Blockchain et autorité de certification

Janvier 2020

Table des matières

Le programme Java va simulation d'une autorité de certification "décentralisée" à l'aide d'une blockchain. En fait si une blockchain de type Bitcoin est vraiment décentralisée c'est qu'elle est gérée par peer to peer. Ici nous allons implémenter une blockchain centralisée mais on va faire comme si.

1 Présentation

1.1 Principe d'une blockchain

La blockchain ou chaîne de blocs est une structure de données informatique qui permet de sécuriser des données, supposées être publiques, contre toute tentative de modification ou altération. Pour cela on utilise un algorithme de hash code qui permet d'associer à une donnée une valeur numérique unique. Si H est la fonction qui implémente un de ces algorithmes, et x une donnée quelconque, on a :

— $H(x)$ donne toujours la même valeur pour le même x
— $H(x) \neq H(x')$ même si la différence entre x et x' est minime.
— connaissant $H(x)$ est très difficile de retrouver x.

L'algorithme de hash que nous utiliserons est le SHA256 (nombres de 256 bits). Nous allons manipuler son résultat sous forme d'une chaîne de caractères hexadécimaux. La longueur de la chaîne est fixe (32 octets, donc caractères).

La blockchain est constituée de blocs chaînés entre eux sous forme de liste. Jusque là ça reste une SD très classique en informatique. Pour éviter qu'un bloc disparaisse sans laisser de trace, on va ajouter (signer) chaque bloc avec le hash code de son contenu. Et pour qu'il laisse une trace en cas de disparition on va inclure dans le contenu du bloc le hash code de son prédécesseur. Et donc, le hash code du bloc va intégrer le hash code du bloc précédent dans la chaîne.

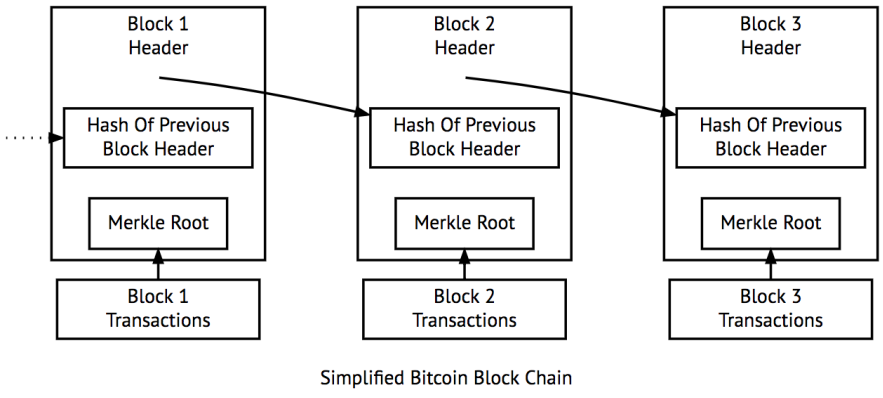

FIGURE 1 – La blockchain

Que se passe-t-il si on veut supprimer un bloc quelque part au milieu de la chaîne ? De toute évidence on perturbe la continuité des hash codes. Il faut donc recalculer le hash de tous les blocs suivant celui qui disparaît et refaire le chaînage. La sécurité de la chaîne tient ici à la quantité de travail à faire pour altérer la chaîne.

Même sur un grand nombre de block, on peut imaginer pouvoir le faire avec un gros PC. Ça reste jouable me direz-vous. On va donc pimenter le tout avec la "difficulté". Pour faire simple, disons que la difficulté est un critère que doit remplir le hash code et qui assure qu'il n'est pas trop facile à calculer même avec une machine performante. Dans la pratique (Bitcoin) la difficulté est adaptée à l'évolution de la puissance des machines. Elle est choisie pour qu'il faille 10mn pour la calculer. A itérer autant de fois qu'il y a de blocs à recalculer pour altérer la chaîne sans que cela ne se voit. Comme un nouveau bloc arrive toutes les 10mn, il faudrait recalculer l'ensemble de la chaîne en 10mn pour que cela passe inaperçu (ou presque car il y a aussi d'autres mécanismes dans un univers peer2peer où il y a autant de copies de la chaîne que de noeuds dans le réseau).

Nous allons simplifier ce mécanisme en choisissant une difficulté représentée par un entier d, et en de-

mandant que le hash du bloc commence par d zéros dans sa représentation en ascii hexadécimal. Et si cela n'est pas le cas ?

1.1.1 Le minage

On introduit dans le contenu de chaque bloc un entier initialisé à zéro et appelé "nonce". Si le hash du bloc (incluant la nonce) ne satisfait pas le critère, on incrémente la nonce et on recommence le calcul du hash. On répète ce processus jusqu'à ce que le hash satisfasse le critère (la difficulté). Ce calcul est appelé «minage» [mining].

1.1.2 Contenu des blocs

Que met-on dans les blocs ? Ce que l'on veut en fait. Le Bitcoin gère des transactions d'argent (crypto-monnaie [crypto-currency]). Nous allons imiter ce principe bien qu'il soit annexe dans ce sujet. Donc il y aura des transactions de Duckcoincoin (Dcc) du genre «Astérix envoie 10 Dcc à Obélix». Il y a un nombre variable de transactions dans un bloc qui est normalement (Bitcoin) dépendant de la taille maximale du bloc et de celle des transactions. Nous allons fixer arbitrairement le nombre maximal de transactions (constante du programme). De plus, le bloc contient la date de création (timestamp).

Le premier bloc de la chaîne est particulier. Il ne peut pas intégrer le hash de son prédécesseur car il n'a pas de prédécesseur. On l'appelle bloc «génésis». Le hash de son prédécesseur est zéro, sa nonce reste à zéro, il a une unique transaction réduite à la chaîne « Genesis » [1]. Par contre il a bien un hash. Il n'est pas nécessaire de le miner.

Que se passe-t-il si on veut altérer une transaction d'un bloc sans toucher aux hash de ce bloc ?

1.1.3 Arbre de Merkle [Merkle tree]

Si on altère une transaction d'un bloc sans toucher aux hash de ce bloc, le hash du bloc va quand même être modifier puisque qu'il prend en compte le contenu de tout le bloc, et la liste des transactions fait partie du contenu (pour notre sujet car le Bitcoin est légèrement différent).

Pour corser le tout on va calculer l'arbre de Merkle des hash des transactions :

1. supposons que nous ayons quatre transactions : $t1, t2, t3, t4$,
2. on calcule les hash de chaque transaction. On obtient $h1, h2, h3, h4 = H(t1), H(t2), H(t3), H(t4)$
3. puis on calcule le hash de $h1$ concaténé avec $h2$ et le hash de $h3$ concaténé avec $h4$. Soit $h1.2$ et $h3.4$ les résultats,
4. puis on calcule $H(h1.2$ concaténé $h3.4)$, soit $h1.2.3.4$ le résultat.

Le résultat final $h1.2.3.4$ est le racine de l'arbre [Merkle tree root]. C'est un hash code. Ce hash sera inclus dans le contenu du bloc.

Si le nombre de transactions (ou de hash intermédiaires) est impair, on duplique simplement la dernière transaction (ou hash).

Connaissant les transactions qui sont écrites en clair on peut recalculer la racine de l'arbre et vérifier que c'est bien le même hash root que celui qui apparaît dans le bloc, et que donc, aucune transaction n'a été altérée

1. Pour reprendre le nom donné à ce block par Satoshi Nakamoto (聡中本) l'inventeur du Bitcoin.

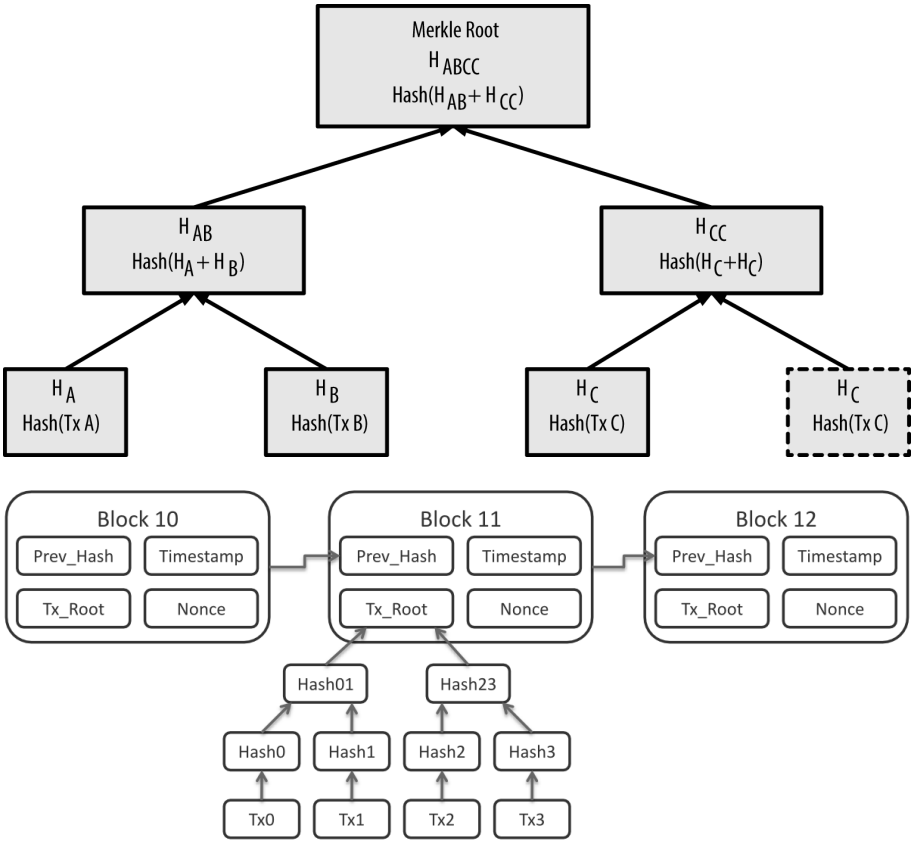

FIGURE 2 – Merkle tree

1.1.4 Sécurité de la blockchain

Quelques attaques (dans la cas de notre blockchain simplifiée) :

— Supprimer ou modifier une transaction : Cela modifie le merkleroot et donc le sha du block et provoque une inconsistance de la chaîne.

— Rejouer une transaction déjà présente. Normalement le timestamp et le numéro de tx doit empêcher ça car il sont dans le hash de la signature. Il faudrait modifier ces données. De plus cela modifie le merkle tree. Le test de hash root le détecte.

— Agir sur les blocks. Supprimer un block pour faire disparaître une transaction. Cela supprime aussi les autres transactions. Et le chaînage de hash est perturbé. Il faut tout recalculer.

1.1.5 Les transactions

Les transactions seront des documents à archiver par la blockchain :

— déclaration de clef de cryptage ou de signature associée à une identité

— déclaration de révocation de clef

Pour créer une transaction, c'est à dire enregistrer un document, une personne doit envoyer une requête. Nous ferons l'impasse sur la vérification de l'identité de l'émetteur car cette phase est délicate. Chaque transaction intègre, de plus, une date (timestamp).

1.1.6 Vérification

L'intégrité de la blockchain peut être vérifiée à tout moment.

1. On vérifie que la chaîne commence bien par le bloc génésis et que le chaînage des hash est valide, et que le hash du bloc est bien celui annoncé (vérification 1).
2. On vérifie pour chaque bloc que le hash Merkle root correspond bien aux transactions de ce bloc (vérification 2).

1.1.7 Qui veut la peau de la blockchain ?

Bien sûr notre blockchain n'est pas inaltérable, ni celle du Bitcoin d'ailleurs. Puisque que la nôtre est centralisée on peut faire ce que l'on veut avec ce qui n'est pas le cas de celle du Bitcoin qui est répartie et soumise au principe du consensus (mais c'est une autre histoire).

1.1.8 Problème de la certification et de la vérification de l'identité

Astérix est un Youtubeur connu mais qui a négligé les autres réseaux sociaux et ne connait rien à la cryptographie. Il n'a donc pas de compte FB ni insta ou twiter. Acidecloridrix (nom de code HCL), l'espion de César, ouvre un compte mail [astérix@gmail.com], envoie une tx à la blockchain avec des adresses crypto et signature qu'il a créés, ouvre des comptes FB et insta avec cette adresse mail, communique en se faisant passer pour Astérix et incite les gens à utiliser ses adresses crypto.

Assurancetourix, qui vient de créer l'oeuvre de sa vie, un opéra épique et magnifique, au dire de son auteur, décide de l'envoyer à Astérix pour avoir son avis. Il fait confiance à Astérix, qu'il connait par ailleurs, mais décide d'utiliser la cryptographie pour sécuriser la transmission de son message. Il cherche les adresses crypto d'Astérix sur la BC, trouve celle envoyée par HCL, et ne fait pas attention à l'adresse mail qui n'est pas celle qu'il connait. Il envoie donc son opéra, crypté avec la clef d'HCL à l'adresse créée par HCL, en croyant correspondre avec Astérix. HCL peut donc lire le message et envoie l'opéra à César, ceo de Rome and co limited, une multinationnale sans scrupule déjà condamnée pour avoir spoilé de nombreux auteurs.

2 Travail à faire

2.0.1 Cahier des charges JAVA

La programmation par objets se prête particulièrement à la programmation d'une blockchain. Java fournit des structures de données prédéfinies comme des *arraylist* génériques qui sont très efficaces. Les packages et classes fournis pour la lecture/écriture au format JSON et le calcul du SHA256 sont *plug and play*. Votre programme Java devrait se passer sans trop de soucis et aller assez vite.

1. Interface de gestion (voir ci-dessous)
2. Structure de données pour la blockchain centralisée
3. Structure de données pour les blocs
4. Structure de données pour les listes de transactions
5. Calcul du hash256 d'un bloc
6. Minage des blocs
7. Test d'intégrité (présence du Génésis, les ash des blocks sont cohérents, le chaînage aussi, les racines de Merkle aussi)
8. Calcul de l'arbre de Merkle d'une liste de transactions
9. Sauvegarde/lecture fichier JSON (en utilisant le package Google)
10. Lecture les transactions en attente dans un fichier et les traiter par ordre de lecture
11. Génération d'un blocks à partir des nouvelles transactions
12. Faire une classe main qui gère l'ensemble

3 Détails techniques

Cette partie apporte des précisions sur la structure des différents objets. Une partie des documents, les programmes et liens cités sont disponible sur Moodle. Moodle sera par ailleurs le medium privilégié de communication de documents, de news, de mise à jour etc.

3.1 Spécifications générales

3.1.1 Structure de la blockchain

— Champs difficulté : entier = critère de difficulté pour le minage
— Champs : nombre de blocks : entier = nombre de blocks de la chaîne (en comptant le génésis).
— Champs liste des blocks : liste de blocks

3.1.2 Structure des blocks

— champs : index : entier = numéro du block dans la chaîne (le génésis est numéroté zéro)
— champs : timestamp : chaîne de caractères = la date au moment de la création
— champs : hash du block précédent : chaîne de caractères = hash du block précédent dans la chaîne
— champs : nombre de transactions : entier
— champs : liste des transactions
— champs : hash root de l'arbre de Merkle des transactions : chaîne de caractères
— champs : hash du block courant : chaîne de caractères
— champs : nonce : entier

3.1.3 Structure des transactions

— champs : index : entier = numéro de la transaction dans la liste
— champs : timestamp : chaîne de caractères = la date au moment de la création de la transaction
— champs : émetteur : chaîne de caractères = identité de l'émetteur
— champs : email de l'émetteur
— champs : clef publique de chiffrage
— champs : clef de vérification de signature

3.1.4 Interface de gestion

C'est une partie plus créative du projet, il n'y a pas de véritable contrainte si ce n'est que l'interaction devra être la plus légère et efficace possible. Cette interface devra permettre les actions suivantes :

— Génération d'une blockchain en choisissant le nombre de blocs, la difficulté, le nombre maximal de transactions par bloc,
— sauvegarde ou lecture dans un fichier au format JSON,
— vérification de l'intégrité de la blockchain,
— sortir du programme,
— affichage de la blockchain ou d'un bloc d'un numéro donné
— publication d'une clé publique avec signature
— revocation d'une clé avec signature
— recherche d'une clé pour trouver le propriétaire
— recherche de toutes les clés de quelqu'un
— savoir si une clé est révoquée
— connaître si la clef est vérifiée et par qui (mail d'ack quand la transaction est traitée) XXX
— Comparer toutes les clés d'une personne et dire si elle ont été déclarées pas cette personne
— sauver/importer en JSON
— sauver des clefs

— ajout de blocs et minage
— historique et vérification d'une clef, d'une identité
— évènements : revocation, nouvelle clef,
La structure de l'interface est libre.

3.2 SHA256

L'algorithme de hash que nous allons utiliser est celui qu'utilise la blockchain du Bitcoin, le sha256. Un programme réalisant ce calcul est fourni en Java comme en C. Il reçoit une chaîne de caractères et renvoie une chaîne contenant le hash en caractères hexadécimaux.

3.3 Le format JSON

Le format JSON vient du Javascript et est un format d'échange de données alternatif au XML. Wikipédia en parle très bien, je vous invite à lire la page :

`https://fr.wikipedia.org/wiki/JavaScript_Object_Notation`

En fait, il n'y a pas grand chose à savoir sur ce format. Il s'agit juste de l'utiliser. Des packages et librairies sont fournies pour cela.

3.4 Les packages Java

3.4.1 GSON : le package de lecture/écriture JSON de Google

`https://github.com/google/gson`

Le package $gson - 2.2.2.jar$ est à télécharger sur Moodle.

3.4.2 Classe sha256

La classe est à télécharger sur Moodle.

3.4.3 Le package google pour les adresses et les clefs

bitcoinj-core-0.14.5-bundled.jar sur Moodle

3.5 Constantes des programmes et performances attendues

La difficulté sera de 4 pour les tests. Le nombre de transactions doit pouvoir être de 1 à 10 (nombre aléatoire pour chaque bloc). Le nombre de blocs n'excèdera pas de 10 pour les tests. Cependant, programme doit pouvoir supporter la génération de 100 blocks.