TP2 - Les transactions

L'objectif de ce TP est de vous faire implémenter le concept de transaction.

[Temps estimé : ***20 ~ 30 minutes*** (sans l'exercice bonus).]

Introduction

1) Chargez dans un navigateur web le fichier ***/TP2/src/index.html***.

2) Chargez dans un éditeur de code le TP2 à partir de son répertoire racine ***/TP2***.

3) Rendez-vous dans le fichier du code métier ***/TP2/src/index.js*** *dans votre éditeur de code***.**

Exercice 1 : les transactions

1) Copiez les classes ***Block***, et ***Blockchain*** du TP1 dans le code métier de votre TP2.

1) Déclarez une classe ***Transaction***.

2) Déclarez un ***constructeur*** dans votre classe ***Transaction***. Il doit initialiser ces attributs :

- ***sender*** : l'expéditeur de la transaction ;
- ***privateKeySender*** : la clé privée de l'expéditeur de la transaction ;
- ***recipient*** : le destinataire de la transaction ;
- ***amount*** : le montant de la transaction ;
- ***creationDate*** : la date de création de la transaction.

Le constructeur doit prendre en paramètre ***sender***, ***privateKeySender***, ***recipient***, et ***amount***. Pour creationDate, Date.now() suffira, et les attributs : ***sender***, ***recipient***, ***amount***, et ***creationDate*** ne sont pas en hexadécimal.

3) Déclarez des **getters** pour chacun des attributs, sauf pour l'attribut ***privateKeySender***.

4) Déclarez une fonction ***encrypt()***, qui retourne la transaction chiffrée avec la clé privée de l'expéditeur. La fonction doit concaténer les attributs ainsi : ***creationDate + « : » + sender + « -> » + amount + « -> » + recipient***, puis elle doit chiffrer le résultat de la concaténation avec la fonction ***encrypt()*** et la clé privée de l'expéditeur.

5) Déclarez un attribut ***encryption*** dans le constructeur qui prend la valeur retournée par ***encrypt()***, et déclarez aussi un **getter** pour ce nouvel attribut.

6) Déclarez une méthode de classe : ***static decrypt(encryptedTransaction, publicKey)***, qui doit retourner le déchiffrement d'une transaction chiffrée, avec la fonction ***decrypt*** et la clé publique de l'expéditeur.

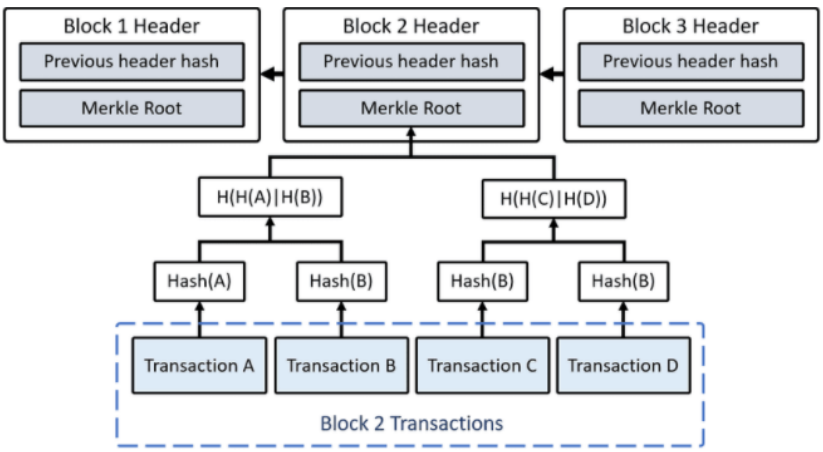
7) Testez votre classe ***Transaction*** dans la fonction ***handler*** : créez une blockchain, créez plusieurs transactions, puis ajoutez des blocks dans la blockchain prenant les encryptions de ces transactions dans leur tableau de hashs de transactions, et affichez les blocks de la blockchain un par un. Observez.

8) Envoyez via une messagerie de votre choix, l'encryption d'une de vos transactions et votre clé publique à votre binôme, demandez à votre binôme de déchiffrer la transaction avec la fonction ***Transaction.decrypt(encryptedTransaction, publicKey)***, a-t-il bien retrouver la description originale de la transaction ?

Exercice 2 (bonus) : le Merkle Tree

1) Redéfinissez dans la classe ***Block*** la fonction ***computeTransactionsMerkleRootHash()***, l'idée est de la réécrire correctement.

   On appelle ***Merkle Tree***, l'algorithme qui permet de calculer le ***Merkle Root***, qui est le hash résumé de tous les hashs des transactions d'un block.



L'idée est qu'il faut hacher la concaténation des encryptions des transactions, couple de transaction par couple de transaction, et continuez récursivement jusqu'à n'avoir plus qu'un seul hash, le nœud racine du ***Merkle Tree*** : le ***Merkle Root***. Pour hacher : on utilisera dans notre implémentation la fonction ***sha256(content).***

Pour commencer, vous pouvez partir du prérequis que vous avez un nombre n **pair** de transactions, en tant que feuilles de l'arbre. C'est un prérequis faux, mais qui permet de commencer simplement.

Si vous avez du mal avec cette question, demandez des compléments d'information à un des deux responsables de l'atelier.