# Titre du laboratoire

Auteurs: Edwin Häffner, Arthur Junod

## Description des fonctionnalités

Ce logiciel permet de cracker un hash MD5 en bruteforce avec un nombre customisable de threads.

## Choix d'implémentation

### Parallélisation avec des threads

Le coeur de l'algorithme de bruteforce a été placé dans une fonction exécutée par chaque thread. Chaque thread reçoit en paramètre des pointeurs permettants de retourner des valeurs, des variables qui sont juste récupérer des paramètres de ThreadManager et des variables qui sont initialisées dans ThreadManager afin qu'elle ne soit pas reintialisées à chaque thread.

Pour éviter les calculs redondants, chaque thread commence avec un mot de passe différent en incrémentant le premier mot de passe testé par l'ID du thread. Par exemple, le thread 1 commence par "aaaa", le thread 2 par "baaa", etc.

De plus, à chaque nouvel essai, le mot de passe testé est incrémenté du nombre total de threads. Ainsi, les threads testent des mots de passe complètement différents.

Lorsqu'un thread trouve le mot de passe, la variable *found* est mise à true pour indiquer aux autres threads de s'arrêter.

Le mot de passe trouvé et le nombre total d'essais sont passés par adresse pour être mis à jour par tous les threads.

### Barre de progression

Une barre de progression affiche l'avancement en se basant sur le nombre total d'essais effectués. Elle est mise à jour toutes les 1000 opérations par les threads en parallèle. Nous nous sommes rendus compte qu'elle n'étais pas totalement précise en la testant avec, par exemple, le mdp "****" car elle arrive à 100% un peu avant que le programme n'ait vraiment trouvé la réponse. Nous avons essayé de régler le problème mais ne trouvant pas mieux avons laissé notre solution la plus proche de la réalité.

## Tests effectués

Nous avons essayé différentes valeurs de hash qui nous semblaient représentatives tout en changeant le nombre de threads.

Nous avons commencé par tester différents hash avec différentes valeurs de threads afin de vérifier qu'augmenter le nombre de threads réduisait bien le temps d'exécution. Nous nous sommes rendus compte que le temps baissait bien en incrémentant le nombre de threads jusqu'à 3, mais qu'une fois cette valeur dépassée le temps réaugmentais. Nous pensons que cela peut venir du fait que nous executions le programme sur une VM.

Nous avons aussi testé la valeur maximale "****" et cela nous a permis de mettre en lumière le fait que nous n'arrivions jamais à l'atteindre. Nous avons donc pu régler un problème que nous avions au niveau de la condition de la boucle while principale des threads (Nous sortions de la boucle avant d'avoir tester tous les cas).

Nous avons pu confirmer que le crackage marchait avec un sel.