ISIFC3-esanté, Sécurité Appliquée
Fonctions de hachage

Jean-François Couchot

Université de Franche-Comté, UFR-ST

Plan

Plan

Plan

Objectif : comparer des messages m_1 et 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

```
# coding: utf-8
import hashlib
txt = input('Txt 1: ')
hash_object = hashlib.sha256(txt.encode('utf-8'))
print(hash_object.hexdigest())

#Txt 1: ISIFC securite, le cours indispensable pour tout etudiant serieux
#5d1f7a0e20746e61256bacc4e185bcd094e62a893bb48accbf26c28ab8187617
#Txt 2: ISIFC securite, le cours indispensable pour tout etudiant serieuy
#659b4c0a5b39b3d867694fefaa089ec4b8d0a9e0df82c49aae1cbcd544e3f888
```

Applications des fonctions de hachage

Applications cryptographiques

- Intégrité d'un message lors d'une transmission (vu)
- Vérification d'un mot de passe dont est stockée uniquement l'empreinte
- Signature de l'empreinte plutôt que du message lui-même (après crypto asymétrique)

Applications non-cryptographiques

- Table de hachage

Plan

Hachage de mots de passe

Remarque préliminaire importante

On ne stocke jamais un mot de passe 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 est la suivante :

1. l'utilisateur définit son mot de passe ;
2. le système enregistre une empreinte du mot de passe ;
3. quand l'utilisateur se connecte, il donne son mot de passe ;
4. le système calcule l'empreinte du mot de passe et la compare à l'empreinte stokée.

Attaque sur une base de données de mots de passe hachés

- 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.

$\rightarrow$ 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

Examples (Hachage cryptographique pour les mots de passe)

- bcrypt, 1999, basé sur Blowfish
- PBKDF2, 2000, RFC 2898
- scrypt, 2012, RFC 7914
- Argon2, 2015, gagnant du Password Hashing Competition

Plan

Code d'Authentification de Message

Définition (Code d'Authentification de Message (MAC))

Un Message Authentication Code (MAC) est une association entre une fonction de compression et une clef secrète permettant de vérifier à la fois l'intégrité et l'authenticité des données reçues.

Utilisation

1. Alice veut envoyer un message m à Bob
2. Elle calcule le MAC avec une clef secrète K qu'elle partage avec Bob
3. Elle envoie son message et le MAC à Bob
4. Bob recalcule le MAC et vérifie qu'il est identique à celui d'Alice

Hash-based Message Authentication Code (HMAC)

Hash-based Message Authentication Code (HMAC)

- Défini dans la RFC 2104
- Utilisé dans IPSec et TLS
- $\mathrm{HMAC}(K, m) = H\left((K \oplus opad) \,||\, H\left((K \oplus ipad) \,||\, m\right)\right)$
  - H est une fonction de hachage cryptographique
  - K est la clef secrète
  - $opad = 0\text{x}5\text{c}5\text{c}5\text{c}\ldots5\text{c}5\text{c}$
  - $ipad = 0\text{x}363636\ldots3636$
  - $||$ est la concaténation
- Plusieurs variantes suivant H : HMAC MD5 (bof), HMAC-SHA-256

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 empreinte, le résumé ou la somme de contrôle de m
- m : appelé 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 $H : \mathbb{B}^* \to \mathbb{B}^n$

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 $H : \mathbb{B}^* \to \mathbb{B}^n$, 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

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

  Avec un dictionnaire (`-a 0 example.dict`) et des règles (`-r d3ad0ne.rule`) $\rightsquigarrow$ 646 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

```
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"""
```