

# FONCTIONS DE HACHAGE CRYPTOGRAPHIQUES

## LA PRIMITIVE SANS CLEF TRÈS UTILE

Yann Rotella

UVSQ - Université Paris-Saclay

12 mars 2026



université PARIS-SACLAY

# PLAN DU COURS

DÉFINITIONS

UTILISATION

EXEMPLES

CONSTRUCTIONS

DÉFINITIONS

UTILISATION

EXEMPLES

CONSTRUCTIONS

Le padding

Merkle-Damgard

Davies-Meyer

Autres constructions

# FONCTIONS DE HACHAGE

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



# FONCTIONS DE HACHAGE CRYPTOGRAPHIQUES

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

## PROPRIÉTÉ (FONCTION DE HACHAGE CRYPTOGRAPHIQUES)

*H est une fonction de hachage cryptographique si*

- (1) la fonction est **résistante aux collisions** : il est « difficile » de trouver  $m$  et  $m'$  avec  $m \neq m'$  tels que  $H(m) = H(m')$ .
- (2) la fonction est **résistante aux pré-images** : connaissant  $h \in \{0,1\}^n$ , il est « difficile » de trouver  $m$  tel que  $H(m) = h$ .
- (3) la fonction est **résistante aux secondes pré-images** : connaissant  $h \in \{0,1\}^n$  et  $m$  tels que  $H(m) = h$ , il est « difficile » de trouver  $m'$  tel que  $H(m') = H(m) = h$  avec  $m \neq m'$ .

-  Est-ce qu'il existe un algorithme en  $O(1)$  qui renvoie une collision ?
-  Est-ce que  $\forall h \in \{0,1\}^n$  il est difficile de trouver  $m$  tel que  $H(m) = h$  ?

# COMPLEXITÉ GÉNÉRIQUE

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

- (1) Complexité générique pour trouver une collision ?
- (2) Complexité générique pour trouver une pré-image ?
- (3) Complexité générique pour trouver une seconde pré-image ?

DÉFINITIONS

UTILISATION

EXEMPLES

CONSTRUCTIONS

Le padding

Merkle-Damgard

Davies-Meyer

Autres constructions

# OÙ SONT LES FONCTIONS DE HACHAGE ?

Les fonctions de hachage cryptographiques sont utilisées dans  
« toute » la crypto


- ▶ Blockchain (preuve de travail)
- ▶ Signatures (Cours 10)
- ▶ MACs (La semaine prochaine)
- ▶ Certificats (Cours 11)
- ▶ Stockage de mots de passe (Ce cours)

# LE STOCKAGE DES MOTS DE PASSE


## PROBLÈME


- ▶ *Les utilisateurs ne mettent pas de mots de passe complexes.*
- ▶ *Les utilisateurs mettent des mots de passe « similaires » entre les différents sites.*
- ▶ *Et ce même s'il y a des règles plus ou moins complexes.*
- ▶ *Les serveurs ont besoin de stocker les mots de passe.*
- ▶ *Les bases de données peuvent fuiter.*

# CAS 1 - STOCKAGE DES MOTS DE PASSE DIRECTEMENT

 Est-ce que les mots de passe peuvent être lus en observant la connexion ?

On distingue deux choses : les attaques « **online** » et les attaques « **offline** ».

 Comment gérer le cas où les utilisateurs mettent des mots de passe simples dans des attaques de type « online » ?





 Que se passe-t-il si la base de données du serveur fuit ?

## CAS 2 - UTILISER UNE FONCTION DE HACHAGE

On suppose maintenant que le serveur va stocker

$$H(\text{mdp})$$

au lieu de stocker  $\text{mdp}$ .




-  Donner le processus d'authentification associé à cette méthode.
-  Est-ce possible qu'un utilisateur puisse se connecter avec un autre mot de passe ?
-  Maintenant si la base de données fuite, l'attaquant a accès à  $(H(\text{mdp}_i))_{i \geq 0}$ . Est-ce faisable de retrouver  $\text{mdp}_i$  si
  1. Les mots de passe sont générés aléatoirement sur 256 bits ;
  2. La taille de sortie de la fonction de hachage est de 256 bits ;
  3. La fonction de hachage est « parfaite » (i.e. il n'y a pas d'attaque connue meilleure que l'attaque générique) ?
-  Les mots de passe n'étant pas générés aléatoirement et sont « simples », pensez-vous qu'il est possible d'en retrouver une partie ?

## CAS 2 - UTILISER UNE FONCTION DE HACHAGE ET UN SEL

Pour tout utilisateur  $i$ , on génère une valeur aléatoire dans  $\text{salt}_i \in \{0, 1\}^n$ . Le serveur va alors stocker

$$H(\text{mdp}_i || \text{salt}_i)$$

au lieu de stocker  $\text{mdp}$ .

-  Donner le processus d'authentification associé à cette méthode.
-  L'attaquant peut-il connaître  $\text{salt}_i$  ?
-  En utilisant la stratégie précédente, donner le coût de réussite de l'attaque afin de retrouver les mots de passe dits « simples ».

# STOCKAGE DE MOTS DE PASSE

On distingue trois types d'attaques :

- ▶ Attaque par dictionnaire et force brute : tester tous les mots de passe simples. Le coût, avec du sel, est de  $|D| \times N_{\text{user}}$ .
- ▶ Attaque par Rainbow Tables : si pas de sel, hacher tous les mots de passe, et réaliser le match entre cette table et la base de données.

Utiliser un gestionnaire de mots de passe, avec génération aléatoire des mots de passe, et les changer « fréquemment ».

DÉFINITIONS

UTILISATION

EXEMPLES

CONSTRUCTIONS

Le padding

Merkle-Damgard

Davies-Meyer

Autres constructions

# QUELQUES FONCTIONS DE HACHAGE

- ▶ MD5
- ▶ SHA-1
- ▶ RIPEMD-160
- ▶ SHA-2 (SHA-256)
- ▶ SHA-3 (construction vue à la fin du cours)
- ▶ BLAKE2 and BLAKE3

# SÉCURITÉ DE MD-5

## Message Digest 5, Ronald Rivest 1992

- ▶ 1996 : Première attaque (Hans Dobbertin), les cryptographes suggèrent de passer à RIPEMD où à SHA-1
- ▶ 2004 : Attaque pratique en utilisant le paradoxe des anniversaires
- ▶ 2004 : Wang, Feng, Lai and Yu : une heure sur un cluster IBM pour des collisions
- ▶ 2005 : Lenstra, Wang et de Weger montrent comment produire deux certificats TLS valides
- ▶ 2008 : Alexander Sotirov, Marc Stevens, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, et Benne de Weger créent un certificat qui apparaît légitime.
- ▶ 2012 : attaque massive (Flame malware) - Windows code signature.

# SÉCURITÉ DE SHA-1

NSA design (SHA-2 also)

- ▶ 2005 : Rijmen et Oswald 53 cassent 53 des 80 tours ( $2^{80-\epsilon}$ )
- ▶ 2005 : Wang, Yin et Yu (CRYPTO 2005) Full SHA-1 en  $2^{69}$ , 58 tours en pratique
- ▶ 2006 (Asiacrypt) : De Cannière et Rechberger 64 tours en  $2^{35}$  appels
- ▶ 2010 : Marc Stevens. Almost Full SHA-1 en  $2^{57.5}$
- ▶ 2015 : SHAppening : IV choisi - collision complète (Marc Stevens, Pierre Karpman and Thomas Peyrin), 2000\$
- ▶ 2017 : SHattered, première collision sur deux fichiers PDFs, CWI et Google,  $2^{63}$  évaluations, 6500 années d'un seul CPU.
- ▶ Avril 2019 : Gaetan Leurent, Thomas Peyrin,  $2^{68}$  pour des collisions à préfixe choisi.
- ▶ 5 janvier 2020 : Gaetan Leurent et Thomas Peyrin, deux certificats valides produits.

DÉFINITIONS

UTILISATION

EXEMPLES

CONSTRUCTIONS

- Le padding

- Merkle-Damgard

- Davies-Meyer

- Autres constructions

# PADDING (OU BOURRAGE)

## PROBLÈME

On ne sait construire des **primitives** que sur des entrées de taille fixe et les messages sont de longueur arbitraires.


$$\text{Pad} : \{0, 1\}^* \rightarrow (\{0, 1\}^n)^*$$

## DÉFINITION (PADDING)

Un **padding** est une application de  $\{0, 1\}^* \rightarrow (\{0, 1\}^n)^*$  *injective*.

On veut aussi que ce soit facile, connaissant  $\text{Pad}(m)$  de retrouver  $m$ .

# PADDING (OU BOURRAGE) - EXEMPLES

 Parmi les descriptions suivantes, donner ceux qui sont des bons systèmes de padding.

1. On rajoute des zéros à  $m$  jusqu'à avoir un multiple de  $n$  en taille.
2. On rajoute des uns à  $m$  jusqu'à avoir un multiple de  $n$  en taille.
3. On rajoute toujours un 1 puis que des 0 jusqu'à avoir une taille multiple de  $n$ .
4. On rajoute au message autant de zéros qu'il faut et on concatène son nombre de bits effectif en binaire.
5. On rajoute au message autant de zéros qu'il faut et on garde toujours les  $\log_2(n)$  derniers bits pour écrire la taille effective du dernier bloc.

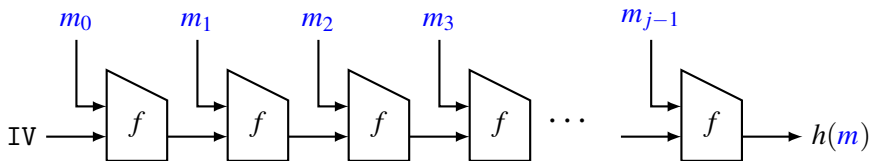
# LA CONSTRUCTION MERKLE DAMGARD


Soit  $f$  une fonction de compression.

$$f : \{0,1\}^{n+\ell} \rightarrow \{0,1\}^n$$

Soit  $\text{Pad} : \{0,1\}^* \rightarrow (\{0,1\}^\ell)^*$  un bon système de padding.

Soit  $j$  le nombre de blocs de longueur  $\ell$  après bourrage.



 Donner le pseudo-code de la construction Merkle Damgård.

On peut montrer que s'il est « difficile » de trouver des collisions sur  $f$ ,  
il l'est aussi pour  $H$  construite à partir de  $f$ .

- L'analyse se réduit alors à la seule fonction  $f$ , construite sur un principe itératif...

# PROBLÉMATIQUES

- ▶ Ce n'est pas simple de construire une fonction de compression (MD-4, MD-5, SHA-1, SHA-2).
- ▶ On peut vouloir utiliser le même circuit qu'un chiffrement par bloc (SPN)
  - ✍ Pourquoi ?
- ▶ Prouver la construction sous l'hypothèse de chiffrement par bloc idéal ?

## SOLUTION

*Construire une fonction de compression à partir d'un chiffrement par bloc.*

# FONCTIONS DE COMPRESSION À PARTIR DE CHIFFREMENT PAR BLOC

$$f : \{0,1\}^m \rightarrow \{0,1\}^n$$

avec  $m > n$ .

$$E : \{0,1\}^{k+n} \rightarrow \{0,1\}^n$$

Avec Merkle Damgard, on a principalement deux choix « immédiats » :

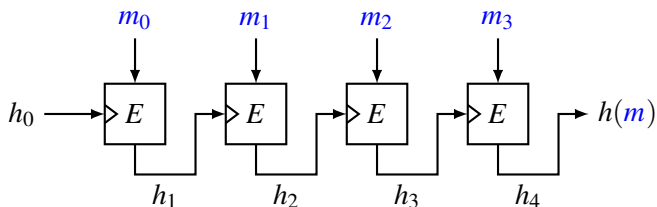
$$h_{i+1} = E_{h_i}(m_i)$$

ou

$$h_{i+1} = E_{m_i}(h_i)$$

# PREMIER CAS

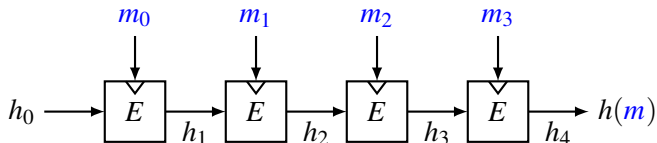
$$\forall i, h_{i+1} = E_{h_i}(m_i)$$




Expliquer pourquoi ce n'est pas une bonne idée.

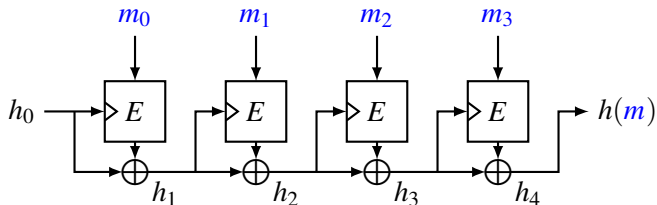
## DEUXIÈME CAS

$$\forall i, h_{i+1} = E_{m_i}(h_i)$$



 A priori OK, mais pas complètement. Pouvez-vous trouver une stratégie pour trouver une (seconde) pré-image en  $2^{\frac{n}{2}}$  ?

# DAVIES-MEYER DANS MERKLE DAMGARD



- ✍ Donner la fonction de compression associée dite de Davies-Meyer.
- Analyse de la sécurité de la construction en TD.

$$m_{i+1} = E_{h_i}(m_i) \oplus m_i$$



Dessiner cette construction.



Analyser cette construction (cf TD).

$$f = E_{h_i}(m_i) \oplus m_i \oplus h_i$$



Dessiner cette construction.

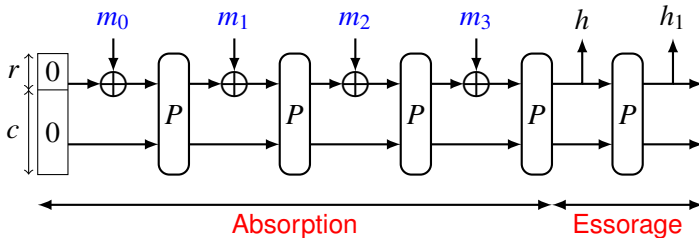


Analyser cette construction (cf TD).

# LA CONSTRUCTION EN ÉPONGE (2007)

$$P : \{0,1\}^{r+c} \rightarrow \{0,1\}^{r+c}$$

- ▶ G. Bertino, J. Daemen, M. Peeters et G. Van assche
- ▶ Si  $P$  est une permutation ou une fonction tirée aléatoirement, la construction en éponge peut être considérée comme indistinguable d'un **oracle aléatoire**.



# CONCLUSION

## IMPORTANT

- ▶ *On ne peut pas prouver la sécurité de nos fonctions de hachage.*
- ▶ *Il est compliqué de construire des fonctions de hachage (plusieurs anciennes constructions cassées).*
- ▶ *On a des standards actuels dans lesquels on a confiance (e.g. SHA-3).*
- ▶ *C'est important car utilisé presque partout en cryptographie.*
- ▶ *On a toujours  $2^{n/2}$  pour des collisions (anniversaires),  $2^n$  pour pré-image et seconde pré-image.*