M2 IF Apprentissage projet composant

Spécifications composant 4

Hacheur AES

|  |
| --- |
| **Groupe B** |
| Sahar HERWAL |
| Faouz ESSERHIR EL FASSI |
| Othmane NCIRI |
| Alexandre PATRY |

|  |  |  |  |
| --- | --- | --- | --- |
| Version doc | Date | Auteur(s) | Modifications |
| 0.0 | 27/01/2016 | Groupe B | Version initiale |
| 0.5 | 12/02/2016 | Groupe B | Définition du fonctionnement général |
| 0.8 | 24/02/2016 | Groupe B | Spécification du composant |
| 1.0 | 02/03/2016 | Groupe B | Description des algorithmes |

Objectifs :

Dans ce document nous allons spécifier le composant 4 : Hacheur AES.  Pour ce faire, nous allons dans un premier temps décrire le fonctionnement du composant, puis nous allons définir et normaliser ses entrées et sorties puis nous décrirons le processus d’interaction avec les autres composants et nous finirons par la présentation d’un plan de test du composant.

**I - Fonctionnement du composant**

Un mécanisme innovant permet de construire un historique des transactions. Ce mécanisme se base sur la connaissance de l’arborescence des blocs identifiés par leur hash. Chaque bloc à une référence d’un bloc plus ancien.

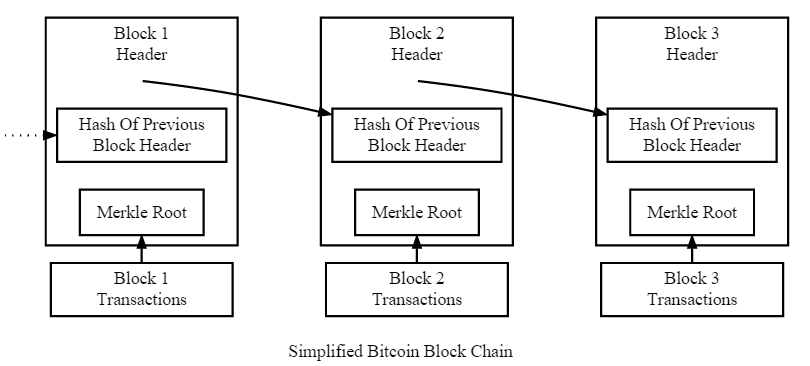


Figure : Block Chain simplifié, source Bitcoin Dev’s Guide

  Ce mode de fonctionnement permet d’être plus robuste qu’un « timestamp » stocké dans un bloc qui pourrait être modifié par un acteur malveillant du réseau.

Le module Hacheur AES à deux fonctionnalités :

* un générateur de hash,
* vérificateur de hash.

Le générateur permet de calculer et retourner un haché à partir d’un bloc envoyé par le Miner (composant 3). Le vérificateur renvoie un booléen à partir de la vérification du bloc et de son hash passés en paramètres.

**II - Générateur de digest**

Fonctionnement, exemple sur une chaîne de caractères :

**II.1 - Cas d’un message de longueur inférieur à 512bits**

Soit le message à hacher “*ProjetComposants*”, sa représentation ASCII en binaire correspond à :

*0000000001010000011100100110111101101010011001010111010001000011011011110110110101110000011011110111001101100001011011100111010001110011*

 L’algorithme de chiffrement que l’on va utiliser est Secure Hash Algorithm - SHA 1. Ce dernier manipule des blocs de 512 bits.

Les 64 derniers bits sont réservés à la longueur du message. Il faut donc ajouter à notre message de 136 bits suffisamment de bits pour atteindre 448 bits, soit 312.

On concatène à droite un ‘1’ et 311 ‘0’ et on ajoute la longueur ici 16 =² 0001 0000, on obtient :

*0000000001010000011100100110111101101010011001010111010001000011011011110110110101110000011011110111001101100001011011100111010001110011100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010000*

**II.2 - Cas général**

 Si les messages sont plus longs que 512 bits, ils doivent être découpés en bloc de 512 bits. Voici la spécification de l’algorithme de hachage :

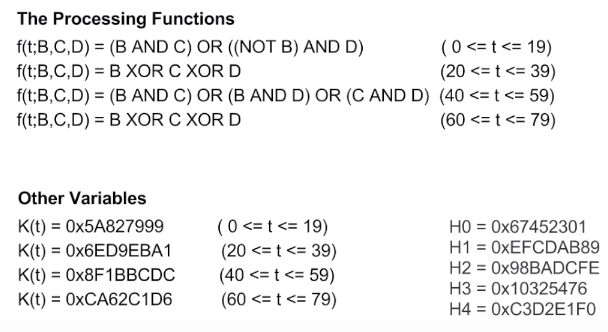


Figure : Algorithme de hachage

**Algorithme :**

*// Soit M[1,2,...,N] le message décomposé en N blocs de 512 bits.*

*SHAOne(Bloc b)*

*// Initialisation : Découper b en blocs de 512 bits et définir l'opération I.1*

*Pour tout les blocs de 512 bits :*

*Diviser M[k] (k 1,..,N) en 16 mots (w0 à w15)*

*Construire les mots w16 à w79, pour chaque mot t de 16 à 79 :*

*// opérations algébrique et décalage à gauche ( <<<)*

*W(t) = (W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16)) <<< 1*

*Définir a, b, c, d, e, variables initialement égales à H0, H1, H2, H3, H4, respectivement.*

*Pour t de 0 à 79 :*

*tmp = a <<< 5 + f(t;b;c;d) + e + w(t) + K(t);*

*e = d;*

*d = c;*

*c = b <<< 30;*

*b = a;*

*a = TEMP;*

*RETOURNER*

*H0 = H0 + a;*

*H1 = H1 + b;*

*H2 = H2 + c;*

*H3 = H3 + d;*

*H4 = H4 + e;*

On obtient avec cet algorithme le digest (concaténation de H0 à H4) :

*fce124600e7d888c1ca51d42de0cad8a603003b8*, avec le message “*ProjetComposants*” spécifié en 1.

**III - Vérificateur de bloc**

Le vérificateur reçoit un bloc et son digest, l’algorithme défini en 2.2 est utilisé pour calculer le digest du bloc. Ce résultat est comparé au digest passé en paramètre.

**Algorithme:**

VerificateurBloc(Bloc b, Digest d)

h = SHAOne(b)

si h = d

RETOURNER "vrai";

sinon

RETOURNER "faux";

**IV - Description des erreurs**

Chaque fonctionnalité du composant doit gérer ces cas d’erreurs :

**IV.1 - Fonction SHAOne de génération de digest**

|  |  |
| --- | --- |
| Nom Fonction : SHAOne (Bloc bloc) | |
| Erreur : données corrompue | Message : Malformed block |
| Erreur : longueur de message nulle | Message : Incorrect length |

**IV.2 - Fonction verificateurBloc qui retourne si oui ou non le hashé du bloc correspond au digest passé en paramètre**

|  |  |
| --- | --- |
| Nom Fonction : verificateurBloc(Bloc bloc, Bytes[] digest ) | |
| Erreur : données corrompue | Message : Malformed block |
| Erreur : digest.length != 160 bits | Message : Malformed digest |

**V - Plan de tests**

**V.1 - Interaction vis-à-vis des autres composants**

Les composants 1, 6 et 7 sont créés dans une version simplifiée reprenant les fonctions essentielles. Ces composants renvoient des valeurs qui nous permettent de tester les différents scénarios.

**V.2 -  Test du composant**

Il faut tester la fonction de hashage en utilisant une distribution similaire que celle sur laquelle on souhaite travailler. Toutefois, si on génère des données d’entrées, on doit être facilement être en mesure de créer un test unitaire qui génère une quantité importante de celui-ci et ensuite vérifier que la propagation correspond bien à la définition.