Groupe 2
ARIZANOVA Simona
DIARSO Laye Kemo
HADI Ismaïl
JOLIMAN Iris

Compte Rendu

Composant n°2

Blockchain Wallet

Version	Date	Modifications	Auteur
1.0	26/04/2020	Initiation du document	Groupe 2

Table des matières

I. Contexte général

Une blockchain est une nouvelle technologie permettant de stocker et transmettre des informations de manière transparente, sécurisée et non centralisée. Sa principale application relève des cryptomonnaies telles que le bitcoin (2008) ou l'éther. Depuis, les entreprises prennent confiance en cette technologie dont la notoriété ne cesse de grandir. On assimile de façon plus simple une blockchain à un livre comptable public, anonyme et impossible à falsifier.

Ce projet consiste à créer une blockchain contenant donc un ensemble de transactions et permettant d'en enregistrer de nouvelles tout en respectant les principes de base utilisés par cette technologie (minage, cryptographie, clé privée ou publique, etc). Au sein de ce projet, différents composants interagissent entre eux. Nous avons la responsabilité du composant « wallet / portefeuille », fonctions fournies en python (seul composant nécessitant d'être codé en python ; tous les autres composants étant en C++).

Un wallet peut en réalité prendre différentes formes ; il peut s'agir d'une application mobile, d'une clé usb (hardware wallet) ou encore d'un logiciel. Dans tous les cas, ce composant est vulnérable aux cyber-attaques car il contient en général des cryptomonnaies. Son fonctionnement est décrit dans la suite de ce document.

II. Principe du composant – « Wallet »

Chargés de réaliser les fonctionnalités liées aux composants, nous nous sommes intéressés aux principaux usages d'un portefeuille :

- Un Wallet est une sorte de « compte bancaire » ; pour ainsi dire il est nécessaire de pouvoir s'authentifier sur compte. Le concept de connexion et déconnexion est donc pris en compte.

- Comme dans une institution bancaire classique, nos crypto monnaies peuvent être réparties dans de différents comptes (matérialisés par plusieurs clés publiques). Il faut donc préciser de quel compte on parle, pour la réalisation d'une transaction ou lorsqu'on souhaite obtenir son solde.

- Les informations importantes doivent être facilement accessibles : une visualisation simple de son solde de compte (que cela soit sur un seul compte ou sur l'ensemble de ses comptes).

- L'utilisateur doit pouvoir effectuer une transaction vers un autre utilisateur. Il doit donc préciser le compte depuis lequel la crypto monnaie sera débitée, le compte à créditer ainsi que le montant.

Les choix techniques concernant ces fonctionnalités seront décrits dans la partie suivante. ***Chaque fonction est associée à une fonction de test, dont le résultat a été positif (sauf celles qui ont de fortes dépendances avec les autres équipes).***

III. Définition des fonctionnalités

A. Fonctions de consultation

La fonction de consultation consiste à explorer et à analyser les différentes transactions contenues dans les différents blocs.

Pour cela, il faut mettre en place une stratégie permettant de lire toutes les transactions par bloc et d'établir le lien entre chacune d'elles. Cela consiste à consolider par clés publiques afin de déterminer les montants correspondants.

- *Lecture des UTXO qui ne sont présents dans des TXI :*

Détaillée plus bas (cf ici).

Cette fonction est le pivot de la fonction de consultation de solde ; toute UTXO non dépensée dont l'adresse publique correspond à l'utilisateur permet de déterminer le solde du client.

- *Consultation de solde :*

```
def balance(self, blocList, publicAddress = None):
```

En input : la liste de blocs et le compte (si rien n'est mis on renvoie la somme de tous les comptes de l'utilisateur)

En output : La somme de tous les montants correspondants à la clé publique entrée en paramètre (si rien n'est mis, la somme de tous les comptes du portefeuille).

B. Fonction d'émission de transaction

Pour qu'une transaction soit valide, il est nécessaire qu'elle vérifie les trois conditions suivantes :

- Au cours d'une transaction, il est nécessaire que les deux parties soient en possession d'un portefeuille.
- L'acheteur doit effectivement posséder la somme qu'il souhaite envoyer.
- L'acheteur n'a pas envoyé cette somme à une autre personne avec les mêmes UTXO.

Ces conditions seront vérifiées ultérieurement grâce au *composant #3 « Mineur »*.

Pour émettre une transaction, valide ou non, nous devons avoir connaissance d'un montant et de l'adresse publique du récepteur. L'émetteur doit « signer » la transaction avec son adresse privée.

Nous avons donc les fonctions suivantes :

- *Envoyer les paramètres de la transaction au Composant 6 :*

Cette fonction permet de convertir les données pour qu'elles soient utilisées dans la fonction signature (développée par le groupe du composant n° 6).

```
def concatTransactionParameters(self, sender_publicAdress, recipient_adress,
amount)
```

En input : L'adresse publique de l'émetteur, le destinataire du virement et le montant

En output : une chaine de caractère qui retourne la concaténation des données en entrées + date de la transaction.

Choix des UTXO pour réaliser la transaction :

Pour réaliser une transaction, il faut réussir à trouver des UTXO qui correspondent au montant de la transaction.

```
def selectUtxoForTransaction(listUtxoNotSpend, amount):
```

En input : La liste des UTXO non dépensés et le montant à chercher.

En output : La liste des UTXO ou une erreur (dans le cas où le montant des UTXO est insuffisant).

- *Modifier le format UTXO en TXI pour la transaction :*

Dès lors que les UTXO ont été trouvés (voir fct « *Choix des UTXO pour réaliser la transaction »)*; pour qu'on puisse les utiliser dans la transaction on créer une nouvelle liste qui reprend les informations des UTXO dans une nouvelle liste de TXI.

```
def convertUtxoInTxi(self, listUtxos, sign):
```

En input : La signature et la liste des UTXO (fonction « Choix des UTXO pour réaliser la transaction »).

En output : La liste des TXI.

- *Générateur de transaction :*

Cette fonction permet de créer une transaction en la signant et transmet la liste des TXI correspondants à la liste de tous les UTXO sélectionnés pour la transaction. Elle renvoie les deux UTXO créés pour l'émetteur et le destinataire.

Une transaction contient des inputs et des outputs :

```
def transaction(self, sender_privateAdress, recipient_adress, amount, blocList):
```

En Input : L'adresse privée de l'émetteur de la transaction, l'adresse publique du destinataire, le montant et la blockchain.

En Output : La liste des TXI correspondantes, 1 UTXO pour l'émetteur et 1 UTXO pour le destinataire.

Dès lors que la transaction est créée et signée, elle est renvoyée aux composant 4.

C. Lecture des blocs et recensement des UTXO

Pour permettre la visualisation des éléments de la blockchain, il est nécessaire de créer des fonctions qui nous permettent de récupérer les informations de ces derniers.

Ainsi, nous avons les fonctions suivantes :

- *Lecture des TXI :*

Récupère la liste des TXI issue de la liste de bloc en entrée.

```
def retrieveTXIs(self, blocList):
```

En input : La liste des blocs (fonction « *Lecture de bloc* »)..

En output : La liste des TXI.

- *Lecture des UTXO :*

Récupère la liste des UTXO issue de la liste de bloc en entrée.

```
def retrieveUTXOs(self, blocList)
```

En input : La liste des blocs (fonction « *Lecture de bloc* »).

En output : La liste des UTXO.

- *Lecture des UTXO qui ne sont présents dans des TXI :*

Cette fonction est nécessaire pour estimer le solde de compte. Il faut prendre toute la chaine et repérer toutes les transactions non dépensés (qui concerne le compte de notre utilisateur) afin de déterminer le solde.

```
def UTXO_not_in_TXI(self, blocList, publicAddress = None):
```

En input : La liste des blocs (fonction « *Lecture de bloc* ») ainsi que le compte (si l'on ne met rien, on retourne le solde total des crypto monnaies liées à cet utilisateur).

En output : La liste des UTXO correspondantes.

IV. Échanges menés

Notre composant se trouvant au centre du projet, nous interagissons avec beaucoup de composants. Nous devons donc clairement définir les objets que nous allons recevoir et ceux que nous devons traiter de notre côté. Pour ce faire, nous avons donc organisé des réunions avec d'autres équipes :

A. Composant 1 : liste des blocs

Les équipes qui sont en charge de développer le composant 1 ont pour objectif principal de définir tout ce qui en rapport avec *l'interface fichier.* Pour ainsi dire, ils s'occupent de représenter la blockchain et toutes les fonctionnalités qui autour de son utilisation :

- Gestion des fichiers
- Créer la blockchain (ajout du bloc « Genesis » en début de chaîne)
- Ajout de bloc en fin de chaîne

Une interdépendance sur un point primordial a été soulevée : qu'elle équipe s'occupe de la *gestion de la liste de bloc* ? Pour effectuer quelque fonction du Wallet soit-il, il est nécessaire de posséder la blockchain en argument. Pour récupérer le solde par exemple, il faut récupérer toutes les UTXO non dépensées, information uniquement disponible dans la liste de bloc.
Nous avons donc contacté l'équipe du composant 1 pour essayer de comprendre qu'elle sera la solution la plus logique et propre. Comme le composant 1 se charge de gérer la

blockchain en sa globalité, il parait raisonnable de s'attendre à une fonction qui nous renvoie la liste des blocs, que nous traiterons à la suite.
Après discussion, nous nous sommes accordés sur cette décision : le *composant 1* nous fournit une fonction capable de nous retourner une liste de blocs.

B. Composant 4 : fonction de hash + création de blocs

Le composant 4 possède une double fonctionnalité : il est en charge de réaliser la fonction de hash ; ce qui permet aux blocs d'obtenir leur hash individuel.
Dans un second temps, l'équipe devrai s'assurer de la création des blocs.
La problématique soulevée auprès de notre équipe a été de déterminer quel composant aurait besoin du retour de notre fonction « *transaction* ».
Comme attendu, la conclusion des discussions a été qu'après la formation d'une transaction et sa signature, la transaction est transmise au *composant 4.*

C. Composant 6 : Signature

Nous avons contacté l'équipe chargée du composant 6 pour déterminer le format de la fonction signature ; c'est-à-dire ce que la fonction signature attends comme paramètre et ce qu'elle renvoie comme résultat.