TP 1 et 2

Fonctions de hachage

Contexte

Le but de ce TP est de trouver des collisions sur des fonctions de hachage connues, mais dont la sortie est tronquée (réduite à quelques bits). Nous allons utiliser plusieurs stratégies et tronquer la sortie à un nombre variable de bits, puis nous allons comparer et analyser nos résultats.

Le travail sera réalisé en Python.

Ensuite, nous allons utiliser des fonctions de hachage dans un contexte d'échange de clé, avec l'algorithme HMAC. Celui-ci prend en entrée une clé et une entrée, et peut (à partir de ces entrées) retourner une sortie d'octets qui peuvent servir de clé partagée entre deux entités. Pour une petite clé, $HMAC(K;m) = MAC(H(pad_o \oplus K) \,||\, H((pad_i \oplus K)||m)$. Nous allons premièrement utiliser la fonction de hachage SHA1 pour cet exercice, en observant la différence lorsqu'on utilise deux clés très similaires pour un même message en entrée, ou deux messages similaires pour une même clé. Finalement, nous allons utiliser une fonction de hachage différente.

En tant que note finale, veuillez noter que, même si nos exercices vont utiliser la fonction de hachage SHA1, celle-ci n'est plus assez sécurisée pour être utilisée en pratique. Aujourd'hui, c'est mieux d'utiliser SHA-256, SHA-512, ou Keccak.

Exercice 1

En Python une bibliothèque de base implémentant des fonctions de hachage est hashlib.

https://docs.python.org/3/library/hashlib.html

Dans ce premier exercice nous allons calculer un message digest (un hash) d'un message, puis tronquer cette sortie à 5 bits. Ensuite, nous allons prendre en entrée un fichier et calculé le digest tronqué du premier mot du dictionnaire.

1. En regardant la documentation de la bibliothèque hashlib, trouvez comment on pourrait calculer le digest SHA1 d'un message.
2. Ecrivez du code Python qui calcule le digest SHA1 du mot « cryptis ».
3. Ajoutez ensuite du code qui tronque la sortie du digest aux 5 premiers bits du hash et affichez-les. Indice : pour trouver les 5 premier bits d'une valeur (par exemple une valeur a) vous pouvez utiliser la syntaxe % (1 << 5) : par exemple a % (1<<5) .
4. Pour l'instant le mot que vous hachez est le mot cryptis, entré à la main dans le code. Maintenant vous allez trouvez la liste de mots dans le fichier http://www-

personal.umich.edu/~jlawler/wordlist, que vous allez stocker dans une collection (par exemple une liste). Ensuite, calculez le haché du premier mot de cette liste. (N'oubliez pas l'encoding de vos mots, qui devrait être UTF-8)

Exercice 2

Dans ce deuxième exercice nous allons utiliser deux méthodes différentes pour trouver une collision sur le digest à 5 bits, en utilisant les mots stockés dans la liste de l'exercice précédent.

1. Pour la première méthode écrivez du code qui fait les opérations suivantes :
    a. Calcule le haché tronqué à 5 bits du mot « cryptis »
    b. Itère sur les mots de la liste de mots de l'exercice 1, mot par mot, calcule leur hash tronqué, et, si le hash tronqué est le même que celui de « cryptis », indique qu'une collision a été trouvée.
    c. Fait afficher les mots dont les digests coïncident et le nombre d'essais nécessaires pour trouver la collision.

2. Et si vous changiez le mot de la question 1a de cryptis à poisson ?

3. Ecrivez une deuxième méthode, qui trouve des collisions de la façon suivante :
    a. On itère sur le code suivant :
        i. On choisit deux mots distincts de la liste de mots
        ii. On calcule leur hash tronqué
        iii. Si ça coïncide, on a trouvé une collision
    b. On affiche les mots dont les digests coïncident et le nombre d'essais nécessaires pour trouver la collision

Exercice 3

Dans cet exercice nous allons modifier la taille du digest pris en compte et analyser comment la taille du digest influence notre probabilité de trouver une collision.

1. Pour les deux méthodes de l'exercice 2, modifiez votre code pour que vous puissiez entrer, en tant que paramètre à la méthode, une taille désirée du digest.
2. Faites exécuter vos méthodes pour trouver des collisions pour des tailles de digest de 6,7,8,… 15. A chaque fois, affichez le nombre d'essais utilisés.
3. Que remarquez-vous concernant le nombre d'essais ? Quelle est la probabilité théorique de trouver telles collisions si la fonction de hachage était un vrai oracle aléatoire ?

Exercice 4

Dans cet exercice nous allons utiliser la fonction de dérivation de clés HMAC. En Python, une bibliothèque qui nous y permet accès est hmac :

https://docs.python.org/3/library/hmac.html

Regardez premièrement la documentation de cette bibliothèque. Vous aurez besoin des méthodes new et hexdigest principalement.

1. Vous allez premièrement utiliser la clé « cryptis » et le message « Ceci est mon premier HMAC SHA1 ». Calculez la sortie HMAC sur ces deux entrées, en utilisant la fonction de hachage SHA1. Faites afficher votre sortie en hexadécimal.
2. Maintenant, vous allez changer la clé, de « cryptis » à « kryptis ». Refaites les mêmes calcules. Combien de caractères hexa les deux sorties ont-elles en commun ?
3. Utilisez maintenant « cryptis en tant que clé, mais le message « Ceci est ton premier HMAC SHA1 ». Refaites le calcul de la question 1. Combien de caractères hexa les deux sorties ont-elles en commun ?
4. Maintenant remplacez la fonction de hachage par SHA512 et refaites la question 1. Quelles différences observez-vous ?