R3.09 -- Cryptographie
TP 3+4

Durée : 2 séances de TP travaillées seuls ou en binôme

Langage : indifférent entre Java, Python, Rust

Date limite dépôt de livrables : 19/10 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. 6 pages (avec des bouts du code, des tableaux, des graphes, etc.) plus une page de réflexion et de conclusions, qui :
    a. Détaille votre recherche sur les bibliothèques (libraries) utilisables et détaille et justifie votre choix.
    b. Explique comment vous générez et comment vous tronquez les mots de passe dans le cas simple (pas de contraintes autre que la taille), et dans le cas d'une taille variable.
    c. Explique le fonctionnement algorithmique de votre attaque contre les mots de passe, en discutant les vulnérabilités exploitées
    d. (Extensions) Si jamais vous avez travaillé sur les extensions, expliquez comment vous gérez le stockage des mots de passe.
    e. (Page réflexion) Discute vos conclusions sur l'utilisation d'un générateur de mots de passe, ainsi que les bonnes pratiques à avoir si un tel générateur est utilisé.

Contexte

Votre travail aujourd'hui concerne les mots de passe, et plus particulièrement le défi de générer des mots de passe (pseudo-)aléatoires pour chaque service utilisé. Il faut souligner que le travail que vous allez faire aujourd'hui ne peut que gratter la surface de cette problématique – toutefois, l'idée est de pouvoir

comprendre les verrous ainsi que d'explorer certaines pistes qui peuvent permettre à un utilisateur d'avoir une hygiène responsable de ses mots de passe, sans pour autant obliger l'utilisateur à se souvenir de nombreux mots de passe difficiles à retenir.

Le but sera de mettre en place un algorithme de génération de mots de passe fonctionnant localement sur chaque machine, qui prend en entrée un mot de passe maître (que l'utilisateur doit mémoriser) ainsi qu'un tag spécifique à chaque contexte où un mot de passe est utilisé : par exemple Unilim pour un mot de passe de l'Université de Limoges, Facebook pour un mot de passe Facebook, etc. Le mot de passe pour chaque contexte sera calculé comme étant égal à H(mod de passe maître | tag) où | indique la concaténation.

La sécurité du générateur ne doit pas dépendre du secret de ce deuxième input (tag), seulement du secret du premier input (le mot de passe maître).

Préparation : les bibliothèques

Pour ce TP vous aurez besoin de plusieurs primitives cryptographiques, dont a minima les fonctions de hachage SHA1 et SHA256. Trouvez, pour les 3 langages ci-dessus, des bibliothèques (libraries) qui peuvent vous fournir ces fonctionnalités. Attention : pour les extensions éventuelles prévues au travail du TP, vous aurez peut-être besoin d'autres primitives intéressantes, telles que le chiffrement, les MAC…

Une fois votre état de l'art réalisé, veuillez lister les candidats que vous avez choisis et faites votre choix de langage et de bibliothèque. Ces choix doivent être détaillés et justifiés dans votre rapport.

Exercice 1 : Des mots de passe tout simples

Dans un premier temps, vous allez devoir produire une application qui doit, en prenant en entrée deux chaines de caractères de taille arbitraire, produire une chaine de 8 caractères, qui peuvent être : des lettres majuscules ou minuscules, des chiffres, ainsi que des caractères spéciaux arbitraires (mais pas d'espaces, des caractères invisibles, etc.).

Vous n'avez pas besoin d'une interface particulière graphique, vous pouvez gérer les entrées et les sorties dans la console.

L'algorithme de génération de mots de passe devra utiliser une fonction de hachage et suivre les étapes suivantes :

- Concaténer les deux chaines de caractères en entrée
- Faire exécuter la fonction de hachage avec en entrée la suite de caractères obtenue ci-dessus
- Traduire (si besoin) la sortie dans une suite de 8 caractères comme détaillé dans la consigne. En fonction de la fonction de hachage utilisée, vous devrez potentiellement tronquer l'output de la

fonction de hachage, ou justement changer de fonction de hachage car l'output est trop petit. Détaillez ces étapes dans votre rapport

Particularités : Évaluez d'une façon convaincante votre programme, à la fois en termes du respect de la consigne (est-ce que l'output est vraiment la bonne sortie de la fonction de hachage et à la bonne taille ?) et en termes de la robustesse des mots de passe par rapport aux entrées (distributions des outputs). Décrivez vos tests et résultats d'une façon à la fois lisible et succincte dans le rapport.

Exercice 2 : Des mots de passe d'une taille demandée

Faites une extension à votre programme qui permet de prendre en entrée un entier N, ainsi que les deux chaines de caractères utilisés dans l'exercice précédent. L'entier N peut varier entre 1 et 12 et il représente la taille attendue par l'output. Si besoin, changez de fonction de hachage pour avoir une sortie plus longue.

La validation de l'état actuel de votre programme se fera par des jeux d'essais. Veuillez indiquer les jeux d'essais les plus importants, ainsi qu'une liste de cas de tests, dans votre rapport.

Exercice 3 : Mot de passe maître

Retravaillez votre code pour qu'il prenne (et stocke) dans un fichier un mot de passe maître, qui correspond à la première des deux chaines de caractères prises en entrée. Ce fichier s'appellera mpwd (l'extension sera de votre choix). Votre générateur de mot de passe devra désormais prendre en entrée seulement la taille du mot de passe attendu et une chaine de caractère.

Le mot de passe sera calculé à partir du mot de passe maître et de la chaine de caractères pour un output de taille N.

Particularités : Attention à la gestion du mot de passe maître ! Les règles suivantes doivent être observées par votre programme :

- Un mot de passe maître ne peut pas contenir des espaces (mais peut contenir les autres caractères permis dans l'exercice 1)
- S'il n'y a aucun mot de passe dans le fichier ou si le fichier n'existe pas : demandez un nouveau mot de passe maître
- L'utilisateur doit pouvoir demander un changement de mot de passe dans la console, au début de chaque exécution (on ne veut pas lui demander de changer le fichier directement !)

Extension possible : si jamais vous avez fini le TP et vous souhaitez avancer d'avantage, vous pouvez également penser à mettre en place un fichier de stockage pour les tags utilisés. Ce type de fichier pourrait par exemple stocker la correspondance entre l'URL ou le service pour lequel on utilise un mot de passe et le tag utilisé.

Exercice 4 : Attaques sur des mots de passe

Dans cet exercice, vous allez tenter de trouver des préimages possibles pour le mot de passe maître, c'est-à-dire, vous allez tenter de casser la sécurité de votre générateur de mots de passe.

Votre tâche sera de trouver un mot de passe valide en connaissant le tag utilisé mais pas le mot de passe maître.

Assurez-vous premièrement que votre fichier de mots de passe maître contient un mot de passe maître de 10 caractères (à vous de les choisir). Dans votre rapport discutez quelle est la probabilité qu'un attaquant retrouve ce mot de passe maître par force brute.

Générez, en utilisant votre générateur de mots de passe et le mot de passe maître, des mots de passe d'un caractère (N=1) pour les tags suivants : Unilim, Amazon, Netflix.

Dans un premier temps votre attaque devra trouver un mot de passe maître qui donnera le même mot de passe que celui produit par votre vrai mot de passe sur le tag Unilim.

La stratégie sera celle d'une attaque par dictionnaire. Votre dictionnaire doit contenir tous les mots de passe possibles, avec toutes les possibilités pour chaque caractère. Itérez sur les mots de passe possibles sur 10 caractères, chaque caractère provenant de votre dictionnaire, jusqu'à trouver un mot de passe identique (pour le tag Unilim) à celui produit par le gestionnaire de mots de passe. Combien de possibilités avez-vous essayées ? Quelle est la probabilité théorique de trouver un mot de passe maître qui provoque une collision et comment votre nombre d'essais se compare à cette valeur ?

Dans un deuxième temps, votre attaque devra retrouver un mot de passe maître qui provoque une collision sur les 3 tags : Unilim, Amazon et Netflix. Comparez le nombre d'essais nécessaire pour votre attaque dans ce cas par rapport au cas précédent.

Finalement, essayez la même attaque pour N = 2 et N = 3, en enregistrant le nombre d'essais.

Analyse : Dans votre rapport utilisez les données sur le nombre d'essais pour N = 1, 2 et 3 pour pouvoir formuler une hypothèse sur le nombre d'essais demandé pour retrouver un mot de passe maître qui donne des collisions pour des mots de passe de taille N = 8, 10 et 12. Comparez ces chiffres à la probabilité théorique de réussir, ainsi qu'à la probabilité de trouver le mot de passe maître (quia avait 10 caractères). Quelles sont vos conclusions par rapport à un design sécurisé et une utilisation correcte de votre générateur de mots de passe ?

(Pour les plus rapides) Exercice 5 : Sécuriser le mot de passe maître

Sauvegarder le mot de passe maître dans un fichier peut être une bonne idée. Malheureusement, cela met en danger sa sécurité.

Une façon de sécuriser le fichier serait de le chiffrer. Pouvez-vous trouver une façon de chiffrer votre fichier, sans pour autant le rendre inutile ? (vous voulez toujours pouvoir déchiffrer le fichier pour l'utiliser dans la génération des mots de passe, changer le mot de passe maître, etc…)

Détaillez, dans votre rapport, la méthode de chiffrement choisie ainsi que votre mise en œuvre dans le langage que vous avez choisi. N'oubliez pas de mentionner des éventuelles bibliothèques dont vous aurez besoin.