Compte-rendu projet période 1

–

Algorithmie

Evan Excoffier – Clément Juventin

27/10/2021

1) Introduction

Question 1

La fonction md5 est codée sur 128 bits mais le hash lui n'est codé que sur 64. La fonction de hashage ne possède donc que 2^64 hashs différents.

Question 2

Dans le cas où il y a des mots de passes de taille M comprenant uniquement des lettres minuscules, il existe 26^M mots de passes différents.

Dans le pire des cas, nous devrons tester les 26^M mots de passe.

Dans le meilleur des cas, nous devrons tester 1 seul mot de passe.

Donc en moyenne, nous allons tester (26^M + 1)/2 mots de passe.

Question 3

Nous pensons qu'il est judicieux d'utiliser un tableau composé de pointeurs sur les mots de passe. L'indice de chaque élément du tableau serait le hash du mot de passe auquel il se rapporte, stocké à ce même indice.

La complexité temporelle serait alors o(26^M) et la structure serait de taille 26^M car il y a 26^M mots de passe.

Question 4

Pour M = 6,

26^M = 26^6 = 308915776

La complexité temporelle, la complexité de l'attaque et la taille de la base de données sont donc de 308915776.

Pour M = 8,

26^M = 26^8= 208827064576

La complexité temporelle, la complexité de l'attaque et la taille de la base de données sont donc de 208827064576.

Utiliser cette technique ne semble donc pas réaliste. Pour des mots de passes composés uniquement de lettres minuscules et de petite taille, la quantité de stockage nécessaire et le temps d'exécution semblent beaucoup trop important. Il est de l'ordre de plusieurs giga-octets dans notre simple cas.

2) Méthode des hashs chainés : compromis temps – mémoire

Question 5

Non, le candidat n'est pas forcément un antécédent. Dans le cas où il y a plus de mots de passes que de hash, c'est à dire 26^M > 2^64. Un même hash est issu de 2 mots de passes différents. Supposons que 2 mots de passe finaux ($pass_{i,L}$ et $pass_{j,L}$ avec i =/= j) sont identiques ($pass_{i,L} = pass_{j,L}$ avec i =/= j). Alors, il se peut qu'à un certain rang n < L, $pass_{i,n}$ =/= $pass_{j,n}$. Donc les hashs précédents sont différents. Donc, pour un même rang k <= n, les mots de passes sont différents. Or si un hash au rang k est le hash attaqué, nous ne saurons pas quel mot de passe final choisir. Un candidat ne sera donc pas un antécédent de notre hash attaqué.

Voyons un exemple, plaçons-nous dans le cas où L=4, N=1, voici la situation :

$pass_{1,0} \rightarrow hash_{1,0} \rightarrow pass_{2,0} \rightarrow hash_{2,0} \rightarrow pass_{3,0} \rightarrow hash_{3,0} \rightarrow pass_{4,0}$

$pass_{1,1} \rightarrow hash_{1,1} \rightarrow pass_{2,1} \rightarrow hash_{2,1} \rightarrow pass_{3,1} \rightarrow hash_{3,1} \rightarrow pass_{4,1}$

Maintenant, nous savons que $pass_{4,0} = pass_{4,1}$ donc $hash_{3,1} = hash_{3,0}$ mais la situation est telle que $pass_{3,1}$ =/= $pass_{3,0}$. Donc $hash_{2,1}$ =/= $hash_{2,0}$ et $pass_{2,1}$ =/= $pass_{2,0}$, si notre hash attaqué correspond à $hash_{2,0}$ alors nous aurons 2 candidats qui sont $pass_{2,0}$ et $pass_{2,1}$ or $pass_{2,1}$ n'est pas un antécédent de $hash_{2,0}$.

Question 6

Dans le cas très favorable où il n'y a pas de redondance, il suffit de tester un mot de passe candidat. En effet, chaque mot de passe aura un hash qui, réduit, donnera un mot de passe unique. Il y aura donc un match unique car les mots de passe sont uniques.

Question 7

Pour stocker les couples ($pass_{x,0}$, $pass_{x,L}$), nous pouvons utiliser une hashtable, cela rendra la recherche pertinente, efficace et rapide. Notre hashtable sera un tableau de taille N et comprenant des pointeurs vers des listes de couples ($pass_{x,0}$, $pass_{x,L}$).

La complexité temporelle de l'attaque d'un hash sera alors o(n) dans le pire des cas.

- Explication :
  - ➔ Hachage – o(1)
  - ➔ Accès à une case d'un tableau – o(1)
  - ➔ Parcours de la liste chainée – o(n)
  - ➔ On suppose que nous disposons déjà de données rangées dans une hashtable.

Question 8

Si $pass_{x,i} = pass_{y,j}$ avec x=/=y cela aura pour conséquence de créer plusieurs antécédents. Cela nous obligera à tester tous les pass afin de trouver les différents antécédents, ainsi notre complexité sera majorée au nombre maximum de cas à chaque fois.

Question 10

26^6 = 308 915 776

26^8 = 208 827 064 576

$26^6 < 26^8 < 2^{64}$ donc deux hash peuvent être réduits et donner le même pass. C'est donc la réduction qui pose un problème. En revanche si 26^M était supérieur à 2^{64}, ce serait l'étape de hachage qui risquerait de donner deux fois le même hash pour deux pass distincts.

Question 11

Non, si $pass_{x,i} = pass_{y,j}$ avec x=/=y vu qu'ils ne sont pas au même rang (j=/=i), cela signifie que les fonctions de réduction qui ont mené à ces pass sont différentes et donc qu'ils proviennent de hash différents. On évite la collision et le processus reste déterministe.

Question 12

Si $pass_{x,i} = pass_{y,i}$ avec x=/=y alors $pass_{x,L} = pass_{y,L}$ car on appliquera les mêmes fonctions de réduction à travers le processus.

Question 13

La probabilité d'obtenir une collision en fin de chaine (rang L) est la somme des probabilités d'obtenir une collision au rang i<=L.

Avec pour hypothèse que $2^{64}>26^M \Leftrightarrow M < (64\log(2) / \log(26)) \sim 13.6 \Leftrightarrow M <= 13$

$$P(X < L) = \sum_{i=1}^{L} P(X = i) \quad = \sum_{i=1}^{L} p * (1-p)^{i-1}$$

Avec X qui suit une lois géométrique de paramètre $p = (2^{64} - 26^M) / (2^{64} * 26^M)$

Question 14

On sait que les probabilités de la suite géométrique diminue lorsque l'indice augmente. Donc on peut donner un ordre de grandeur tel que….

$$P(X < L) \cong \sum_{i=1}^{1} P(X = i) \; = P(X = 1) = \frac{1}{26^M} - \frac{1}{2^{64}} \cong \frac{1}{26^M}$$

Ainsi pour M=6 on a P(X < L) ~= 3,24e-9

Donc on peut facilement créer jusqu'à 100 000 lignes avec une chance d'environ 1/1000 de créer une collision.

Question 16

Nous n'avons pu attaquer que 260 hash (faute de temps à ~1 min/hash) et nous avons trouvé 229 mots de passes correspondants. Soit un taux de réussite d'environ 88%.

Malgré le temps qui a été une contrainte pour ce projet, nous pensons que notre taux de réussite est très encourageant car le succès d'une attaque de hash repose plus sur la découverte du mot de passe que sur le temps passé à le chercher (dans la limite du raisonnable).

Nous n'avons pas pu tester différentes valeurs de R, N, L.

3) Recherche documentaire

Question 17

De nos jours, on cherche sans cesses à augmenter la complexité des mots de passes, plus de caractères et des hash plus grands.

On utilise aussi des techniques telles que la double authentification, qui permet de se prémunir des attaques au mot de passe en nécessitant une action supplémentaire de la part de l'utilisateur.

En plus de ces techniques, il est courant d'essayer de limiter le nombre de tentatives de vérification de mot de passes, et les fonctions de hachages peuvent être gardées secrètes par les autorités les manipulant.

Enfin, le détenteur d'un mot de passe peut bien souvent le changer et donc protéger son accès en changeant son hash.

Question 18

Les principaux usages des fonctions de hachages sont :

- Le stockage de mots de passes / données sensible
- La communication chiffrée
- Les blockchains

La fonction md5 semble obsolète car facilement attaquable, de nos jours il existe des fonctions bien plus complexes et efficace (sha-256).

De plus, le nombre de hash différents devient vite contraignant au regard des mots de passes actuels (128 caractères ASCII, longueur pouvant aller jusqu'à 80 caractères).