R3.09 -- Cryptographie
TP 3

Durée : 1 séance de TP

Langage : indifférent entre Java, Python, Rust

Date limite dépôt de livrables : 11/10 à 23h 59

Livrables : 2 livrables :

1. Le code source dans une archive .zip ou .7z. Convention de nommage : <nom1>_<nom2>TP_1_et_2
2. Un rapport de max. 5 pages (avec des bouts de (pseudo)code), qui décrit les algorithmes utilisés pour retrouver des collisions ou des préimages et spécifie le speed-up de l'algorithme qui retrouve des collisions dans le cadre de la fonction H*. Vous pouvez utiliser jusqu'à une page en plus d'images qui montrent les résultats de votre travail.

Exercice 1 : implémenter une mauvaise fonction de hachage

Chaque caractère (lettre majuscule ou minuscule, caractère spécial) a un encodage en UTF-8. Voici une liste de ces encodages : https://www.utf8-chartable.de/.

On suppose qu'il y a une fonction de hachage H qui prend en entrée un caractère `car` qui est soit une lettre majuscule, soit une lettre minuscule, soit un caractère spécial, parmi : `!, #, (, ), *, +, /, ?`. Ces valeurs ont des encodages correspondant à des valeurs entre 20 et 80, donc elles sont représentables sur un octet. La fonction de hachage H, ayant en entrée un caractère `car`, retourne un octet (8 bits) représentant l'encodage du caractère `car`. Par exemple l'encodage UTF-8 de la lettre A est 41, ce qui peut être représenté en binaire par 01000001.

Implémentez cette fonction de hachage.

Exercice 2 : des préimages et des collisions

Dans cet exercice l'idée est d'estimer, de façon empirique, la facilité de trouver une préimage et une collision. Voici les deux expériences que nous allons regarder aujourd'hui :

- **Trouver des collisions** : produire deux caractères différents qui donnent la même valeur hachée.
- **Trouver des préimages** : choisir aléatoirement un caractère parmi les caractères possibles en entrée. Obtenez son haché, qu'on dénote h. Trouvez un caractère qui, lorsqu'il est haché, donne la valeur h.

Estimez, dans un premier temps, de façon théorique la probabilité de trouver des collisions, ainsi que la probabilité une préimage.

1. Implémentez un algorithme qui essaie de trouver une collision. Confirmez ou infirmez votre hypothèse par rapport à la probabilité théorique calculée avant.
2. Implémentez un algorithme qui prend en entrée une valeur h (correspondant à l'image d'une entrée aléatoirement choisie) et qui recherche une préimage de cette valeur.
3. Répétez votre recherche de préimage un nombre significatif de fois (100, ou plus), en comptabilisant à chaque fois le nombre d'essais pour trouver la préimage. Comparer cette valeur à la probabilité calculée ci-dessus.

Exercice 3 : une deuxième fonction de hachage, aussi mauvaise

Implémentez une fonction de hachage H* qui prend en entrée un texte de taille arbitraire, comportant des majuscules, des minuscules et les caractères spéciaux listées dans l'exercice 1, et qui ressort un octet. Pour une valeur en entrée phrase, on calcule H* en tant que le où exclusif, bit par bit, des hachées avec la fonction H (donnée à l'exercice 1) des caractères qui la comportent.

Par exemple $\mathrm{H}^*(\mathrm{texte}) = \mathrm{H}(\mathrm{t}) \oplus \mathrm{H}(\mathrm{e}) \oplus \mathrm{H}(\mathrm{x}) \oplus \mathrm{H}(\mathrm{t}) \oplus \mathrm{H}(\mathrm{e})$

Exercice 4 : des attaques ciblées

Utilisez le fichier texte présent sur Moodle et votre algorithme de l'exercice 2 pour trouver une collision. (Remarque : les mots du dictionnaire pourront être utilisées avec les lettres mises en majuscules, en minuscules, ou des minuscules précédées par une majuscule) Combien de valeurs avez-vous essayées ?

Pouvez-vous trouver une attaque qui exploite la fonction de hachage pour trouver des collisions plus facilement parmi les mots présents dans le dictionnaire ? (plusieurs speed-ups possibles)