Document de spécifications du composant 3

Auteurs: LEPERS Antoine, XU Victor, MEGHARA Lyes, LAKHDARI Adlane Badr

HISTORIQUE DES VERSIONS

Dernière modification le :	Par :	Objet de la modification :	Version
18.02.2019	Antoine LEPERS	Création du document	1.0
20.03.2019	Antoine LEPERS	Première version	1.1
06.04.2019	Antoine LEPERS	Change composant 4, améliore la mise en forme, et met à jour la description du tester	1.2

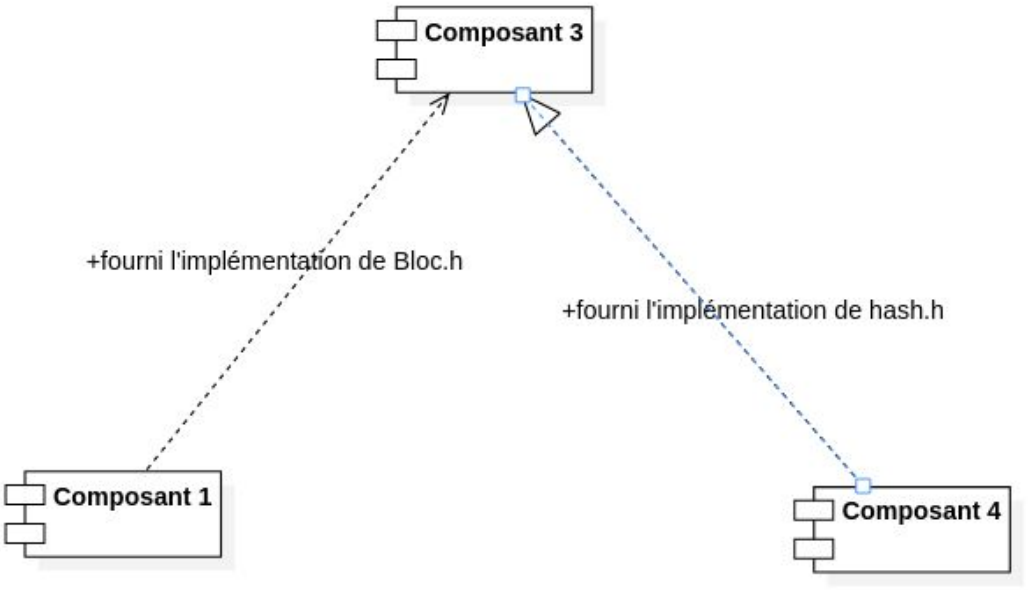
Contexte

La sécurité de la plupart des cryptomonnaies repose en grande partie sur des hashs. Le hash de chaque bloc de la blockchain doit contenir un certain nombre de zéros. Et comme les blocs sont liés (d'où le nom de blockchain), le hash de chaque bloc dépend du bloc précédent. Ainsi altérer un bloc est quasiment impossible car cela implique de recalculer tous les blocs suivant le bloc altéré pour que leurs hashs gardent le bon nombre de zéros.

Nous allons ici décrire le fonctionnement du composant 3. C'est le mineur. Il prend en entré un bloc et une nombre de zéros et renvoie le bloc en ayant modifié le nonce pour que le hash comporte le bon nombre de zéros.

Ce composant fait appel à deux autres composants. Le composant 1 qui définit notamment la classe Bloc. Et le composant 4 qui définit la fonction hash.

Schéma des interactions avec les composants annexes



Résumé des headers

Bloc.h (composant 1)

Défini la classe Bloc de M. Luu.

```
ToString()
```

Traduit un bloc en string. Ce string sera ensuite utilisé par le Hacheur

Hacheur.h (composant 4)

```
std::string Hacheur::hacher(std::string input)
```

Prend en entrée le string d'un bloc et renvoie son hash. Cette fonction est contenue dans la classe Hacheur, il faut donc initialiser une instance de Hacheur pour pouvoir l'utiliser.

Mineur.h (Composant 4)

```
Bloc miner(unsigned int difficulty, Bloc b)
```

Fait une boucle jusqu'à trouver un hash terminant par un nombre de zéros supérieur au paramètre difficulty. Le paramètre difficulty utilisé par le tester est 3, il permet une recherche de hash prenant quelques secondes.

Tester

Le composant 1 n'étant pas fonctionnel, nous utilisons *DummyBloc.cpp* pour implémenter la fonction ToString() et le constructeur d'un bloc.

Le tester créé un bloc vide à l'aide de *DummyBloc.cpp*, il appelle ensuite le composant 3 en lui donnant le bloc nouvellement créé et une difficulté de 3 en paramètre.