. Nécessité d'un padding $M \to \text{Pad}(M)$.
  Padding standard :
  - Ajouter un 1 au bout.
  - Compléter par zéro ou plusieurs 0 jusqu'à un multiple de b bits.
  - Ajouter la longueur en bits du message d'origine.
- Étant donné $H(X)$, on peut déduire $H(\text{Pad}(X)\|Y)$ aisément, puisque $H(X)$ est une valeur de l'état interne.

Attention avec Merkle-Damgård

La construction Merkle-Damgård a ses défauts :

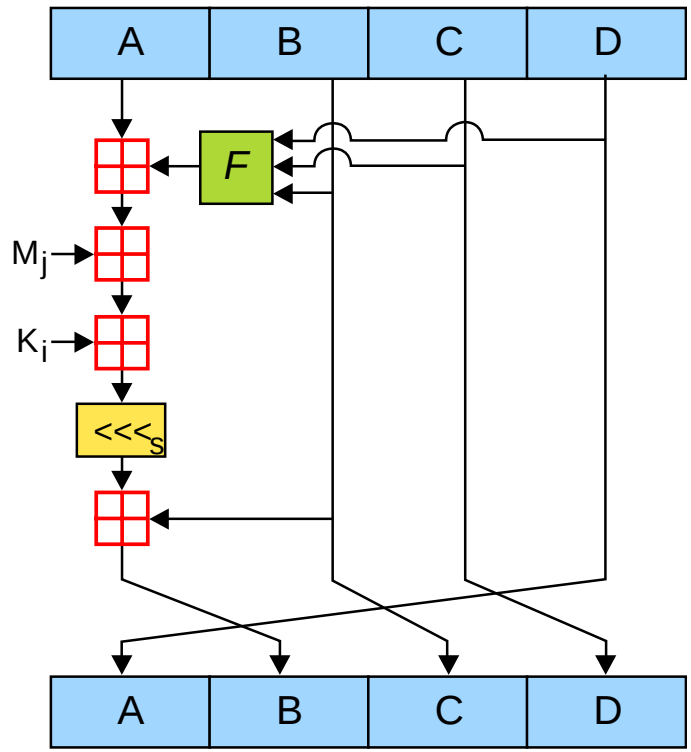
- En réalité, les messages ont une longueur qui n'est pas multiple de quelque chose. Nécessité d'un padding $M \to \text{Pad}(M)$.
  Padding standard :
  - Ajouter un 1 au bout.
  - Compléter par zéro ou plusieurs 0 jusqu'à un multiple de b bits.
  - Ajouter la longueur en bits du message d'origine.
- Étant donné $H(X)$, on peut déduire $H(\text{Pad}(X) \| Y)$ aisément, puisque $H(X)$ est une valeur de l'état interne.

La construction Merkle-Damgård reste néanmoins la plus utilisée à ce jour.

Exemple : MD5

MD5 : Message Digest 5 – Ronald Rivest – 1991 – cassé !

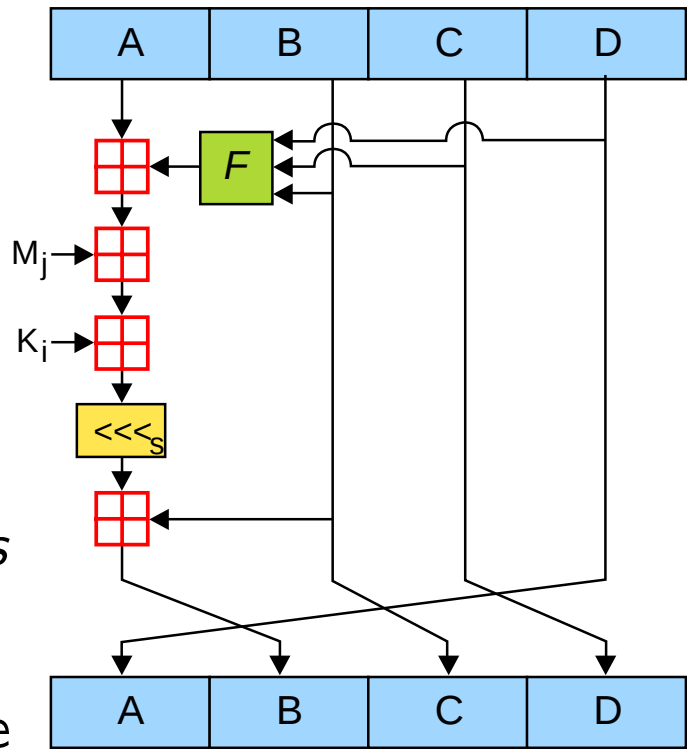
- Merkle-Damgård avec fct de compression : 640 bits $\rightarrow$ 128 bits.
  - Taille de IV = taille de l'état interne = $v = 128$ bits
  - Taille des blocs du message $= b = 512$ bits
    Eux-mêmes découpés en 16 petits blocs de 32 bits.
  - Empreinte finale sur 128 bits.
- Un fct de compression = 64 fois le dessin.



Exemple : MD5

- Etat interne 4 blocs A, B, C, D de 32 bits.
- F_1, F_2, F_3, F_4 fct non linéaire qui varient. Ex : $F_1 = (B \wedge C) \vee (\neg B \wedge D)$
- M_j l'un des 16 petits blocs, varient aussi : $M_1, M_1, \cdots M_{16}$.
- K_i cste de 32 bits, il y en a 64.
- Rotation de s bits vers la gauche, s cste entre 1 et 64, il y en a 64.

On a $4x16 = 64$ tours au total pour une seule application de la fct de compression.



MD5 : éléments

MD5 est construit à parti d'éléments très simples :

- Opérations bit à bit.
- Opérations arithmétiques.
- Décalages.

MD5 s'implante très bien.

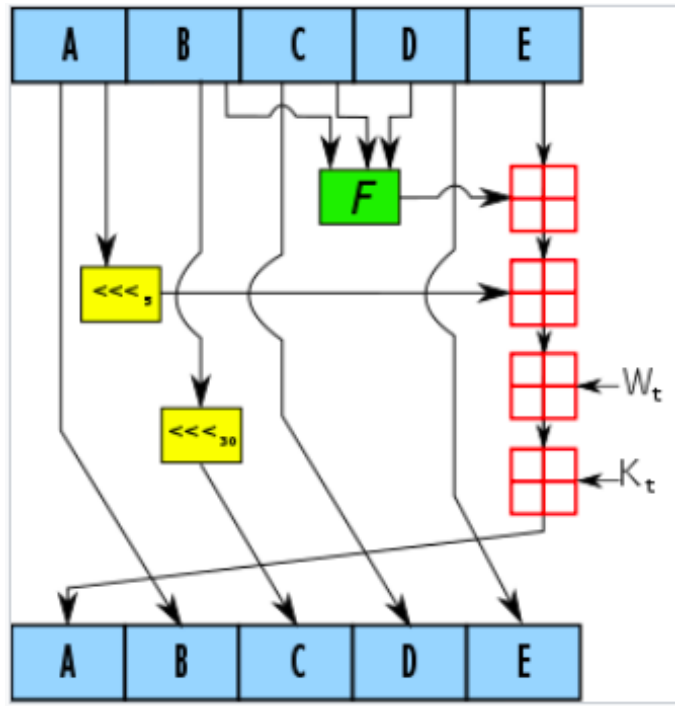
MAIS : MD5 ne satisfait pas les critères souhaités.

- Résistance aux collisions : NON.
  Il est en fait très facile d'exhiber des collisions (2004).
- Résistance au calcul de préimage : non encore cassé.

SHA-1

SHA-1 n'est pas le successeur de MD5, mais un contemporain.

- Basé sur Merkle-Damgård
- État interne et empreinte finale sur 160 bits.
- Schéma et opérations semblables à MD5.
- Cassé. En 2017, premières collisions entre 2 documents (CWI, Amsterdam).



Plan

MD5 et SHA1 $\rightarrow$ poubelle !

MD5 et SHA1 sont encore aujourd'hui les fonctions les plus utilisées.
La facilité à créer des collisions pose problème :

- Malgré leur lien peu évident avec le réel, elles peuvent créer des attaques intéressantes en utilisant le fait qu'il y a énormément de place dans un fichier pour ajouter des données inutiles.
  Attaque Nostradamus : "prédiction" du président américain en 2008. Avec une PS3 :)
- Création de faux certificats de sites web.

MD5 et SHA1 $\rightarrow$ poubelle !

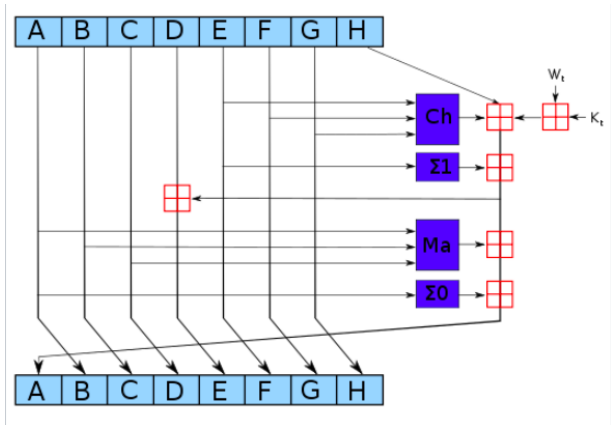
MD5 et SHA1 sont encore aujourd'hui les fonctions les plus utilisées.
La facilité à créer des collisions pose problème :

- Malgré leur lien peu évident avec le réel, elles peuvent créer des attaques intéressantes en utilisant le fait qu'il y a énormément de place dans un fichier pour ajouter des données inutiles.
  Attaque Nostradamus : "prédiction" du président américain en 2008. Avec une PS3 :)
- Création de faux certificats de sites web.

Conclusion : il faut une nouvelle fonction de hachage.

Qui sont les remplaçants ?

- Dans la même lignée que MD5 et SHA1 : SHA2 - non cassé !
  - SHA2 n'est pas UNE fct, mais une famille de fct de hachage.
  - Standardisée par les Etats-Unis en 2002 dans FIPS (Federal Information Processing Standard)
  - SHA-256, SHA-224, SHA-384, SHA-512. L'indice donne le nbr de bits de l'empreinte.



- Dans une veine tout à fait différente : SHA 3.

Finalistes SHA-3

- Compétition lancée par le NIST. Détermination du vainqueur en 2012.
- 5 Finalistes au bout de 2 tours (et 2 ans d'attaques intenses de la part de la communauté scientifique) :
  - Blake.
  - JH.
  - Keccak.
  - Skein.
  - Grøstl.

Finalistes SHA-3

- Compétition lancée par le NIST. Détermination du vainqueur en 2012.
- 5 Finalistes au bout de 2 tours (et 2 ans d'attaques intenses de la part de la communauté scientifique) :
  - Blake.
  - JH.
  - Keccak. (heureux gagnant !)
  - Skein.
  - Grøstl.
- Keccak devenu SHA3 a été décrite par Bertoni, Daemen (coinventeur de l'AES !), Peeters et Van Assche.

Finalistes SHA-3

- Compétition lancée par le NIST. Détermination du vainqueur en 2012.
- 5 Finalistes au bout de 2 tours (et 2 ans d'attaques intenses de la part de la communauté scientifique) :
  - Blake.
  - JH.
  - Keccak.
  - Skein.
  - Grøstl.
- Keccak devenu SHA3 a été décrite par Bertoni, Daemen (coinventeur de l'AES !), Peeters et Van Assche.
- Keccak est une fonction éponge.

SHA3 : Fonction éponge

Une fonction éponge c'est une amélioration d'une fonction de compression.

- Originalité : prendre des tailles aléatoires et sortir des tailles fixes.
- Généralise à la fois les fonctions de hachage et le chiffrement par flot.
- Permet de construire des preuves de sécurité.
- Fonctionnement : A partir d'une chaine de r blocs, et d'un état initial = état interne
  - Absorption : XOR du bloc 1 avec les r premiers bits de l'état, puis fonction f sur tout l'état, puis XOR du bloc 2 etc.
  - Essorage : on construit pas à pas les bits de la sortie. La taille de la chaine peut être choisie.

Ce qu'il faut définir : qui est cette fonction f de l'absorption.

SHA3 en bref

- Fonction éponge.
- Empreinte de taille variable : l'utilisateur choisit. SHA3-356, SHA3-512...
- Efficace et rapide (plus que SHA2).
- Effet avalanche : changer un seul bit dans le message en modifie en moyenne la moitié dans l'empreinte.
- Sécurité qui dépend de la version. Par exemple pour SHA3-256 :
  - Résistance aux attaques par collision : 2^{128}
  - Résistance aux attaques par pré-image et seconde pré-image : 2^{256}
  - Et les attaques quantiques ? Collisions en 2^{85}.
  - Pour résister au quantique $\rightarrow$ SHA3-384 minimum.

Plan

Fonctions de hachage

Attaques simples et réductions

Constructions de fonctions de hachage

Hachage actuel : SHA2 et SHA3

MACs

Les MACs

Définition

MAC = message authentication code = garantir l'authenticité grâce à une clé en commune entre 2 entités.

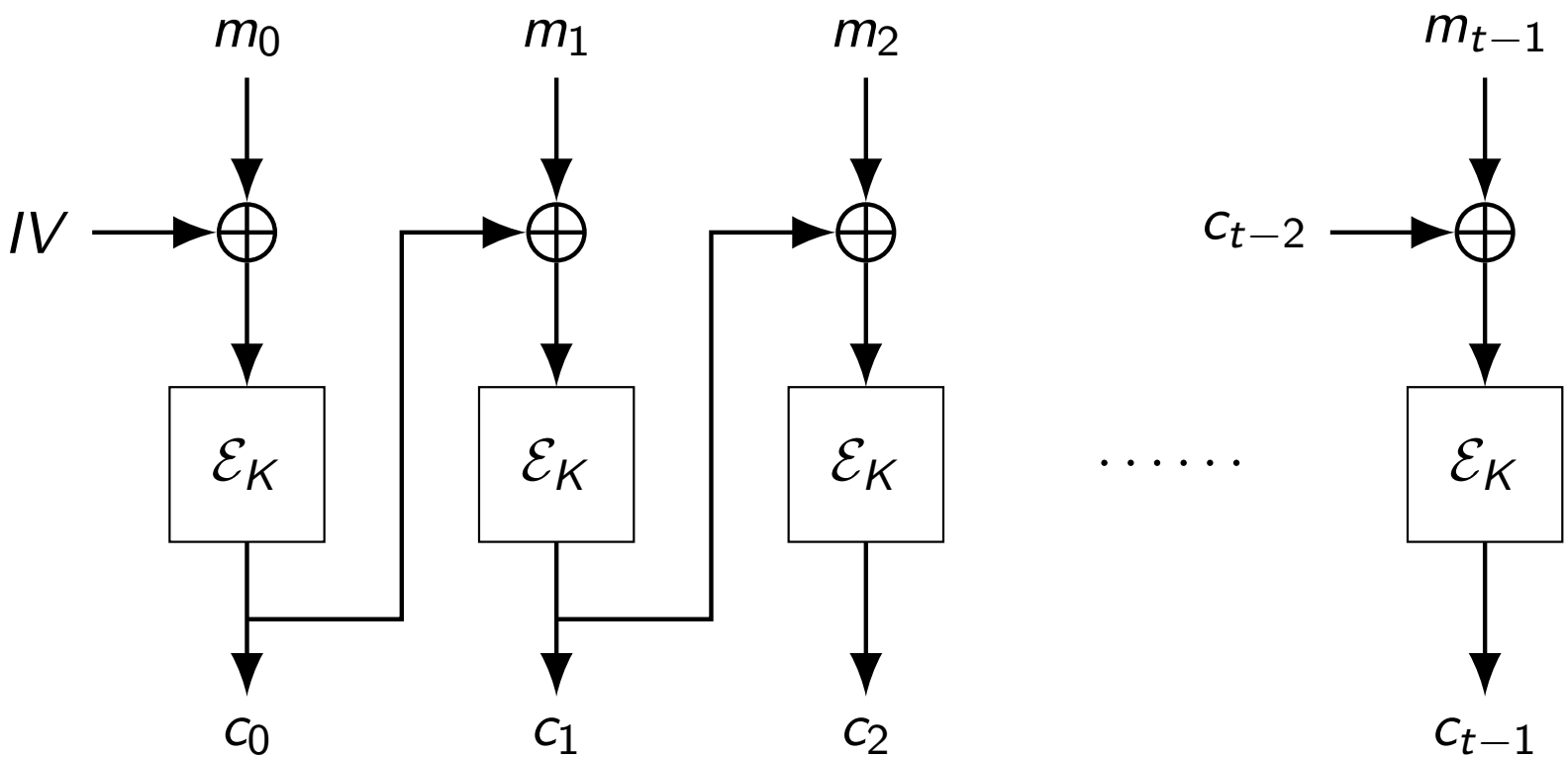
Cas d'usage typique :

- au sein d'une communication chiffrée par ailleurs avec une clé de session.
- Penser TLS, SSH, IPSEC, ...

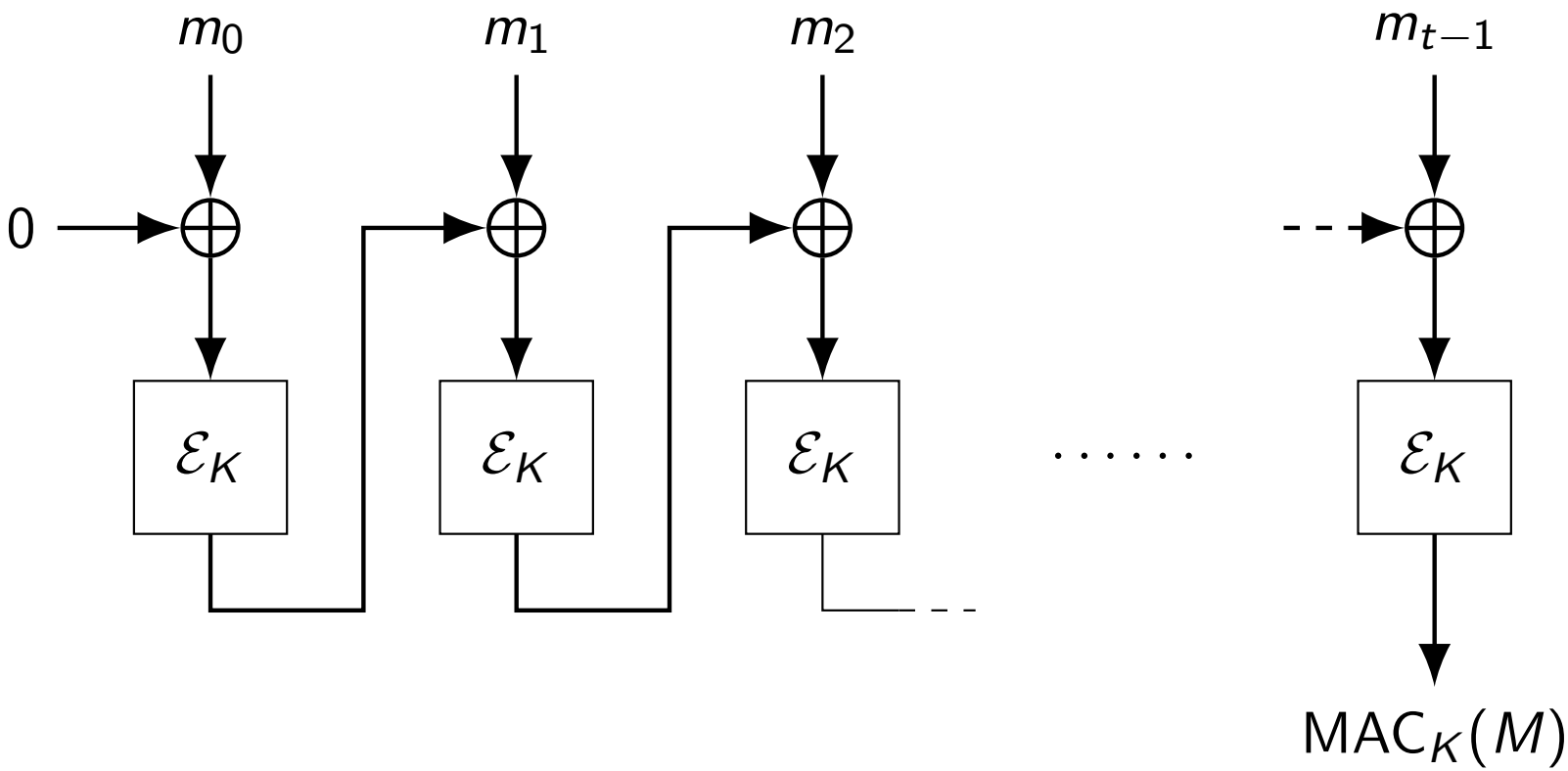
3 façons correctes de faire du MAC, qui utilisent les fct de hachage :

- CBC-MAC (attention à ses subtilités)
- HMAC
- GMAC

Le mode CBC standard, petit rappel



CBC-MAC


m_0
m_1
m_2
m_{t-1}
0
$\mathcal{E}_K$
$\mathcal{E}_K$
$\mathcal{E}_K$
$\mathcal{E}_K$
$\mathrm{MAC}_K(M)$

CBC-MAC : dangers

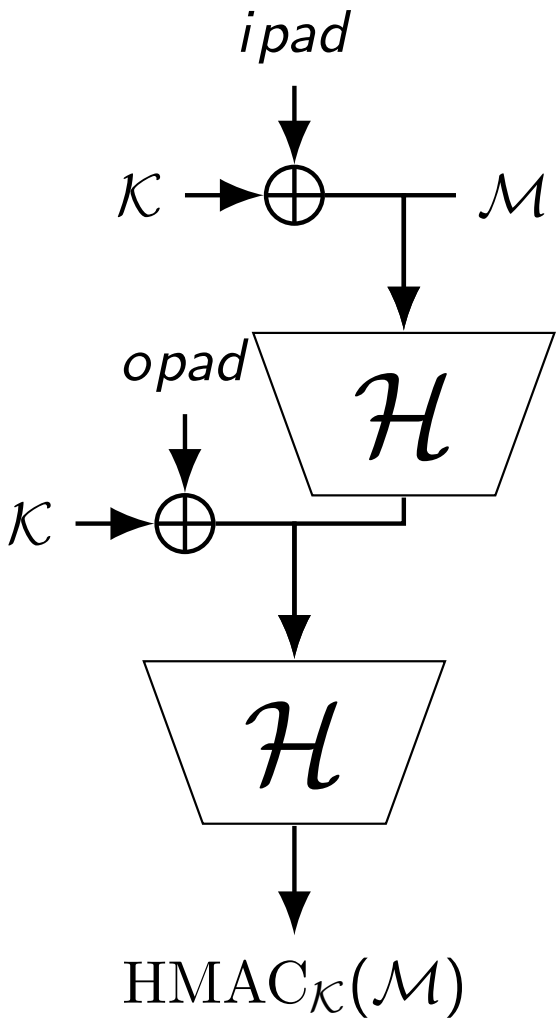
- Interdiction d'utiliser la même clef pour chiffrer par bloc un message et pour en extraire un CBC-MAC.
  Sinon Eve peut créer des faux MAC par adaptation du dernier bloc.
- Comme souvent il faut être vigilant avec l'IV : ici, étrangement, il doit vraiment être initialisé à 0.
  Sinon m_0 n'est pas authentifié.
- Attention, si la taille du message est indiqué nulle part, Eve peut concaténer deux messages dont elle connait les MAC, et produire un 3^e MAC valable.

CBC-MAC : réparations

Plusieurs choses peuvent être faites :

- Faire que la clé k dépende de la longueur du message : pas toujours pratique, mais faisable.
- Préfixer le message par sa longueur.

HMAC



Idée simple : on réduit la taille d'un message en le hachant (2 fois). Ensuite on peut utiliser un MAC qui ne fonctionne qu'avec des tailles fixes et petites.

GCM et GMAC

Galois Counter Mode (GCM) peut aussi être modifié en une variante pour l'authentification : GMAC.