



Rapport : Blockchain

Touraya EL HASSANI
Adjanie KENGNI
MAIN4

Encadré par : François PÉCHEUX

31 mai 2019

Table des matières

1	Introduction	2
2	Qu'est ce que la Blockchain ?	2
3	SHA 256	4
4	Le minage	5
4.1	Comment faire ?	6
5	Blockchain	7
6	Notre code pour la blockchain	9
6.1	Structure d'un bloc	9
6.2	Génération du hash	9
6.3	Le premier bloc	9
6.4	Création de la chaîne de blocs	10
7	Résultat	10

1 Introduction

De nos jours, tout le monde utilise sa carte bancaire. Il faut savoir que quand on utilise notre carte bancaire, la transaction est stockée dans le compte de notre banque. Mais imaginons qu'un jour, ce compte est piraté et disparaît.

Ceci n'est pas un problème avec la crypto-monnaies puisqu'elle repose sur des livres de compte librement accessibles et décentralisés. Tout le monde peut aujourd'hui télécharger le livre de compte des transactions Bitcoin.

Ce grand livre de compte porte un autre nom, la **blockchain**. Notre but durant ce projet est d'implémenter des transactions blockchain.

2 Qu'est ce que la Blockchain ?

La **blockchain** est une technologie de stockage et de transmission d'informations sécurisées, transparentes et décentralisées (fonctionnant sans organe central de contrôle). La Blockchain constitue une base de données qui contient l'historique de tous les échanges effectués entre les utilisateurs depuis le début du blockchain. Cette base est partagée par les différents utilisateurs, sans intermédiaire, ce qui permet à chacun utilisateur de vérifier la validité de la chaîne.

Il existe des blockchains publiques, ouvert à tous, et des blockchains privés dont l'accès et l'utilisation sont réservés à certains nombres d'acteurs.

Dans le **blockchain public**, la clé publique est connue par tout le monde et permet de vérifier que les transactions faites par un utilisateur en privé avaient bien lieu d'être.

Par exemple, si un utilisateur utilisant une clé privée et souhaite faire une transaction de \$20, pour cela il devra utiliser deux clés, une privée et l'autre publique. La transaction est faite avec une signature que lorsque celui-ci change dans le blockchain, on reçoit un message d'erreur comme on peut le voir ici.

Block:

5

Nonce:

7355

Coinbase:

\$ 100.00 -> 04cc17dc129331c1cbb9c32cf4dc2

Tx:

\$ 2.00 From: 04d4080959e3795b7 -> 04d84dae793a8253

Sig: 304502207fcc9d79c7894a4fa246f3b1ee8b21a40ae7f195e8f08ffe253d1631

\$ 6.00 From: 0451d4a9c44a2dec -> 043e17e5095e878b

Sig: 30450220454632e38948141be2c1b75e6c08b2b98dfc6e95d1691cdd6cd0a31

\$ 4.00 From: 0451d4a9c44a2dec -> 04020d6fe7aeabd3

Sig: 3046022100e5e8cb0d2ac042cc8c026c52622a191780da1bdca41ebe2b6190f

\$ 9.95 From: 040b4c84f02bfec4 -> 04148850d1edbd66

Sig: 304502203f18249ae65e941f0571cc58debf3455700f2508c6ad04ba45194a6

Prev:

0000e0e3d78d093313f15936fb3d08f06b2bd095044342a1c896a3ee8b10a7bf

Hash:

0000056aeaa2c748970c80ac2412be73e0cec4593862b710c59d0d3f90764cd

Mine

SHA 256

Dans cette blockchain, on peut voir comment la transaction se met en place et la signature reçue par celle-ci.

Block:

5

Nonce:

7355

Coinbase:

\$ 100.00

->

04cc17dc129331c1cbb9c32cf4dc2

Tx:

\$ 25.00

From: 04d4080959e3795b

->

04d84dae793a8253

Sig: 304502207fcc9d79c7894a4fa246f3blee8b21a40ae7f195e8f08ffe253d163

\$ 6.00

From: 0451d4a9c44a2dec

->

043e17e5095e878b

Sig: 30450220454632e38948141be2c1b75e6c08b2b98dfc6e95d1691cdd6cd0a31

\$ 4.00

From: 0451d4a9c44a2dec

->

04020d6fe7aeabd3

Sig: 3046022100e5e8cb0d2ac042cc8c026c52622a191780da1bdca41ebe2b6190f

\$ 9.95

From: 040b4c84f02bfec4

->

04148850d1edbd66

Sig: 304502203f18249ae65e941f0571cc58debf3455700f2508c6ad04ba45194a6

Prev:

0000e0e3d78d093313f15936fb3d08f06b2bd095044342a1c896a3ee8b10a7bf

Hash:

0f21b5f2519df9c407755743ba2dc237a7a126ddf0b4939b2752481f2313bb78

Mine

SHA 256

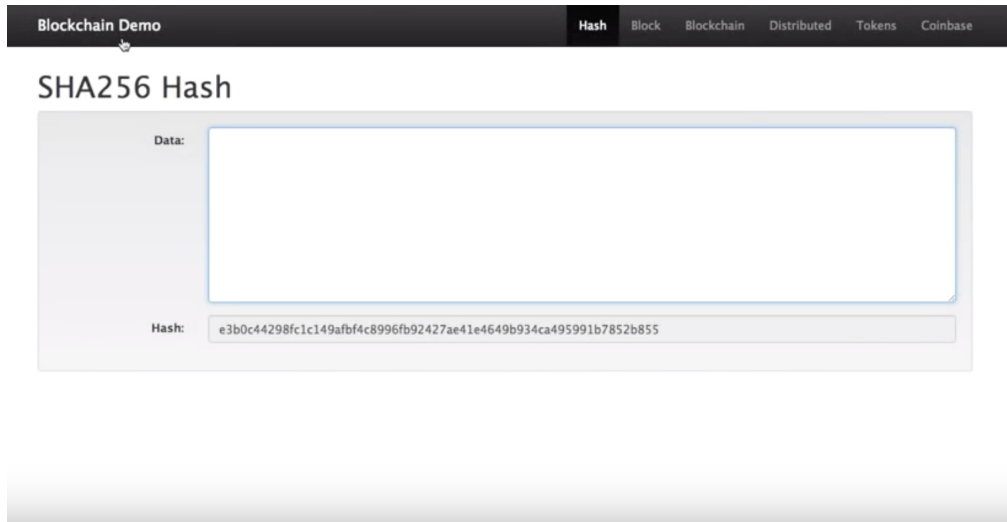
Comme on peut le voir ici, quand la signature change, le bloc n'est plus attribué et la transaction ne pourra se poursuivre.

3 SHA 256

SHA-2 (Secure Hash Algorithm) est une **famille de fonctions de hachage** qui ont été conçues par la National Security Agency des États-Unis (NSA). SHA-2 comporte les fonctions, **SHA-256** et SHA-512 dont les algorithmes sont similaires mais opèrent sur des tailles de mot différentes (**32**

4

bits pour **SHA-256** et 64 bits pour SHA-512). Dans notre projet, nous allons utiliser SHA-256.



The screenshot shows a web interface for a 'Blockchain Demo'. At the top, there is a navigation bar with links: 'Hash', 'Block', 'Blockchain', 'Distributed', 'Tokens', and 'Coinbase'. The 'Hash' link is currently selected. Below the navigation bar, the title 'SHA256 Hash' is displayed. The main content area contains two input/output fields. The first field, labeled 'Data:', is a large empty text box. The second field, labeled 'Hash:', contains the hexadecimal string 'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855'.

SHA 256

L'image ci-dessus montre que pour hacher des lettres, des mots ou encore des phrases, il suffit de les écrire dans l'espace data et la fonction va automatiquement les hacher. Le hash change à chaque fois qu'une lettre est modifiée, supprimée, etc. Peu importe la taille de data, la taille du résultat hash aura toujours la même taille.

-> Nous pouvons retrouver ce code dans le fichier test.c. Pour le compiler et le lancer il suffit de suivre les étapes suivantes :

- gcc sha.c test.c -o test
- ./test

Nous avons écrit comme data : Touradj
et nous obtenons comme hash : fa 5a f9 a4 8d bb 30 fc 10 le a ff fb 2d 8f 89 5c 82 6e c9 5b bb 6 5a 1c c3 ld l8 2c 6b 7 83.

Le but maintenant est d'étendre cela à un bloc.

4 Le minage

Dans cette section le but est de voir ce qu'est le bloc. La seule différence entre la fonction précédente et le bloc est que cette fois ci dans le bloc, la data est séparée en 3 cases :

- Le **bloc**, c'est un nombre qui désigne le numéro du bloc,
- Le **Nonce**, qui sera expliqué plus tard,

- La **data** comme précédemment.

Block

Block: # 1

Nonce: 72608

Data:

Hash: 0000f727854b50bb95c054b39c1fe5c92e5ebcfa4bcb5dc279f56aa96a365e5a

Mine

SHA 256

Comme on peut le voir ici, le hash **commence avec 0000**. On peut appeler ça "un bloc attribué" (assigned block). Mais, si on ajoute à Data, le mot "bonjour" par exemple, le hash va changer et ne commencera plus par 0000, il ne sera donc plus attribué. C'est à partir de là qu'on peut parler du **nonce**. Le but est de trouver le nonce qui fait en sorte d'avoir un hash qui commence avec 0000.

4.1 Comment faire ?

- **Choisir le numéro du block** : 1 par exemple,
- **Mettre la data** pour laquelle on veut trouver un hash qui commence par 0000,
- **Mettre Nonce à 1** au début et vérifier si le hash commence par 0000.
 - Si oui, alors on a trouvé le nonce,
 - Sinon on incrémente le nonce.

—> Nous pouvons retrouver ce code dans le fichier test1.c. Dans notre code, nous faisons une boucle while qui s'arrête une fois que le hash qui commence par 0000 est trouvé. Une fois qu'il est trouvé, on récupère le nonce. Pour le compiler et le lancer il suffit de suivre les étapes suivantes :

- gcc sha.c test1.c -o test1
- ./test1

Le premier nonce trouvé pour une data=Touraya, est égale à **20 077**.

-> C'est ce qu'on appelle "**Le minage**".

Quand on trouve le nonce qui nous donne un hash avec 0000, le bloc est donc attribué (assigned block).

Cela nous mène à la notion de blockChain que l'on verra dans la partie suivante.

5 Blockchain

La blockchain est une chaine de plusieurs bloc. Le but ici est de voir **comment on les combines ensemble**. Ci-dessous, une illustration de la blockchain avec plusieurs blocs.

Blockchain

Block	#	Nonce	Data	Prev	Hash
1	1	11316		00000000000000000000000000000000	000015783b764259d382017d91a36d206d0
2	2	35230		000015783b764259d382017d91a36d206d0	000012fa9b916eb9078f8d98a7864e697ae83
3	3	12937		000012fa9b916eb9078f8d98a7864e697ae83	0000b9015ce2a08b61216ba

Suite des blocs

- Nous pouvons voir qu'il y a bloc 1 avec un certain nonce, puis le bloc 2, bloc 3, etc.
- Le **prev** (qui veut dire précédent) du dernier bloc commence par 0000ae8 et correspond au hash du block précédent et ainsi de suite pour les autres blocs.
- Du coup, pour le tout premier bloc, le prev est égale à que des 0 car il n'a pas de bloc précédent.

Que se passe-t'il si on change la data du dernier block ?

-> Le dernier block a un hash qui change et du coup qui ne commence plus par 0000 et donc il n'est plus attribué.

Maintenant que se passe-t'il si on change la data de l'avant dernier block ?

-> Le hash de ce block change et du coup le prev du dernier block aussi ce qui fait que le hash du

dernier block change aussi et ne commence plus par 0000.

Block:	#	4
Nonce:	35990	
Data:	h	
Prev:	0000b9015ce2a08b61216ba5a0778545bf4d	
Hash:	dc5e8a5df1667c3383de6250db431a444a11c	

Block:	#	5
Nonce:	56265	
Data:		
Prev:	dc5e8a5df1667c3383de6250db431a444a11c	
Hash:	a0491b2ef9aa0ad2eae430af4fdb2db158872	

Blockchain avec bug

Block:	#	3
Nonce:	12937	
Data:	T	
Prev:	000012fa9b916eb9078f8d98a7864e697ae83	
Hash:	0000b9015ce2a08b61216ba5a0778545bf4d	

Block:	#	4
Nonce:	35990	
Data:		
Prev:	0000b9015ce2a08b61216ba5a0778545bf4d	
Hash:	0000ae8bbc96cf89c68be6e10a865cc47c6c4f	

Blockchain sans bug

Par contre, si on veut changer la data et trouver un nouveau hash qui commence par 0000, il faut le changer dans le dernier block et reminer.

6 Notre code pour la blockchain

6.1 Structure d'un bloc

Il faut d'abord créer une classe Block qui va contenir les éléments suivants :

- **public variable sPrevHash** (car chaque bloc est lié au bloc précédent)
- le constructeur de bloc avec nIndexIn et sDataIn,
- la méthode **GetHash**,
- la méthode **MineBlock**,
- la méthode **CalculateHash**,
- **nIndex**,
- **nNonce**,
- **sData**,
- **sHash**,
- **tTime**.

6.2 Génération du hash

La technologie des **blockchain** a été rendue populaire lorsqu'elle a été conçue pour la monnaie numérique Bitcoin, car le registre est à la fois immuable et public ; ce qui signifie que lorsqu'un utilisateur transfère des Bitcoin à un autre utilisateur, une transaction de transfert est écrite dans un bloc de la blockchain par des nœuds du réseau Bitcoin. Un **nœud** est un autre ordinateur exécutant le logiciel Bitcoin et, dans la mesure où le réseau est peer-to-peer, il peut s'agir de n'importe qui dans le monde. Ce processus est appelé "**extraction**" car le propriétaire du nœud est récompensé par Bitcoin chaque fois qu'il crée avec succès un bloc valide dans la blockchain.

Pour réussir à créer un bloc valide, et donc être récompensé, un mineur doit créer un **hachage cryptographique** du bloc qu'il souhaite ajouter à la blockchain, ceci est réalisé en comptant le nombre de zéros au début du hachage, si le nombre de zéros est égal ou supérieur au niveau de difficulté défini par le réseau bloqué est valide. Si le hachage n'est pas valide, une variable appelée nonce est incrémentée et le hachage est créé à nouveau. Ce processus, appelé preuve de travail, est répété jusqu'à la production d'un hachage valide.

→ C'est ce que fait notre méthode MineBlock

Nous commençons par la signature de la méthode MineBlock, que nous avons spécifiée dans le fichier d'en-tête Block.h, et créons un tableau de caractères d'une longueur de 1 à la valeur spécifiée pour nDifficulty. Une boucle for est utilisée pour remplir le tableau avec des zéros, puis le dernier élément de tableau reçoit le caractère de fin de chaîne (). Une boucle do... while est ensuite utilisée pour incrémenter nNonce et sHash est affecté à la sortie de CalculateHash. La partie avant du hachage est ensuite comparée à la chaîne de zéros que nous venons de créer ; si aucune correspondance n'est trouvée, la boucle est répétée jusqu'à ce qu'une correspondance soit trouvée. Une fois la correspondance trouvée, un message est envoyé au tampon de sortie pour indiquer que le bloc a été exploité avec succès.

6.3 Le premier bloc

Nous commençons par la signature du constructeur de la chaîne de blocs que nous avons spécifiée dans Blockchain.h. Au fur et à mesure que des blocs sont ajoutés à la blockchain, ils doivent référencer le bloc précédent à l'aide de son hachage, mais comme la blockchain doit commencer

quelque part, nous devons créer un bloc pour le prochain bloc à référencer, nous appelons cela un bloc de genèse. Un bloc de genèse est créé et placé sur le vecteur `vChain`. Nous définissons ensuite le niveau de difficulté `n`. Ici nous voulons 0000 donc `n=4`.

6.4 Création de la chaîne de blocs

La signature que nous avons spécifiée dans `Blockchain.h` pour `AddBlock` est ajoutée, suivie de la définition de la variable `sPrevHash` pour le nouveau bloc à partir du hachage du dernier bloc de la chaîne de caractères obtenue à l'aide de `GetLastBlock` et de sa méthode `GetHash`. Le bloc est ensuite exploité à l'aide de la méthode `MineBlock` vue précédemment, suivie de l'ajout du bloc au vecteur `vChain`, complétant ainsi le processus d'ajout d'un bloc à la chaîne de blocs.

7 Résultat

Notre main crée une nouvelle chaîne de blocs et informe l'utilisateur qu'un bloc est en cours d'exploration en imprimant dans le tampon de sortie, puis crée un nouveau bloc et l'ajoute à la chaîne; le processus d'extraction de ce bloc sera alors lancé jusqu'à ce qu'un hachage valide soit trouvé. Une fois le bloc extrait, le processus est répété pour deux autres blocs.