TP2

Contact :
Marios.Fanourakis@unige.ch
Chen.Wang@unige.ch
Clement.Jeannet@etu.unige.ch
Sajaendra.Thevamanoharan@etu.unige.ch

October 2020

1 Objectifs

L'objectif général du TP est de créer un programme qui permet de générer des hashs, également appellés digests (MD5, SHA1, etc.) à partir de chaines de caractères ou de fichiers. Le but des digests est de pouvoir contrôler l'intégrité de données transmises entre deux entités.

Les objectifs pédagogiques de ce TP sont de:

- manipuler des chaines de caractères en C;
- manipuler les paramètres argc / argv de la fonction main;
- se familiariser avec la fonction getopt qui permet de manipuler les paramètres passés au programme;
- effectuer un lien vers des librairies externes (ici la librairie EVP de openssl);
- se familiariser avec les fonctions de hachage qui est un mécanisme de base de la cryptographie.

2 Cryptographie

2.1 Concepts

D'une manière générale, les mécanismes de cryptographie permettent de communiquer des messages de manière protégée. La protection des données implique trois aspects:

- authenticité: s'assurer que l'on communique bien avec la personne/machine souhaitée;
- confidentialité: chiffrer les messages envoyés pour qu'ils ne soient pas lisibles par n'importe qui (i.e. seule une personne possédant une clé de décryptage peut lire le message);
- intégrité: s'assurer que le message reçu est bien identique au message envoyé et qu'il n'a pas été corrompu lors du transfert.

Le point qui nous intéresse pour ce TP est **l'intégrité** des données. L'intégrité de données est assurée en utilisant une fonction de hachage h sur le message m à envoyer (le message peut être le contenu d'un fichier ou n'importe quelle chaine de caractères). Cette fonction de hachage retourne une valeur $h(m)$ qui peu ensuite être vérifiée à la réception en utilisant la même fonction de hachage sur le message reçu m'. Dans le cas ou les deux valeurs $h(m)$ et $h(m')$ sont différentes on peu alors conclure que les données ont été corrompues. Une fonction de hachage se doit d'avoir (au moins) les propriétés suivantes:

- il doit être improbable de trouver deux messages ayant le même hash ($h(m_1) = h(m_2)$);
- une petite modification du message m doit donner lieu à un important changement de $d = h(m)$;
- le message ne doit pas pouvoir être reconstruit à partir de la valeur retournée par la fonction de hachage (pour des raisons de sécurité non évoquées ici).

Plusieurs fonction de hachage ont été developpées parmis les quelles ont peu trouver SHA1, MD5, SHA512.

Exercice: Votre première tâche consiste à vous familiariser avec deux outils de calcul de hachage *sha1sum* et *md5sum*. Ces outils vous permettrons par la suite de vérifier que votre programme fonctionne correctement. Afin de vous familiariser avec ces deux programmes:

- lisez le manuel des commandes *sha1sum* et *md5sum*;
- créez un fichier contenant le texte "Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux" sans retour à la ligne. Utiliser les commandes ci-dessus pour calculer les hashs du contenu du fichier.
- combiner la commande *echo "Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux".* avec les commandes ci-dessus pour calculer les hashs sans utiliser de fichier;
- le résultat est différent, pourquoi? Comment résoudre le problème?

2.2 La librairie openssl

Openssl est un ensemble d'outils permettant d'implémenter le protocol de sécurité TLS/SSL. Dans ce TP nous n'allons utiliser qu'une petite partie des fonctions proposées par openssl afin de calculer les digests/hashs de fichiers et de chaines de caractères d'une manière similaire aux commandes *sha1sum* et *md5sum*. Les fonctions utilisées sont regroupées dans la librairies EVP de openssl et commencent toutes par *EVP_Digest*.

Exercice:

- lisez le manuel des fonction EVP_Digest... (man EVP_DigestInit);
- implémentez l'exemple proposé dans les pages du manuel;
- compilez le programme en prenant soin de lier les librairies libssl et libcrypto;
- testez le programme avec la phrase "Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux."

3 Gestion des paramètres d'un programme

Vous avez vu en cours qu'il est possible de manipuler les paramètres d'un programme en utilisant les variables *argc / argv* de la fonction *main*. Nous vous proposons ici d'utiliser la fonction POSIX *getopt* qui permet de standardiser la gestion de paramètres. Comme vous l'avez vu la plupart des programmes utilisent des paramètres précédés du signe moins pour indiquer une option. De plus certaines options peuvent prendre des arguments. Par exemple pour la commande *grep*:

- l'option -i permet de ne pas considérer les différences de case (i.e. majuscules / minuscules), elle ne contient pas d'argument;
- l'option -f FILE prend un argument FILE qui indique un fichier contenant une liste de pattern à rechercher;

- les paramètres supplémentaires à la commande *grep* sont les fichier dans lesquels chercher le(s) pattern(s) indiqué(s).

Voici le fonctionnement global de la fonction *getopt*. Cette fonction prend en paramètres les arguments *argc* et *argv* que l'on peu directement rediriger depuis la fonction main. De plus l'argument *optstring* permet d'indiquer quelles sont les lettres qui correspondent à des options (e.g. "fot" indique à la fonction que le programme à trois options possibles -f, -o et -t). Le caractère ':' indique que l'option précédent le caractère contient un argument (e.g. "fo:t" indique que l'option -o sera suivie d'un argument).

A chaque appel de la fonction *getopt* celle-ci retourne un des caractères d'option qui à été trouvé dans la ligne de commande du programme. Dans le cas ou il n'y a plus ou pas de caractère d'option disponible la fonction retourne -1; si une option inconnue (e.g. -z suivant la chaine *optstring* "fo:t") ou mal formulée (e.g. -f argument) à été trouvée la fonction retourne '?'.

Lorsqu'une option contenant un argument est trouvé on peu alors utilisé la variable globale *optarg* pour obtenir un pointeur sur l'argument. Ce pointeur pointe en fait soit sur le prochain élément *argv* (cas ou l'option et l'argument sont séparés d'un espace) soit sur le texte suivant l'option dans le même élément *argv*.

Finalement, les arguments du programme qui ne correspondent pas à des options sont transférés à la fin du tableau *argv*. La variable gloable *optind* pointe alors sur l'élément de *argv* qui contient le premiers de ces arguments.

Ces informations devraient être suffisantes pour utiliser la fonction *getopt*. Toutefois il est vivement recommandé de lire le manuel de cette fonction (man 3 getopt) pour voir un exemple d'utilisation ainsi que d'autres fonctionnalités.

Exercice:

- lisez le manuel de la fonction *getopt*;
- implémentez l'exemple proposé dans les pages du manuel et testez le.

4 Intégration: le programme à réaliser

Le but de ce tp est de créer un programme permettant de générer des hash de fichiers et de chaines de caractères. Le programme pourra donc être appellé de deux manières. Avec l'option -f les paramètres d'entré seront considérés comme des fichiers et un hash code par fichier sera calculé. Sans l'option -f les paramètres du programmes seront considérés comme une unique chaine de caractère. Ainsi:

```
1 hash -f ficher1 fichier2 ...
```

retournera par exemple (cela depend bien sure du contenu du fichier):

```
1 8f85e20c95a5a73b9670883fd2cfc5ecd23e9b8b        fichier1
2 57ba123423aadc8a65a3a83066f0c7523787c211        fichier2
```

alors que:

```
1 hash ceci est une chaine
```

retournera le hash de la chaine “ceci est du chaine”.

De plus l'option -t permettra de choisir quel type de digest sera utilisé pour calculer les hashs (e.g. MD5, SHA1, ...). Dans le cas ou l'option -t n'est pas spécifiée par l'utilisateur le digest “SHA1” sera utilisé.

Le programme devra être implementé en deux modules contenant chacun un .c et un .h ainsi que du fichier .c contenant la fonction main (5 fichiers au total). Le premier module se chargera de gérer les options passées par l'utilisateur. Le second module se chargera de calculer le hash de fichier et de chaines de caractères. Il est important de bien pensé quelles parties du codes seront disponibles à l'utilisateur (i.e. présentes dans le .h) et quelles fonctions seront privées.