TP2 Systèmes Informatiques

2.1) Concepts

Les commandes sha1sum et md5sum permettent les deux de générer des checksum à partir d'un fichier source. C'est utilisé pour détecter des erreurs, vérifier l'intégrité des données, ils sont plus performants que des simples CRC.

J'ai lancé sous bash les commandes suivantes :

```
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ touch file
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ nano file
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ cat file
Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ sha1sum file
aa91cf14c329a5fe9b1158ae2665b39370f57e37  file
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ md5sum file
120227d6118cfddbd21639644aa0884d  file
```

J'ai refait la même chose en pipant le standard output de echo "Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux"

```
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ echo "Le manuel disait: Nécessite Windo
ws 7 ou mieux. J'ai donc installé Linux" | sha1sum
aa91cf14c329a5fe9b1158ae2665b39370f57e37  -
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ echo "Le manuel disait: Nécessite Windo
ws 7 ou mieux. J'ai donc installé Linux" | md5sum
120227d6118cfddbd21639644aa0884d  -
```

vers sha1sum et md5sum. Le résultat fût :

Donc exactement le même ! On s'attendait à avoir une différence, car selon l'éditeur de texte les fins de fichier ou de ligne peuvent être signalés par un caractère additionnel rajouté par l'éditeur, ce qui changerait le hash fondamentalement. Pour y remédier, on se sert d'un éditeur qui ne le fait pas, comme nano dans ce cas. Les hash seront, comme on s'y attendait, les mêmes. Si on utilisait LibreOffice, ce serait différent.

2.2) Librairie Open SSL

Afin de pouvoir consulter le manuel sur EVP_DigestInit, il a fallu installer les manuels sur la librairie ssl. Nous avons donc utilisé apt install libssl-doc et apt install libssl-dev.

Pour compiler, il a fallu lier les deux librairies libssl et libcrypto, pour ce faire on utilise -l qui est une option qui s'autocomplète en

```
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ gcc impl.c -lssl -lcrypto
```

lib, on fournit les librairies en paramètre à gcc.

En étudiant le programme et en lisant le manuel, j'ai appris qu'il faut appeler ./a.out HASH_FUNCTION et il appliquera cette fonction sur le

message dans la source. J'ai simplifié le code afin de seulement hacher le message "Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux", le code source est devenu :

```c
#include <stdio.h>
#include <string.h>
#include <openssl/evp.h>

int main(int argc, char *argv[])
    {
    EVP_MD_CTX *mdctx;
    const EVP_MD *md;
    char mess[] = "Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé
Linux";
    unsigned char md_value[EVP_MAX_MD_SIZE];
    unsigned int md_len, i;

    if (argv[1] == NULL) { //Si on a pas fourni de fonction de hash.
        printf("Usage: mdtest digestname\n"); //On dit à l'utilisateur d'en fournir.
        exit(1);
    }
//Sinon, on vérifie qu'elle existe.
    md = EVP_get_digestbyname(argv[1]);
if (md == NULL) { //Si elle existe pas, on exit.
        printf("Unknown message digest %s\n", argv[1]);
        exit(1);
    }

    mdctx = EVP_MD_CTX_new(); //alloue et renvoie un "context de digest"
  //configure le context de digest à utiliser un digest du ENGINE impl.
    EVP_DigestInit_ex(mdctx, md, NULL);
   //hash nb octets de données dans le context mdctx.
    EVP_DigestUpdate(mdctx, mess, strlen(mess));
   //récupère la valeur du digest de mdctx et la place dans md_value
    EVP_DigestFinal_ex(mdctx, md_value, &md_len);
    EVP_MD_CTX_free(mdctx);

    printf("Digest is: ");
    for (i = 0; i < md_len; i++)
        printf("%02x", md_value[i]);

    printf("\n");

    exit(0);
}
```

En le testant, j'ai reçu les résultats suivants, j'ai également comparé avec un hash256 du programme *hashalot* et les résultats sont bien les mêmes.

```
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ ./a.out sha256
Digest is: 92450fc4fb4a4f29c2be9780eb5c15ca3f9b83a7dc76d2f197ce44d9bd3abd4c
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$ sha256 -x
mlockall: Cannot allocate memory
Warning: couldn't lock memory, are you root?
Enter passphrase:
92450fc4fb4a4f29c2be9780eb5c15ca3f9b83a7dc76d2f197ce44d9bd3abd4c
gordon@LAPTOP-J9DDPHVI:~/Documents/Systemes_Informatiques/TP2$
```

3) Gestion des paramètres d'un programme

J'ai lû le manuel sur getopt mais la référence GNU expliquait son utilisation mieux[i], je vais parler de cette fonction dans le cadre de l'utilisation que j'en fait dans mon programme comme exemple. La signature de getopt est la suivante,

int getopt *(int argc, char *const *argv, const char *options)*

Ou argc est le nombre d'arguments, char *argv le tableau d'arguments fourni au programme et const char *options sont les options que l'on cherche à gérer. Dans le cas des options -f et -t, on s'attends à pouvoir récupérer le type de digest ainsi que les noms de fichier.

Mais attention -f est juste un flag, c.à.d si on se sert de -f, tout ce qui suivra devra être traité comme fichier, excepté ce qui suit -t ! Afin d'implémenter ce comportement, on utilise :

```
opt = getopt(argc, argv, "ft:")
```

Opt prendra la valeur de f et de t (en int ASCII) et on traite les cas via un case, avec ce choix, le nom de hash suivant -t sera traité en tant qu'argument d'option, ainsi après avoir traité toutes les options, optind sera égal à la position de argv à partir de laquelle les arguments non liés aux options sont fournis, qui sont soit les noms de fichier, soit un message ! (On distingue les deux cas dans le case si -f est présent).

4) Intégration : le programme à réaliser

Afin de créer des modules en C, il a fallu que j'apprenne comment on déclarait un module, chose que l'on n'avait pas abordé en cours. Je me suis donc référé à un site expliquant la démarche pour créer une interface[ii]. J'ai appris que c'était en fait assez simple.
Soit un fichier interface.h, ce fichier contient juste les **déclarations** des fonctions et variables en utilisant le keyword **extern** DECLARATION. Ces déclarations seront disponibles à n'importe quel fichier qui veut s'en servir.

Le fichier qui les **implémente** est le fichier interface.c, qui lui importe l'interface à implémenter, et si il doit créer des fonctions ou variables qui doivent rester propre à ce fichier et pas partie de l'interface, on utilise le keyword **static** DECLARATION (comme fait avec get_text dans hash.c). C'est comme cela que j'ai procédé pour créer mes deux modules, le seul fichier C contenant un main est donc main.c, tous les autres fichiers contenant uniquement des déclarations ou des implémentations. Par contre il ne faut pas oublier de lier les modules lors de la compilation de main.c ! Avec la commande :

gcc main.c -lssl -lcrypto hasher.c gereoptions.c

Les implémentations des fonctions de hashage sont pratiquement les mêmes que dans la section 2.2), avec quelques ajouts de paramètres, pour les noms de fichier et type de digest. Il a fallu que je lise un fichier et que je puisse le mettre dans un buffer, stack overflow[iii] est venu à ma rescousse et j'ai appris à me servir des fonctions fseek, fread et le fonctionnement général d'un stream[iv], avec l'indicateur de position que l'on déplace afin de lire un fichier morceau par morceau de taille souhaitée avec fread.

Je vous laisse évaluer l'implémentation jointe, qui est abondamment commentée. Les références sont en dernière page.

i https://www.gnu.org/software/libc/manual/html_node/Using-Getopt.html
ii https://www.embedded.com/modular-programming-in-c/
iii https://stackoverflow.com/questions/174531/how-to-read-the-content-of-a-file-to-a-string-in-c
iv https://www.gnu.org/software/libc/manual/html_node/Streams.html