


# Générateur/Recherche de condensats C

François MARTINEL

# Objectifs

- (G) Lire un dictionnaire de mots de passe
- (G) Calculer le condensat cryptographique (SHA-256 ou autre)
- (G) En faire un fichier T3C (Hash      MdpEnClair)
- (L) Puis à partir d'un hash retrouver le mot de passe clair

# Architecture

 hellofunc.c  
hellomake.h  
main.c

- Mode G: -i (input) -o (output) -a (algo)
- ./main G -i list.txt -o output.t3c -a sha256
- Mode L: -t (fichier t3c) -c (fichier condensats)
- ./main L -t output.t3c -c condensats.txt

```
typedef struct Node {  
    char *condensat;  
    char *chaine;  
    struct Node *gauche;  
    struct Node *droite;  
} Node;  
Node* create_node(char *hash, char *valeur);  
Node* insert_node(Node *noeud, char *hash, char *valeur);  
Node* find_node(Node *node, char *hash);  
void free_node(Node *node);
```

# Arbre binaire

```
Node* create_node(char *hash, char *valeur) {
    Node *node = (Node*)malloc(sizeof(Node));
    node->condensat = strdup(hash);
    node->chaine = strdup(valeur);
    node->gauche = NULL;
    node->droite = NULL;
    return node;
}

Node* insert_node(Node *noeud, char *hash, char *valeur) {
    if (!noeud) {
        return create_node(hash, valeur);
    }
    int calcul = strcmp(hash, noeud->condensat);
    if (calcul < 0) {
        noeud->gauche = insert_node(noeud->gauche, hash, valeur);
    } else if (calcul > 0) {
        noeud->droite = insert_node(noeud->droite, hash, valeur);
    } else {
        char *valid = strdup(valeur);
        if (valid) {
            free(noeud->chaine);
            noeud->chaine = valid;
        }
    }
    return noeud;
}
```

```
Node* find_node(Node *node, char *hash) {
    while (node) {
        int calcul = strcmp(hash, node->condensat);
        if (calcul == 0) {
            return node;
        }
        if (calcul < 0) {
            node = node->gauche;
        } else {
            node = node->droite;
        }
    }
    return NULL;
}

void free_node(Node *node) {
    if (!node) {
        return;
    }
    free_node(node->gauche);
    free_node(node->droite);
    free(node->condensat);
    free(node->chaine);
    free(node);
}
```

```
ubuntu:~/c-projet$ valgrind --leak-check=full ./main G -i list.txt -o output.t3c -a sha256
==4510== Memcheck, a memory error detector
==4510== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==4510== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==4510== Command: ./main G -i list.txt -o output.t3c -a sha256
==4510==
```

hachage utilise: sha256

Generate termine.

```
==4510==
==4510== HEAP SUMMARY:
==4510==    in use at exit: 0 bytes in 0 blocks
==4510==   total heap usage: 1,566,816 allocs, 1,566,816 frees, 87,285,973 bytes allocated
==4510==
==4510== All heap blocks were freed -- no leaks are possible
==4510==
==4510== For lists of detected and suppressed errors, rerun with: -s
==4510== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

# OpenSSL

- Choix du hachage

```
const EVP_MD* choix(char *name);
```

- Fonction simple donnée dans la doc

```
// https://wiki.openssl.org/index.php/EVP_Message_Digests
void digest_message(const unsigned char *message, size_t message_len, unsigned char **digest, unsigned int *digest_len, char *name) {
    EVP_MD_CTX *mdctx;
    const EVP_MD *hachage = choix(name);
    if(!hachage) {
        return;
    }
    if((mdctx = EVP_MD_CTX_new()) == NULL) return;
    if(1 != EVP_DigestInit_ex(mdctx, hachage, NULL)) return;
    if(1 != EVP_DigestUpdate(mdctx, message, message_len)) return;
    if((*digest = (unsigned char *)OPENSSL_malloc(EVP_MD_size(hachage))) == NULL) return;
    if(1 != EVP_DigestFinal_ex(mdctx, *digest, digest_len)) return;
    EVP_MD_CTX_free(mdctx);
}
```

```
const EVP_MD* choix(char *name) {
    if (strcmp(name, "sha1") == 0) {
        return EVP_sha1();
    } else if (strcmp(name, "ripemd160") == 0) {
        return EVP_ripemd160();
    } else if (strcmp(name, "sha224") == 0) {
        return EVP_sha224();
    } else if (strcmp(name, "sha256") == 0) {
        return EVP_sha256();
    } else if (strcmp(name, "sha384") == 0) {
        return EVP_sha384();
    } else if (strcmp(name, "sha512") == 0) {
        return EVP_sha512();
    } else if (strcmp(name, "md5") == 0) {
        return EVP_md5();
    }
    return NULL;
}
```

# Generate

- Lecture d'un fichier puis conversion avec OpenSSL

```
int generate(char *fichier, char *fichier_output, char* hachage) {
    FILE *fp = fopen(fichier, "r");
    if(!fp) {
        return 1;
    }

    FILE *fo = fopen(fichier_output, "w");
    if(!fo) {
        return 1;
    }

    printf("\nhachage utilise: %s\n", hachage);
    char ligne[1024];
    while(fgets(ligne, sizeof(ligne), fp)) {
        filtre(ligne);
        if (ligne[0] == '\0') continue;
        unsigned char *digest = NULL;
        unsigned int digest_len = 0;
        digest_message((unsigned char*)ligne, strlen(ligne), &digest, &digest_len, hachage);
        if(!digest) {
            fclose(fp);
            fclose(fo);
            return 1;
        }
        for(unsigned int i = 0; i<digest_len; i++) {
            fprintf(fo, "%02x", digest[i]);
        }
        fprintf(fo, "\t%s\n", ligne);
        OPENSSL_free(digest);
    }
    fclose(fp);
    fclose(fo);
    printf("\nGenerate termine.\n");
    return 0;
}
```

# Lookup

- Comparaison entre le fichier t3c et les condensats

```
int lookup(char *fichier_t3c, char *cible) {
    Node *node = load(fichier_t3c);
    if(!node) {
        return 1;
    }

    FILE *c = fopen(cible, "r");
    if(!c) {
        free_node(node);
        return 1;
    }
    char ligne[1024];
    while(fgets(ligne, sizeof(ligne), c)) {
        filtre(ligne);
        Node *n = find_node(node, ligne);
        if(n) {
            printf("%s\n", n->chaine);
        } else {
            printf("\n");
        }
    }
    fclose(c);
    free_node(node);
    printf("\nLookup termine.\n");
    return 0;
}
```

```
int lookup_entree(char *fichier_t3c) {
    Node *node = load(fichier_t3c);
    if(!node) {
        return 1;
    }
    char ligne[1024];
    while(fgets(ligne, sizeof(ligne), stdin)) {
        filtre(ligne);
        Node *n = find_node(node, ligne);
        if(n) {
            printf("%s\n", n->chaine);
        } else {
            printf("\n");
        }
    }
    free_node(node);
    printf("\nLookup termine.\n");
    return 0;
}
```



# Questions ?