Université de Genève

-

Sciences Informatiques

Systèmes d'Exploitation - TP 02

Noah Munz - Gregory Sedykh

Octobre 2022

TP 02

2 Cryptographie

2.1 Concepts

Exercice: Hasher le string "Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux" avec SHA1 puis MD5. La première fois en lui passant un fichier contenant la string, la deuxième fois en combinant echo avec les SHA1 et MD5.

SHA1 dans fichier: 74bac9f1611450f44e0a377da1b0732034d20f59
MD5 dans fichier: ef81ae80ec507096f761cefa49f7b63e

SHA1 avec echo: aa91cf14c329a5fe9b1158ae2665b39370f57e37
MD5 avec echo: 120227d6118cfddbd21639644aa0884d

On voit que les 2 hashs different dans les 2 cas car on a un retour à la ligne dans le fichier mais pas lorsque l'on pipe directement l'output de `echo` dans l'input de SHA1 et MD5.

2.2 La libraire OpenSSL

La fonction de l'exemple man est dans le fichier `examples/openssl_example.c` .
On voit bien que en changeant le message à "Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux", on recoit le même hash que dans la partie (2.1).

3 Gestion des paramètres d'un programme

La fonction de l'exemple man est dans le fichier `getopt_example.c`

4 Intégration: Le programme à réaliser

4.1 Contenu du projet

(1) Code

Le code de l'implémentation, qui se compile avec la *"target"* `all` du `Makefile` , se trouve dans:

- Le fichier `main.c` qui contient la fonction main qui permet de lancer le programme.
  Son fonctionnement sera expliqué en détail ci-après (l'executable qui lui fait directement appel est compilé, à la racine, sous le nom `digest` .)
- Le module `OptionParser` qui définit les différentes options disponibles et contient tout le nécessaire au traitement des options et arguments spécifiés par l'utilisateur.
- Le module `hash_calc` qui s'occupe de calculer les hashs de tout ce qui a été demandé.

(2) Reste

Le reste du projet contient :

- Un dossier `exemples` qui contient l'implémentation demandée des examples du manuel. Ils peuvent être compilés avec les targets `<nomDuTest>_example` du Makefile.
- Un dossier `res` qui contient des fichiers à être utilisé pour tester le calcul des hashs.
- Un dossier `out` qui contient le code compilé et les `.o` des différents modules.
- Et finalement, le Makefile.

4.2 Manuel

La commande `make all` permet de compiler tous les fichiers.

Comme dit précédemment, le makefile donne aussi d'autres possibilités de compilation (i.e. compiler seulement les `.o` situé dans `out/`, les exemples dans `exemples/` ...)

On peut exécuter le programme de 4 manières différentes:

1-2) En spécifiant un ou plusieurs fichiers avec `-f`, en spécifiant (ou pas) la fonction de hashage `-t` : `./digest -f file1 file2 ...[-t <hashMethod>]`.

Ce qui retournera **1** hash **par** fichier. E.g.:
`$ ./digest -f res/test.txt res/string-tohash.txt -t md5`

```
Hashing Method:  md5
Hashing file "res/string-tohash.txt"...
120227d6118cfddbd21639644aa0884d File:  res/string-tohash.txt
```
(ou `ef81ae80ec507096f761cefa49f7b63e` sans retour à la ligne final)
```
------------

Hashing file "res/test.txt"...
4e1243bd22c66e76c2ba9eddc1f91394e57f9f83 File:  res/test.txt
------------
```

3-4) En spécifiant une ou plusieures chaînes de caractères, en spécifiant (ou pas) la fonction de hashage `-t` :
`./digest string1 string2 ...[-t <hashMethod>]`.

Ce qui retournera **1** hash **pour** la concaténation de **tous** les strings. E.g.:

```
$ ./digest -t sha256 max linux, windows = linux

Hashing Method:  sha256
String to hash:  "max linux, windows = linux"
Digest:  069da09aa2f501f87b1603db31659261bddd89adae1bd6e895b3d36bbb495250
```

(l'ordre de l'option `-t` n'est pas important, elle peut être au début ou à la fin.)

Pour vérifier, `sha1sum string-tohash.txt` et `./digest -f res/string-tohash.txt` donnent le même hash digest (voir 2.1)