Université d'Ottawa
Faculté de génie

École d'ingénierie et de
technologie de l'information

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

Devoir 4 (9%) CSI2110/2510

A remettre sur le Campus Virtuel avant 23h55 le 3 décembre 2018
30% de pénalité pour un devoir soumis avec 1min-24hs de retard.
Devoirs non acceptés au-delà de cette période
C'est un devoir individuel : attention au plagiat

Cracker les mots de passe!

Un bon gestionnaire informatique doit connaitre les techniques utilisées par les pirates informatiques afin de percer les systèmes d'une entreprise. Il doit même pouvoir mettre en pratique ces techniques afin d'identifier toute faille de sécurité et de tester la vulnérabilité d'un système. C'est ce que nous ferons dans ce devoir.

Dans votre devoir de programmation précédent, vous avez utilisé le hachage cryptographique afin de coder une transaction dans une chaine de blocs. Une autre application de cette technique est l'encryption des mots de passe donnant accès à un système logiciel.

En effet, un système logiciel n'enregistre jamais les mots de passe dans sa base de données. A la place, il sauvegarde la version encryptée de ce mot de passe, en utilisant, le plus souvent, un hachage cryptographique, comme le SHA-256. Lorsque que vous voulez vous connecter à un système en spécifiant votre mot de passe, celui-ci est encrypté en utilisant le même algorithme et comparé au code enregistré dans la base de données. De cette façon, si un pirate réussit à accéder à la base de données, il/elle ne verra que les mots de passe encryptés, et comme nous l'avons expliqué, il n'est pas possible de renverser les codes de hachage afin de retrouver les mots de passe originaux.

Toutefois, si vous avez pu mettre la main sur une liste de mots de passe encryptés, il peut être possible de tenter de deviner quels sont ces mots de passe. Ceci est possible parce que la plupart des utilisateurs ont tendance à sélectionner des mots de passe très simples. Un pirate peut alors simplement encrypter un mot de passe commun (par exemple 12345) et comparer le code obtenu avec tous les mots de passe encryptés de la liste. Cette technique sera particulièrement efficace si un très grand nombre de mots communs sont stockés dans un dictionnaire en utilisant le mot encrypté comme clé. Les codes de la liste de mots de passe obtenus sont alors recherchés dans le dictionnaire. Lorsqu'un match est trouvé, le mot de passe original associé est alors identifié.

Création d'un dictionnaire de mots de passe encryptés

La première étape consiste à créer un dictionnaire de mots de passe. La clé utilisée pour stocker ces mots de passe sera donc la version encryptée du mot de passe. L'encryption se fera avec l'algorithme SHA-1.

Afin de créer cette liste, vous vous baserez sur la liste des 10,000 mots de passe les plus communément utilisés. Cette liste de mots de passes se trouve dans le fichier `commonPwd10K.txt`.
(Avertissement: cette liste peut contenir des mots offensants ou vulgaires; ils sont là parce qu'ils ont été identifiés comme mots de passe courants)

Cette liste de mots de passe doit ensuite être augmentée en appliquant les règles suivantes:

1. Mettre en majuscule la première lettre de chacun des mots de passe (si ils commencent par une lettre), *e.g. dragon devient Dragon*
2. Ajouter l'année courante au mot de passe, *e.g. dragon devient dragon2018*
3. Remplacer les a par le caractère @, *e.g. dragon devient dr@gon*
4. Remplacer les e par le chiffre 3, e.g. *baseball devient bas3ball*
5. Remplacer les i par le chiffre 1, e.g. *michael devient m1chael*

Ces règles peuvent être combinées afin de produire encore plus de mots de passe candidats.

Instructions

Un certain nombre de programmes testeur vous sont fournis (e.g. `Tester.java`) qui devront interagir avec la classe à créer, `PasswordCracker`, et l'une de vos classes dictionnaires (`DatabaseStandard` et `DatabaseMine` réalisant l'interface `DatabaseInterface`). Vous aurez aussi accès à un fichier (`commonPwd10K.txt`) contenant les mots de passe communs et un autre contenant une liste d'utilisateurs et leurs mots de passe encryptés (`leakedAccounts.txt`).

Deux versions de dictionnaires doivent être construits. Ceux-ci doivent contenir des paires : mot de passe et sa version encrypté avec SHA-1. La clé est le code SHA-1 qui sera utilisé pour la recherché des mots de passe contenu dans le fichier de mots de passe piratés.

Partie 1: Dictionnaire Java.util.HashMap

Créer un dictionnaire de mots de passe à partir de la liste des 10000 mots de passe communs augmentée avec les règles spécifiées plus haut. Ce dictionnaire se trouvera dans la classe `DatabaseStandard` réalisant l'interface `DatabaseInterface` et utilisera la structure de données `java.util.HashMap`. Pour ce faire, utiliser la méthode suivante de la classe `PasswordCracker`:

```
void createDatabase(ArrayList<String> commonPasswords,
                    DatabaseInterface database)
```

Cette classe reçoit les mots de passe originaux et est responsable de son augmentation. Cette classe insère tous les mots de passe dans le dictionnaire (vous devriez ainsi obtenir un minimum de 60 000 mots de passe).

Utiliser ce dictionnaire afin de cracker les mots de passe du fichier `leakedAccounts.txt`. Tous ces mots de passe ont été générés à partir de la liste de mots donnée, modifiée en utilisant une combinaison des règles données précédemment. Votre mission est donc de cracker ces mots de passe. Le programme testeur contient le code permettant de lire un fichier et d'effectuer des appels multiples aux méthodes suivantes de la classe `PasswordCracker`:

```
String crackPassword(String encryptedPassword,
                     DatabaseInterface database)
```

Vous devez programmer cette méthode qui doit vérifier si le mot de passe encrypté se trouve dans le dictionnaire, et si c'est le cas, retourne le mot de passe non-encodé (null sinon).

Partie 2: Dictionnaire avec Table de Hachage

Refaire la même tâche mais en utilisant cette fois une table de hachage que vous devez programmer vous-même. Tout le code produit en Partie 1 pourra être ré-utilisé. Vous avez seulement à produire un nouveau dictionnaire `DatabaseMine` réalisant aussi l'interface `DatabaseInterface`.

Votre table de hachage doit utiliser le sondage lineaire afin de résoudre les collisions et la fonction de hachage à utiliser est `key.hashCode() mod N`, ou `N` est un nombre premier.

Les classes à utiliser (ne doivent pas être modifiées)

Tester.java: elle permet de tester votre programme. Vous devez concevoir vos classes de façon à ce qu'elle soit compatible avec ce testeur.

Sha1.java: la classe d'encryption du devoir 2 (sauf que cette fois le code est donné en majuscule).

DatabaseInterface.java: c'est l'interface de vos dictionnaires `DatabaseStandard` et `DatabaseMine`; ces deux dernières classes doivent définir les méthodes de cette interface.

```
public interface DatabaseInterface {
     public String save(String plainPassword, String encryptedPassword);
     public String decrypt(String encryptedPassword);
     public int size();
     public void printStatistics();
}
```

Les classes à concevoir

Voici les spécifications pour les classes à concevoir. Vous pouvez ajouter d'autres méthodes ou attributs au besoin.

Partie 1:
DatabaseStandard.java réalisant **DatabaseInterface** et qui doit encapsuler une instance de java.util.HashMap. Cette classe doit se présenter ainsi:

```
public class DatabaseStandard implements DatabaseInterface {
…
public DatabaseStandard() {…}
       // this constructor must create the initial hash map
…
}
```

La méthode `printStatistics` doit montrer les statistiques telles que montrées dans l'exemple ci-dessous.

PasswordCracker.java: cette classe va insérer des mots de passe dans le dictionnaire. Elle va aussi essayer de décoder les mots de passe.

Les méthode suivantes doivent être présentes:

```
public class PasswordCracker {
…
void createDatabase(ArrayList<String> commonPasswords, DatabaseInterface
database)  {
// receives list of passwords and populates database with entries consisting
// of (key,value) pairs where the value is the password and the key is the
// encrypted password (encrypted using Sha1)
// in addition to passwords in commonPasswords list, this class is
// responsible to add mutated passwords according to rules 1-5.

String crackPassword(String encryptedPassword, DatabaseInterface database) {
//uses database to crack encrypted password, returning the original password
```

Voici un exemple simple d'utilisation de ces classes:

```
PasswordCracker testCracker=new PasswordCracker();
DatabaseStandard database1=new DatabaseStandard();
ArrayList<String> commonPass=new ArrayList<String>();
commonPass.add("123456");
commonPass.add("password");
commonPass.add("12345678");
commonPass.add("brady");
testCracker.createDatabase(commonPass,database1);
database1.printStatistics();
String code=new String("F35D55B3ACF667911A679B44918F5D88B2400823");
String discoverPassword=testCracker.crackPassword(code,database1);
System.out.println("Decrypt: "+code+ " Password: "+discoverPassword+";");
```

```
  Our output for the program above is:
  *** DatabaseStandard Statistics ***
  Size is 20 passwords
  Initial Number of Indexes when Created 37
  *** End DatabaseStandard Statistics ***
  Decrypt: F35D55B3ACF667911A679B44918F5D88B2400823 Password: brady;
```

Partie 2:

DatabaseMine.java réalisant **DatabaseInterface** et qui doit réaliser un dictionnaire avec une table de hachage. La fonction de hachage est appliquée sur une chaine encryptée en utilisant la méthode `hashCode` et en appliquant une opération modulo. Attention de choisir la dimension de la table afin de rencontrer un facteur de charge adéquat. Une liste de 1 million de nombres premiers se trouve ici:
http://www.mathematical.com/primes0to1000k.html

```
public class DatabaseMine implements DatabaseInterface {
  int N; // this is a prime number that gives the number of addresses
// these constructors must create your hash tables with enough positions N
// to hold the entries you will insert; you may experiment with primes N
  public DatabaseMine() {…}      // here you pick suitable default N
  public DatabaseMine(int N) {…} // here the N is given by the user
…
  int hashFunction(String key) {
          int address=key.hashCode()%N;
          return (address>=0)?address:(address+N);
  }
…
public void printStatistics() { … }
// important statistics must be collected (here or during construction) and
printed here: size, number of indexes, load factor, average number of probes
and average number of displacements (see output example below)
}
```

Vous pouvez réutiliser le testeur de la première partie en changeant la second ligne pour:
`DatabaseMine database1=new DatabaseMine(37);`
L'exécution devrait produire le résultat suivant à l'aide de la méthode `printStatistics`:

```
*** DatabaseMine Statistics ***
Size is 20 passwords
Number of Indexes is 37
Load Factor is 0.5405405
Average Number of Probes is 1.5
Number of displacements (due to collisions) 4
*** End DatabaseMine Statistics ***
Decrypt: F35D55B3ACF667911A679B44918F5D88B2400823 Password: brady;
```

Le nombre exact de mots de passe depend de la stratégie adoptee.

A soumettre:

DatabaseInterface.java (as given)
Tester.java (as given)
Sha1.java (as given)
DatabaseStandard.java (part 1)
PasswordCracker.java (the same for part1 and 2)
DatabaseMine.java (part 2)