TD6 – « Cassage » de mots de passe

4ASTI – Programmation Java – B.Nguyen

Introduction : Retrouver un mot de passe en connaissant tous les paramètres

Données : *Vous disposez de la liste des 20.000 mots de passes les plus fréquents (triés du plus fréquent au moins fréquent*) *dans le fichier passwords.txt (dite top 20K).*

Le chiffrement AES en mode ECB nécessite 2 ingrédients : une clé (String) et un sel (String).

Le chiffrement AES en mode CBC nécessite 3 ingrédients : une clé (String), un sel (String) et un vecteur d'initialisation (byte[16]).

Suite à une intrusion dans une base de données d'utilisateurs, vous avez récupéré plusieurs mots de passe chiffrés. Vous savez que ces mots de passe ont été chiffrés avec l'algorithme suivant : AES/ECB/PKCS5Padding

Vous retrouverez le code pour les opération de chiffrement et déchiffrement AES dans le package cmChiffrement.

Exercice : Cassage distribué

Vous avez récupéré un nouveau mot de passe chiffré de l'utilisateur *bnguyen*, toujours en mode AES/ECB/PKCS5Padding mais vous ne connaissez ni la clé, ni le sel utilisés, qui vous permettront de vous introduire dans un système plus important. Vous savez que cet utilisateur a sûrement utilisé un mot de passe très sécurisé. Le texte que vous avez récupéré est *cible*= XLtXaPNv67TJqQ0d8xCiMR7oyVumZsn4nnVv3P/W4ZA=

Malheureusement pour *bnguyen*, vous avez aussi récupéré le mot de passe de l'utilisateur *stagiaire* qui est le suivant : $mdp_{\text{stagiaire}}$= zyLj+eqxjzTh9eEz/Ga4Lw==

Grâce à de l'ingénierie sociale, nous avons réussi à obtenir le mot de passe en clair du stagiaire : il s'agit de « sangoku » (703^{e} mdp le plus fréquent).

Afin de déchiffrer le mot de passe *cible* vous allez essayer de trouver les paramètres de chiffrement du mot de passe $mdp_{\text{stagiaire}}$ puis appliquer ces mêmes paramètres au mot de passe *cible*.

En voyant que le site où vous avez réupéré les mots de passes chiffrés n'utilise qu'un simple chiffrement ECB, vous faites l'hypothèse que la clé et le sel sont des mots de passe fréquents dans la liste top 20K (pour accélérer les choses, je les ai pris dans le **top 1K**)

Nous allons donc mettre en place un programme *multi-threadé* pour accélérer les opérations de test. Il faut développer :

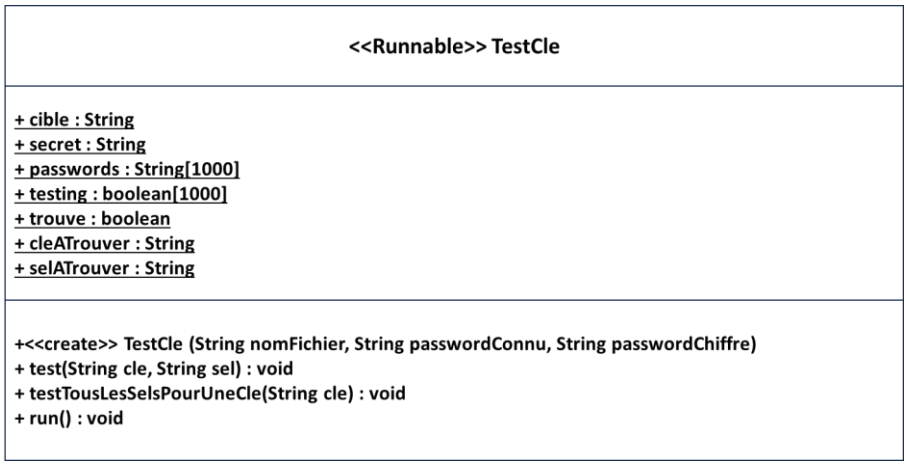
1) Une méthode « de base » de signature `String chiffre(String cle, String sel)` qui essaie de chiffrer le mot de passe « sangoku » en utilisant une clé et un sel issus de la liste des fichiers `passwords.txt`, puis qui compare ce chiffré à $mdp_{\text{stagiaire}}$.
2) Un algorithme non distribué qui teste exhaustivement l'ensemble des valeurs possibles cle/sel et qui affiche ces valeurs une fois trouvées.
3) En utilisant des `Thread` (le nombre dépend du nombre de cœurs disponibles dans votre machine ou VM), et réduisez le temps de cassage. Evaluez expérimentalement le gain en temps et comparez le avec le nombre de cœurs utilisés. (Voir plus bas pour des indications).

Développement de la classe qui va implémenter Runnable (indications) :

Il faut se poser un certain nombre de questions lorsqu'on développe un algorithme distribué.

1) Quelle est exactement la tâche « atomique » qu'on souhaite distribuer ? Ici, l'opération atomique est le chiffrement d'un texte, en passant comme paramètre une clé et un sel. Nous avons une liste de 1000 valeurs possibles pour cette clé et ce sel. Une proposition c'est qu'une tâche atomique sera de rechercher pour une clé donnée l'ensemble des sels possibles. C'est ça que devra faire la méthode `void run()`
2) Comment savoir, lorsqu'on démarre un nouveau `Thread` ce que font déjà les autres `Threads` ? Ce peut être votre programme principal qui décide de paramètres, mais avec `Runnable` on ne peut pas directement utiliser un paramètre car la méthode run() n'en prend pas[1] ! Nous proposons donc d'utiliser une variable statique (partagée donc à travers tous les Threads) qui permet de savoir quelles sont les clés déjà testées ou en cours de test. Cette variable (**`testing`**) sera un tableau de booleens de taille 1000, initialisé à `false` pour tous les indices. `testing[b] == true` signifie qu'on a testé ou qu'on teste le n^{ieme} mot de passe.
3) Bien vérifier que vous permettez l'arrêt de tous les `Thread` une fois qu'un `Thread` a déjà trouvé le résultat. Ceci peut être fait en testant une variable partagée (statique) dans chaque Thread avant de lancer un nouveau test.
4) Combien faut-il lancer de Threads ? Pour simplifier on va lancer 1000 Thread, mais il faut savoir qu'il ne peut pas y avoir plus de deux fois le nombre de cœurs de votre processeur lancés en même temps. Observez également l'utilisation mémoire de votre machine (e.g. en utilisant la commande `htop`)

Voici le format de la classe proposé :



Le constructeur prend en paramètre le fichier contenant les mots de passe (on a codé en dur le nombre de mdp), et remplir la variable statique `passwords` avec les 1000 premières valeurs. Le tableau `testing` doit être initialisé à `false`. On stocke également le mot de passe connu **sangoku** et le mot de passe chiffre **zyLj+eqxjzTh9eEz/Ga4Lw==** et la variable `trouve` est positionnée à false au départ et prend la valeur true lorsqu'on a trouve le sel et la clé, qui doivent être stockées dans les champs `cleATrouver` et `selATrouver`.

La méthode `test(String cle, String sel)` teste si la cle et le sel sont ceux qu'on cherche.

[1] La solution est d'utiliser des lambdas expressions, mais nous n'en parlons pas dans ce cours.

La méthode `testTousLesSelsPourUneCle (String cle)` va appeler la méthode `test(String, String)` et doit tester l'ensemble des sels pour une clé donnée.

Conclusion :

Déduisez-en le mot de passe super sécurisé de *bnguyen*, et réfléchissez au problème de réutilisation de mots de passe, même très sécurisés, sur de multiples sites.

Pour aller plus loin :

Essayez de déchiffrer les mots de passe suivants, chiffrés avec AES/CBC/PKCS5Padding sachant que vous connaissez la clé (secret) et le sel (123456), mais pas le vecteur d'initialisation (les 2 mots de passe ont des vecteurs d'initialisation différents). Une approche distribuée sera sans doute nécessaire pour Mdp_2.

Mdp_1= oXEMdm/fMWEuYpOZkMKQqg==

Mdp_2= zcxCgCulDAALIGB1FuXCqA==