Cryptographie

Fonctions de hash

Léo Colisson Palais
Master CSI 2024 – 2025

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

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

Fonctions de hash

Qu'est-ce qu'une fonction de hachage ?

Fonction de hachage

Une fonction de hachage est une fonction $h : \{0,1\}^* \to \{0,1\}^n$.

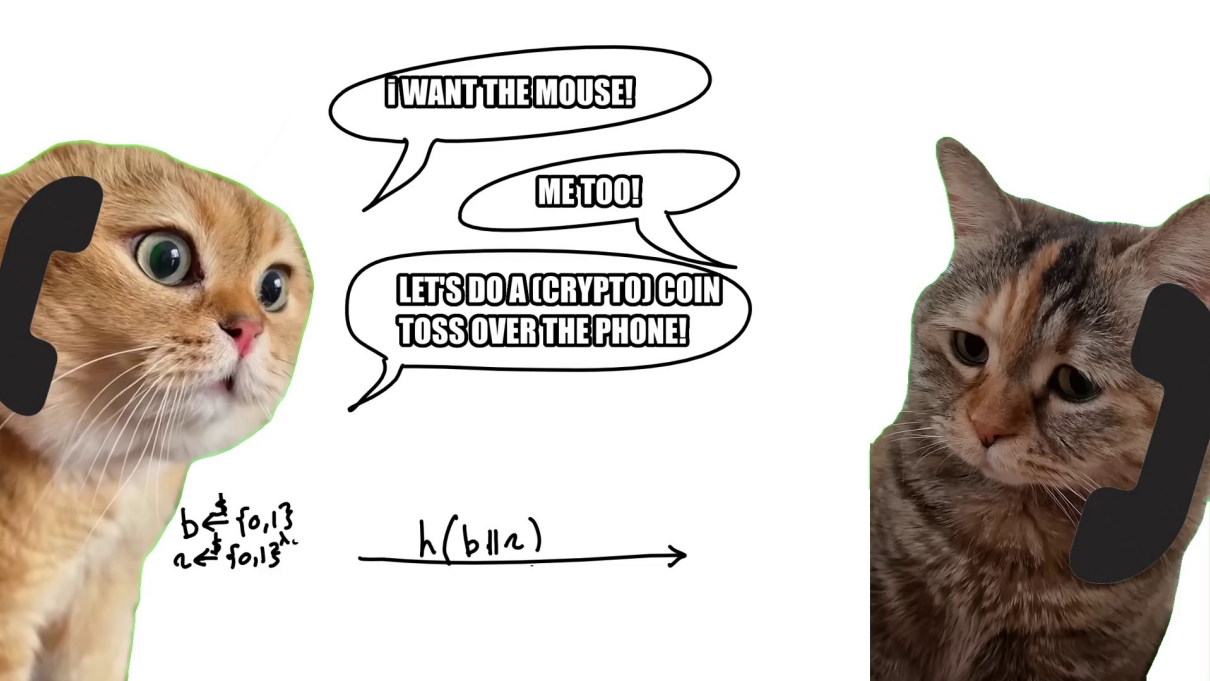
Telle qu'elle, définition peu utile, mais en fonction des applications on peut ajouter **plusieurs propriétés** :

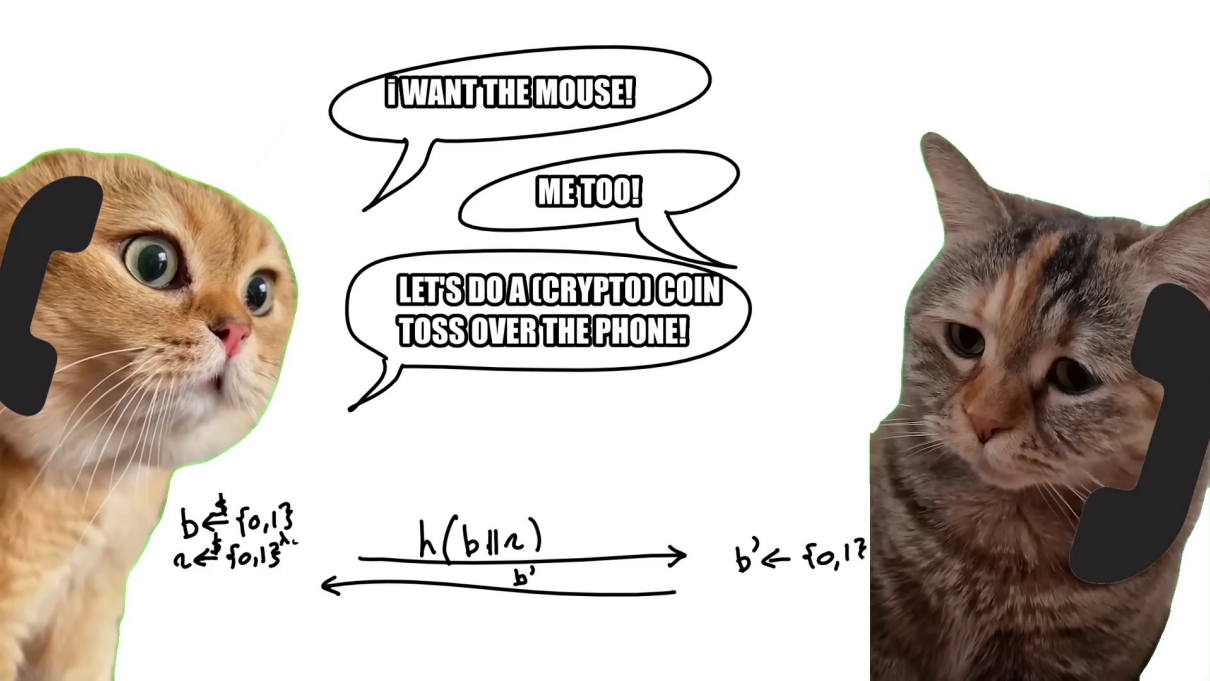
- **Résistance aux collisions**: il est difficile de trouver une collision, c'est-à-dire $x \neq x'$ tel que $h(x) = h(x')$
- **Résistance à la première préimage** (à sens unique = one-way): étant donné y, il est difficile de trouver x tel que $h(x) = y$
- **Résistance à la seconde préimage**: étant donné un x aléatoire, il est difficile de trouver $x' \neq x$ tel que $h(x) = h(x')$
- **Masquage (hiding)**: étant donné $h(r\|x)$ pour un r suffisamment long et aléatoire, il est difficile de retrouver x
- **Universalité**: Hypothèse sur l'uniformité de la distribution de sortie

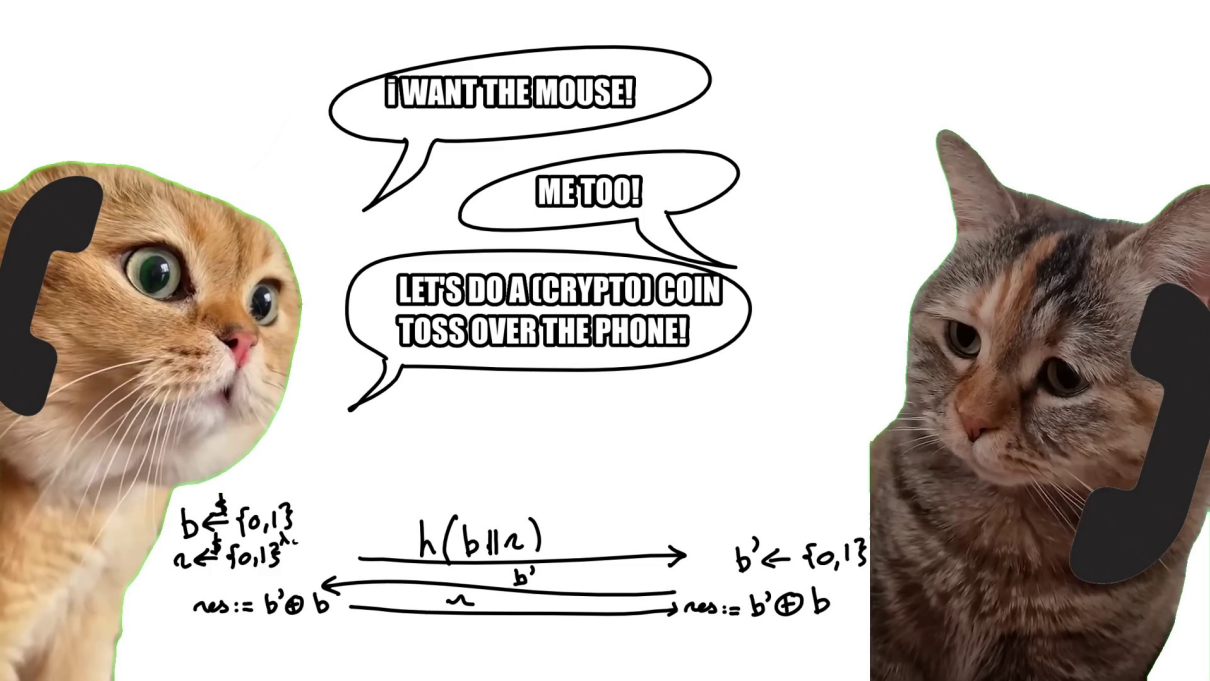

I WANT THE MOUSE!


I WANT THE MOUSE!
ME TOO!


I WANT THE MOUSE!
ME TOO!
LET'S DO A (CRYPTO) COIN TOSS OVER THE PHONE!


I WANT THE MOUSE!
ME TOO!
LET'S DO A (CRYPTO) COIN TOSS OVER THE PHONE!
$b \xleftarrow{\$} \{0,1\}$
$a \xleftarrow{\$} \{0,1\}^{\lambda}$
$h(b \| a)$


I WANT THE MOUSE!
ME TOO!
LET'S DO A (CRYPTO) COIN TOSS OVER THE PHONE!
$b \xleftarrow{\$} \{0,1\}$
$r \xleftarrow{\$} \{0,1\}^{\lambda}$
$h(b \| r)$
b'
$b' \leftarrow \{0,1\}$


I WANT THE MOUSE!
ME TOO!
LET'S DO A (CRYPTO) COIN TOSS OVER THE PHONE!
$b \xleftarrow{\$} \{0,1\}$
$r \xleftarrow{\$} \{0,1\}^{\lambda}$
$h(b \| r)$
$b' \leftarrow \{0,1\}$
b'
r
$res := b' \oplus b$
$res := b' \oplus b$

Quelle propriété de sécurité est requise ?

Avons-nous besoin de la résistance aux collisions ici ?

A. Non

B. Oui, pour se protéger contre le chat gauche malveillant

C. Oui, pour se protéger contre le chat droit malveillant

Quelle propriété de sécurité est requise ?

?

Avons-nous besoin de la résistance aux collisions ici ?

- A Non ✗
- B Oui, pour se protéger contre le chat gauche malveillant ✓ Si le chat gauche connaît r, r' tels que $h(0\|r) = h(1\|r')$ et souhaite que le résultat soit 0 : il révèle r si $b' = 0$, sinon r' (res $= b' \oplus b = 0$)
- C Oui, pour se protéger contre le chat droit malveillant ✗

Quelle propriété de sécurité est requise ?

Avons-nous besoin de la propriété de masquage ici ?

A. Non

B. Oui, pour se protéger contre le chat gauche malveillant

C. Oui, pour se protéger contre le chat droit malveillant

Quelle propriété de sécurité est requise ?

Avons-nous besoin de la propriété de masquage ici ?

A. Non ✗

B. Oui, pour se protéger contre le chat gauche malveillant ✗

C. Oui, pour se protéger contre le chat droit malveillant ✓ Si le chat droit peut retrouver b à partir de $h(b\|r)$ et souhaite que le résultat soit 0, il suffit qu'il envoie $b' := b$ (res $= b' \oplus b = 0$).

Applications des fonctions de hachage

Fonctions de hachage = nombreuses **applications** :

- Vérifier efficacement l'intégrité d'un fichier (empreinte)
- Authentification (HMAC, NMAC, Envelope MAC…)
- **Constructions IND-CCA**
- Stockage sécurisé des mots de passe
- Organiser, retrouver et/ou mettre en cache des données de manière efficace et/ou sécurisée (git, nix, …)
- Blockchain (preuve de travail)
- Engagement (=*commitments*) cryptographiques
- Pile ou face sécurisé
- Preuves à divulgation nulle de connaissance (=*Zero-Knowledge proofs*)
- Calcul multipartite
- …

Définition formelle

Comment définir formellement la résistance aux collisions ?

Première tentative : h est résistante aux collisions ssi :

$\mathcal{L}_0$

TEST(x, x'):
if $x \neq x'$ and $h(x) = h(x')$: return `true`
else: return `false`

$\approx$

$\mathcal{L}_1$

TEST(x, x'):
return `false`

?

Définition formelle

Est-ce une définition raisonnable ?

1. Oui
2. Non, car essentiellement toutes les fonctions seraient résistantes aux collisions
3. Non, car essentiellement aucune fonction ne serait résistante aux collisions

Définition formelle

Est-ce une définition raisonnable ?

1. Oui
2. Non, car essentiellement toutes les fonctions seraient résistantes aux collisions
3. Non, car essentiellement aucune fonction ne serait résistante aux collisions

✔ Très **subtile** problème d'ordre des quantificateurs. Cette définition dit : h résistante aux collisions ssi $\forall \mathcal{A}$, $|\Pr[\mathcal{A} \diamond \mathcal{L}_0 = 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_1 = 1]| \leq \text{negl}(\lambda)$. Puisque $\mathcal{A}$ apparaît **après** h, $\mathcal{A}$ peut dépendre arbitrairement de h. Ainsi $\mathcal{A}$ pourrait simplement **coder en dur** une collision (x, x'), comme :

$\mathcal{A}$
retourner TEST(x, x')

Il est vraiment difficile de trouver le code de $\mathcal{A}$, mais $\mathcal{A}$ s'exécute toujours en temps polynomial !

$\Rightarrow$ Nous aimerions fixer h **après** $\mathcal{A}$: "grain de sel" (**salt**) publique

Définition formelle

Grain de sel = valeur publique aléatoire échantillonnée pour “personnaliser” la fonction h.

Résistance aux collisions (version 1)

Une fonction de hachage $h\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ est résistante aux collisions si :

$\mathcal{L}^{h}_{\text{cr-real}}$

$s \xleftarrow{\$} \{0,1\}^\lambda$

GETSALT():

  return s

TEST(x, x'):

  if $x \neq x'$ and $h(s,x) = h(s,x')$: return `true`

  else: return `false`

$\approx$

$\mathcal{L}^{h}_{\text{cr-fake}}$

$s \xleftarrow{\$} \{0,1\}^\lambda$

GETSALT():

  return s

TEST(x, x'):

  Return `false`

Définition formelle

Problème : cette définition est **rarement utile** car nous ne vérifions jamais explicitement s'il y a une collision : nous supposons simplement qu'il n'y en a pas.

Plutôt **utilisé dans les réductions** : si $\mathcal{A}$ peut distinguer $\mathcal{L}_0$ de $\mathcal{L}_1$, alors nous pouvons construire $\mathcal{A}'$ (appelant $\mathcal{A}$ en interne) qui trouve une collision contre h (il est alors trivial de distinguer $\mathcal{L}^h_{\text{cr-real}}$ de $\mathcal{L}^h_{\text{cr-fake}}$). Par conséquent, cette définition équivalente pourrait être plus facile à utiliser :

Résistance aux collisions (version 2)

Une fonction de hachage h est résistante aux collisions si pour tout $\mathcal{A}$ borné polynomialement en temps de calcul :

$$\Pr_{\substack{s \leftarrow \{0,1\}^\lambda \\ (x,x') \leftarrow \mathcal{A}(s)}} \left[h(s,x) = h(s,x') \right] \leq \text{negl}(\lambda)$$

Spécificité du hachage de mot de passe

Hachage de mot de passe

Alice crée un site web et, pour fournir une sécurité supplémentaire, elle décide de stocker les mots de passe des utilisateurs en les chiffrant avec AES en mode CTR. Est-ce une bonne idée, pourquoi ?

A. Oui

B. Non

Hachage de mot de passe

Alice crée un site web et, pour fournir une sécurité supplémentaire, elle décide de stocker les mots de passe des utilisateurs en les chiffrant avec AES en mode CTR. Est-ce une bonne idée, pourquoi ?

A. Oui ✗

B. Non ✓ Pour vérifier les mots de passe, elle a besoin de la clé de déchiffrement, et cette clé restera sur le serveur. Si le serveur est corrompu (base de données volée…), la clé sera probablement aussi volée, révélant les mots de passe.

Hachage des mots de passe

Vous devriez toujours hacher les mots de passe que vous stockez dans une base de données !

Hachage des mots de passe

Si nous ne pouvons pas “déchiffrer” le mot de passe $(s_{\text{Alice}}, h_{\text{Alice}})$ (fonction de hachage), comment pouvons-nous vérifier si le mot de passe p est correct ?

Hachage des mots de passe

Si nous ne pouvons pas “déchiffrer” le mot de passe $(s_{\text{Alice}}, h_{\text{Alice}})$ (fonction de hachage), comment pouvons-nous vérifier si le mot de passe p est correct ?

$\Rightarrow$ Vérifier si $h(s_{\text{Alice}}, p) = h_{\text{Alice}}$!

Grain de sel

Le grain de sel : utile en théorie… Mais **le salt est aussi utile en pratique** ! (changer le hachage pour chaque mot de passe) Sinon

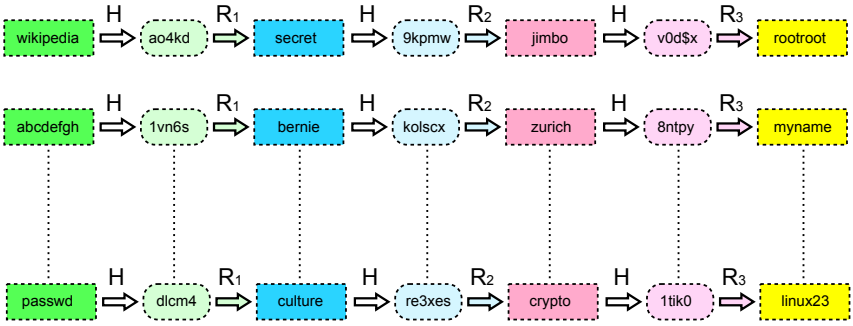
- Facile de voir si deux utilisateurs ont des mots de passe identiques
- Limiter les attaques par **pré-calcul**

Tables arc-en-ciel (rainbow tables)

Comment (essayer de) récupérer un mot de passe haché sans salt ?

- **Méthode 1**: force brute, recommencer de zéro pour chaque nouveau mot de passe
  $\Rightarrow$ inefficace en temps $O(\#_{\text{mots de passe}})$, efficace en espace $O(1)$
- **Méthode 2**: force brute et stockage pour réutilisation ultérieure
  $\Rightarrow$ efficace en temps une fois la table générée $O(\log \#_{\text{mots de passe}})$, mais nécessite un énorme stockage $O(\#_{\text{mots de passe}})$
- **Méthode 3**: **tables arc-en-ciel** = compromis temps/espace
  $\Rightarrow$ par exemple, temps modéré $O(\sqrt{\#_{\text{mots de passe}}})$, stockage modéré $O(\sqrt{\#_{\text{mots de passe}}})$

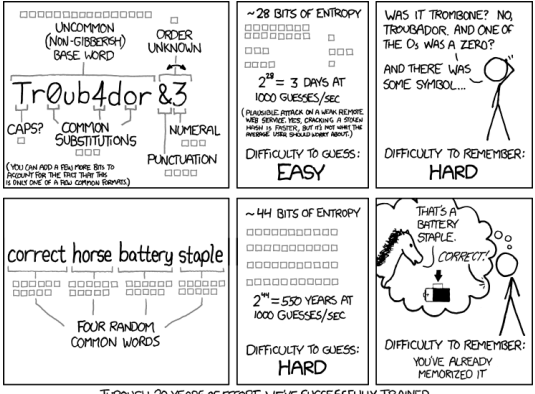
Tables arc-en-ciel



Fonction de réduction différente pour chaque colonne = évite les longues chaînes de collision

Le salage suffit-il ?

Le salage est nécessaire (coût d'attaque de n mots de passe = $n\times$ coût d'attaque d'un mot de passe), mais ce n'est pas suffisant : faible entropie des mots de passe = peu de mots de passe utilisés :



Recommandations pour le hachage de mots de passe

Mitigation : limiter les attaques par force brute avec des fonctions de hachage lentes
(idéalement sur n'importe quel matériel = les fonctions résistantes à la mémoire sont de bons candidats : supposons que la mémoire est également coûteuse/rapide partout)

✓ Recommandation OWASP	✗ À ne pas utiliser!
Argon2 (idéalement Argon2id)	NTLM (Windows : trop rapide)
Scrypt (si Argon2 n'est pas disponible)	MD5 (cassé)
Bcrypt (systèmes hérités)	SHA1 (cassé)
PBKDF2 si la conformité FIPS-140 est requise	SHA256 (trop rapide)

+ il est bon d'ajouter **poivre** (HMAC du hachage, avec une clé stockée en dehors de la base de données en cas d'injection SQL/accès de sauvegarde)

Recommandations pour le hachage de mots de passe

Mitigation : limiter les attaques par force brute avec des fonctions de hachage lentes
(idéalement sur n'importe quel matériel = les fonctions résistantes à la mémoire sont de bons candidats : supposons que la mémoire e

De nombreux excellents guides : plus de détails dans le Password Storage Cheat Sheet, testing guide…

✓ Recommandation OWASP	✗ À ne pas utiliser!
Argon2 (idéalement Argon2id)	NTLM (Windows : trop rapide)
Scrypt (si Argon2 n'est pas disponible)	MD5 (cassé)
Bcrypt (systèmes hérités)	SHA1 (cassé)
PBKDF2 si la conformité FIPS-140 est requise	SHA256 (trop rapide)

+ il est bon d'ajouter **poivre** (HMAC du hachage, avec une clé stockée en dehors de la base de données en cas d'injection SQL/accès de sauvegarde)

Construction de fonctions de hachage

Merkle-Damgård

Merkle-Damgård est utilisé dans :

- MD5 (cassé)
- SHA-1 (cassé)
- SHA-2 (toujours sûr)

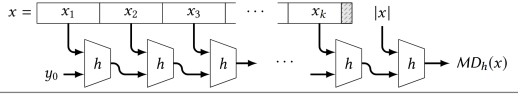
Merkle-Damgård

Construction de Merkle-Damgård

Soit $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$ une fonction de compression. Alors la transformation de Merkle-Damgård de h est $\mathrm{MD}_h\colon \{0,1\}^* \to \{0,1\}^n$ avec :

$\underline{\mathrm{MDPAD}_t(x)}$
$\ell := |x|$, as length-t binary number
while $|x|$ not a multiple of t:
  $x := x \| 0$
return $x \| \ell$

$\underline{\mathrm{MD}_h(x)}$:
$x_1 \| \cdots \| x_{k+1} := \mathrm{MDPAD}_t(x)$
// *each* x_i *is* t *bits*
$y_0 := 0^n$
for $i = 1$ to $k+1$:
  $y_i := h(y_{i-1} \| x_i)$
output y_{k+1}



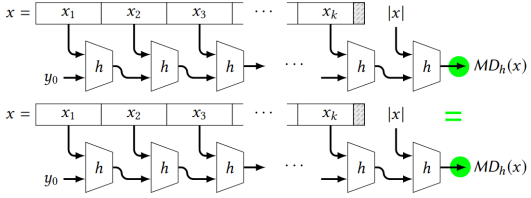
(en réalité, $h(x)$ est défini seulement si $x < 2^t$ ici, mais nous pouvons améliorer la partie de padding)

Merkle-Damgård

Théorème (Merkle-Damgård est résistant aux collisions)

Soit $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$ une fonction de compression résistante aux collisions. Alors la transformation de Merkle-Damgård MD_h est résistante aux collisions.

Preuve. Par contradiction, supposons que nous ayons trouvé une collision $x \neq x'$ contre MD_h, nous voulons trouver une collision contre h :

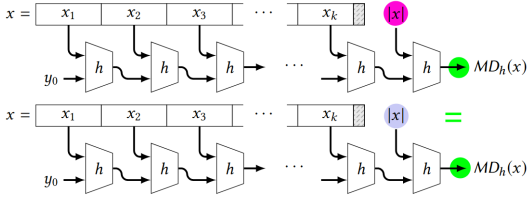


Merkle-Damgård

Théorème (Merkle-Damgård est résistant aux collisions)

Soit $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$ une fonction de compression résistante aux collisions. Alors la transformation de Merkle-Damgård MD_h est résistante aux collisions.

Preuve. Par contradiction, supposons que nous ayons trouvé une collision $x \neq x'$ contre MD_h, nous voulons trouver une collision contre h :

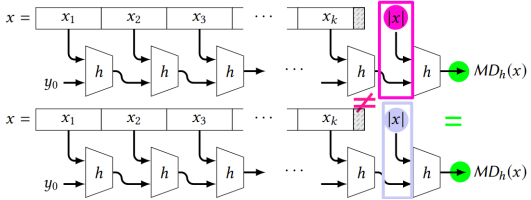


Merkle-Damgård

Théorème (Merkle-Damgård est résistant aux collisions)

Soit $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$ une fonction de compression résistante aux collisions. Alors la transformation de Merkle-Damgård MD_h est résistante aux collisions.

Preuve. Par contradiction, supposons que nous ayons trouvé une collision $x \neq x'$ contre MD_h, nous voulons trouver une collision contre h :

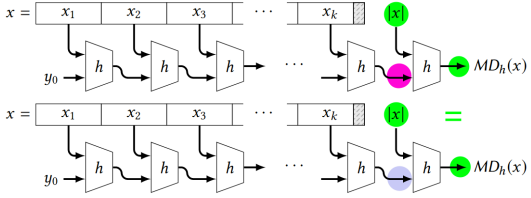


Merkle-Damgård

Théorème (Merkle-Damgård est résistant aux collisions)

Soit $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$ une fonction de compression résistante aux collisions. Alors la transformation de Merkle-Damgård MD_h est résistante aux collisions.

Preuve. Par contradiction, supposons que nous ayons trouvé une collision $x \neq x'$ contre MD_h, nous voulons trouver une collision contre h :

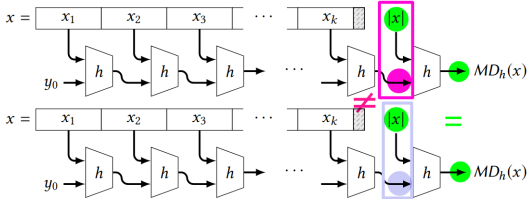


Merkle-Damgård

Théorème (Merkle-Damgård est résistant aux collisions)

Soit $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$ une fonction de compression résistante aux collisions. Alors la transformation de Merkle-Damgård MD_h est résistante aux collisions.

Preuve. Par contradiction, supposons que nous ayons trouvé une collision $x \neq x'$ contre MD_h, nous voulons trouver une collision contre h :

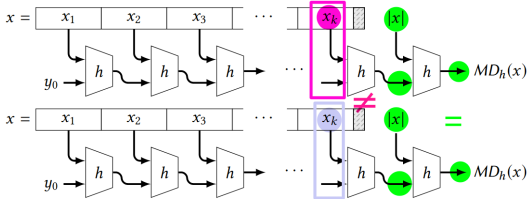


Merkle-Damgård

Théorème (Merkle-Damgård est résistant aux collisions)

Soit $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$ une fonction de compression résistante aux collisions. Alors la transformation de Merkle-Damgård MD_h est résistante aux collisions.

Preuve. Par contradiction, supposons que nous ayons trouvé une collision $x \neq x'$ contre MD_h, nous voulons trouver une collision contre h :

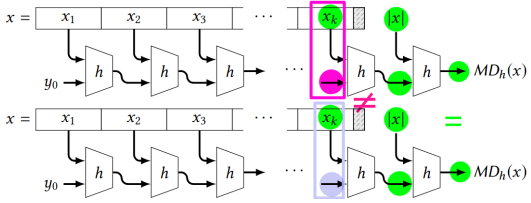


Merkle-Damgård

Théorème (Merkle-Damgård est résistant aux collisions)

Soit $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$ une fonction de compression résistante aux collisions. Alors la transformation de Merkle-Damgård MD_h est résistante aux collisions.

Preuve. Par contradiction, supposons que nous ayons trouvé une collision $x \neq x'$ contre MD_h, nous voulons trouver une collision contre h :

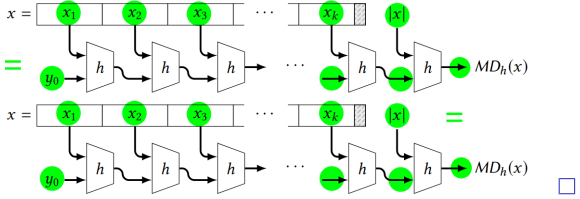


Merkle-Damgård

Théorème (Merkle-Damgård est résistant aux collisions)

Soit $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$ une fonction de compression résistante aux collisions. Alors la transformation de Merkle-Damgård MD_h est résistante aux collisions.

Preuve. Par contradiction, supposons que nous ayons trouvé une collision $x \neq x'$ contre MD_h, nous voulons trouver une collision contre h :



□

Attaque par extension de longueur

Objectif : Obtenir un code d'authentification de message (MAC, = "signature", voir le prochain cours) à partir de fonctions de hachage via $h(k\|m)$.
Problème : **Attaque par extension de longueur** : Avec la construction MD, il est possible d'obtenir $h(k\|m')$ à partir de $h(k\|m)$ (= signer un message différent sans connaître k)

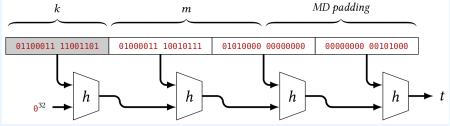
Comment ?

?

Attaque par extension de longueur

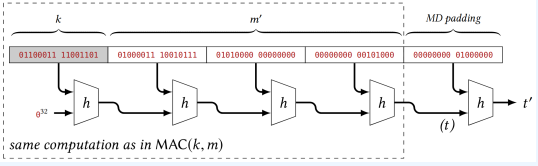
Objectif : Obtenir un code d'authentification de message (MAC, = "signature", voir le prochain cours) à partir de fonctions de hachage via $h(k\|m)$.
Problème : **Attaque par extension de longueur** : Avec la construction MD, il est possible d'obtenir $h(k\|m')$ à partir de $h(k\|m)$ (= signer un message différent sans connaître k)

Comment ?
Observation : "Connaître $H(x)$ permet de prédire le hachage de toute chaîne commençant par MDPAD(x)".



?

⇒

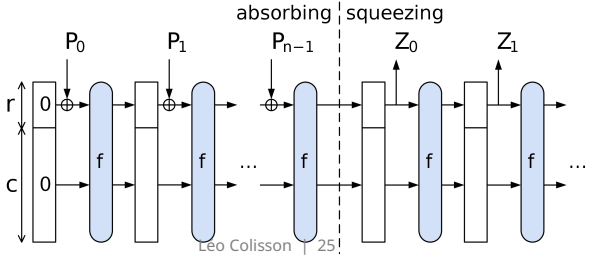


Mitigation de l'attaque par extension de longueur

L'attaque par extension de longueur est possible car le hachage **contient l'état interne complet**

Solutions :

- **Construction Wide-pipe** : Ne renvoyer, par exemple, la moitié du hachage final. Utilisée dans SHA-512/224 et SHA-512/256 (famille SHA-2), tandis que SHA-512 et SHA-256 sont vulnérables à cette attaque.
- **Construction éponge (=Sponge)** : deux phases d'absorption et d'extraction, utilisée dans SHA-3



Construction d’une fonction de compression

Comment obtenir des fonctions de compression ?

Construire une fonction de compression $h\colon \{0,1\}^{n+t} \to \{0,1\}^n$:

- À partir de zéro
- À partir d'un chiffrement par blocs (par exemple, AES) $\mathcal{E}$, choisir ce qui définit la clé/message, ajouter une rétroaction (sinon inversible) :
  - Davies–Meyer : $h(x\|k) \coloneqq \mathcal{E}_k(x) \oplus x$ (par exemple, utilisé dans SHA-2 avec un chiffrement personnalisé)
  - Matyas–Meyer–Oseas : $h(x\|x') \coloneqq \mathcal{E}_{g(x)}(x') \oplus x'$
  - Miyaguchi–Preneel : $h(x\|x') \coloneqq \mathcal{E}_{g(x)}(x') \oplus x' \oplus x$
  - Hirose

Les preuves sont généralement faites dans le **modèle Ideal Cipher Model** pour modéliser $\mathcal{E}$

Modèles de sécurité

Modèles idéaux

Dans le modèle de chiffrement idéal (resp. modèle d'oracle aléatoire), nous supposons qu'une fonction se comporte comme une permutation (resp. fonction) uniformément échantillonnée aléatoirement, à laquelle les parties (y compris l'attaquant) ne peuvent accéder que de manière boîte noire (pour les chiffrements idéaux, les parties peuvent également demander l'inverse de la fonction). Mais :

- En pratique, nous devons l'instancier avec une permutation réelle (resp. fonction), par exemple AES (resp. SHA-3) :
  $\Rightarrow$ nous avons alors une **sécurité heuristique** (pas de réduction)
- Il existe des schémas (pathologiques) sécurisés dans le ROM [Bellare, Boldyreva, Palacio 03] mais impossibles à instancier
- Pourtant, aucune construction non pathologique n'est connue pour être sécurisée dans le ROM mais non sécurisée en pratique

Modèle idéalisé $\neq$ modèle standard

Microsoft avait besoin d'une fonction de hachage pour la vérification d'intégrité ROM pour la XBOX :

- Ils ont utilisé l'algorithme de chiffrement Tiny Encryption Algorithm (TEA, chiffrement par blocs) comme chiffrement de base avec Davies-Meyer [1]
- Problème : pour tout k, il est facile de trouver k' tel que $\text{TEA}_k(m) = \text{TEA}_{k'}(m)$ (comme inverser un bit de k), et $\Rightarrow$ Trivial d'obtenir une collision :

$$\text{DM} - \text{TEA}(x \| k') = \text{TEA}_{k'}(x) = \text{TEA}_k(x) = \text{DM} - \text{TEA}(x \| k)$$

- Pourtant, TEA est toujours un bon PRP (une fois que nous choisissons un k aléatoire) !

[1] Détails de l'attaque dans [Steil, 2005]
`https://events.ccc.de/congress/2005/fahrplan/events/559.en.html`

Modèle de l'oracle aléatoire

Modèle de l'oracle aléatoire (ROM)

Un protocole est dit défini dans le modèle de l'oracle aléatoire (ROM) si toutes les parties (y compris les parties honnêtes lors de la définition du protocole et les adversaires) ont un accès oracle à H défini comme suit :

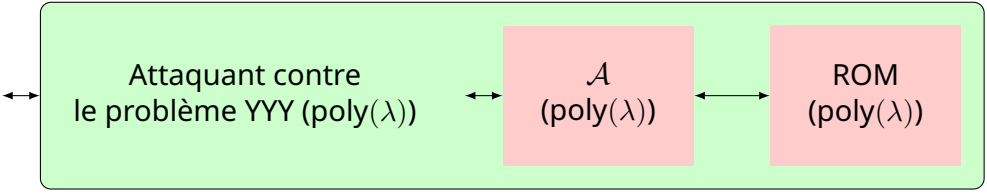
Random Oracle

$T :=$ empty assoc. array
$H(x \in \{0,1\}^*)$:
  if $T[x]$ undefined:
    $T[x] \xleftarrow{\$} \{0,1\}^{\mathsf{out}}$
  return $T[x]$

Modèle de l'oracle aléatoire

Remarques :

- **Échantillonnage paresseux** plutôt que de déterminer complètement H au début = nécessaire dans les réductions pour avoir un adversaire polynomial :



- **ROM $\neq$ PRF !!!** Dans le modèle ROM, les parties ont seulement un accès oracle à H, alors que dans le cas d'une PRF, les parties peuvent également voir le "code" de H. Cela permet de nouvelles techniques de preuve !

Modèle du chiffrement idéal

Modèle du chiffrement idéal = choisir une permutation aléatoire pour chaque clé

Modèle du chiffrement idéal (ICM)

Un protocole est dit défini dans le modèle du chiffrement idéal (ICM) si toutes les parties (y compris les parties honnêtes lors de la définition du protocole et les adversaires) ont un accès oracle à F et F^{-1} définis ainsi :

Ideal-Cipher Model

$T :=$ empty assoc. array of assoc. array
$F(k \in \{0,1\}^\lambda, x \in \{0,1\}^{\mathtt{blen}})$:
  if $T[k][x]$ undefined:
    $T[k][x] \xleftarrow{\$} \{0,1\}^{\mathtt{blen}} \setminus T[k].\text{values}$
  return $T[k][x]$
$F^{-1}(k \in \{0,1\}^\lambda, y \in \{0,1\}^{\mathtt{blen}})$:
  if $\exists x$ s.t. $T[k][x] = y$:
    return x
  else:
    $x \xleftarrow{\$} \{0,1\}^{\mathtt{blen}} \setminus T[k].\text{keys}$
    $T[k][x] := y$
    return x

Fonctions de hash principales

Comparaison des principales fonctions de hachage

Pour résumer : utilisez SHA-3, ou SHA-2 (mais pas pour les MAC). **N'utilisez jamais MD5**, SHA-0, SHA-1

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Rounds	Operations	Security against collision attacks (bits)	Security against length extension attacks (bits)	Performance on Skylake (median cpb) [61]		First published
									Long messages	8 bytes	
MD5 (as reference)		128	128 (4 × 32)	512	4 (16 operations in each round)	And, Xor, Or, Rot, Add (mod 2^{32})	≤ 18 (collisions found)[62]	0	4.99	55.00	1992
SHA-0		160	160 (5 × 32)	512	80	And, Xor, Or, Rot, Add (mod 2^{32})	< 34 (collisions found)	0	≈ SHA-1	≈ SHA-1	1993
SHA-1							< 63 (collisions found)[63]		3.47	52.00	1995
SHA-2	*SHA-224*	224	256 (8 × 32)	512	64	And, Xor, Or, Rot, Shr, Add (mod 2^{32})	112	32	7.62	84.50	2004
	SHA-256	256					128	0	7.63	85.25	2001
	SHA-384	384	512 (8 × 64)	1024	80	And, Xor, Or, Rot, Shr, Add (mod 2^{64})	192	128	5.12	135.75	2001
	SHA-512	512					256	0[64]	5.06	135.50	2001
	SHA-512/224	224					112	288	≈ SHA-384	≈ SHA-384	2012
	SHA-512/256	256					128	256			
SHA-3	*SHA3-224*	224	1600 (5 × 5 × 64)	1152	24[65]	And, Xor, Rot, Not	112	448	8.12	154.25	2015
	SHA3-256	256		1088			128	512	8.59	155.50	
	SHA3-384	384		832			192	768	11.06	164.00	
	SHA3-512	512		576			256	1024	15.88	164.00	
	SHAKE128	*d* (arbitrary)		1344			min(*d*/2, 128)	256	7.08	155.25	
	SHAKE256	*d* (arbitrary)		1088			min(*d*/2, 256)	512	8.59	155.50	