Licence L2 STS Mention SPI Parcours Informatique
Unité 174EN007
Sécurité Informatique

Correction TD2

Ce second TD porte sur l'algorithme de hachage SHA-256 et son implémentation en C.

Exercice n°1

Etudiez les principes de fonctionnement de l'algorithme SHA-256 à partir de la documentation issue de la fiche Wikipédia.

Cf. documentation jointe au TD.

Exercice n°2

- Dans la construction de Merkle-Damgård a quoi correspond *IV* ?

IV : Initialization Vector Ensemble des valeurs initiales pour le premier bloc de données de 512 bits. Ces valeurs sont fixées quelque soit la taille du message. Ils servent pour la première exécution de la fonction de compression sur le premier bloc de données de 512 bits.

- Dans le prétraitement, à quoi sert de compléter (bourrage ou *padding*) le message M ?

L'algorithme est conçu pour fonctionner par bloc de données de taille de 512 bits. Le bourrage est là pour compléter les messages dont la taille n'est pas un multiple de 512 bits. De plus, il permet d'introduire un peu d'aléa pour réduire les risques de collisions (deux messages différents ayant la même valeur de hachage).

- Calculer la valeur de k pour le message "Hello World !" (codage ASCII étendu)

Message : "Hello World !" Longueur du message $l = 8 \times 12 = 96$ *bits* (un caractère ASCII = 8 bits) Nombre de zéro $k = 448 - (l + 1) = 448 - 97 = 351$ *bits* Longueur du message + bourrage : $96 + 1 + 351 + 64 = 512$

- Expliciter l'objectif et le fonctionnement de la fonction de compression.

L'idée est d'avoir une empreinte du message, c'est-à-dire une valeur représentative de tout le message de telle sorte que si le message change l'empreinte calculée précédemment n'est plus valable. Pour cela, on va « compresser » les informations qui représentent le message dans le sens où chaque information élémentaire participe au calcul de l'empreinte. On a dont donc bien une sorte de compression (bien que l'opération de décompression n'existe pas).

Un bloc de données de 512 bits est traité par la fonction de compression. La fonction de compression a deux entrées : le bloc de données et soit le résultat de la compression précédente soit un vecteur d'initialisation pour le premier bloc. La fonction a une sortie : une valeur de hachage sur 256 bits qui est pris compte comme entrée de l'application suivante de la fonction.

Les mots de 32 bits A, B, C, D, E et F sont initialement fixés par le vecteur *IV* pour l'application de la fonction sur le premier bloc. Pour les applications suivantes, ces mots proviennent de l'application précédente. La valeur de sortie de ces mots (une valeur de hachage) est calculée par un ensemble d'opérations binaires sur ces valeurs elles-mêmes et sur les données.

Les données sont pris en compte différemment selon la valeur du tour t :
- si $0 \leq t \leq 15$ alors $W_t = M_t^{(i)}$
- si $16 \leq t \leq 63$ alors $W_t = \sigma_0^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16}$

Quand $0 \leq t \leq 15$ les 512 bits de données sont pris en compte directement ($512 = 16 \times 32$).
Quand $16 \leq t \leq 63$ les données sont de nouveau prises en compte via la somme calculée.

K_t sont des constantes dont la valeur change selon la valeur de t.

Exercice n° 3

(a) Etudiez l'implémentation des fonctions suivantes :
- fonction décalage binaire à gauche $ROTR^n(x)$: $(x >> n) \vee (x << (32 - n))$
- fonction décalage binaire à droite $SHR^n(x)$: $x >> n$
- fonction $Ch(x, y, z)$: $(x \wedge y) \oplus (\neg x \wedge y)$
- fonction $Maj(x, y, z)$: $(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
- fonction $\sum_0^{\{256\}}(x)$: $ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$
- fonction $\sum_1^{\{256\}}(x)$: $ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$
- fonction $\sigma_0^{\{256\}}(x)$: $ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$
- fonction $\sigma_1^{\{256\}}(x)$: $ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$

D'après vous, pourquoi sont-elles définies comme macros ?

Ces fonctions sont définies comme macro pour une question de performance. En effet, lorsque le compilateur rencontre l'utilisation d'une fonction macro dans le code, il remplace directement l'appel de la fonction macro par son code. Ceci permet de ganger du temps en évitant l'appel d'une méthode (empilement et dépilement des paramètres et des valeurs de retour).

(b) Etudiez l'implémentation de la fonction de compression *sha256_compress*.

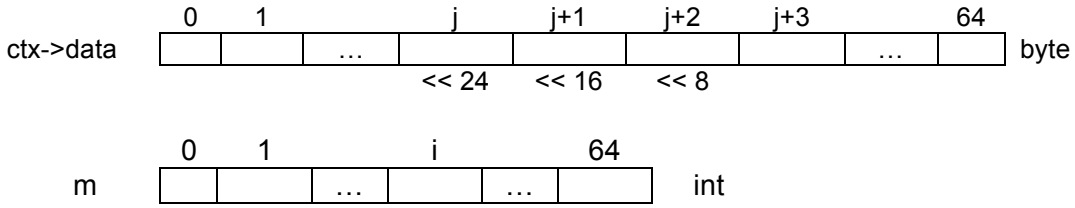
Comment est implémentée la boucle *Pour i = 1 à N* de l'algorithme ?

Cette boucle n'est pas implémentée dans cette fonction. Elle est implémentée dans la fonction *sha256_compute*. La fonction *sha256_compress* n'implémente que le calcul d'une valeur de hachage pour un bloc de 512 bits (c'est-à-dire la fonction de compression).

Comment est calculée la valeur W_t ?

Les valeurs de Wt sont stockées dans le tableau *WORD m[64]* est sont liées aux données.

Pout $0 \leq i \leq 15$, m[i] est le résultat d'un OU binaire de quatre cases adjacentes *ctx->data[j]* à *ctx->data[j+3]* décalées à droite de la manière suivante :



Quatre cases de 8 bits décalées forment un mot de 32 bits (8 x 4) mis dans m[i].

Pour $16 \leq i \leq 63$, m[i] est le résultat de somme définie dans l'algorithme.

(c) Etudiez la fonction *sha256_compute.*

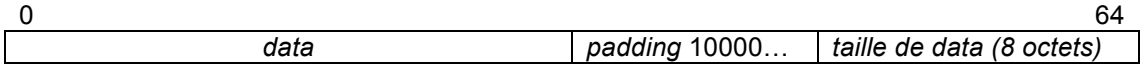
Comment se fait le découpage des données en bloc de 512 bits ?

Le découpage se fait dans la première boucle effectuée sur la taille des données : les données sont copiées dans le tableau *ctx->data* interne à la structure *ctx*. Dès que la longueur atteint les 64 octets (soit 512 bits), on fait une compression.

Lorsque l'on sort de la boucle, soit tous les blocs de 512 bits ont été « compressés », soit il reste des données mais la taille est < à 512 bits.

Expliquer comment le bourrage a été implémenté.

La fonction de compression ne fonctionne que sur des blocs de 512 bits. Lorsqu'un bloc n'est pas complet, il faut le compléter par du bourrage mais en tenant compte du fait qu'il faut réserver 64 bits (8 octets) pour ajouter en fin de bloc la taille des données en bits.



Par conséquent, sur les 64 octets d'un bloc, 8 octets sont réservés pour indiquer la taille des données. Il reste donc 56 octets pour les données et le padding.

Si la taille des données est < 56, alors il y a suffisamment de place.

Si la taille des données est > 56, il n'y aura pas assez de place pour indiquer la taille. Dans ce cas, il faut débuter le *padding* dans le bloc en cours, faire une compression sur ce bloc, puis ajouter un bloc « vide » constitué essentiellement de la suite du *padding* et de la taille.
Une foi la taille ajoutée (expansion de *ctx->bitlen*), il faut compresser ce dernier bloc « vide ».

L'expansion de *ctx->bitlen* est obligatoire puisqu'il est codé sur 64 bits. Il est étendu en 8 octets (*ctx->data[56]* à *ctx->data[63*]).

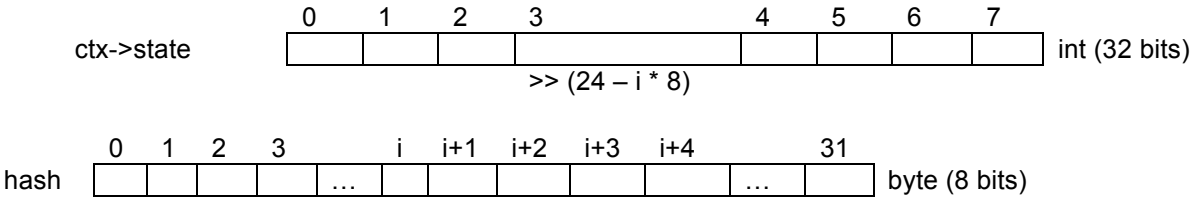
(d) Etudiez la fonction *sha256_convert*.

Quelle est l'utilité de cette fonction ?

La réponse attendue doit être donnée sous forme de tableau de 32 octets (256 bits). La valeur de hachage stockée dans un tableau de 8 entiers *int* (256 bits) doit donc être convertie.

Donnez un exemple de fonctionnement.

Le tableau de 8 x 32 bits (256) doit être converti dans un tableau de 32 x 8 bits (256).



Chaque case/valeur de ctx->state est découpée en 4 parties. Les bits de poids forts sont mis en premier et ainsi de suite sur 4 cases adjacentes du tableau hash.