M2 IF Apprentissage projet composants

Spécifications composant 5

Vérificateur de bloc

|  |
| --- |
| **Groupe E** |
| Bastien ALBERT Etudiant1 |
| Sven DUMAY Etudiant2 |
| Qian ZHANG Etudiant3 |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| Version doc | Date | Auteur(s) | Modifications |
| 1.0 | 27/01/2015 | Jose Luu | Version initiale |
| 1.1 | 27/01/2015 | Jose Luu | Modification pour exemple |
| 1.2 | 18/02/2016 | Groupe E | Création des spécifications du composant  « Vérificateur de bloc » |
| 1.3 | 26/02/2016 | Groupe E | Mise à jour des fonctions et erreurs |
| 1.4 | 28/02/2016 | Groupe E | Mise à jour des erreurs |
| 1.5 | 07/03/2016 | Groupe E | Mise à jour des erreurs et Plan de tests |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Ce document a pour but de rassembler les spécifications concernant le composant 5 « Vérificateur de bloc » du projet composant.

1. **Fonction du composant**

La « blockchain » est le concept de base du Bitcoin. Il s’agit d’un ensemble de blocs appelé également « chaine de blocs ». Son fonctionnement est le suivant.

À chaque fois qu’un utilisateur utilise une technologie basée sur la « blockchain », les transactions (date, le compte débiteur, le compte créditeur et le montant de la transaction, etc…) qu’il réalise sont stockées sur une plateforme décentralisée, puisque cette technologie est basé sur le modèle « pair à pair » (P2P).

Plusieurs de ces transactions constituent un “bloc” et l’ensemble des blocs constituent la “blockchain”.

Après avoir stocké l’ensemble des données récentes, le bloc s’intègre à la chaîne globale. Dès lors, les informations n’ont plus la possibilité d’être effacées, car chaque bloc garde une trace des transactions et se lie à la suite des autres en respectant un ordre chronologique.

Bloc1

Transaction 1,2

Bloc2

Transaction 3,4,5

BlocN-1

Transaction …

BlocN

Transaction 100,101,102

**Futures transactions**

**Transactions fixées**

Transaction1

Transaction2

Transaction3

Transaction4

Transaction5

Les fonctions du composant « Vérificateur de bloc » correspondent à la vérification de la composition du chaque bloc et la vérification des transactions.

C’est un moyen de s’assurer que les informations représentées par la blockchain sont de qualité.

Ce composant est en interactivité avec d’autres composants (1 ; 4 et 6).

Voici un schéma incluant les composants connexes :

Composant5

Composant4

Composant1

Composant6

Vérification de bloc

Recherche de bloc

Vérification de hash

Vérification de clé

Retour de vérification

Retour de vérification

Dans la suite du document, chacune des étapes du schéma sont explicitées afin de mieux comprendre le fonctionnement du composant et son mode de réalisation.

Vérification de transaction

1. **Les étapes**
   1. **Réception des blocs du composant 1 pour faire la vérification**

Notre composant a besoin d’un bloc en entrée pour pouvoir effectuer des opérations de vérification.

La fonction blockVerify suivante traite le bloc en entrée :

**boolean blockVerify(String bloc, String chainAES)**

La fonction blockVerify s’assure que toutes les données du bloc sont cohérentes :

* le bloc doit commencer par un hash (celui du bloc précédent)
* la deuxième information doit être un numéro (celui du bloc)
* la troisième information doit être un nonce (complément)
* en quatrième position se trouvent les transactions
* pour terminer, le gain du mineur (transac, sans provenance)

Une fois la vérification effectuée, il faut valider le bloc. C’est le rôle du composant 4. Il faut envoyer au composant 4 le bloc vérifié par notre composant ainsi que la chaine AES.

Notre composant a également pour rôle de vérifier les transactions. C’est l’objectif de la fonction transfertVerify suivante :

**boolean transfertVerify(String transaction, String chainAES)**

La fonction transfertVerify vérifie la transaction :

* les entrées UTXI doivent être valides
  + le nombre d’entrée doit être valide
  + une entrée doit être : le numéro de bloc, le numéro UTXO et la signature UTXO

A partir des UTXI, la fonction doit retrouver les UTXO correspondantes.

Pour chaque entrée distincte, le composant 1 nous fournit le bloc correspondant. Avec le numéro d’UTXO que nous possédons, nous pouvons récupérer l’UTXO du bloc.

Il faut vérifier que l’UTXO n’a pas déjà été référencée par une UTXI dans un bloc de numéro inférieur.

Il faut ensuite appeler le composant 6 afin de faire valider la signature accompagnant l’UTXI grâce à la clé publique de UTXO.

Si la signature est bonne, nous devons maintenant la décrypter et si on retrouve le hash contenu de la UTXO, alors la transaction est valide.

* 1. **Appel du composant 6**

Notre module appelle le composant 6 pour valider tous les signatures de bloc, la fonction booléenne valideSignature suivante servi à traiter cette vérification :

**bool valideSignature( String bloc, String clePub, String sign)**

* 1. **Appel du composant 4**

Notre module appelle le composant 4 pour valider la chaine AES de bloc, la fonction booléenne valideHash suivante servi à traiter cette vérification :

**bool valideHash( String bloc, String chainAES)**

1. **Description des erreurs**
2. **boolean blockVerify(Block bloc, String chainAES)**

Cette fonction a pour objectif de vérifier si le bloc est au bon format, les seules erreurs possibles sont donc celles qui sont directement liées au contenu du bloc :

* Mauvais format du bloc
* Mauvais format des données dans le bloc
* Problème au niveau du hash

1. **boolean transfertVerify(String bloc, String chainAES)**

La fonction transfertVerify a pour objectif de vérifier que la transaction est au bon format.

Les erreurs possibles sont :

* mauvais format de la transaction
* mauvais format des données dans le bloc
* problème au niveau du hash

1. **Plan de tests**

Afin de pouvoir tester ce composant, nous feront appel à un « main » qui appellera plusieurs les différentes fonctions du composant afin de tester les cas passants ainsi que les cas avec erreurs.

Afin de pouvoir mener à bien ces tests, il faudra utiliser une version simplifiée des composants 1, 4 et 6 afin de simuler la demande de bloc, le hash et la clé.

Cela permettra de vérifier si les sorties du composant sont bien équivalentes aux sorties qui étaient attendu (cas passant sans erreur, les bonnes erreurs remontées pour chaque cas avec erreur).

**bool blocVerify(Block bloc, String chainAES)**

Si une des valeurs (bloc, chainAES) est nulle alors le message d’erreur suivant doit être renvoyé :

« NULL VALUE » ;

Si une donnée manque dans le tableau alors le message d’erreur suivant doit être :

« MISSED VALUE » ;

Si une donnée n’a pas le bon format, le message d’erreur suivant doit être envoyé :

« INVALIDE TYPE OF DATA » ;

**bool transfertVerify(Block bloc, String chainAES)**

Si une des valeurs (bloc, chainAES) est nulle alors le message d’erreur suivant doit être renvoyé :

« NULL VALUE » ;

Si une donnée manque dans le tableau alors le message d’erreur suivant doit être :

« MISSED VALUE » ;

Si une donnée n’a pas le bon format, le message d’erreur suivant doit être envoyé :

« INVALID TYPE OF DATA » ;