femto-st SCIENCES & TECHNOLOGIES

ISIFC 3, Crypto
Fonctions de hachage

Jean-François Couchot
Université de Franche-Comté, UFR-ST

cnrs UBFC UNIVERSITÉ BOURGOGNE-FRANCHE-COMTÉ UNIVERSITÉ DE FRANCHE-COMTÉ SUP MICRO TECH ENSMM utbm université de technologie Belfort-Montbéliard

Plan

Plan

Plan

Objectif d'intégrité : vérifier l'égalité $m_1 = m_2$

Pour ne pas analyser tous les bits de m_1 et m_2 (fastidieux)

- Souhait : une fonction H
  - Réduisant m à son empreinte $H(m)$ de taille fixe
  - Implantant l'effet d'avalanche : même si m_1 et m_2 ne diffèrent que d'1 bit $\rightsquigarrow$ $H(m_1)$ et $H(m_2)$ complètement différents
  - Reproductible : $H(m)$ ne change pas au cours du temps

Exécution de la fonction sha256 sur 2 messages proches

```
import hashlib
txt1 = " ISIFC Crypto".encode('utf-8')
print(txt1,'\n',hashlib.sha256(txt1).hexdigest())
txt2 = " ISIFB Crypto".encode('utf-8')
print(txt2,'\n',hashlib.sha256(txt2).hexdigest())
```

```
b' ISIFC Crypto'
 f5fea6cc095c19cb722f1731560bb8765971810b64b863331169220863be27a1
b' ISIFB Crypto'
 8aac87441cbf30978adf41f8347b5cb3a35e876d1551dfc79d2851ba5396373d
```

Plan

Mot de passe : jamais stocké en clair !

Hachage de mots de passe

Le hachage de mots de passe est un cas particulier très usuel d'utilisation des fonctions de hachage. L'idée principale :

1. Utilisateur ou utilisatrice : définit son mot de passe m ;
2. Système : enregistre une empreinte $h = H(m)$ du mot de passe ;
3. Lors d'une connexion : mot de passe m' saisi ;
4. Système : calcule l'empreinte $h' = H(m)$ du mot de passe puis, comparaison avec h.

Stockage de mot de passe dans une base

```
CREATE TABLE ETUDIANT
  (Numero INTEGER PRIMARY KEY,
   Nom VARCHAR(50) NOT NULL,
   Prenom VARCHAR(50) NOT NULL,
   DateNaissance DATE NOT NULL,
   Login CHAR(8) NOT NULL UNIQUE,
   MotDePasse BINARY(64) NOT NULL) -- a discuter
```

```
INSERT INTO ETUDIANT
VALUES (32823,'Marche','Claire',
'1998-10-01','cmarche',
SHA2('@L1Sc1enc3!',512))-- a discuter
```

Hachage de mots de passe : attaques

- l'espace de recherche est-il suffisamment grand ?
- la fonction de hachage utilisée est-elle rapide ?

Espace de recherche

Espace de recherche

Ce paramètre dépend de l'utilisateur. Si l'utilisateur utilise un mot du dictionnaire, l'espace de recherche est ridiculement faible !

- mots du dictionnaire : $130000 \approx 2^{17}$
- deux mots du dictionnaire : 2^{34}
- trois mots du dictionnaire : 2^{51}

Ordres de grandeur

Règles en vigueur sur l'ENT : a minima

- 8 caractères
- 1 caractère spécial (parmi 26)
- 2 chiffres
- 1 caractère alphabétique

Longueur	6	7	8	9	10
a-z (26)	2^{28}	2^{32}	2^{37}	2^{42}	2^{47}
a-zA-Z (52)	2^{34}	2^{39}	2^{45}	2^{51}	2^{57}
a-zA-Z0-9 (62)	2^{35}	2^{41}	2^{47}	2^{53}	2^{59}
+ [punct] (94)	2^{39}	2^{45}	2^{52}	2^{58}	2^{65}
ENT	-	-	2^{48}	2^{55}	2^{61}

Vitesse de la fonction de hachage

Vitesse de la fonction de hachage

- Une fonction de hachage est normalement destinée à être rapide pour être capable de hacher de grandes quantité de données.
- Pour limiter la portée d'une attaque sur des mots de passe, on a intérêt à ce que la fonction de hachage soit la plus lente possible.

→ On n'utilise pas les mêmes fonctions de hachages pour les mots de passe et pour les autres utilisations cryptographiques

Hachage cryptographique pour les mots de passe

Généralités

Dans les fonctions suivantes, on peut souvent régler :

- le temps d'exécution
- l'espace mémoire utilisé

Hachage cryptographique de mots de passe

- bcrypt, 1999, basé sur Blowfish
- PBKDF2, 2000
- scrypt, 2012
- Argon2, 2015, gagnant du Password Hashing Competition

Plan

Pseudonymiser un attribut identifiant

Pseudonymiser n'est pas anonymiser

- Remplacer la valeur d'un attribut "identifiant" par un pseudonyme.
- Pseudonyme peut être dérivé de la valeur identifiante (par exemple, en appliquant une fonction de hachage).
- Risque de mise en corrélation de données avec l'identité originale d'un sujet : réduit mais possible
- ré-identification indirecte possible en utilisant les autres attributs$\rightsquigarrow$ Pas une méthode d'anonymisation.

Pseudonymiser un attribut avec pandas

```
import pandas as pd
import hashlib
df = pd.read_csv('BankChurners.csv',
                 usecols=['CLIENTNUM', 'Customer_Age','Total_Trans_Amt'])
df['CLIENTNUM'] = df['CLIENTNUM'].astype(str)
df['CLIENTNUM_HASH'] = df['CLIENTNUM'].apply(lambda x:
                                             hashlib.sha256(x.encode('utf-8'))
                                             .hexdigest())

df.drop(['CLIENTNUM'],inplace=True)
```

Plan

Plan

Définitions

Définition (Fonction de hachage $H : \mathbb{B}^* \rightarrow \mathbb{B}^n$)

Fonction qui transforme un message m de taille quelconque en une valeur $h = H(m)$ de taille fixe n.

- h : appelé.e empreinte, le résumé ou la somme de contrôle de m
- m : appelé.e l'antécédent ou la préimage de h

Définition (Collision)

Il y a collision entre m_1 et m_2, $m_1 \neq m_2$ si $H(m_1) = H(m_2)$

Propriétés pour $H : \mathbb{B}^* \rightarrow \mathbb{B}^n$

Définition (Résistance à la préimage)

Pour toute empreinte h, il est difficile de trouver m tel que $H(m) = h$.
Conséquences :

- Difficile de retrouver un mdp m étant donnée une empreinte h stockée
- On parle de fonction à sens unique

Définition (Résistance à la seconde préimage)

Pour tout m_1, il est difficile de trouver $m_2 \neq m_1$ tel que $H(m_1) = H(m_2)$.
Remarque : on a ici une connaissance supplémentaire qui est m_1

Définition (Résistance aux collisions)

Il est difficile de trouver m_1 et $m_2 \neq m_1$ tels que $H(m_1) = H(m_2)$.
Remarque : comme $H : \mathbb{B}^* \rightarrow \mathbb{B}^n$, il y a nécessairement des collisions

Plan

Collision : nombre de tirages suffisant

Combien de tirages pour avoir une collision avec une probabilité 1/2 ?

Quel est la probabilité, en tirant au hasard k valeurs dans un ensemble de N valeurs, de tirer deux fois la même valeur ?

$$p(k) = 1 - \frac{N!}{(N-k)!} \cdot \frac{1}{N^k} \approx 1 - e^{-\frac{k(k-1)}{2N}}$$

Et inversement : $k(p) \approx \sqrt{2 \cdot N \ln\left(\frac{1}{1-p}\right)}$

Paradoxe des anniversaires

Combien de personnes sont suffisantes pour que la probabilité qu'au moins deux personnes aient leur anniversaire le même jour soit supérieure à $\frac{1}{2}$? 23 !

Conséquences pour H sur n bits

Collision avec une probabilité supérieure à $\frac{1}{2}$ en $O(\sqrt{2^n}) = O(2^{\frac{n}{2}})$: seulement $\frac{n}{2}$ bits de sécurité.

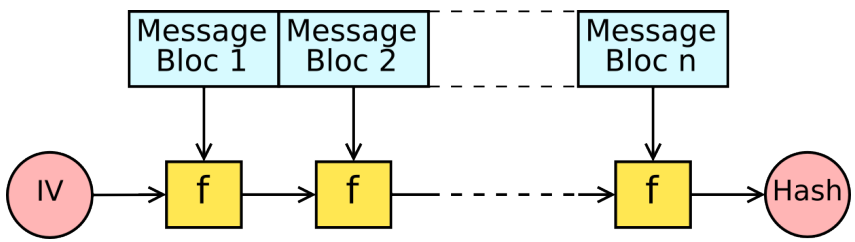
Plan

Plan

Construction de Merkle-Damgård

Idées principales

- On découpe le message m en blocs m_i de taille égale
- On utilise une fonction de compression f : $h_0 = IV$, $h_{i+1} = f(m_i, h_i)$
- Propriété : si f est résistante aux collisions, alors la construction l'est
- Appliquées dans MD5, SHA-1, SHA-2.....

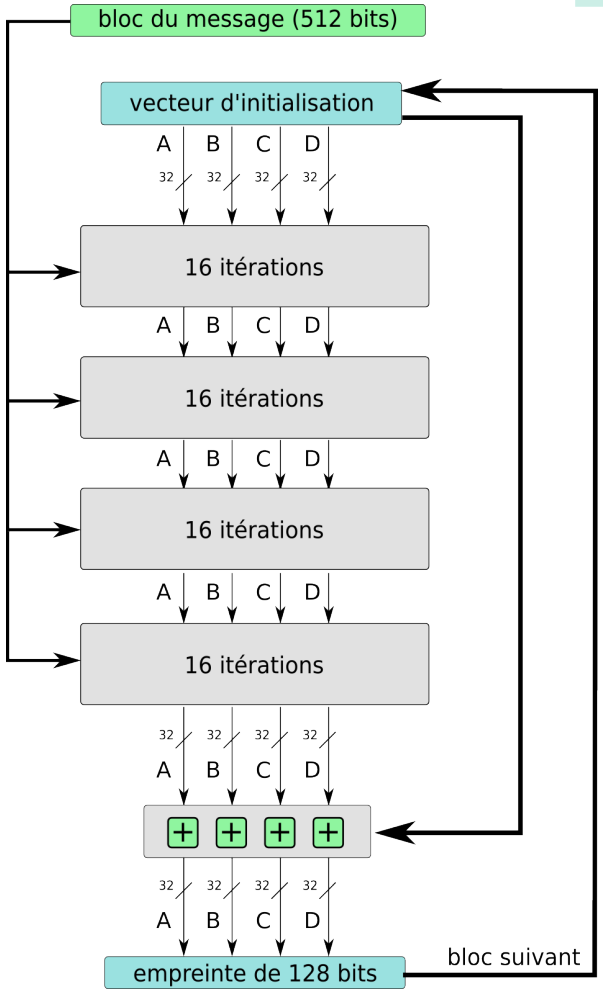


MD5

Aperçu général

Généralités

- Inventé par Ronald Rivest en 1991
- Amélioration de MD4
- Bloc de 512 bits
- Empreinte de 128 bits
- 64 rondes (ou itérations)



MD5

pas résistance à la préimage

Historique

- 2009 : Attaque en $2^{123.4}$

Attaque `hashcat` de `example0.hash` stockant des empreintes MD5 (`-m 0`)

- `hashcat -m 0 example0.hash -a 0 example.dict -r d3ad0ne.rule --show`

  Avec un dictionnaire (`-a 0 example.dict`) et des règles (`-r d3ad0ne.rule`) $\rightsquigarrow$ 1380 mdp reconstruits en qques. secondes

- `hashcat -m 0 -a 3 example0.hash`

  Brute force (`-a 3`) $\rightsquigarrow$ 1700 mdp reconstruits en qques. heures

MD5

pas résistante aux collisions

Historique $\rightsquigarrow$ MD5 n'est plus une fonction cryptographiquement sûre

- 1996 : une faille est trouvée dans la fonction de compression
- 2004 : premières collisions générées

Code de collision

```
import hashlib
from array import array
input1 = array('I',  [1634021390, 2275908181, 4156951504, 889109771, 1694748420, 2236575902,
                      4212113460, 2269944933, 798280768, 362884587, 1544942755, 1225230011,
                      1835370099, 2754879357, 2413254944, 4023487834])
input2 = array('I',  [1634021390, 2275908181, 4156951504, 889109771, 1694748420, 2236576926,
                      4212113460, 2269944933, 798280768, 362884587, 3692426403, 1225230011,
                      1835370099, 2754879357, 2413254944, 4023487834])
print(""+str(input1[5])+" vs "+ str(input2[5]))
print(""+str(input1[10])+" vs "+ str(input2[10]))

print(hashlib.md5(input1).hexdigest())
print(hashlib.md5(input2).hexdigest())
```

```
2236575902 vs 2236576926
1544942755 vs 3692426403
cee9a457e790cf20d4bdaa6d69f01e41
cee9a457e790cf20d4bdaa6d69f01e41
```

SHA-1

Aperçu général et attaques

Algorithme similaire à MD-5, mais

- 1995 : complexification de SHA-0
- Empreinte de 160 bits
- 80 rondes (ou itérations)

Attaques

- 2005 : construction d'une attaque théorique en 2^{69} (comparer à 2^{80})
- 2017 : attaque de la seconde préimage réaliste (Google, vue en TD)
- 2020 : construction d'une attaque théorique 2^{63} opérations.
- Déconseillées depuis 2010, mais :
  - Signatures SHA-1 (vues plus tard) : encore acceptées dans TLS...
  - HMAC-SHA-1 : dans plus de 8% des 1M de serveurs Alexa (2020)
- Il est important que vous ne l'utilisiez plus !

SHA-256

Aperçu général

Algorithme similaire à SHA-1, mais

- 2002 : création par la National Security Agency
- Empreinte de 256 bits
- 64 rondes (ou itérations)

Attaques

- Aucune connue ! A utiliser (et sa version SA-512)