Compression et hachage

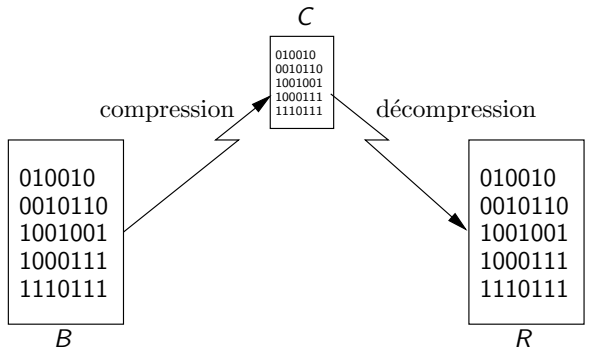
Bruno MARTIN,
Université Côte d'Azur

Plan

Pourquoi la compression de données ?

- les supports de stockage de données se remplissent en même temps que leur taille croît.
- formats de fichiers intègrent la compression :
  - images (gif, jpeg)
  - texte (pdf)
- réseaux : augmenter la bande passante en diminuant le nombre de bits transmis (pb des chiffres)
- télécommunications : utilisée dans le fonctionnement des modems (protocole V42 par exemple) et pour les transmissions par télécopie.

Compression & décompression



R n'est pas forcément identique à B. Quand

- $B = R$ on parle de compression **sans perte**
- $B \neq R$ on parle de compression **avec pertes** (plus efficace)

Types d'algorithmes de compression

- **algorithmes statistiques**, p.e. codes de Huffman, construisent un dictionnaire en effectuant une analyse statistique du document en entier :
- **algorithmes dynamiques**, p.e. Lempel et Ziv, construisent dynamiquement un dictionnaire et remplacent les données répétées par des liens vers une entrée du dictionnaire.
- **méthodes heuristiques** essayent de « deviner » les éléments du bloc de données. Ce sont les plus récentes.

Mesures d'efficacité

- **rapport de compression**, $\frac{|C|}{|B|}$ normalement < 1. Une valeur de 0,6 signifie que $|B|$ a été réduit de 40%.
- **facteur de compression**, rapport inverse du rapport de compression est normalement > 1. Plus la compression est grande, plus le facteur de compression croît.
- l'expression $100 \times (1 - \text{rapport de compression})$ est souvent utilisée. Une valeur de 40 signifie que $|B|$ à été réduit de 40%.

Méthodes naïves – Compression des espaces

Texte avec espaces non adjacents ; les retirer et mémoriser leur position par un mot binaire : 1 = un espace et 0 = autre caractère
Pour réduire la longueur $\mapsto$ 000010000000100100000000
le texte comprimé est :

000010000000100100000000|Pourréduirelalongueur

Faible nombre d'espaces dans m : $\sharp_1 m << \sharp_0 m$ et m pourra être comprimé, p.e. $0^4 10^7 10^2 10^8$.

Méthodes naïves – Compression de tête

Liste de mots triés dans l'ordre lexicographique (p.e. dictionnaire), 2 mots consécutifs partagent souvent un même n-préfixe p. On remplace p dans le second mot par n, longueur du préfixe commun.

Cod a	Coda
Codé	3é
Code -Barres	3e-Barres
Coder	4r
Cod eur	4ur
Codicille	3icille

RLE (pour *Run Length Encoding*)

Idée : répétition de $n \ll a \gg$ remplacée par na. (long. de répétition ou *Run length*)

les chaussettes de l'archiduchesse

et la transforme en

les chau@2se@2tes de l'archiduche@2se

Observation

Sortie > entrée : dans l'entrée, on n'a que des répétitions de longueur 2 compressées avec 3 caractères !
Tactique valable quand un caractère est répété plus de 3 fois.

Parmi les méthodes dans la compression MNP5 des modems [5].

Méthode statistique, le code de Huffman

But : construire un code **préfixe** optimal pour une source S_r comprenant r symboles sur un alphabet binaire basé sur :

- **réduction :** transforme S_r en une source à $r-1$ symboles S_{r-1} pour obtenir finalement une source à deux symboles : son code préfixe optimal est $\{0,1\}$
- **propriété :** sur $\{0,1\}$ si $C = \{c_1, \ldots, c_r\}$ code préfixe optimal pour une source S_r, alors $C' = \{c'_1, \ldots, c'_r, c'_{r+1}\}$:

$$\begin{cases} c'_i = c_i & 1 \leq i \leq r-1 \\ c'_r = c_r.0 \\ c'_{r+1} = c_r.1 \end{cases}$$

est un code préfixe optimal pour la source S'_{r+1}.

Principe de réduction

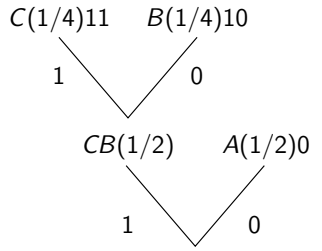
1. ordonner les symboles de source par proba. décroissantes
2. fusionner les 2 derniers symboles, avec proba = somme des proba des 2 derniers symboles
3. réordonner la liste et répéter le processus
4. arrêt quand il n'y a plus que 2 éléments

ensuite, appliquer successivement la propriété en remontant de la source à 2 symboles codés par 0 et 1 vers S_r pour avoir un code préfixe optimal pour S_r.

Exemple de code optimal

A, B et C t.q. $P(A) = 1/2$, $P(B) = P(C) = 1/4$.
après l'étape (4) de l'algorithme, on obtient :

A	$1/2$	$1/2$	$\rightarrow$	0
B	$1/4$	$1/2$	$\rightarrow$	1
C	$1/4$	$\nearrow$		



$A \mapsto 0$, $B \mapsto 10$ $C \mapsto 11$.
$ABAACABC \mapsto 010001101011$.

Principe des algorithmes dynamiques

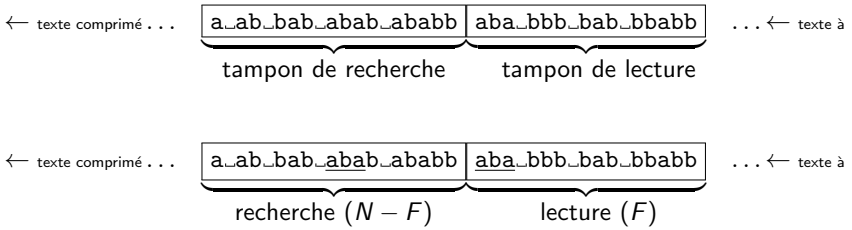
Remplacer des **facteurs** par des codes courts : indices des facteurs dans un dictionnaire construit dynamiquement.
Reposent sur une des méthodes proposées par Lempel et Ziv [8, 9].
LZ77 et LZ78 parcourent l'entrée à compresser de la gauche vers la droite en remplaçant les facteurs répétés par des pointeurs vers l'endroit où ils sont déjà apparus dans le texte.
Nombreuses variantes sur la manière de mémoriser et repérer les facteurs répétés.
LZ77 : utilisé dans `pkzip`, `gzip` ;
LZ78 : utilisé dans `compress` d'UNIX et le format d'images `gif`.

Principe de LZ77

Utilise une partie de la donnée d'entrée comme un dictionnaire.
L'algorithme fait glisser une fenêtre de N caractères sur la chaîne d'entrée de gauche à droite. Fenêtre en deux parties :

- à gauche : tampon de recherche, $N - F$ caractères : dictionnaire des lettres lues et comprimées récemment ;
- à droite : tampon de lecture, F caractères à comprimer.

Exemple



Algorithme de compression LZ77

Chercher dans les $N - F$ premiers caractères du tampon de recherche le plus long facteur répété au début du tampon de lecture de taille au plus F. Le codage est (p, ℓ, c) où :

- p : distance entre le début du tampon de lecture et la position de répétition dans le dictionnaire ;
- ℓ : la longueur de la répétition ;
- c : 1^{er} caractère du tampon de lecture différent du caractère correspondant dans le rampon de recherche.

Répétition peut chevaucher le dictionnaire et le tampon de lecture.
Après le codage, la fenêtre glisse de $\ell + 1$ caractères vers la droite.
Si on ne trouve pas de répétition, on code $(0, 0, c)$.

␣␣␣␣␣	le␣ma	ge␣dit␣abracadabra	(0, 0, l)
␣ ␣␣␣␣l	e␣mag	e␣dit␣abracadabra	(0, 0, e)
␣␣ ␣␣␣le	␣mage	␣dit␣abracadabra	(3, 1, m)
␣␣␣␣ ␣␣le␣m	age␣d	it␣abracadabra	(0, 0, a)
␣␣␣␣␣ ␣le␣ma	ge␣di	t␣abracadabra	(0, 0, g)
␣␣␣␣␣␣ le␣mag	e␣dit	␣abracadabra	(5, 2, d)
␣␣␣␣␣␣le␣ mage␣d	it␣ab	racadabra	(0, 0, i)
␣␣␣␣␣␣le␣m age␣di	t␣abr	acadabra	(0, 0, t)
␣␣␣␣␣␣le␣ma ge␣dit	␣abra	cadabra	(4, 1, a)
␣␣␣␣␣␣le␣mage ␣dit␣a	braca	dabra	(0, 0, b)
␣␣␣␣␣␣le␣mage␣ dit␣ab	racad	abra	(0, 0, r)
␣␣␣␣␣␣le␣mage␣d it␣abr	acada	bra	(3, 1, c)
␣␣␣␣␣␣le␣mage␣dit ␣abrac	adabr	a	(5, 1, d)
␣␣␣␣␣␣le␣mage␣dit␣a bracad	abra␣	␣	(4, 1, b)
␣␣␣␣␣␣le␣mage␣dit␣abr acadab	ra␣␣␣	␣	(0, 0, r)
␣␣␣␣␣␣le␣mage␣dit␣abra cadabr	a␣␣␣␣	␣	(5, 1, "")

Décompression de LZ77

A partir des triplets (p, ℓ, c), le décodage se fait en faisant glisser la fenêtre comme pour le codage. Le dictionnaire est reconstruit.

Algorithme

1. *lire un lexème*
2. *chercher la correspondance dans le tampon de recherche*
3. *écrire le facteur trouvé au début du tampon de lecture*
4. *écrire la 3^e composante du lexème à la suite*
5. *décaler le contenu des tampons de $\ell + 1$ cases vers la gauche*

	␣␣␣␣␣	l␣␣␣␣	(0, 0, l)
	␣␣␣␣l	e␣␣␣␣	(0, 0, e)
	␣␣␣le	␣m␣␣␣	(3, 1, m)
	␣␣le␣m	a␣␣␣␣	(0, 0, a)
	␣le␣ma	g␣␣␣␣	(0, 0, g)
	le␣mag	e␣d␣␣	(5, 2, d)
le␣	mage␣d	i␣␣␣␣	(0, 0, i)
le␣m	age␣di	t␣␣␣␣	(0, 0, t)
le␣ma	ge␣dit	␣a␣␣␣	(4, 1, a)
le␣mage	␣dit␣a	b␣␣␣␣	(0, 0, b)
le␣mage␣	dit␣ab	r␣␣␣␣	(0, 0, r)
le␣mage␣d	it␣abr	ac␣␣␣	(3, 1, c)
le␣mage␣dit	␣abrac	ad␣␣␣	(5, 1, d)
le␣mage␣dit␣a	bracad	ab␣␣␣	(4, 1, b)
le␣mage␣dit␣abr	acadab	r␣␣␣␣	(0, 0, r)
le␣mage␣dit␣abra	cadabr	a␣␣␣␣	(5, 1, "")

Faiblesses de LZ77

LZ77 suppose que les motifs répétés sont proches dans l'entrée.

Autre inconvénient : la taille limitée F du tampon de lecture. De ce fait, la taille de la plus longue correspondance ne peut excéder $F - 1$. F ne peut croître beaucoup, car le temps de compression croît proportionnellement à F. Il en est de même avec la taille du tampon de recherche.

Algorithme de compression LZ78

Fonctionnement analogue à LZ77 ; dictionnaire n'est plus une fenêtre coulissante. Constitué de l'intégralité du texte déjà traité. Au départ, aucun facteur n'est connu ; ajouter au dictionnaire tous les facteurs rencontrés en les numérotant. Chercher le plus long préfixe p en correspondance avec un facteur f du dictionnaire. Deux cas :

- $f \notin$ dictionnaire ; le texte restant à traiter s'écrit $c.m$ avec c caractère inconnu au dictionnaire et m le reste du texte. L'algorithme rend $(0, c)$ et ajoute c au dictionnaire.
- $f \in$ dictionnaire à la position $i > 0$; le texte restant à traiter s'écrit alors comme $f.c.m$. L'algorithme rend (i, c) et ajoute $f.c$ au dictionnaire.

Exemple

Soit la chaîne *aabbababababbbbabbbabb* à comprimer.

Dictionnaire		léxème
0	`null`	
1	a	$(0, a)$
2	ab	$(1, b)$
3	b	$(0, b)$
4	aba	$(2, a)$
5	ba	$(3, a)$
6	bb	$(3, b)$
7	bba	$(6, a)$
8	bbb	$(6, b)$
9	abb	$(2, b)$

Algorithme de décompression de LZ78

Reconstruire le dictionnaire au fur et à mesure du décodage. Les indices des facteurs seront identiques à ceux du codage et les facteurs pourront être interprétés.

Compression et décompression utilisent le même dictionnaire sans que celui-ci soit transmis. Il est entièrement reconstruit au cours de la décompression.

La suite $(0, a)(1, b)(0, b)(2, a)(3, a)(3, b)(6, a)(6, b)(2, b)$ reconstruit le dictionnaire :

Dictionnaire		léxème
0	`null`	
1	a	$(0, a)$
2	ab	$(1, b)$
3	b	$(0, b)$
4	aba	$(2, a)$
5	ba	$(3, a)$
6	bb	$(3, b)$
7	bba	$(6, a)$
8	bbb	$(6, b)$
9	abb	$(2, b)$

LZ78 en pratique

En pratique, LZ78 a un dictionnaire de taille bornée ; quand il est plein, on l'efface et on continue avec un nouveau dictionnaire.
Compression moins bonne mais cette méthode peut être employée, même si le dictionnaire n'est pas de taille suffisante pour contenir l'ensemble des facteurs du texte.
LZ78 a un grand nombre de variantes, p.e. :

- LZW par par T. Welch [7] (contrôleurs de disque dur)
- LZC dans `compress` d'UNIX.

Limites de la compression : données incompressibles

$\forall n \in \mathbb{N}$:

- 2^n mots binaires distincts de longueur n
- $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ descriptions plus courtes (mots comprimés de longueur strictement inférieure à n)

Pour tout n, il existe donc au moins un mot binaire de longueur n incompressible.
Cas des suites finies (vraiment) aléatoires [4].
On ne peut trouver aucune régularité dans une suite aléatoire (complexité de Chaitin-Kolmogorov).

Le meilleur algorithme de compression

1000 algorithmes de compression et de décompression : $C_1, C_2, \ldots, C_{1000}$ $D_1, D_2, \ldots, D_{1000}$; on construit un algorithme de compression capable de comprimer toute suite de symboles B aussi bien, à 10 bits près que le meilleur des 1000 compresseurs [1].
Comprimer B : essayer chacun des 1000 compresseurs. Mémoriser k, numéro du meilleur algorithme de compression.
Nouvelle donnée comprimée : suite C' composée du codage en binaire de k suivi du résultat de l'algorithme de compression C_k appliqué à B.
Coder k nécessite 10 bits $= \lfloor \log_2 1000 \rfloor + 1$.
Décomprimer : algorithme de décompression effectue le travail inverse. Il décode k, le nombre binaire porté sur les 10 premiers bits de C' puis applique D_k sur la donnée de C' privé de ses 10 premiers bits.

Généralisation à un nombre N quelconque

Algorithme de compression construit sera aussi efficace que les N compresseurs à $= \lfloor \log_2 N \rfloor + 1$ bits près.

Remarque : le temps de fonctionnement de notre algorithme correspond au temps cumulé des N algorithmes de compression

Plan

Classification

- MDC ou MIC (*message integrity code*) à IV=0
  - fonctions de hachage à sens unique (OWHF)
  - fonctions de hachage résistantes aux collisions (CRHF)
- MAC
  - assure authentification et intégrité
  - IV≠0

Conditions à satisfaire

Une fonction de hachage h calcule

$$z = h(x)$$

où x message de taille arbitraire et z empreinte de taille fixe.
h **à sens unique** (OW), i.e.

- $h(x)$ rapide à calculer à partir de x
- z difficile à inverser.

Résistance aux collisions

collision : 2 mots $x \neq x'$ tels que $h(x) = h(x') = z$.
h vérifie

- **préimage-résistance** si, donné z, il est difficile de trouver un x tq $h(x) = z$
- **2e préimage-résistance** si, donnés h, x, il est difficile de trouver x' tq $x \neq x'$ et $h(x) = h(x')$
- **résistance aux collisions** s'il est difficile de calculer la moindre collision (x, x').

Difficile = sécurité calculatoire (sait-on faire mieux ?)

Relations entre les propriétes de résistance

1. résistance aux collisions $\Rightarrow$ 2e préimage-résistance
2. résistance aux collisions ne garantit pas la préimage résistance
3. pour MAC h_k ; h_k pour des attaques CPA doit être à la fois :
   - 2e préimage-résistante et résistante aux collisions
   - préimage-résistante

Formalisation d'une fonction de hachage

Définition

Une fonction de hachage Π est la donnée de 2 algos PPT (Gen, h) qui satisfont :

- *Gen est un algo PPT reçevant 1^n et produisant s utile pour choisir une fonction de hachage au sein d'une famille*
- *$\exists \ell$ polynôme tq h_s sur l'entrée $x \in \{0,1\}^\star$ fournit $h_s(x) \in \{0,1\}^{\ell(n)}$ où n est la valeur du paramètre de sécurité implicite dans s*

Si h_s n'est définie que pour des entrée $x \in \{0,1\}^{\ell'(n)}$ pour $\ell'(n) > \ell(n)$, (Gen, h) est une fonction de hachage à longueur fixée sur des entrées de taille $\ell'(n)$ (fonction de compression).

Formalisation de la sécurité

Définition (Expérience de recherche de collisions Hash-coll$_{A,\Pi}(n)$)

1. *Engendrer s par Gen(1^n)*
2. *l'adversaire A reçoit s et construit x, x'. Si Π est une fonction de compression d'entrées de taille $\ell'(n)$, $x, x' \in \{0,1\}^{\ell'(n)}$*
3. *la sortie de l'expérience est 1 ssi $x \neq x'$ et $h_s(x) = h_s(x')$. Dans ce cas, A a trouvé une collision.*

Formalisation de la résistance aux collisions

Définition

Une fonction de hachage $\Pi = (Gen,h)$ est résistante aux collisions si pour tout adversaire A PPT, il existe une fonction negl tq

$$Pr(Hash\text{-}coll_{A,\Pi}(n) = 1) \leq negl(n)$$

C'est la plus forte condition de sécurité qu'on puisse demander à une fonction de hachage.

Attaques

- contre MDC :
  - OWHF : donné z, trouver x tq $h(x) = z$.
  - CRHF : trouver deux entrées $x \neq x'$ tq $h(x) = h(x')$ (attaque des anniversaires).
- contre MAC :
  - sans connaître k, donnés $(x_i, h(x_i))$, calculer $x, h_k(x)$, $x \neq x_i$
  - KPA, CPA, ...
  - falsification sélective et existentielle

(beaucoup plus de résultats de sécurité prouvée)

Objectifs de sécurité

Type de H	Conception	sécurité	Adversaire
OWHF	préimage-res	2^n	trouve préimage
	2-préimage-res	2^n	trouve 2^e préimage
CRHF	rés. collisions	$2^{n/2}$	trouve collision
MAC	key non-recovery	2^n	trouve clé MAC
	sécurité calc.	$\min\{2^n, 2^{\sharp(k)}\}$	trouve nouv. MAC.

n : taille empreinte
$\sharp k$: taille clé (IV)

Quelques attaques de base

- tentatives répétées : une fonction de hachage de long. n a une robustesse idéale si elle satisfait les bornes sup. de OWHF et CRHF
- recherche exhaustive de clé MAC (KPA) demande $2^{\sharp k}$ opérations
- deviner MAC demande 2^n opérations
- précalcul d'empreintes `http://gdataonline.com/` (fini)
- parallélisation pour 2^e préimage

Attaque par force brute

En $O(|D|)$ sur $h : \{0,1\}^k \times D \to \{0,1\}^n$ tq $(D, <) = (D_1, \ldots D_d)$.

Algorithme

$x_1 \xleftarrow{\mathcal{U}} D$; $y := h_k(x_1)$
pour $i := 1 \ldots d$ faire
  si $(h_k(D_i) = y \wedge x_1 \neq D_i)$ alors
    retourne x_1, D_i
  fsi
retourne *echec*

Paradoxe des anniversaires

Donnée : $B = (b_1, \ldots, b_k) \in \{1, 2, \ldots, n\}^k$.
Problème : Proba p qu'il existe au moins 2 élts identiques de B ?

$$1 - p = q = 1\left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\ldots\left(1 - \frac{k-1}{n}\right) = \prod_{i=1}^{k-1}\left(1 - \frac{i}{n}\right)$$

1 proba de tirer b_1 ; $\left(1 - \frac{1}{n}\right)$ proba de tirer $b_2 \neq b_1$ car $P(b_2 = b_1) = \frac{1}{n}$;
$\left(1 - \frac{i}{n}\right)$ proba de tirer $b_i \neq b_1, \ldots, b_{i-1}$
Comme $1 - x \leq e^{-x}$: $q \leq \prod_{i=1}^{k-1} e^{-\frac{i}{n}} = e^{-\frac{1}{n}\sum_{i=1}^{k-1} i} = e^{-\frac{(k-1)k}{2n}}$.
Passage au logarithme : $2n \ln q = -(k-1)k \Leftrightarrow 2n \ln \frac{1}{q} \propto k^2$
Pour $p > \frac{1}{2}, k \geq \sqrt{2n \ln 2}$ et $k \propto O(\sqrt{n})$ car $\sqrt{2 \ln 2} = 1,17$
Exemple : $n = 365$, $k = 23$ personnes. ($n = 12$, $k = 5$).
Utilité : dimensionner la longueur n de l'empreinte d'une fonction de hachage pour éviter les collisions.

Attaque basée sur le paradoxe de $h : \{0,1\}^\star \to \{0,1\}^n$

Calculer et trier des couples $(x, h(x))$ pour détecter des collisions.
Combien de couples pour en trouver 2 identiques parmi 2^n à $p > 1/2$?
Hyp : les images par h suivent une distribution uniforme.
Pour k entrées on a $p > 1/2$ d'avoir une collision avec $k \propto O(\sqrt{2^n}) \approx 2^{\frac{n}{2}}$

n	50	100	150	200
k	25	50	75	100

En calculant $> 2^{n/2}$ empreintes, on a une collision avec proba $> 1/2$.
Pour que h soit CR, on choisit n pour que le calcul de $2^{n/2}$ images par h soit irréaliste. A ce jour, $n \geq 128$ voire même $n \geq 160$.

Fonctions de hachage efficaces

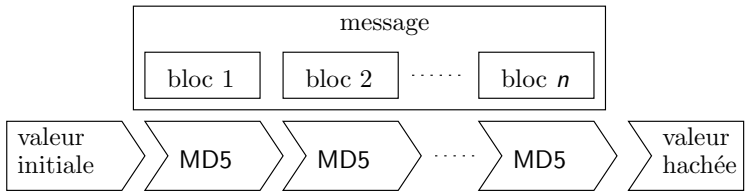
Les plus courantes :

nom	bits	tours×étapes	vitesse relative
MD5	128	4×16	1
SHA	160	4×20	0,28

A quoi servent-elles et comment sont-elles construites ?

Hachage par compression

Dans la plupart des algos, on découpe x en n blocs et on effectue :



Construire une fonction de hachage cryptographique

Partir d'un chiffre symétrique e_k, construire une fonction de compression :

$$g : \{0,1\}^m \to \{0,1\}^n \qquad \text{pour } m, n \in \mathbb{N}, \quad m > n$$

Etendre la fonction de compression en fonction de hachage :

$$h : \{0,1\}^\star \to \{0,1\}^n \qquad \text{pour } n \in \mathbb{N}$$

Construction d'une fonction de compression

A partir de $e_k : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$
on construit une fonction de compression

$$g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \qquad \text{pour } n \in \mathbb{N}$$

Le chiffre est utilisée soit directement s'il est résistant aux collisions soit en le $\ll$ perturbant $\gg$ un peu plus comme p.e.

$$g(k,x) = e_k(x) \oplus x$$
$$g(k,x) = e_k(x) \oplus x \oplus k$$
$$g(k,x) = e_k(x \oplus k) \oplus x$$
$$g(k,x) = e_k(x \oplus k) \oplus x \oplus k$$

Construction d'une fonction de hachage (Merkle)

Soit $r = m - n > 1$. On construit $h : \{0,1\}^\star \to \{0,1\}^n$ à partir de $g : \{0,1\}^m \to \{0,1\}^n$ (compression).
Soit $x \in \{0,1\}^\star$ et ℓ sa longueur en binaire.

- compléter x avec des "0" : $u = 0^i x$ t.q. $|u| \equiv 0 \mod r$
- compléter ℓ avec des "0" : $y = 0^j \ell$ t.q. $|y| \equiv 0 \mod r-1$
- découper y en blocs de $r-1$ bits et ajouter un "1" au début de chacun des blocs pour former le mot v
- construire $w = u0^r v$ composé de t blocs de longueur r.

Exemple

$r = 4$, $x = 11101$, $\ell = 101$. *On forme* $u = 0001\,1101$, $v = 1101$.

$$w = 0001\,1101\,0000\,1101 = w_1 w_2 w_3 w_4 \ (t = 4)$$

H déf. inductive : $H_0 = 0^n$; $H_i = g(H_{i-1} w_i)$, $1 \le i \le t$, $h(x) = H_t$

Plan

Fonctionnement MD5

Proposition

h est résistante aux collisions si g l'est aussi.

MD5 suit la construction MD et sa sécurité est équivalente à celle de sa fonction de compression MD5c qui produit 128 bits d'empreinte avec en entrée :

- CV : 128 bits de valeur de chaînage $= cv_0, cv_1, cv_2, cv_3$
- bloc M de 512 bits

Sur un message $M = M_0 M_1 \dots M_k$ découpé en blocs de 512 bits, MD5(m) est calculé inductivement par $H_{i+1} = \text{MD5c}(H_i, M_i)$ pour $1 \leq i \leq k$ avec MD5(M)=H_{k+1}

MD5c

64 valeurs d'étapes intermédiaires Q_i, $0 < i < 64$ (4 tours de 16 étapes) :

$$T_i := \Phi(Q_{i-1}, Q_{i-2}, Q_{i-3}) + Q_{i-4} + w_i + y_j$$
$$Q_i := Q_{i-1} + (T_i << s_i)$$
$$Q_4 := cv_0,\ Q_3 := cv_3,\ Q_2 := cv_2,\ Q_1 := cv_1$$

s_i et y_i ctes dépendant de l'étape, w_i le i^{e} bloc de l'expansion initiale du message (pour $0 \leq i < 64$, $w_i = m_j$ pour un certain $0 \leq j < 16$). L'opération $+$ est $\mod 2^{32}$ et $x << y$ décalage circulaire x de y positions à gauche. Initialisation CV cte.

$$\begin{array}{ll} \Phi_i(x,y,z) = F(x,y,z) = (x \wedge y) \vee (\neg x \wedge z) & 0 \leq i < 16 \\ \Phi_i(x,y,z) = G(x,y,z) = (x \wedge z) \vee (y \wedge \neg z) & 16 \leq i < 32 \\ \Phi_i(x,y,z) = H(x,y,z) = x \oplus y \oplus z & 32 \leq i < 48 \\ \Phi_i(x,y,z) = I(x,y,z) = y \oplus (x \oplus \neg z) & 48 \leq i < 64 \end{array}$$

A la fin, MD5c calcule :

$$cv'_0 := cv_0 + Q_{60},\ cv'_1 := cv_1 + Q_{63},\ cv'_2 := cv_2 + Q_{62},\ cv'_3 := cv_3 + Q_{61}$$

Attaque de Wang et al [6]

$$\delta_0 = (0,0,0,0,2^{31},0,0,0,0,0,0,2^{15},0,0,2^{31},0)$$
$$\delta_1 = (0,0,0,0,2^{31},0,0,0,0,0,0,-2^{15},0,0,2^{31},0)$$

$M := M_0 M_1$ de 1024 bits. Pour chaque M, soit $M'_0 = M_0 + \delta_0$ (idem pour M'_1) et $M' := M'_0 M'_1$ (additions $\mod 2^{32}$).
L'attaque de [6] décrit un algorithme efficace pour trouver de tels M tq MD5(M)=MD5(M') en pistant les différences dans les valeurs étapes intermédiaires par une attaque différentielle.
L'attaque invalide la résistance aux collisions de MD5.

Si Q_i (resp Q'_i) est la sortie de la i^{e} étape du calcul de M (resp M'), Wang produit 128 valeurs (64 par bloc) a_i tq si leur méthode trouve une collision, $Q'_i - Q_i = a_i$ pour tous les Q_i calculés pendant MD5c(M_0) et MD5c(M'_0) et $Q'_i - Q_i = a_{i+64}$ pour les Q_i calculés pendant MD5c(M_1) et MD5c(M'_1). Ce sont les différentiels des valeurs a_i. Elle donne 4 valeurs supplémentaires qui fixent les différentiels pour les CV ie les sorties intermédiaires des fonctions de compression ou les sorties de MD5c(M_0) et MD5c(M'_0).
L'article ne décrit pas la manière de trouver les a_i. Cependant, ils décrivent comment trouver efficacement de tels M en posant les conditions sur les Q_i. Si ces conditions sont satisfaites, alors les différentiels occurrent avec forte proba. Les détails ont été ensuite décrits/améliorés par plusieurs autres chercheurs.

Recherche de collisions

Algorithme

tant que *non collision* faire
chercher M qui satisfait la plupart des conditions sur Q_i
(germes tirés aléatoirement puis méthodes dét.)
calcul Q_i, Q_i' pour vérifier la correction des différentiels
collision := (différentiels-validés == vrai)
ftq
retourne M

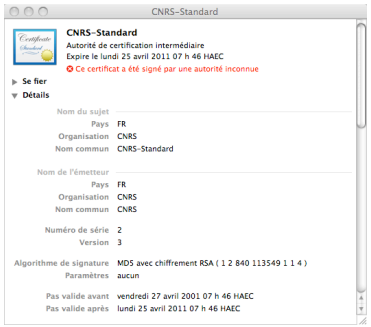
algo exécuté pour chaque bloc de M. Trouve d'abord un bloc M_0 qui satisfait les différentiels puis un bloc M_1.

Notes bibliographiques

Beaucoup d'améliorations de cette attaque !

- Klima [2] a construit un premier algorithme améliorant l'attaque de Wang et al. et l'a amélioré : `http://cryptography.hyperlink.cz/MD5_collisions.html`
- Joscak `http://cryptography.hyperlink.cz/2006/diplomka.pdf` : mémoire de master (algèbre) avec étude de la complexité des attaques
- Stevens `http://www.win.tue.nl/hashclash/` freewares et mémoire de master (p.e. pour des collisions X509)
- et il y en a encore d'autres (attaques $<$ minute)

Confiance ?



Plan

Utilisation

- mécanisme de base dans les protocoles cryptographiques
- outil essentiel pour des applications
  - cryptographiques
  - et autres...

Vérification de mots de passe

- mdp de A est p et le serveur mémorise $\overline{p} = h(p)$
- $A \to B$: p sur canal sûr ; B vérifie $h(p) = \overline{p}$

sûr tq Eve qui obtiendrait $\overline{p}$ ne peut retrouver p (OWHF)
attaques possibles par dictionnaire.

Comparaison par hachage

- A (resp. B) possède un grand fichier F_A (resp. F_B)
- ils veulent savoir si $F_A = F_B$
- $A \to B$: $h(F_A)$ et B teste $h(F_A) = h(F_B)$

Identification & authentification

identification affirmation de l'identité d'une entité au moyen de son identifiant. (Je suis Bruno)
authentification procédé de vérification de l'identité d'une entité. (Je suis Bruno et en voici la preuve)

L'identification permet de connaitre l'identité d'une entité et l'authentification de vérifier cette identité.

Mots de passe jetables de Lamport [3]

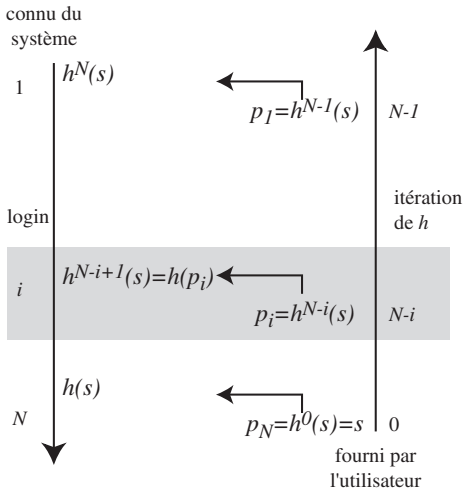
Application du hachage pour le service de contrôle d'accès.
Accès distant : identification par login/ mot de passe. Un pirate peut découvrir le mdp de l'utilisateur de 3 manières :

- en accédant aux informations contenues au sein du système, pe en accédant au fichier des mots de passe ;
- en interceptant l'envoi du mot de passe par l'utilisateur par l'espionnage de la ligne de connexion ou par un programme d'espionnage sur l'ordinateur de l'utilisateur ;
- par la divulgation du mot de passe de l'utilisateur si ce dernier a choisi un mot de passe facile à deviner.

Remèdes

- conserver les mots de passe chiffrés. C'est ce qui est réalisé à l'heure actuelle dans la plupart des systèmes.
- ne pas utiliser un unique mot de passe mais une suite de mots de passe. Méthode utilisée dans les systèmes à mot de passe jetables (OTP : *One time passwords*) comme `S/Key`, `Opie`,... au moyen d'une fonction de hachage cryptographique.
- De plus en plus appel à des systèmes biométriques (empreintes vocales, digitales ou rétiniennes).

Fonctionnement des OTP



Fonctionnement des OTP

On fixe N nombre maximal de connexions et s secret initial. Le système et l'utilisateur partagent soit une fonction à sens unique soit une fonction de hachage cryptographique h.
Le i^{e} mot de passe p_i est $h^{N-i}(s)$. La suite des N mots de passe de connexion fournis par l'utilisateur est :

$$h^{N-1}(s), h^{N-2}(s), \ldots, h(h(h(s))), h(h(s)), h(s), s$$

et celle du système pour identifier l'utilisateur par :

$$h^{N}(s), h^{N-1}(s), \ldots, h(h(h(s))), h(h(s)), h(s)$$

Fonctionnement des OTP

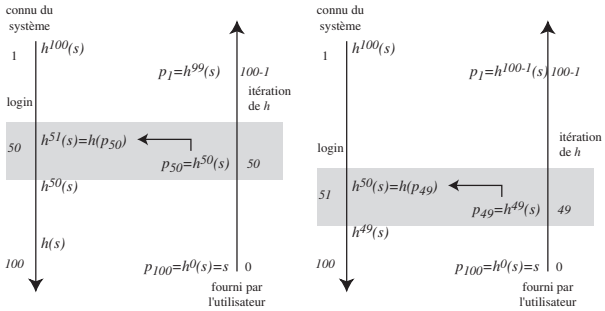
Connexion i/N ; connus du système : h^{N-i+1} et i n° de connexion.

Demande de connexion
- Système envoie un *défi* : numéro de connexion, i
- utilisateur utilise h en fonction de s, N et i : calcule $p_i = h^{N-i}(s)$ et le transmet au système

Vérification système reçoit $p_i = h^{N-i}(s)$, calcule $h(p_i) = h(h^{N-i}(s)) = h^{N-i+1}$ qu'il connaît. S'il y a égalité, il accepte la connexion et mémorise la valeur p_i reçue de l'utilisateur pour la prochaine connexion.

Chaque mdp est la valeur requise par le système pour la connexion suivante.

Deux connexions consecutives



Et maintenant ?

Avenir du hachage :
- MD5 : no future
- SHA-0 : collisions touvées
- SHA-1 ; recherche collisions en 2^{69} ; boincstats (over). Premières collisions construites en 2017 requièrent 1 siècle GPU mais plus rapide qu'une recherche exhaustive d'un facteur 100000

Et maintenant –suite– ?

- NIST : nouveau concours pour fonctions de hachage `http://csrc.nist.gov/groups/ST/hash/index.html` ; paramètres demandés (deadline était 20081031 !) :
  - conception : famille de fns, sorties de 224, 256, 384, 512 bits
  - compatible avec les standards cryptographiques
  - sécurité : CR, OW, autres critères de résistance
  - efficace : au moins aussi rapide que SHA-256
- SHA-3 : gagnant : Keccak ; connu depuis le 2 octobre 2012. Nouvelle construction (sponge)

J.P. Delahaye.
La compression des données.
Pour la science, 217 :177–189, 1995.

V. Klima.
Finding MD5 collisions - a toy for a notebook, 2005.

L. Lamport.
Password authentication with insecure communication.
Communications of the ACM, 24(11) :770–772, 1981.

M. Li and P. Vitányi.
An introduction to Kolmogorov complexity and its applications.
Springer Verlag, 1993.

D. Salomon.
Data compression, the complete reference.
Springer Verlag, 1998.

X. Wang and H. Yu.
How to break MD5 and other hash functions.
In *EUROCRYPT*, pages 19–35, 2005.

T.A. Welch.
A technique for high performance data compression.
Computer, 17(6) :8–19, 1984.

J. Ziv and A. Lempel.
A universal algorithm for sequential data compression.
IEEE Transactions on Information Theory, IT–23(3) :337–343, 1977.

Compression
Hachage
Paradoxe des anniversaires
Construction de Merkle-Damgard
Attaques contre MD5
Utilité des fonctions de hachage

J. Ziv and A. Lempel.
Compression of individual sequences via variable-rate coding.
IEEE Transactions on Information Theory, IT–24(5) :530–536, 1978.