Document de spécification du “composant 4”

Auteurs :

- Victor Fritz
- Mickaël Peeren
- Quentin Sauvage

Historique des versions :

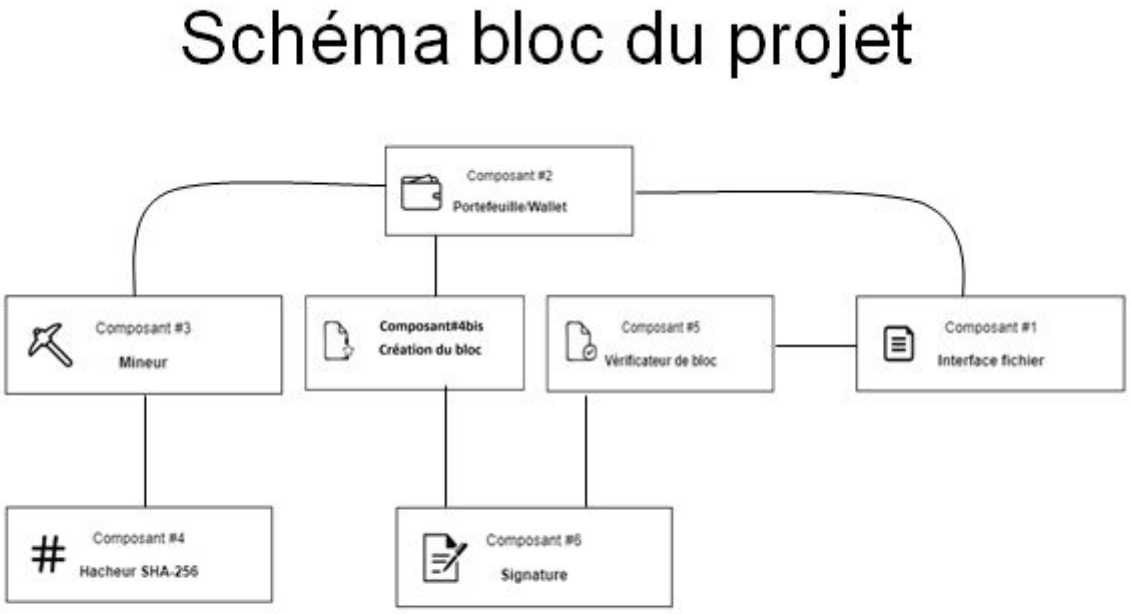
Date	Numéro de version	Modification(s)	Auteur(s)
20/03/2020	1.0.0	Création	Sauvage
05/04/2020	1.0.1	Contexte	Fritz et Sauvage
10/04/2020	1.0.2	Objets et interfaces	Fritz, Peeren et Sauvage
15/04/2020	1.0.3	Ajout schéma bloc	Peeren
26/04/2020	1.0.4	Ajout des négociations intergroupe	Peeren
26/04/2020	1.0.5	Mise a jour des interfaces	Peeren

I. Description du composant

A. Contexte et objectif de l'outil

Ce composant a pour but de créer un hash, à partir d'un bloc d'une blockchain. Il peut également vérifier la validité d'un hash et du bloc qui lui est associé.

B. Schéma bloc incluant les composants connexes



C. Objets Spécifiques du composant

L'objet Bloc est un objet créé par notre composant et sera décrit de la façon suivante

```
class Bloc
{
 public:
    char hash[HASH_SIZE]; // hash des autres champs, hash of the entire bloc
    unsigned int nonce;

    char previous_hash[HASH_SIZE]; // hash of the previous bloc
    int num; // numero du bloc, commence a zero
    TX tx1; //  transaction du bloc
    TXM tx0; // transaction du mineur (coinbase)

    std::string toString(); // a implementer par le groupe "Fichier"
    std::string toStringSansHash(); // a implementer par le groupe "Fichier"
};
```

D. Interface et interactions avec chaque autre composant

Notre composant ne présente qu'une seule interface avec le composant n°3 : le mineur.

Les interactions qu'il effectue avec celui-ci sont la réception d'un bloc de sa part ainsi que l'envoi de la chaîne SHA créée par le "hacheur SHA-256" au composant "mineur.

E. Interfaces Python

public **string** **hashBloc**(**Bloc** bloc);

hashBloc est une méthode qui reçoit un objet de type Bloc et qui va construire la chaîne de caractère correspondante en SHA-256. Elle retourne cette chaîne.

public **bool** **verifHash**(**Bloc** bloc, **string** hashCode);

verifHash est une méthode qui permet de vérifier si une chaine de caractère en SHA-256 passée en paramètre (hashCode) correspond bien au bloc passé en paramètre (bloc). Elle encode le bloc en SHA-256 puis compare les 2 chaînes. La méthode renvoie **True** si les deux sont identiques, **False** sinon.

public **Bloc** **buildBloc**(TX transaction);

buildBloc est une méthode qui permet d'obtenir un objet de type Bloc.

public Bloc **buildBlocFull**(char hash[], unsigned int nonce, int num, char prevHash[], TX transaction, TXM trasnsactionMineur);

buildBlocFull est une méthode qui permet d'obtenir un objet de type Bloc.

F. Négociations avec les autres composants

Composant 2 (wallet)

La fonction qui sera utilisée sera la fonction buildBloc() qui prend en paramètre une transaction. Ces transactions seront signées. A partir de cela, la fonction construira le hash des UTXO de l'objet TX passé en paramètre, récupèrera le hash du bloc précédent et retournera le bloc nouvellement créé.

Composant 3 (minage)

Le minage a besoin de la fonction de hachage, ils utiliserons la fonction hashBloc() qui prend en paramètre un bloc et qui leur retourne le hash associé.

Composant 1 (fichier)

Le composant 1 a besoin de construire le bloc 0, pour cela la fonction buildBlocFull() leur est fournie afin de créer un bloc par défaut. Ils peuvent transmettre tout, une partie, ou aucun paramètre afin de créer le bloc de départ.

Composant 5 (vérification de bloc)

Afin de garantir l'intégrité de la blockchain, il y a besoin d'une méthode de vérification verifBloc() avec en paramètre un bloc, et une chaîne SHA256 qui renvoie vrai si le hash correspond au bloc, et faux sinon.