M2 IF Apprentissage projet composant

Spécifications composant 3

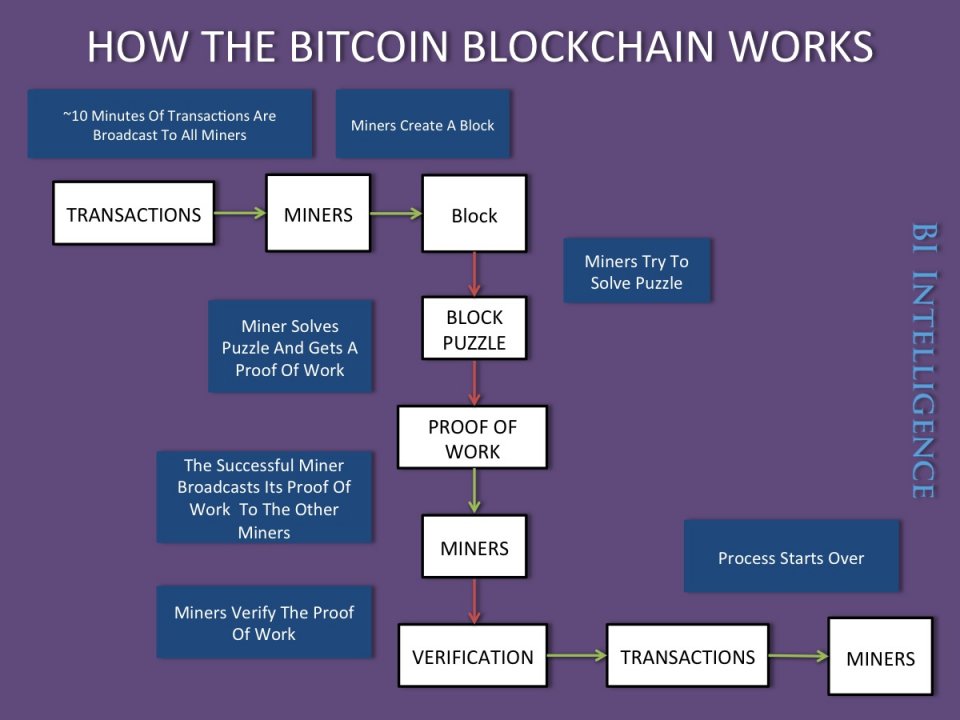
Blockchain Bitcoin

|  |
| --- |
| **Groupe C** |
| Dorra JRAD |
| Yassine BEN AMMAR |
| Loubna LARAJE |
| Anna LEE |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | 29/02/2016 | Groupe C | Création des spécifications du composant 3« mineur» |
| 1.3 |  | Groupe C | Mise à jour des fonctions et erreurs |
| 1.4 |  | Groupe C | Mise à jour des erreurs |
| 1.5 |  | Groupe C | Mise à jour des erreurs et Plan de tests |
|  |  |  |  |
|  |  |  |  |

Ce document a pour but de rassembler les spécifications concernant le composant 3 « Mineur» du projet de classe.

**1- Blockchain Bitcoin: Comment ça marche?**



2- Fonctionnement du composant 3

 Les mineurs sont des entités dont la fonction est d’alimenter le réseau en puissance de calcul, afin de permettre la mise à jour de la base de données décentralisée. Pour mettre à jour la base de données, les mineurs doivent confirmer les nouveaux blocs en décryptant les données.

Une concurrence existe entre les mineurs pour le décryptage des transactions, permettant à la puissance disponible sur le réseau de croître. N’importe qui peut prêter sa puissance de calcul pour miner, mais plus les mineurs sont nombreux plus la résolution des preuves est difficile à s’attribuer. Ainsi, le protocole peut devenir quasi-inviolable dès lors que la concurrence est forte à chaque nœud du réseau c'est-à-dire qu’aucun groupement de mineurs ne devient majoritaire.

Les fonctions du mineurs sont:

* Reçoit une difficulté (nombre de zéros)
* Un bloc
* Appelle le hacheur AES
* Incrémente le nonce
* Sortie: bloc hash

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.

1. Les étapes

1.1 Composant 1

Reçoit une difficulté (nombre de zéros)

1.2 Composant 4

* 1. Réception du composant 1

Notre module reçoit du composant “XLL interface” un certain nombre de paramètres :

* La difficulté de minage (Nombre de zéro)
  1. Géneration des nonces

Pour être rémunéré, un mineur doit trouver un bloc. Pour ce faire, il doit exécuter l’algorithme de nombreuses fois jusqu’à ce qu’il tombe sur un résultat qui montre de façon incontestable qu’il a bien trouvé un bloc.

***2.2.1 Appel du hashage***

Les mineurs effectuent des hashs cryptographiques SHA-256(une chaîne de caractères pseudo-aléatoire) Cette chaîne de caractères correspond à l’identifiant du bloc qu’ils essayent de trouver.

double CallHashage () ;

***2.2.2 Incrémentation du nonce***

Pour qu’un mineur trouve un bloc, il faut que le hash de ce bloc commence par un certain nombre de zéros. Plus la difficulté est forte, plus le nombre de zéros par lequel le hash doit commencer est grand.

Les mineurs utilisent l’identifiant du bloc, y ajoutent un incrément, puis appellent de nombreuses fois le hashage en augmentant l’incrément à chaque fois. Ils font cela jusqu’à ce qu’ils trouvent un résultat qui commence par le bon nombre de zéros

double incrementNonce(double identifiant) ;

***2.2.3 Sortie bloc hash***

Une fois les mineurs trouvent un hash inférieur à la difficulté cible ils envoient leur découverte aux autres nœuds du réseau.

Double getNonce ();

1. Description des erreurs

Pour le composant, chacune des erreurs est gérée par les exceptions (throws) suivantes :

* 1. Etape 2.1

|  |  |
| --- | --- |
| Nom Fonction : CallHashage *()* | |
| *Erreur :* données négatives | *Negative value* |
| *Erreur :* données manquantes | *Missed data* |

* 1. Etape 2.2

***4.3.1 Etape 2.2.1***

|  |  |
| --- | --- |
| Nom Fonction : incrementNonce *(double maturity, double spot )* | |
| *Erreur :* données négatives | *Negative value* |
| *Erreur :* données manquantes | *Missed data* |

*4.3.2 Etape 2.2.2*

|  |  |
| --- | --- |
| Nom Fonction :getNonce *()* | |
| *Erreur :* données négatives | *Negative value* |
| *Erreur :* données manquantes | *Missed data* |
| *Erreur :* résultat non cohérent avec  le bloc précédent | *Non consistent result* |

1. Plan de tests
   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.

* 1. Test du composant

Un main fait appel aux fonctions des différents composants et compare le comportement du composant avec celui attendu. Si le composant n’a pas le comportement souhaité pour ce cas de test, il y aura un message nous indiquant l’erreur. Si tout est correct, il n’y aura donc pas de message affiché d’erreur affiché.

NB : Pour les trois fonctions listées ci-dessous, il est possible qu’une exception soit retournée alors qu’elle n’était pas attendue. Dans ce cas, le message d’erreur suivant apparaitra :  
"FAILED: ‘XXX’ EXCEPTION TRIGGERED WHEREAS DATA ARE CORRECT";  
XXX : Name of the exception

* CallHashage:

Si une des valeurs est négative alors le message d’erreur suivant doit être renvoyé :   
"NEGATIVE VALUE";

Si une donnée manque dans le tableau alors le message d’erreur suivant doit être renvoyé :   
"MISSED DATA";

* incrementNonce :

Si une des valeurs (maturity, nombre de test, strike) est négative alors le message d’erreur suivant doit être renvoyé :   
"NEGATIVE VALUE EXCEPTION UNTRIGGERED ";

Si une donnée n’a pas le bon type, le message d’erreur suivant doit être renvoyé :   
"WRONG DATA EXCEPTION UNTRIGGERED ";

* getNonce :

Si une donnée manque dans le tableau alors le message d’erreur suivant doit être renvoyé :   
"MISSING DATA EXCEPTION UNTRIGGERERD ";

Si une des valeurs est négative alors le message d’erreur suivant doit être renvoyé :   
"NEGATIVE VALUE EXCEPTION UNTRIGGERED ";

Si une valeur ressorti est trop importante, le message d’erreur suivant doit être renvoyé :   
"FAILED : VALUE > 1.10^6 EXCEPTION UNTRIGGERERD ";