

Password Cracker (Casseur de mots de passe)

Description

Programme qui permet de cracker (casser) des mots de passes encryptés selon une fonction de hachage. L'utilisateur doit fournir un fichier texte contenant la liste des utilisateurs ainsi que leur mot de passe haché, formaté comme suit :

```
bob:8b433670258f79578f9a4e5ea388b007
jean:08da50bd109c7fb1bec49d15ae86e55f
clauda:a8f6830bce790a8a67fc2e84e12093ba
superuser007:a1234b3161b4fbfd96dd576b65bbea
awesomeMan:6d4db5ff0c117864a02827bad3c361b9
superCoolMan:9460370bb0ca1c98a779b1bcc6861c2c
```

Format : utilisateur:motDePasseHaché

On peut aussi spécifier le caractère séparateur manuellement (voir [Maquette #1](#)).

L'utilisateur a ensuite 3 choix :

- Fournir un dictionnaire de mots
 - Un fichier texte avec un mot par ligne. Il y a de nombreux dictionnaires de disponibles en ligne.
- Fournir une Rainbow Table (table arc-en-ciel) *[J'ai des doutes qu'on puisse être capable de faire ça et de rendre ça optimisé. Voir [Idées](#)]*
 - C'est une liste de toutes les combinaisons possibles d'un certain nombre de caractères, avec leur hash.
 - Par exemple, pour un mot de passe en 1 et 10 caractères utilisant des lettres minuscules (a-z) ou un chiffre (0-9) (ou les deux) il y a **3,760,620,109,779,060** combinaisons possibles. La table arc-en-ciel pèserait environ 316 Go! *[Comment lire ça de manière optimisée?]*.
- Brute Force (essayer toutes les combinaisons possibles)
 - On peut spécifier les caractères qu'on veut tester (Ex. : toutes les lettres majuscules, toutes les lettres minuscules, longueur max, liste de caractères spéciaux, etc.)
 - Le tout serait présenté en checkbox (on coche les options qu'on veut)
 - L'utilisateur pourra aussi fournir la liste des caractères qu'il voudrait inclure dans le brute force
 - C'est moins optimal qu'une table arc-en-ciel, puisqu'on doit générer nous-même les différentes combinaisons, mais au moins l'utilisateur n'a pas besoin de fournir un fichier de 316 Go! Ici, la vitesse dépend beaucoup du processeur de l'utilisateur.

L'utilisateur clique ensuite sur un bouton « Crack All the Passwords! » et attends entre 0 et ∞ secondes.

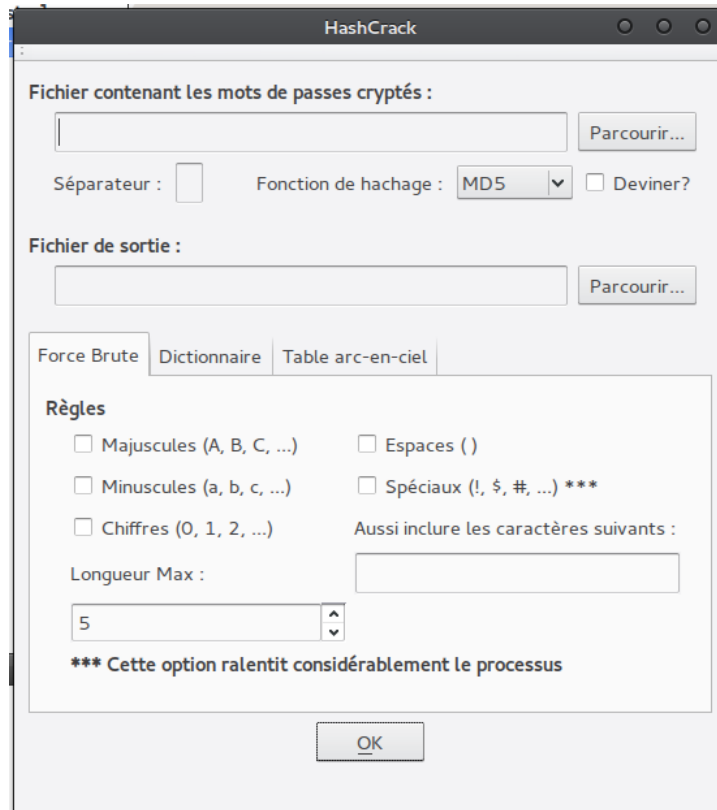
Il serait intéressant d'avoir une fonctionnalité qui permet de générer une table de correspondance (Lookup Table) entre les mots d'un dictionnaire et son hash. Essentiellement, l'utilisateur fournit une liste de mots, et le programme s'occupe de calculer le hash de chaque mot, et l'enregistre en mémoire. Ensuite on les trie, et finalement on exporte le tout dans un fichier texte. L'utilisateur peut ensuite utiliser ce fichier texte pour faire une attaque par dictionnaire.

Objectifs

- MVC
 - L'interface
 - Les fonctions de hachages et autres algorithmes
 - Le projet « Casseur de mots de passe » (Password Cracking).
 - Les tests
- Multithreading
- Implémentation d'au moins 2 fonctions de hachage nous-mêmes (MD5 et SHA-1 sont relativement simples. Il serait intéressant d'avoir aussi le SHA-256).
- Architecture modularisée. C'est-à-dire que le programme est facilement extensible. Par exemple, ajouter une fonction de hachage ne devrait pas changer toutes les classes du programme. Ça veut donc dire : utilisation d'interfaces abstraites.
- Il y a beaucoup de potentiel au niveau des tests. Il y a plusieurs choses à tester.
- C'est un bon travail d'équipe. MVC simplifie la tâche un peu.
- La performance est importante, mais avant tout on veut que ça fonctionne. Il y a de nombreux logiciels qui cassent des mots de passe et qui existent depuis très longtemps (John the ripper, HashCat, rcrack...). Ces logiciels ont plusieurs trucs sous leurs manches, ainsi notre application sera probablement plusieurs ordres de grandeur moins performante. Notre motivation première est de s'éduquer sur le sujet. En codant notre propre application de cassage de mots de passe, on va mieux comprendre comment ça fonctionne.

Maquette #1

Maquette fait avec QT Designer



The image shows a Qt Designer window titled "HashCrack". The window contains a form with the following elements:

- Fichier contenant les mots de passes cryptés :** A text input field and a "Parcourir..." button.
- Séparateur :** A small square input field.
- Fonction de hachage :** A dropdown menu currently showing "MD5".
- Deviner?** An unchecked checkbox.
- Fichier de sortie :** A text input field and a "Parcourir..." button.
- Force Brute**, **Dictionnaire**, and **Table arc-en-ciel**: Three tabs, with "Force Brute" currently selected.
- Règles**: A section containing several checkboxes:
 - ☐ Majuscules (A, B, C, ...)
 - ☐ Minuscules (a, b, c, ...)
 - ☐ Chiffres (0, 1, 2, ...)
 - ☐ Espaces ()
 - ☐ Spéciaux (!, \$, #, ...) ***
- Aussi inclure les caractères suivants :** A text input field.
- Longueur Max :** A spin box currently set to 5.
- *** Cette option ralentit considérablement le processus**: A warning message.
- OK**: A button at the bottom center.

Idées

- Lorsque l'utilisateur fournit son dictionnaire de mots, on va lire le fichier en mémoire, haché tous les mots avec la fonction de hachage voulue (et mettre ça dans une map, ou dans le même fichier dictionnaire).

Ex : Disons que mon dictionnaire (mots.txt) contient les mots

sausage
blubber
pencil

Ce qu'on ferait, c'est qu'on ouvrirait mots.txt, et on calculerait le mot haché à côté du mot, donc ça donnerait :

sausage 8b433670258f79578f9a4e5ea388b007
blubber 08da50bd109c7fb1bec49d15ae86e55f
pencil a8f6830bce790a8a67fc2e84e12093ba

- Avant d'essayer n'importe quoi, on pourrait essayer de voir si le hash du mot de passe existe déjà sur internet (à la BozoCrack <https://github.com/juuso/BozoCrack> version Ruby, <https://github.com/ikkebr/PyBozoCrack> version python). Ou on pourrait garder une liste de sites web avec un API public disponible, et utiliser le service pour trouver rapidement un mot de passe. Pour les mots de passe simples, c'est extrêmement efficace. Pour le fun je l'ai essayé sur une liste de 100 hashes de mots simples (voir annexe) et BozoCrack a été capable de trouver 86% des mots. Drôlement efficace.
- Pour générer des tables arc-en-ciel, on pourrait utiliser une base de données SQLite. En effet, pour chaque combinaisons possible on fait un `INSERT INTO TABLE MD5_PASSWORD_LIST (plaintext, md5) VALUES ({combinaison}, {hashDeLaCombinaison})`. Ensuite pour retrouver le hash lié à un mot on fait une simple requête SQL :

```
SELECT
    password
FROM
    MD5_LIST
WHERE
    MD5_LIST.MD5 = {MD5};
```

L'avantage de ça est qu'on n'aura pas à se casser la tête à développer notre propre algorithme de lecture et d'écriture de table arc-en-ciel (c'est assez compliqué, et on n'a pas le savoir nécessaire pour l'implémenter. Je comprends plus ou moins la page [Wikipédia](#). C'est probablement possible pour nous d'implémenter ça, mais pas en moins d'un mois).

Annexe

10 premiers hash MD5 (sur 100) testé avec BozoCrack

8b433670258f79578f9a4e5ea388b007
08da50bd109c7fb1bec49d15ae86e55f
a8f6830bce790a8a67fc2e84e12093ba
a1234b3161b4fbdfb96dd576b65bbea
6d4db5ff0c117864a02827bad3c361b9
9460370bb0ca1c98a779b1bcc6861c2c
df53ca268240ca76670c8566ee54568a
7516c3b35580b3490248629cff5e498c
91e02cd2b8621d0c05197f645668c5c4
5f9901fc60b769b523d0dd8e79b3fe08

Les mots reliés aux hashes

sausage
blubber
pencil
cloud
moon
water
computer
school
network
hammer

Liste de sites web avec un API

<http://md5crack.com/>

Outils existants

rcracki_mt

Voici un exemple de session de cassage de mots de passe avec rcracki_mt. Ici j'utilise la table arc-en-ciel « md5_loweralpha-numeric-space#1-8 » disponible [ici](#). C'est une table arc-en-ciel de 16 Go qui comprend toutes les combinaisons possibles des mots de passe contenant des caractères alphanumériques minuscule + espace (abcdefghijklmnopqrstuvwxyz0123456789) de 8 caractères et moins (3 610 048 327 640 combinaisons possibles).

Fait amusant : La table arc-en-ciel de toutes les combinaisons possibles des caractères alphanumériques minuscules de 9 caractères et moins pèse 109 Go ! Le fait d'ajouter un seul caractère augmente le nombre de possibilités considérablement (133 571 788 122 717 combinaisons). La plus grosse table disponible sur freerainbowtables.com/fr/tables2/ pèse 1049 Go (1.05 To) et contient toutes les combinaisons possibles des mots de passe de 8 caractères et moins contenant caractères suivant :

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$$%^&*()-
_+=~`[]{}|\;:'"<>.,?/
```

Voici donc une session typique. md5.txt contient les 5 hashes MD5 suivant :

```
a7d864bef5c2f814a2055a87637666ec
cbdb7e2b1ed566ceb796af2df07205a3
6af9ae1fb980f60592c2fb03150b229f
4652b19e09ced75df510bf5a263a2bfe
945e9f0b4e381b13aa70b94b89a28709
```

Voyons si rcracki_mt est capable de trouver les mots associés à ces hashes :

```
[Raphael][cygdrive/f/rainbow_tables/rcracki_mt_0.7.0_win32_mingw] $ ./rcracki_mt.exe -l md5.txt -t 4 -o
results.txt *.rti2
```

```
Using 4 threads for pre-calculation and false alarm checking...
Found 40 rainbowtable files...
```

```
md5_loweralpha-numeric-space#1-8_0_10000x24642670_distrtrtgen[p][i]_09.rti2
Chain Position is now 24642670
147856020 bytes read, disk access time: 2.30s
searching for 5 hashes...
Pre-calculating hash 1 of 5.
Pre-calculating hash 5 of 5.
```

```
md5_loweralpha-numeric-space#1-8_0_10000x67108864_distrtrtgen[p][i]_00.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 5.25s
searching for 4 hashes...
Checking false alarms for hash 1 of 4.
Checking false alarms for hash 2 of 4.
```

```
md5_loweralpha-numeric-space#1-8_0_10000x67108864_distrtrtgen[p][i]_01.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 5.19s
searching for 3 hashes...
Checking false alarms for hash 1 of 3.
```

Checking false alarms for hash 2 of 3.
Checking false alarms for hash 3 of 3.

md5_loweralpha-numeric-space#1-8_0_10000x67108864_distrtrtgen[p][i]_02.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 5.39s
searching for 1 hash...
Checking false alarms for hash 1 of 1.

md5_loweralpha-numeric-space#1-8_0_10000x67108864_distrtrtgen[p][i]_03.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 5.01s
searching for 1 hash...
Checking false alarms for hash 1 of 1.

md5_loweralpha-numeric-space#1-8_0_10000x67108864_distrtrtgen[p][i]_04.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 4.97s
searching for 1 hash...
Checking false alarms for hash 1 of 1.

md5_loweralpha-numeric-space#1-8_0_10000x67108864_distrtrtgen[p][i]_05.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 5.14s
searching for 1 hash...
Checking false alarms for hash 1 of 1.

md5_loweralpha-numeric-space#1-8_0_10000x67108864_distrtrtgen[p][i]_06.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 5.04s
searching for 1 hash...
Checking false alarms for hash 1 of 1.

md5_loweralpha-numeric-space#1-8_0_10000x67108864_distrtrtgen[p][i]_07.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 5.14s
searching for 1 hash...
Checking false alarms for hash 1 of 1.

md5_loweralpha-numeric-space#1-8_0_10000x67108864_distrtrtgen[p][i]_08.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 5.11s
searching for 1 hash...
Checking false alarms for hash 1 of 1.

md5_loweralpha-numeric-space#1-8_1_10000x24849868_distrtrtgen[p][i]_09.rti2
Chain Position is now 24849868
149099208 bytes read, disk access time: 1.97s
searching for 1 hash...
Pre-calculating hash 1 of 1.

md5_loweralpha-numeric-space#1-8_1_10000x67108864_distrtrtgen[p][i]_00.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 5.09s
searching for 1 hash...
Checking false alarms for hash 1 of 1.

md5_loweralpha-numeric-space#1-8_1_10000x67108864_distrtrtgen[p][i]_01.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 9.35s
searching for 1 hash...
Checking false alarms for hash 1 of 1.

statistics

```
-----
plaintext found:           5 of 5(100.00%)
total disk access time:    64.95s
total cryptanalysis time:  3.39s
total pre-calculation time: 21.00s
total chain walk step:     299910006
total false alarm:         11293
```

total chain walk step due to false alarm: 41593767

result

a7d864bef5c2f814a2055a87637666ec	patate08	hex:7061746174653038
cdbb7e2b1ed566ceb796af2df07205a3	bond007	hex:3626f6e64303037
6af9ae1fb980f60592c2fb03150b229f	31337	hex:336c333337
4652b19e09ced75df510bf5a263a2bfe	peanut	hex:7065616e7574
945e9f0b4e381b13aa70b94b89a28709	patate	hex:706174617465

rcracki_mt a donc trouvé 5 mots de passe sur 5. Le tout en moins de 2 minutes (Ce qui est très impressionnant, ça montre comment leur algorithme pour traverser une table arc-en-ciel est performant). MAIS, plus haut je parlais de BozoCrack (et de sa version python PyBozoCrack), et en utilisant la même liste j'ai pu trouver 4 mots de passe sur 5. Pas si mal que ça. Et ça a pris environ le 1/4 du temps. Cependant, avec des mots de passe plus complexes c'est moins efficace (mais, comme je disais plus haut, ça vaut la peine d'essayer ça en premier).