TP 6 : Blockchain

Ludovic Pfeiffer
Axel Chollet
Noria Foukia

Description :

La blockchain est une technologie de stockage et de transmission d'informations, transparente, sécurisée, et fonctionnant sans organe central de contrôle (définition de Blockchain France)[1].

Par extension, une blockchain constitue une base de données qui contient l'historique de tous les échanges effectués entre ses utilisateurs depuis sa création. Cette base de données est sécurisée et distribuée : elle est partagée par ses différents utilisateurs, sans intermédiaire, ce qui permet à chacun de vérifier la validité de la chaîne.

Une blockchain utilise un algorithme de consensus. Plusieurs de ces algorithme ont été mis au point. Les plus prédominants sont les algorithmes Proof of Work (PoW) et Proof of Stake (PoS). Dans ce TP, on s'intéressera à l'algorithme PoW.

1. Calcul d'un hash SHA-256

Le système de hash est à la base de la blockchain. C'est pourquoi il est important d'avoir une méthode dont on est sûr qu'elle ne posera pas de problème par la suite.

Créer une fonction qui permet d'effectuer un hash SHA-256. La librairie Python hashlib est très utile pour répondre à ces attentes. (aller voir ce lien et le lire pour une explication simple de SHA-256 : https://assiste.com/SHA-256.html)

2. Effectuer un hash qui respecte une difficulté

La notion de difficulté dans une blockchain est utilisée uniquement lorsque la blockchain utilise l'algorithme Proof of Work (PoW). Cet algorithme est basé sur un système de puzzles que la machine va devoir résoudre avant de pouvoir envoyer les informations. Le PoW est le modèle principal qui constitue la blockchain Bitcoin. Dans la création de la blockchain, des blocs sont minés par des entités appelées mineurs. Leur rôle est de vérifier et valider les blocs successifs créant la blockchain. Le PoW fondé sur la difficulté est utilisé pour choisir le mineur qui aura le droit d'émettre le prochain bloc de la blockchain en fournissant un travail (work) suffisant en terme de ressource.

Une difficulté peut être définie par un pattern à reproduire. Dans le cadre de l'algorithme PoW, la difficulté constitue le nombre de zéros que l'on trouve en début de hash.

Par exemple, prenons le mot bonjour que l'on va hacher en SHA-256. Le résultat sera « 2cb4b1431b84ec15d35ed83bb927e27e8967d75f4bcd9cc4b25c8d879ae23e18 ». Pour respecter une difficulté de 4, cet hash devrait commencer par 4 zéros. On va donc choisir de rajouter un nombre aléatoire à la fin du mot à hacher que l'on nomme « Nonce ». Tant que le hash ne respecte pas la difficulté, ce nombre sera changé (incrémenté par exemple).

En reprenant notre exemple de bonjour, si on ajoute 479 à la fin (donc « bonjour479 »), le hash deviendra « 0000dbb7e10433cca4235022130e29d6e35fe37eef9c5cb5631f54e0507ed8b7 ». Le hash commence bien par 4 zéros, il respecte la régle demandée en terme de difficulté.

[1] https://blockchainfrance.net/

En utilisant la fonction de hachage créée précédemment, mettre au point un système Proof of Work qui permet de respecter une difficulté.

Quel est le nonce et le hash en difficulté 4 pour le mot « blockchain » ? Même question pour le mot « sécurité ».

3. Calcul du temps selon la difficulté

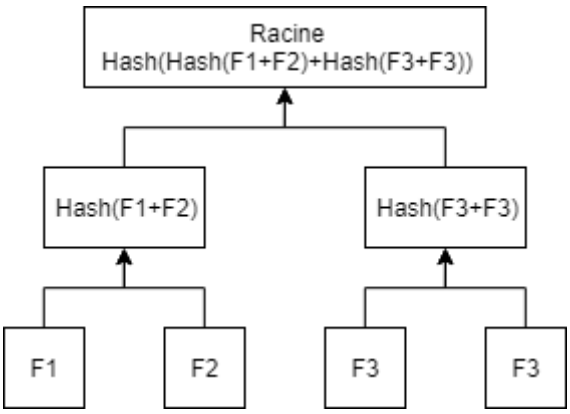
Utilisez une fonction permettant de chronométrer le temps d'exécution afin de pouvoir mesurer le temps selon la difficulté.
Représenter graphiquement les temps obtenus en fonction de la valeur croissante de la difficulté sur le hash du mot « blockchain ».
Que remarquez-vous ? Expliquez pourquoi

4. Calcul de la racine d'un arbre de Merkle

Un arbre de Merkle[2] est un arbre qui permet de regrouper plusieurs hashs ensemble sous la forme d'un hash unique. Cet hash unique est appelé la racine de l'arbre.
Le fonctionnement d'un arbre de Merkle est le suivant :

- On regroupe tous les hashs à notre disposition et on les met ensemble 2 à 2. Si le nombre est impair, le dernier hash est dédoublé.
- Ces paires de hashs sont ensuite hachées ensemble, par exemple hash(f1 + f2).
- Le résultat de ces hashs de paires sont également mis ensemble 2 à 2.
- On recommence ces étapes jusqu'à ce que l'on se retrouve avec un hash unique à la fin.

Un arbre de Merkle avec 3 hashs aurait la forme suivante :



Ecrire une fonction qui permette de réaliser un arbre de Merkle. Cette fonction est une fonction récursive.
Quelle est la valeur de la racine de l'arbre de Merkle constitué des hashs de « bonjour », « blockchain » et « sécurité » (dans cet ordre) ? Répétez la même opération avec « 1234 », « 5678 », « abcd » et « efgh » (dans cet ordre).

5. Construction et minage d'un bloc

Un bloc est un élément essentiel d'une blockchain. Traduit littéralement, le terme « blockchain » veut dire « chaine de blocs ».

[2] https://fr.wikipedia.org/wiki/Ralph_Merkle

Il est important de voir quelles sont les parties qui constituent un bloc. Elles sont au nombre de 3. Le header, la racine de l'arbre de Merkle et la liste des transactions dans le cadre de la blockchain Bitcoin par exemple. Le header contient toutes les informations relatives au bloc ; on retrouve l'index du bloc, le hash du bloc précédent (aucun si c'est le premier bloc), la date du minage du bloc, le nonce utilisé, le hash du bloc.
La liste des transactions est contenue dans le bloc et cette liste est utilisée pour calculer la racine de l'arbre de Merkle.

A l'aide de la classe « transaction.py » fournie avec ce TP, créer une classe « block.py ». Cette classe doit contenir toutes les informations nécessaires au bon fonctionnement d'un bloc qui serait utilisé dans le cadre d'une blockchain.

6. Construction de la blockchain

Créer une classe « blockchain.py » qui contiendra une liste de blocs ainsi que la difficulté de la blockchain.

7. Vérifier la chaine

Pour qu'une blockchain soit fonctionnelle, il est nécessaire d'avoir des fonctions de vérification. Ces vérifications sont utilisées à chaque fois qu'un nouveau bloc est créé par un mineur. Avant de l'ajouter à la chaine, il faut vérifier que ce bloc est correct. La vérification dans une blockchain se fait également en vérifiant les blocs précédents. Par exemple, la blockchain Bitcoin vérifie les 6 blocs précédents avant d'en ajouter un nouveau.
Faire les fonctions de vérification nécessaire qui permettent de savoir quel bloc n'est pas correct.

8. Modifier la chaine

Modifier un bloc de la blockchain et voir ce qui se passe.
Que se passerait-il si un nouveau bloc est miné sur une blockchain altérée ?
Comment le réseau va réagir si un bloc invalide est transmis ?