Projet de développement n°24

2017-2018

Comprendre et implémenter les mécanismes du Bitcoin

Rapport technique - Version 2

Pour l'équipe pédagogique du projet S2

IMT Atlantique – Campus de Brest

13 Juin 2018

ENCADRANTE : Elsa DUPRAZ.

AUTEURS : Maroua EL HOUICHA, Marouane MAACHOU, Mehdi DAHOUMANE et Fayçal HAFID.

Résumé

Les crypto-monnaies ont aujourd'hui réussi à s'imposer dans les milieux financiers et à susciter l'intérêt d'une grande partie de la population. Accessibles à tous -des professionnels du domaine aux simples usagers-, elles sont de plus en plus utilisées. C'est la raison pour laquelle nous avons cherché à comprendre et à implémenter les mécanismes du Bitcoin, qui est la crypto-monnaie la plus répandue. Dans un premier temps, nous nous sommes renseignés sur le fonctionnement des crypto-monnaies, les bases du Bitcoin, les blockchains et sur les protocoles de cryptographie nécessaires pour assurer la fiabilité des transactions effectuées. Ensuite, nous sommes passés à une seconde partie, axée développement, dans laquelle nous avons implémenté notre propre blockchain et notre propre crypto-monnaie. La plateforme réalisée au cours de ce projet s'adresse principalement à des enseignants souhaitant introduire à leurs élèves les concepts relatifs aux crypto-monnaies ou à des professionnels voulant réaliser des tests de sécurité. En outre, elle permet aux utilisateurs de créer un compte, consulter leur solde et effectuer des transactions, en toute sécurité grâce aux protocoles employés.

Rédactrice : Maroua EL HOUICHA
Relecteurs : Fayçal HAFID, Marouane MAACHOU, Mehdi DAHOUMANE

Plan du rapport

Introduction

Depuis sa création en 2009, investisseurs, entrepreneurs, professionnels et particuliers utilisent de plus en plus le Bitcoin. Plus généralement, les crypto-monnaies ont révolutionné le marché monétaire, remettant en question tout le système monétaire courant.

Le but de notre projet est dans un premier temps la compréhension des bases du Bitcoin, puis l'implémentation de ses mécanismes. La détermination des cas d'utilisation de notre crypto-monnaie et des besoins des utilisateurs a été primordiale au commencement du projet. Dès lors, nous avons cherché à développer une plateforme ergonomique permettant aux utilisateurs d'ouvrir un nouveau compte, de consulter leur solde et d'effectuer des transactions depuis ce compte tout en assurant la confidentialité des utilisateurs et la traçabilité des transactions. Pour cela, il faudra faire usage de fonctions de hachage et de chiffrement implémentées dans la blockchain. La plateforme devrait aussi être accessible depuis plusieurs machines.

Dans ce rapport, nous commencerons par expliciter les mécanismes fondamentaux sur lesquels se base le Bitcoin que sont le hachage , la paire clé publique/privée en cryptographie et la Blockchain. Et puisque nous avons choisi d'utiliser la programmation orientée objet en Python, nous décrirons les différentes classes créées pour implémenter ces mécanismes. Ces classes sont au nombre de trois : La classe Bitcoin, la classe User et la classe Transaction. Ensuite, nous détaillerons la programmation de la plateforme et de la blockchain et les résultats obtenus. Pour finir nous exposerons certains aspects souhaitables mais optionnels que nous avons souhaité réaliser . Ces aspects sont : une interface graphique agréable et ergonomique , la communication entre plusieurs utilisateurs avec l'application installées sur différentes machines mais aussi l'implémentation d'une application mobile permettant de gérer les bitcoins.

Rédactrice : Maroua EL HOUICHA
Relecteurs : Fayçal HAFID, Marouane MAACHOU, Mehdi DAHOUMANE

Mécanismes du Bitcoin

1.1 Principes du Bitcoin

Le Bitcoin est la première monnaie virtuelle décentralisée. Le Bitcoin désigne aussi tout le système de paiement pair-à-pair (réseau informatique constituée de nœuds communiquant entre eux, qui sont à la fois des clients et des serveurs) basée sur des principes de cryptographie plutôt qu'un tiers de confiance (banques dans le modèle financier classique).

Chaque individu possédant des bitcoins les stocke dans son porte-monnaie électronique (wallet), et chaque bitcoin est relié à une adresse (hachage de la clef publique de son propriétaire). Les transactions Bitcoin sont intégrées dans la blockchain, qui contient ainsi des adresses de bitcoin. En effet, une transaction Bitcoin est caractérisée par ses "entrées" (input) et ses "sorties" (output). Les inputs désignent les bitcoins qu'on souhaite envoyer, déjà enregistrés dans la blockchain (puisqu'ils proviennent d'une transaction précédente) et qu'on peut déverrouiller. Alors que les outputs ne sont que les adresses "destinataires", vers lesquels on va envoyer les bitcoins de l'input. La figure ci-dessous montre l'exemple d'une transaction avec un seul input et deux outputs.

La signature ECDSA est utilisée pendant une transaction pour prouver qu'on possède bien une adresse Bitcoin. Ainsi, à l'aide de sa clé privée, on signe le bitcoin qu'on veut envoyer. La signature obtenue est envoyée dans le réseau pour être validée. La transaction inclut l'adresse Bitcoin qui n'est en fait que le hachage de la clé publique. Enfin, on valide la signature avec l'adresse Bitcoin. Comme on peut le remarquer, tout le monde peut connaître la clé publique et l'adresse bitcoin, mais la clé privée doit rester secrète. Perdre sa clé privée, c'est perdre ses bitcoins.

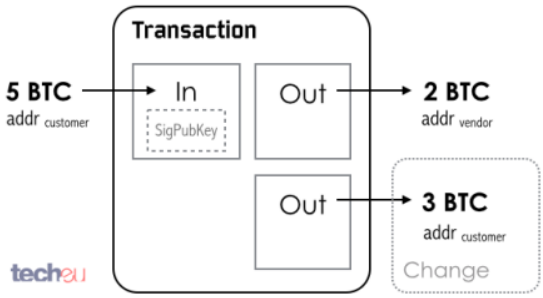

Figure 7: a single input-multiple output Bitcoin transaction
Source: http://tech.eu/features/808/bitcoin-part-one/
(Consulté le 22/05/2018)

Lorsque le "sender" valide la transaction, celle-ci va être validée ou rejetée (en cas de fraude par exemple) par le réseau Bitcoin. Dans le premier cas, elle va être intégrée dans la blockchain et ainsi elle devient définitive.

Rédactrice : Maroua EL HOUICHA
Relecteurs : Fayçal HAFID, Marouane MAACHOU, Mehdi DAHOUMANE
Informations appuyées sur les références [3] et [4]

1.2 Le hachage

1.2.1 Les fonctions de hachage

Soit H une fonction prenant en entrée une donnée de taille aléatoire m et donnant en sortie un condensé de taille n. Une telle fonction est appelée fonction de hachage.

$$H: \{0,1\}^* \rightarrow \{0,1\}^n$$
$$m \mapsto H(m)$$

Pour représenter et identifier une donnée, on utilise son image$H(m)$ par la fonction de hachage: c'est ce que l'on nomme l'empreinte de la donnée. Ces fonctions de hachage sont essentielles en informatique. Elles sont notamment à l'origine des tables de hachage, structure de données permettant une association clé-valeur. Les tables de hachage se présentent sous forme de tableaux stockant des données de natures diverses. Chacune des données possède un identifiant et l'accès à un élément se fait *via* l'empreinte de cet identifiant calculée avec la table de hachage. De cette manière, il est possible de trouver efficacement un élément dans une base de données. Cependant, dans le cadre de notre projet, des propriétés de sécurité sont exigées pour pouvoir accéder à un élément. On n'utilisera donc pas des fonctions de hachage classiques mais des fonctions de hachage cryptographiques

1.2.2 Les fonctions de hachagecryptographiques

a) Propriétés :

Pour qu'une fonction de hachage puisse être cryptographique, il faut qu'elle vérifie les trois propriétés suivantes :

- résistance aux pré images.
- résistance aux secondes pré images.
- résistance aux collisions.

Première propriété : la résistance aux pré images.
La première propriété que doit vérifier une fonction de hachage cryptographique est la résistance aux pré images.Soit h un haché choisi aléatoirement. Trouver une pré image, c'est trouver un message m tel que H(m)=h.Pour des raisons de sécurité, ce problème doit être un problème classé difficile.

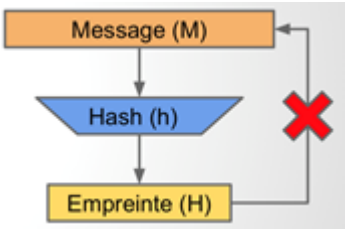

Figure 1

Deuxième propriété : la résistance aux secondes pré images.
Le principe est sensiblement le même que celui de la résistance aux pré images.Soit m un message choisi aléatoirement. Trouver une seconde pré image, c'est trouver un message m' tel que H(m)=H(m').Encore une fois, pour qu'une fonction de hachage ait de bonnes propriétés cryptographiques, il faut que ce problème soit difficile.

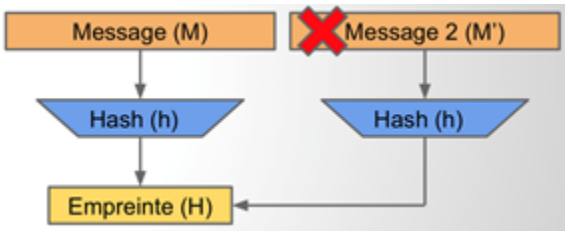

Figure 2

Troisième propriété : la résistance aux collisions.
On parle de collision lorsque deux entrées distinctes conduisent à la même sortie.
Plus formellement, trouver une collision c'est trouver deux messages m et m' tels que $m \neq m'$et$H(m) = H(m')$.

Notons qu'il existe nécessairement des cas de collision. Théoriquement, l'ensemble de départ peut être infini. En pratique, il contient uniquement les messages dont la taille est inférieure à un certain seuil : pour la fonction de hachage SHA-1 par exemple, seuls les messages de taille inférieure à 2^{64} -1 bits sont traitables. L'espace de départ est donc fini et on note Card(E) son cardinal. L'espace de sortie est également fini et on note Card(S) son cardinal. On a donc deux ensembles finis E et S tels que Card(E)>Card(S) et H une application telle que H : E->S. D'après le principe des tiroirs de Dirichlet, il existe un élément de S qui admet au moins deux antécédents par H. Finalement, il existe nécessairement des cas de collision.Dès lors, une fonction de hachage est dite « résistante aux collisions » lorsque trouver une collision est un problème classé difficile.

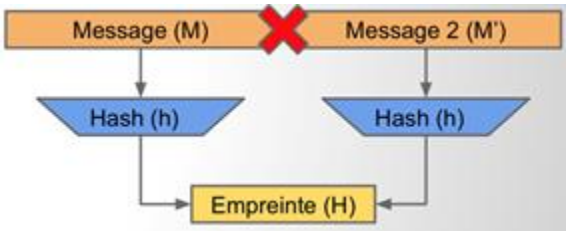

Figure 3

Une fonction de hachage possède de bonnes propriétés cryptographiques quand les trois problèmes énoncés précédemment sont difficiles. Dans le cadre de notre projet, nous serons amenés à utiliser la fonction de hachage SHA-256.

b) SHA-256

La fonction de hachage cryptographique SHA-256 présente l'avantage de présenter de bonnes propriétés cryptographiques. En effet, les trois problèmes présentés précédemment sont difficiles face à cette fonction de hachage. C'est pour cela qu'elle est principalement utilisée par plusieurs protocoles dont le protocole Bitcoin.

Dans le contexte des crypto-monnaies, il faut que les fonctions de hachage fonctionnent sur des entrées de longueur arbitraire. Pour cela, SHA-256 utilise la transformation de Merkle-Damgard. Cette transformation utilise une « fonction de compression » : c'est une fonction qui produit une sortie de taille strictement inférieure à celle de l'entrée de sorte qu'il soit difficile de retrouver l'entrée sachant uniquement la sortie. Supposons que la fonction de compression prenne des entrées de longueur m et renvoie une sortie d'une longueur n plus petite.Dans un premier temps, on divise l'entrée de la fonction de hachage (quelconque) en blocs de longueur (m-n). Ensuite, chaque bloc et chaque sortie de bloc sont passés simultanément dans la fonction de compression. Comme il n'y a pas de sortie de bloc antérieure pour le premier bloc, on utilise à la place un Vecteur d'Initialisation (InitializationVector IV) généré aléatoirement.
Le résultat du dernier bloc est l'haché renvoyé.

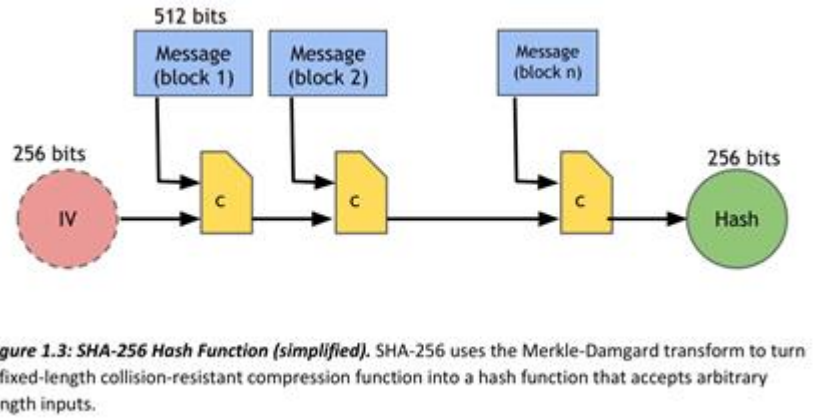

Figure 1.3: SHA-256 Hash Function (simplified). SHA-256 uses the Merkle-Damgard transform to turn a fixed-length collision-resistant compression function into a hash function that accepts arbitrary length inputs.

Figure 4 : Principe simplifié du fonctionnement de SHA-256

SHA-256 utilise une fonction de compression qui prend une entrée de 768 bits et produit des sorties de 256 bits et où chaque bloc a une taille de 512 bits.

Rédacteur : Mehdi DAHOUMANE
Relecteurs : Fayçal HAFID, Maroua EL HOUICHA ,Marouane MAACHOU
Informations appuyées sur la référence [1]

1.3 Méthodes de cryptographie et notions de clés

Dans le monde numérique, il est essentiel de pouvoir prouver l'authenticité d'une information, en l'occurrence lorsqu'on souhaite effectuer des transactions, que la monnaie qu'on veut échanger nous appartient bien et bel, et ainsi éviter les arnaques.Pour ce faire, il existe ce qu'on appelle une signature digitale avec laquelle, en analogie avec la vie réelle, l'acheteur "signe" la transaction qu'il s'apprête à réaliser.

L"ECDSA" (*EllipticCurve Digital Signature Algorithm*) est un algorithme de cryptographie utilisé pour créer une signature digitale pour des données (par exemple un fichier) qui servira à vérifier l'authenticité de celui-ci sans pouvoir la falsifier. C'est-à-dire que n'importe qui peut reconnaître une signature ECDSA mais ne peut la copier ou la trafiquer.L'ECDSA ne crypte pas les données et n'empêche pas les tierces personnes d'accéder à ces données, il ne sert qu'à assurer l'authenticité de ces données-là.Le principe de l'algorithme se base sur les mathématiques cryptographiques : en partant d'une équation qui décrit une courbe, on en sélectionne aléatoirement un point qu'on va considérer comme étant le **point d'origine**.

Ensuite on génère un nombre aléatoire, qui va représenter la **clé privée**. L'algorithme va utiliser le point d'origine choisi avec la clé privée afin de générer un second point sur la courbe : la **clé publique**.Quand on veut signer un fichier, on utilise donc notre clé privée avec le hash du fichier afin d'obtenir la clé publique qu'on pourra diffuser. La signature est divisée en deux parties : **(R, S).**

Pour vérifier que la signature est bien correcte, nous n'avons besoin que de la **clé publique** que l'on va utiliser pour faire des calculs sur la **partie S**. Si la signature est correcte et est issue de la bonne **clé privée**, on doit retrouver la **partie R** comme résultat.
Il est impossible de retrouver la clé privée ou de recréer la signature (le couple R et S) en n'ayant que la clé publique, ce qui assure l'efficacité de l'algorithme.

Principes mathématiques de l'algorithme : (En Annexe 1 figurent plus de détails)

- L'ECDSA n'utilise que les nombres entiers, donc pas de nombres contenant de virgule. La portée de ces nombres dépend de la taille en bits de la signature qu'on souhaite générer. Plus la taille est grande et plus la sécurité est assurée. Avec 32 bits nous avons 4294967296 valeurs possibles, chaque bit supplémentaire ajouté multiplie cette valeur par deux et l'ECDSA utilise généralement 160 bits ce qui donne une valeur longue de 49 chiffres et assure donc une bonne efficacité.
- L'ECDSA se base sur une courbe de la forme :

  $$y^2 \equiv (x^3 + a\,.\,x + b\,)[p]$$

  Ce qu'on remarque rapidement sont la présente du modulo et la quadrature du y. La seconde signifie que pour chaque x, nous avons deux valeurs de y symétriques possibles. Pour le modulo, le nombre p est un nombre premier qui sert à s'assurer que la valeur donnée par $(x^3 + a\,.\,x + b\,)$ ne dépasse pas la valeur à 160bits définie par l'algorithme, et comme c'est un modulo, les valeurs que peut prendre y^2 sont entre 0 et p-1 : on a p valeurs possibles. De plus, comme nous avons précisé qu'on n'utilisera que les valeurs entières, cela nous laisse avec un petit ensemble de nombres qui sont égaux au carré d'un autre nombre (qu'on peut donc écrire sous la forme y^2). On possède alors N valeurs possibles avec $N < p$.
  Pour résumer, l'équation de l'ECDSA fournit une courbe dont le nombre de points sont finis (=N) car l'axe des coordonnées est contraint par le modulo p et par le fait qu'il ne contient que des carrés parfaits avec une symétrie sur l'axe des ordonnées. On a alors un total de $\frac{N}{2}$ valeurs valides pour les x, sans oublier que $N < p$.

- Ainsi pour implémenter l'ECDSA, on doit connaitre les paramètres suivants :
  - a, b, p : paramètres de la courbe.
  - N : nombre de points sur la courbe.
  - G : le point d'origine ou de référence choisi arbitrairement, qui peut être n'importe quel point de la courbe.

  Sans ces paramètres, on ne peut générer ou vérifier une signature.
- La **clé privée** $\boldsymbol{dA}$ est un nombre aléatoire sur 160 bits.
  La **clé publique** $\boldsymbol{Qa}$ est un point de la courbe qu'on génère en multipliant **G** par la **clé privée**.
  On a alors : $\boldsymbol{Qa = dA * G}$**.**
- **Création d'une signature :**

Une signature est codée sur 40 octets sous la forme (R, S), R et S étant chacun sur 20 octets.

1- Premièrement, on génère un nombre aléatoire k sur 20 octets.

2- On retrouve le point P tel que $P = k * G$. La coordonnée de P, sur 20 octets, représentera la partie R de la signature.

3- Pour calculer la partie S, on utilise l'algorithme de hachage SHA1, on appellera Z le hash du fichier donné, qui sera également sur 20 octets.

On retrouve S en utilisant l'équation :

$$S \equiv k^{-1}(Z + dA * R\ )\ [p]$$

Comme nous travaillons avec des nombres entiers, k^{-1} est défini tel que

$$k * k^{-1} \equiv 1[p]$$

Rédacteur : Fayçal HAFID
Relecteurs : Maroua EL HOUICHA, Marouane MAACHOU, Mehdi DAHOUMANE
Informations appuyées sur la référence [2]

1.4 Blockchain et lien avec le Bitcoin

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*).Pour mieux comprendre cela, la blockchain est tout simplement une chaîne (succession) de blocs contenant des informations qui sont accessibles par tous sans intermédiaire, ce qui permet à chacun de vérifier sa validité. On peut donc assimiler la blockchain à un grand livre comptable public, décentralisé et infalsifiable.

La blockchain et le Bitcoin ont été créés ensemble, mais l'usage de la première ne se limite pas qu'aux cryptomonnaies et peut-être généralisé à tout, la limite de son usage se confondant avec les limites de l'imagination humaine. En effet, certaines universités utilisent la blockchain pour stocker les diplômes de ses étudiants à la fin de leurs études afin qu'il ne puisse pas y avoir de faux prétendants de ce diplôme, ou encore un système de vote peut potentiellement être implémenté en passant par la blockchain et éviter ainsi les fraudes.

Dans le contexte du Bitcoin, dans un système traditionnel il faudrait un banquier qui tiendrait ce livre des comptes et à qui il faudrait s'adresser si on souhaitait effectuer une transaction. Comme la blockchain est accessible par tous et qu'il en existe de multiples exemplaires -chacun en possède une copie-, toute écriture effectuée au sein de l'une de ces copies apparaîtra sur les autres. Si la majorité est d'accord, cette écriture y sera validée et deviendra impossible à modifier ou retirer. Ce processus de validation est basé sur la transparence et le consensus, et repose donc sur la preuve au lieu de la confiance comme c'est le cas avec une banque.Un bloc possède un hash : il identifie un bloc et tout son contenu et est toujours unique.Une fois qu'un bloc est créé, son hash est en cours de calcul et modifier son

contenuprovoquera une modification de son hash. Le hash permet donc de repérer les modifications de blocs.En plus de son hash, un bloc contient également le hash du bloc précédent. Grâce à cela,une chaîne de bloc peut être donc être constituée. Chaque bloc contenant son hash et celui du bloc précédent et chaque bloc étant non modifiable (à moins de changer le hash), il est très difficile de falsifier la blockchain. Lorsqu'un utilisateur effectue une transaction, elle est envoyée sur la blockchain et entre en instance de validation. Des utilisateurs viennent alors apporter leur puissance de calcul afin de consulter l'historique des transactions, confirmer si le montant de la transaction estdisponible dans le portefeuille de celui qui souhaite l'effectuer et calculer le hash du nouveau bloc le cas échéant : ce sont les « mineurs ». Après cela, le hash du nouveau bloc est communiqué au réseau et si le résultat est convenable, le mineur est récompensé par des Bitcoins et le bloc est incorporé à la blockchain.

Par ailleurs, une blockchain se doit d'être infalsifiable et l'utilisation des hachages n'est pas suffisante pour empêcher la falsification. Des ordinateurs rapides peuvent calculer des centaines de milliers de hash par seconde si bien qu'il pourrait être possible de falsifier un bloc, recalculer tous les hachages des autres blocs et revalider la blockchain. Pour éviter cela, on utilise la « preuve par travail ».En effet, les blocs forment une sorte de puzzle où chaque pièce ne peut s'enchaîner qu'avec la précédente, ce qui empêche les fraudes. Ceci se concrétise par la "preuve par travail" dont le but est de rendre extrêmement difficile toute réécriture, même partielle, d'un bloc de la blockchain. Pour ce faire, un bloc devra posséder une empreinte particulière afin qu'il soit accepté par l'ensemble des noeuds. Par exemple, si nous fixons notre difficulté à :
difficulty = 0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

Le principe de preuve consiste à trouver un nombre "tare" (en anglais "nonce") telle que : SHA256(texte + tare) <difficulty. Autrement dit, de trouver une chaîne qui, ajoutée à notre texte, donnerait un hash qui commencerait par 000. En tenant compte des caractéristiques d'une fonction de hashage, le seul moyen serait d'essayer des valeurs aléatoires (ou d'en prendre une et ne faire que l'incrémenter) jusqu'à tomber sur une valeur pour laquelle le résultat donnerait ce qu'on cherche.

```
# for i in {0..10000} ;do s=$(echo "$t+$i" | sha256sum -);echo "$s $i" ; [ "$(echo $s|cut -b -3)" == "000" ] && break ;done

99bf33aa466e1c01abe273e84cd161ee743aa4b214ffb25847345247008cb60b  - 0
19f2d870bd82c481813756df8905b1b9d69a364d7ab3ca30d2b3ea5503b1d647  - 1
b3cc684841c8f93527144821e5c2208dfd2879f607ac3855eb6cd7d1449cabd0  - 2
7703903253c2579da22b1b7e417891f62882d2ce3456d6bf5a7bffc151c16480  - 3
99f58bc59f5b49c4a044e66fec809155a26bced8532627c1cdf7ce7247485652  - 4
2d4420d4beea97475ce600daed7669fe936f0af50cfc265b6306b64229aff439  - 5
(...)
898ea41193317248a2cf21c12c13364b7c6897d7dca2da99309aedd48090efc6  - 3071
874fcdc16f84c493a1a3f8ab650093901f93b1e6a6f02bc269ef65da1ce056ad  - 3072
6ebb67f2e4936e178c36ecc029cd447b6245563efc8cc4c2a91f13a77fb1bf77  - 3073
f07b19296c246bcb2eaaba7a69c53c539f4292532273723b2119e58eb99d3768  - 3074
95c9ed201543e4a5ddbfb917ed9bd49626d5ed94218f94033d35881114dc3395  - 3075
000c4a032ca7d39734759da08c672cf50e1554a3671b5a3ddccd389c44bb5db8  - 3076
#
# echo 'Ceci est mon document !+3076' | sha256sum -
000c4a032ca7d39734759da08c672cf50e1554a3671b5a3ddccd389c44bb5db8  -
```

Figure 8 : Exemple de preuve par travail

Ainsi, si le "texte" (auquel il faudra rajouter la tare) est issu du bloc précédent, il sera impossible de rajouter un bloc entre deux ou d'en supprimer un. La preuve par travail rend très difficile la modification des blocs car si un bloc est modifié, il faut recalculer la « preuve par travail » pour tous les blocs suivants, sachant que pendant que ce calcul sera en cours, d'autres blocs seront créés et il faudra alors avoir une puissance de calcul bien supérieure à celle de tous les autres nœuds réunis.On comprend donc que la sécurité d'une blockchain tient sur le hachage et sur la preuve par travail mais pas seulement.

En étant distribuée, n'importe qui peut rejoindre la blockchain et peut donc en obtenir lacopie. Chaque nœud peut donc vérifier que tout est en ordre. Ainsi, quand un bloc est créé, ce bloc est envoyé à tous les nœuds du réseau. Chaque nœud vérifie alors le bloc ets'assure qu'il n'ait pas

été falsifié. Si tout est correct, chaque nœud ajoute ce bloc à sapropre blockchain. Tous les nœuds du réseau forment alors un « consensus » : ils semettent d'accord sur les blocs valides et ceux qui ne le sont pas. Les blocs altérés sontrejetés du réseau.
Finalement, la blockchain est un registre de transactions accessible à tous rendu infalsifiable par les mécanismes de hachage, de « preuve par travail » et par sa structure distribuée.

Rédacteurs : Fayçal HAFID, Mehdi DAHOUMANE
Relecteurs : Maroua EL HOUICHA ,Marouane MAACHOU
Informations appuyées sur les références [3] et [4]
Figure issue de la référence [4]

Développement et implémentation

L'implémentation s'est déroulée en deux étapes principales : La première fut de mettre en place une plateforme d'utilisation interactive qui définisse grossièrement les fonctions devant être implémenté : La consultation du solde et la transaction. Ces deux fonctions ont alors entrainé l'implémentation de trois classes : La classe Bitcoin représentant des objets qui seront notre monnaie virtuelle, La classe User permettant de matérialiser les différents utilisateurs de notre application et la classe transactions permettant de définir les contours d'une transaction donnée et de l'instancier dans un objet propre.

Notre implémentation a repris les mêmes algorithmes de cryptographie et de hachage que le Bitcoin . En effet, nous avons utilisé ECDSA pour la génération des clés privé/publique mais aussi l'algorithme de hachage SHA- 256 .D'autres algorithmes de sécurité beaucoup plus compliqués sont utilisé au niveau du Bitcoin mais nous avons fait le choix de ne pas les implémenter car notre modèle est censé rester un modèle simplifié à but pédagogique. C'est aussi dans un soucis de simplicité que notre Blockchain est très simplifiée. En effet, le minage pour la création des blocks est impossible.

2.1 Plateforme d'utilisation

La plateforme permettant l'utilisation de ces bitcoins est une plateforme à connexion. En effet chaque utilisateur s'authentifie en utilisant un ID unique et un mot de passe. Ces informations sont stockées dans une base de donnée inhérente au système. Cette base de donnée se présente sous la forme d'une base de donnée SQLite que l'on manipule grâce à des commandes python.
Une fois authentifié, l'utilisateur a accès à plusieurs fonctionnalités ou possibilités:
-Effectuer une transaction
-Consulter son solde de Bitcoins
-Consulter l'historique de ses transactions
-Se déconnecter
Toutes ces possibilités sont représentées par des fonctions manipulant l'ensemble des classes et méthodes citées précédemment.

```
Bienvenue dans l'application.
Donnez votre ID : Marouane

donnez votre MDP : 1
Vous êtes connectés. Bienvenue :)

Entrez le numéro de votre opération :
1- Consulter votre solde
2- Deconnexion
3-Consulter l'historique de vos transactions
4-Effectuer une transaction
```

Figure : Plateforme d'utilisation

Rédacteur : Marouane MAACHOU
Relecteurs : Fayçal HAFID, Maroua EL HOUICHA , Mehdi DAHOUMANE

2.2 Les classes de développement

Comme il existe différentes interactions entre Bitcoin, Utilisateurs, et Transactions et qu'il existe plusieurs opérations que l'un peut utiliser sur l'autre ou sur soi, nous avons opté pour un paradigme orienté objet centralisé autour des trois classes : Bitcoin, Utilisateurs et Transactions.

Ces classes sont récursives, dans le sens où on retrouve l'une dans ses propres attributs et dans les attributs des autres. Par exemple, la classe Bitcoin possède comme attribut une liste des différents Bitcoins (donc d'objets de même nature dans la classe elle-même) existants.
Une vue globale des classes implémentées est donnée dans la figure suivante dans laquelle on trouve en rouge les attributs de classe (communs à toutes les instances de la classe) et en bleu les attributs d'objet (spécifiques à chaque instance).

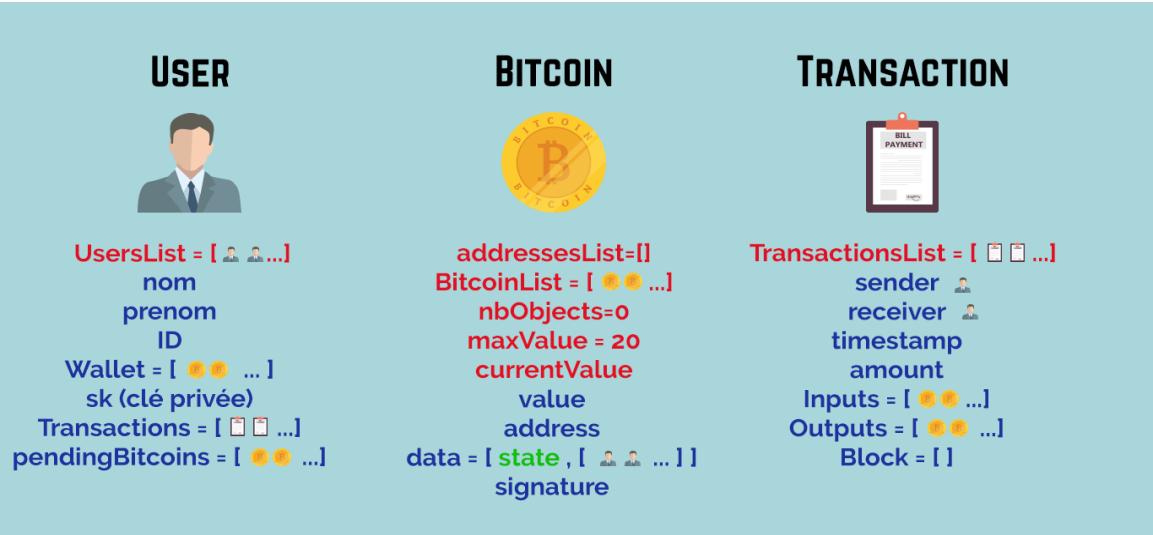

Figure 9 : Vision globale des classes
En rouge : attributs de classe, en bleu : attributs d'instance

2.2.1 La classe "Bitcoin"

Pertinence :

Dans notre démarche de programmation nous avons décidé d'adopter une philosophie de programmation orientée objet pour répartir les variables et les fonctions dans un objet contextuel :il fut donc évident qu'il faudrait concrétiser cela par une classe qui représenterait l'objet Bitcoin; une

classe qui regrouperait des attributs qui font la particularité du bitcoin; mais aussi des méthodes qui permettent de manipuler ces bitcoins de manière aisée.

Implementation :

Les attributs :

Attribut	Type		Utilité
addressesList	Liste	De classe	Cette liste contient les adresses de tous les Objets Bitcoins Instanciés
BitcoinList	Liste d'objets Bitcoin		Cette liste contient tous les objets Bitcoin instanciés
nbObjets	Entier		Représente le nombre d'objets bitcoins instanciés
maxValue	Float		Représente la valeur maximale des valeurs cumulées des bitcoins en circulation
CurrentValue	Float		Représente la valeur cumulée des bitcoins en circulation
value	Float	D'instance	Représente la valeur de l'objet Bitcoin.
address	String		L'adresse du bitcoin créé.
data	Liste		Cette liste contient des informations essentielles sur l'objet puisque le premier élément de la liste est un entier qui représente l'état du Bitcoin . Si ce dernier vaut 0 alors le bitcoin est disponible pour une transaction. Le deuxième élément de la liste data est une autre liste contenant des objets de la classe User qui représente tous les anciens détenteurs de ce Bitcoin
signature	String		La signature unique qui caractérise de façon unique un objet de la classe Bitcoin.

Les méthodes:

La méthode str :

Cette méthode est une méthode classique qui permet de récupérer les informations essentielles sur les objets Bitcoin sous forme de chaîne de caractères.

La méthode getCurrentUser:

Cette méthode permet de récupérer un objet de la classe User qui représente le détenteur actuel de l'objet Bitcoin.

La méthode addUser:

Cette méthode permet de rajouter un élément de la classe User qu'elle prend en paramètre à la liste des détenteurs de l'attribut data.

La méthode giveBitcoin:

Cette méthode permet de donner l'objet bitcoin sur lequel elle agit à un User qu'elle prend en paramètre.

La méthode PreviousOwners:

Cette méthode permet de retourner sous forme de chaîne de caractères une liste de l'ensemble des détenteurs de l'objet bitcoin du plus récent au plus ancien.

La méthode Pending:

Cette méthode est appelée sur chaque bitcoin d'une transaction. Permet de faire basculer le bitcoin cible du portefeuille du sender vers la liste pendingBitcoins du sender en mode "Out" ainsi que vers la liste pendingBitcoins du receiver en mode "In". En plus de cela, elle appelle aussi la méthode sign de la classe User pour faire signer le bitcoin cible par le sender.

La méthode PutInRightPlace:

Cette méthode permet de trier les bitcoins dans le Wallet du User en fonction de leur valeur.

La méthode ShatterBitcoin:

Cette méthode permet de subdiviser un bitcoin en deux bitcoins de valeurs choisies et dont la somme est la valeur du bitcoin divisé. Elle instancie ainsi deux objets de la classe bitcoin qui héritent du bitcoin d'origine la data.

Rédacteur : MarouaneMaachou
Relecteurs : Fayçal HAFID, Maroua EL HOUICHA , Mehdi DAHOUMANE

2.2.2 La classe "User"

Attribut	Type		Utilité
UsersList	Liste d'objets User	De classe	Permet de recenser les différents utilisateurs. A chaque création d'un utilisateur, on rajoute celui-ci à la *UsersList*.
nom	String	D'instance	Identifient l'utilisateur.
prénom	String		
ID	Entier		
Wallet	Liste d'objets Bitcoin		Sera le portefeuille de l'utilisateur, qui représentera une liste des bitcoins qui sont à la disposition de l'*User*.
sk	Objet ECDSA		"*sk*" pour Signing Key, générée grâce à la méthode generate() du module SigningKey du package ECDSA. *sk* représente la clé privée de l'utilisateur, elle est donc invariable et ne doit pas être accessible par une tierce personne. Au début de chaque transaction, elle sera utilisée pour générer une clé publique (une nouvelle à chaque fois) qui servira à signer un bitcoin de la transaction.
Transactions	Liste d'objets Transaction		La liste des transactions auxquelles l'utilisateur a participé, que ce soit en tant qu'acheteur ou en tant que vendeur. Ainsi, elle permettra l'accès à l'historique des transactions d'un utilisateur

			donné.
pendingBitcoins	Liste d'éléments : ***[state,bitcoin, Transaction]*** ***State***:peut valoir soit "In", soit "Out", selon le rôle de l'*User* dans la *Transaction*		Liste temporaire, c'est-à-dire qu'elle est censée être vide la plupart du temps. Lors d'une transaction, les bitcoins que l'acheteur va donner au vendeur quittent son portefeuille et rejoignent la liste pendingBitcoins de l'acheteur en mode "Out", et rejoignent aussi la liste pendingBitcoins du vendeur en mode "In". Une fois la transaction traitée par la Blockchain, cette liste sera vidée. Si elle aura été validée, les bitcoins qu'elle contenait rejoindront le *Wallet* du vendeur, sinon ils retourneront dans celui de l'acheteur.

Les méthodes implémentées pour la classe :

- Méthodes qui permettent de récupérer un utilisateur :

Les méthodes suivantes permettent de récupérer l'instance d'un certain utilisateur à partir de certaines informations, ce qui est essentiel pour pouvoir réaliser des opérations à partir de la plateforme d'usage.

- *getUserFromName(nom, prenom)* : retourne l'instance à partir des Nom et Prénom. (Ne retourne rien s'il n'existe pas)
- *getUserFromID(ID)* : retourne l'instance à partir de l'identifiant ID de l'utilisateur. (Ne retourne rien s'il n'existe pas)

- Méthodes utilisées par une instance User :

Les méthodes suivantes sont utilisées par une instance *user* de la manière suivante : *user.methode(arguments)*.

- *getSolde()* : retourne le solde de l'utilisateur, autrement dit le cumul les valeurs des bitcoins se trouvant dans le *Wallet* de l'*user*.
- *sign(bitcoin)* : permet de faire signer le bitcoin donné en argument et ce par l'acheteurpour prouver qu'il est en sa possession lors d'une transaction.
- *verify(bitcoin, signature)* : permet de faire vérifier la signature du bitcoin tous deux donnés en argument et ce par le vendeur durant la vérification de la transaction dans la blockchain.

- Méthode d'affichage :

Méthode classique et nécessaire afin de pouvoir faire str(user) pour des affichages.

- *__str()__* : retourne en chaine de caractère le nom et le prénom de l'utilisateur qui servent à l'identifier.

Rédacteur : Fayçal HAFID
Relecteurs : Maroua EL HOUICHA ,Marouane MAACHOU, Mehdi DAHOUMANE

2.2.3 La classe "Transaction"

Attribut	Type		Utilité
TransactionsList	Liste d'objets Transaction	De classe	liste des transactions existantes

Sender	Objet User	D'instance	Instance de l'acheteur
R*eceiver*	Objet User		Instance du vendeur
Amount	Float		Valeur que veut transférer le *sender* au *receiver*
Timestamp	Objet		La date de la transaction, nous utilisons la méthode datetime.now() du module date de Python à chaque fois.
Inputs	Liste d'objets Bitcoin		Les bitcoins qui retourneront dans le portefeuille du sender à l'issue de la transaction
Outputs	Liste d'objets Bitcoin		Les bitcoins qui iront dans le portefeuille du receiver à l'issue de la transaction
Block	Liste d'objet Block (Cf. Blockchain)		Liste vide si la transaction n'a été répertoriée dans aucune BlockChain, ou bien contient seulement comme élément le bloc dans lequel elle figure dans le cas contraire. Cet attribut sert surtout à ne pas traiter une même transaction deux fois : elle est ignorée par la blockchain si elle voit qu'elle figure déjà dans un bloc.

.

Contrairement à la classe *User*, on ne rajoute pas une transaction à la liste "*TransactionsList*" dès son instanciation : plusieurs choses se passent à ce moment là.

- Si le **solde du sender** est inférieur à **amount** (la valeur qu'il souhaite transférer), on l'en avertira et on ignorera l'instanciation de la transaction. A lui d'en refaire une nouvelle, celle-ci ne sera pas ajoutée à la liste *TransactionsList* et n'existera donc nul part. Si tout va bien à ce niveau là, on continue :
    - A ce moment, si le sender a suffisamment d'argent, nous n'avons aucune raison de ne pas rajouter la transaction à la liste : on le fait. En effet, si par exemple le Bitcoin n'appartient pas réellement à l'utilisateur, cette vérification ne se fera pas à ce niveau là mais plutôt lors de l'intégration de la transaction à la blockchain.
      A présent, on parcourt le portefeuille du sender à la recherche du bitcoin qui vaut la valeur du amount, ou bien qui vaut la valeur immédiatement au dessus.
      Par exemple si le portefeuille contient [0.01 ; 0.2 ; 2 ; 3.5] et qu'on souhaite transférer 1.1 bitcoin, nous prendront le bitcoin de valeur "2".
      Ceci fait, nous monétiserons au besoin le bitcoin récupéré afin de créer deux nouveaux bitcoins (l'ancien, monétisé, sera détruit) : l'un qui ira dans le pendingBitcoin du receiver et ainsi dans la liste output, et l'autre qui restera en possession du sender et qui ira ainsi dans la liste input.
      Lors du transfert vers les listes pendingBitcoin, le bitcoin est signé par le sender.
      A présent, on affiche au sender que la transaction est valide et est en attente de traitement par la BlockChain.

<u>Les méthodes implémentées pour la classe :</u>

Le travail étant réalisé automatiquement lors de l'instanciation d'une transaction, la seule méthode de classe nécessaire est celle de l'affichage d'une transaction :

- __str()__ : Affichera -> *From : Nom : *nom du sender*Prénom : *prénom du sender* To : Nom : *nom du receiver* Prénom : *prénom du receiver* Amount sent : *amount**

Rédacteur : Fayçal HAFID
Relecteurs : Maroua EL HOUICHA ,Marouane MAACHOU, Mehdi DAHOUMANE

2.3 Implémentation de la Blockchain

La blockchain est elle aussi basée sur le paradigme orienté objet, notre objet en question étant le bloc (*Block*). Une blockchain sera donc une liste de blocs. Un bloc n'a donc pas besoin d'avoir des attributs de classe (il n'y a aucune information qui est censée être commune à tous les blocs) et ne possède que les attributs d'objet suivants :

Attribut	Type	Utilité
Index	Entier	l'index du bloc courant, sert à spécifier le bloc
Timestamp	OBjet	Représente la date de création du bloc, nous utilisons la méthode datetime.now() du module date de Python à chaque fois.
Data	String	Affichage de la transaction qui est contenue dans le block*.
previous_hash	String	Hash du bloc précédent et sert à assurer la continuité et la successivité des blocs.
Hash	String	Représente le hash courant, calculé en utilisant tous les attributs du bloc (ormi le hash courant lui-même, évidemment) grâce au mécanisme sha256.
Proof	Entier	La preuve par travail, c'est un nombre entier qui satisfait des conditions définies plus bas et qui témoigne de l'authenticité du bloc.

**Remarque :* Comme notre implémentation est à but éducatif et ne sera pas utilisée pour effectuer des transactions de la vie réelle, nous avons opté pour une blockchain simpliste dont chaque bloc contiendra une seule transaction contrairement à la vraie blockchain dont chaque bloc contiendrait plusieurs transactions. La contrainte qui nous poussa à faire ce choix est le fait que le cahier des charges stipule que la validation ou le refus d'une transaction ne peut dépasser les 24h, or on se retrouverait face à un problème si, en 24 heures, il n'y aurait pas assez de transactions pour former un bloc. Dans la vraie blockchain, ce problème n'a pas lieu car il y a tellement d'utilisateurs simultanés et de transactions effectuées par jour que la probabilité d'occurrence est négligeable. La chaîne "data" est donc tout simplement un affichage (une représentation sous forme de chaîne de caractères) de la transaction en question. En absence de la contrainte citée, s'il fallait mettre plusieurs transactions par bloc, on aurait pu simplement concaténer les différentes chaînes des différentes transactions.

Méthodes implémentées pour la classe :

Méthode de création d'un bloc :

- *next_block (last_block)* : permet tout simplement de créer un bloc qui sera le successeur du précédent (last_block). En effet, il ne serait pas envisageable d'utiliser directement un constructeur pour ce faire car les blocs sont reliés entre eux par des mécanismes de hashage et de preuve par travail.

Méthode utilisée par une instance de bloc :

- hash_block() : retourne simplement le hash du bloc afin d'instancier l'attribut *hash* de chaque bloc, cette méthode est utilisée dans la méthode *next_block*et doit aussi être utilisée si on fait usage d'un constructeur de la classe.

Méthode d'affichage :

- *__str__()* : cette fonction n'est pas capitale et ne sert qu'à faire des tests en affichant le *hash* du bloc et son attribut *data* (c'est-à-dire la transaction qu'il contient), son usage est purement à but démonstratif.

Méthodes utilisées pour l'implémentation d'une blockchain :

Nous venons de voir la classe Block qui représente l'élément primaire qui constitue notre blockchain, mais il ne se suffit pas à lui même car il est indépendant des classes précédemment implémentées (*User*, *Bitcoin* et *Transaction*). Le traitement est donc séparé de la classe elle-même et se fait par des méthodes extrinsèques :

- create_genesis_block() : permet seulement de créer un nouveau bloc (qui ne contiendra pas de données) qui représentera la tête de la blockchainauquelle seront rattachés les blocs la contenant.
- *routineCheck()* : cette fonction est appelée au début du traitement de chaque nouvelle transaction. Son rôle est de détecter, signaler et supprimer toute fraude de Bitcoin, par exemple le cas où on essaie de créer son propre Bitcoin et l'attribuer à l'un des utilisateurs.
- *proofOfWork(last_proof)* : permet de calculer l'attribut *proof* du bloc qu'on est en train de construire en se basant sur l'attribut *proof* du bloc qui le précède et les *data* du bloc courant. Le calcul se base sur l'incrémentation du proof précédant jusqu'à vérifier la condition décrite dans l'exemple cité dans la partie *1.4 Blockchain et lien avec le Bitcoin*.
- *transactionTreatment(last_block)* : c'est sans doute la fonction la plus importante du programme car tout le traitement (validation ou refus d'une transaction) se fait à ce niveau.
    - Etape 1 : elle crée une nouvelle blockchain si la précédente est complète, sinon elle travaillera sur la blockchain active.
    - Etape 2 : elle parcourt la liste des transactions en quête d'une transaction qui n'a pas encore été traitée (une transaction dont l'attribut *Block* sera vide, c'est-à-dire qui n'appartient encore à aucun bloc), si elle n'en trouve aucun, il ne se passe rien et la fonction sera de nouveau appelée une demi-heure plus tard, sinon on passe à l'étape suivante.
    - Etape 3 : à ce stade nous avons trouvé (au moins) une transaction en attente de validation, on commence par la traiter :
        - On cherche les Bitcoins du *sender* et du *receiver* qui sont en cours de transactions (dont le premier élément de l'attribut *data* vaut 1 et qui sont donc indisponibles)
        - Bitcoin par Bitcoin, nous vérifions la validité de la signature (grâce à la fonction *verify* décrite dans la classe *User*). Il suffit qu'au moins une signature ne soit pas valide pour considérer toute la transaction comme étant frauduleuse.

      Si toutes les signatures sont valides, on passe à l'étape 4, sinon on passe directement à l'étape 5.
    - Etape 4 : La transaction est validée, ce qui signifie que chaque Bitcoin concerné par la transaction existe à la fois dans la liste *pendingBitcoins* du *sender* en mode *"Out"* et dans celle du *receiver* en mode *"In"*. Si le Bitcoin est en mode *"In"*, on le retire de la liste et on le place dans le wallet de l'utilisateur. Si au contraire, le Bitcoin est en mode *"Out"*, on le retire seulement de la liste.

      Ces Bitcoins sont de nouveau disponibles à être dépensé par leur nouveau possesseur.

      A ce moment, on crée un bloc dont l'attribut *data* contiendra la transaction qu'on vient de valider et on le rajoute à la blockchain active.
    - Etape 5 :C'est le cas où au moins une des signatures des Bitcoins échangés n'aura pas été valide et aura donc annulé toute la transaction. Dans ce cas, chaque Bitcoin concerné par la transaction et qui existe dans la liste *pendingBitcoins* des deux

utilisateurs de l'échange doit en être supprimé s'il y figure en mode *"In"* ou *"Out"*, et retourné à son possesseur dans le cas où il y figure en mode *"Out"*.

Rédacteur : Fayçal HAFID
Relecteurs : Maroua EL HOUICHA ,Marouane MAACHOU, Mehdi DAHOUMANE

2.4 Développement optionnel

2.4.1 Communication entre plusieurs machines

Il est évident que faire usage d'une blockchain repose sur la coexistence de plusieurs utilisateurs et plusieurs noeuds/mineurs, et que ceci ne peut se faire sans assurer une communication entre les différentes machines. Aussi, il serait logique de penser à une architecture Client/Serveur, car on aurait besoin d'un serveur qui servira à : stocker les données utilisateurs (les Bitcoins leur appartenant, leurs transactions, etc…), faire appel à la fonction qui cherche des transactions à valider, etc… et qui sera en contact avec les différents nœuds de calcul et les utilisateurs souhaitant faire usage de notre service.Cependant, notre projet vise avant tout à implémenter les principales fonctionnalités d'une crypto monnaie telle que le Bitcoin, qu'on a trouvé plus facile à faire dans un premier lieu en faisant coexister sur une même machine plusieurs utilisateurs et en adoptant un modèle simplifié de la Blockchain qui ne nécessiterait pas plusieurs nœuds de calcul. Le concept de Socket proposé par Python correspond bien au raisonnement serveur/clients auquel nous avons pensé. Un socket est un "trou" en anglais, ce qui peut être visualisé comme la création d'un tunnel entre le serveur et les utilisateurs. Plus techniquement parlant, c'est une association au niveau de l'OS entre un programme qui tourne en boucle et le port de la machine et on dit d'ailleurs que le programme écoute le port qui lui a été réservé. Il écoute le port et répond aux demandes faites par ce port.Aussi, dans ce cas il faudrait départager le serveur et le client en deux scripts Pythons distincts, qui impliquerait beaucoup de modifications de notre code, notamment l'ajout de classes, attributs et méthodes et choisir un protocole réseau adapté entre le TCP et l'UDP.Au jour où ce rapport a été écrit, nous n'avons pas encore réalisé la communication entre plusieurs machines pour les raisons citées ci-dessus.

Rédacteur : Fayçal HAFID
Relecteurs : Maroua EL HOUICHA ,Marouane MAACHOU, Mehdi DAHOUMANE
Informations appuyées sur les références [5]

2.4.2 Interface graphique

La plateforme utilisée représente le minimum requis afin de tester en tant qu'utilisateur les fonctionnalités de notre implémentation et les différents cas. Créer une interface graphique demande un investissement en temps et efforts qu'on a préféré accorder en priorité aux fonctionnalités de bases définies dans le cahier des charges et laisser la création d'une interface graphique plus attirante comme étant un développement optionnel après avoir fini l'aspect technique et implémenté avec efficacité les fonctionnalités prévues.Cependant, ceci ne nous a pas empêchés de nous documenter sur la manière de créer une interface graphique et nous avons opté d'utiliser le module *Tkinter* de Python qui permet de créer des fenêtres dans lequel nous pouvons créer plusieurs types d'objets satisfaisant nos besoins, à savoir des zones de texte et d'entrées de texte, des boutons, des listes, des affichages, des formes, des images, des échelles d'incrémentation et de décrémentation, etc...

Rédacteur : Fayçal HAFID
Relecteurs : Maroua EL HOUICHA ,Marouane MAACHOU, Mehdi DAHOUMANE

2.4.3 Application Mobile

Un de nos objectifs secondaires est de mettre au point une application Android permettant d'avoir accès à des fonctionnalités intéressantes pour les utilisateurs. Une application conçue grâce à android studio qui permet à l'utilisateur de consulter son solde de bitcoins et son historique de transactions.

Rédacteur : Maroua EL HOUICHA, Marouane MAACHOU
Relecteurs : Fayçal HAFID, Mehdi DAHOUMANE

Conclusion

À travers ce projet, nous avons été amenés à comprendre les mécanismes du Bitcoin ainsi que les différentes méthodes de cryptographie et de hachage nécessaires à son développement. Nous nous sommes ainsi familiarisés avec les fonctions de hachage, les signatures électroniques, aux blockchain et à la preuve par le travail.
Par la suite, nous avons implémenté ces méthodes afin de réaliser une crypto-monnaie et une blockchain simplifiées où nous étions les seuls utilisateurs et où le nombre de bitcoin était fixé à l'avance –pas de possibilité de minage-.
Tout au long de l'implémentation, nous avons mené plusieurs séries de tests qui ont permis dans un premier temps de vérifier que les transactions s'effectuaient correctement et que la blockchain s'actualisait convenablement. Par la suite, nous avons axé les tests sur la sûreté de notre crypto-monnaie en simulant des cyber-attaques. Nous avons alors montré qu'il n'était pas possible d'échanger des bitcoins de manière frauduleuse ou d'en créer de nouveaux. Nous avons ainsi mis en évidence la robustesse de notre monnaie.
À court terme, nous avons pour ambition au moins d'explorer de nouvelles pistes pour mener à bien certaines tâches optionnelles comme la communication multi-utilisateurs, l'interface graphique ou l'application mobile et rendre ce projet encore plus abouti. Il pourrait alors servir à long terme à des professionnels souhaitant réaliser des tests de sécurité ou à des enseignants souhaitant aborder le thème des crypto-monnaies avec leurs étudiants.

Rédacteur : Mehdi DAHOUMANE
Relecteurs : Fayçal HAFID, Maroua EL HOUICHA ,Marouane MAACHOU

Annexe 1

Détails supplémentaires sur l'ECDSA

- Addition de points :

L'un des principes de l'ECDSA est la notion d'« addition de points ».

- On définit le point S résultat de l'addition d'un point P à un autre point Q de sorte à ce que, si on trace la droite (PQ), son intersection avec la courbe nous donne un point R. S sera le symétrique de R par rapport à l'axe des ordonnées. [2]

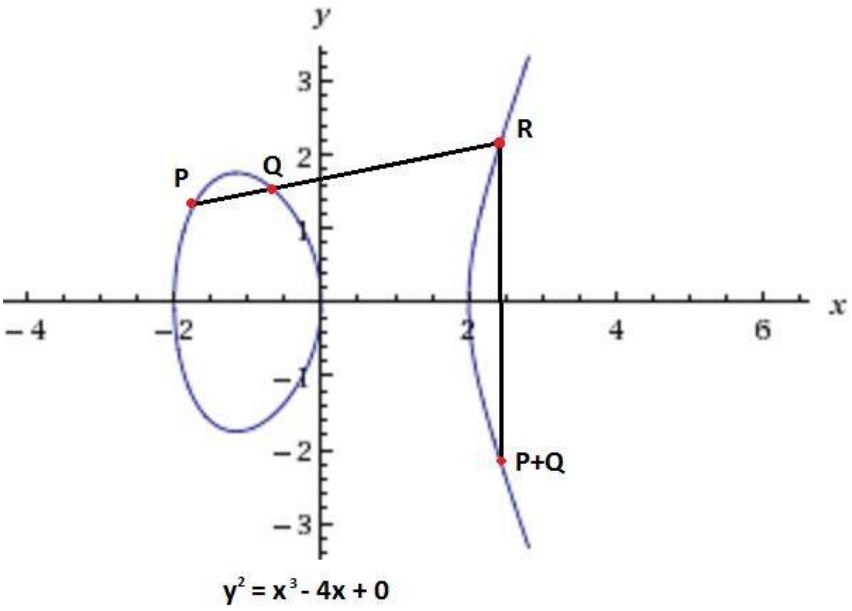

Figure 5 : Addition de points [2]

- Multiplication de points :

On la définit en partant de l'addition. On sait que a*b = a+a+…+a qu'on itère « b fois ».
En additionnant le point P à lui-même, ça sera le symétrique du point R obtenu par l'intersection de la courbe avec la tangente au point P. P+P+P s'obtient en sommant ce qu'on a obtenu de nouveau avec P, et ainsi de suite. [2] Ainsi, si on avait défini le résultat de la somme comme étant directement l'intersection avec la courbe, on obtiendrait le même point à chaque fois qu'on sommerait un point P à lui-même, d'où l'intérêt de prendre son symétrique.

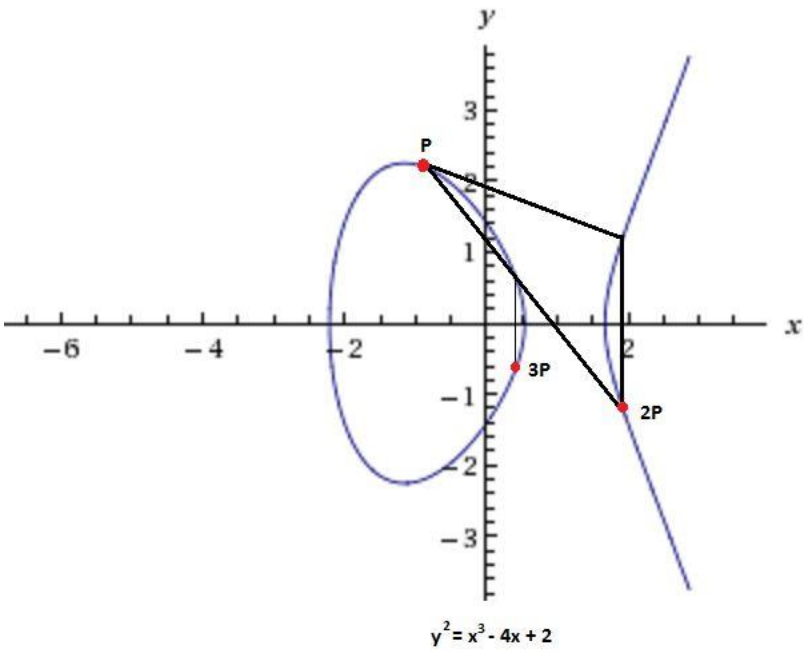

Figure 6 : Multiplication de points [2]

La particularité de la multiplication est que, si on a R=k*P, en connaissant les points R et P, il n'existe aucun moyen de retrouver la valeur de k car la soustraction ou la division de points n'est pas définie, on ne peut résoudre l'équation $k = \frac{R}{P}$.

Vérification d'une signature :

A présent qu'on possède notre signature (R, S), on veut la vérifier. Nous n'aurons besoin que de la clé publique Qa et des paramètres de la courbe. On utilise l'équation suivante pour calculer le point P :

$$P = S^{-1} * Z * G + S^{-1} * R * Qa$$

La signature est valide si et seulement si la coordonnée du point P obtenu est égale à la coordonnée du point R.[2]

Démonstration :

On a :

$$P = S^{-1} * Z * G + S^{-1} * R * Qa$$

Et $Qa = dA * G$

Alors :

$$P = S^{-1} * Z * G + S^{-1} * R * dA * G = S^{-1} * G * (Z + dA * R)$$

La coordonnée de P doit être égale à celle de R, et $R = k * G$,

donc $k * G = S^{-1} * G * (Z + dA * R)$

En simplifiant par G :

$$k = S^{-1} * (Z + dA * R)$$

En intervertissant k et S :

$$S = k^{-1} * (Z + dA * R)$$

Qui représente bien l'équation utilisée pour générer la signature.

Ainsi, on voit qu'on n'utilise que k (nombre aléatoire) et dA (clé privée) pour calculer S, mais qu'on n'a besoin que de R et Qa (clé publique obtenue à partir de la clé privée) pour vérifier la signature. Et comme $R = k * G$ et $Qa = dA * G$ et qu'on ne peut déterminer l'opérande manquant dans une

multiplication de points, nous ne pourrons retrouver la clé privée, et on ne peut générer une fausse signature valide sans celle-ci.[2]

Annexe 2

Exécution d'un scénario de tests

Ci-dessous une simulation avec les utilisateurs suivants :

Utilisateur	Solde initial (en Bitcoins)
Elsa Dupraz	1
Fayçal Hafid	3.9
Maroua EL HOUICHA	2
MarouaneMaachou	0
Mehdi Dahoumane	5

```
        Demonstration
Let's see the users :
Nom : Hafid
Prenom : Faycal
Solde :3.9
Nom : Maachou
Prenom : Marouane
Solde :0
Nom : Dahoumane
Prenom : Mehdi
Solde :5
Nom : Houicha
Prenom : Maroua
Solde :2
Nom : Dupraz
Prenom : Elsa
Solde :1
```

1- Nous effectuons une transaction

De : Fayçal Hafid
A : MarouaneMaachou
Valeur : 1 Bitcoin

```
     Transaction 1 : Hafid Fayçal -> Maachou Marouane : 1 bitcoin
==> <ecdsa.keys.VerifyingKey object at 0x000001D7A480F278>
Verification of the signature :
True
Transaction effectuée avec succès, attente de validation par la BlockChain :
        From :
Nom : Hafid
Prenom : Faycal
        To :
Nom : Maachou
Prenom : Marouane
        Amount sent : 1
```

2- Nous effectuons une transaction

De : Fayçal Hafid
A : MarouaneMaachou
Valeur : 0.2 Bitcoin

```
     Transaction 2 : Hafid Fayçal -> Maachou Marouane : 0.2 bitcoin
==> <ecdsa.keys.VerifyingKey object at 0x000001D7A480F320>
Verification of the signature :
True
Transaction effectuée avec succès, attente de validation par la BlockChain :
        From :
Nom : Hafid
Prenom : Faycal
        To :
Nom : Maachou
Prenom : Marouane
        Amount sent : 0.2
```

3- On donne (illégalement) 500 Bitcoins (fraude) à Maroua EL HOUICHA

4- Nous effectuons une transaction

De : Mehdi Dahoumane

A : Elsa Dupraz

Valeur : 3 Bitcoin

```
     Transaction 2 : Dahoumane Mehdi -> Dupraz Elsa : 3 bitcoin
==> <ecdsa.keys.VerifyingKey object at 0x000001D7A480F668>
Verification of the signature :
True
Transaction effectuée avec succès, attente de validation par la BlockChain :
        From :
Nom : Dahoumane
Prenom : Mehdi
        To :
Nom : Dupraz
Prenom : Elsa
        Amount sent : 3
```

5- On fait valider les transactions en créant une Blockchain :
La fraude est automatiquement supprimée et signalée

```
      BEGINNING THE CREATION OF THE BLOCKCHAIN


            Rountine check Ongoing.


         Forgery found and deleted :
      Bitcoin of value 500 in possession of the user Houicha Maroua
```

On attend le temps que la preuve par travail se fasse :

```
        Transaction en cours :  From :
        Nom : Hafid
        Prenom : Faycal
                To :
        Nom : Maachou
        Prenom : Marouane
                Amount sent : 1
        ==> <ecdsa.keys.VerifyingKey object at 0x000001D7A480F278>
        next
        A new transaction has been verified.
        |
```

Après une quinzaine de secondes :

```
Proof of work just achieved -> Last_proof was : 77194 and current proof is :
55502486
```

La seconde transaction est en cours de validation, on calcule la preuve par travail :

```
        Rountine check Ongoing.

Transaction en cours :  From :
Nom : Hafid
Prenom : Faycal
        To :
Nom : Maachou
Prenom : Marouane
        Amount sent : 0.2
==> <ecdsa.keys.VerifyingKey object at 0x000001D7A480F320>
next
A new transaction has been verified.
```

Après un moment :

```
Proof of work just achieved -> Last_proof was : 55502486 and current proof
is : 111004972
```

La 3ème transaction, calcul de la preuve par travail en cours :

```
                Rountine check Ongoing.

        Transaction en cours :  From :
        Nom : Dahoumane
        Prenom : Mehdi
                To :
        Nom : Dupraz
        Prenom : Elsa
                Amount sent : 3
        ==> <ecdsa.keys.VerifyingKey object at 0x000001D7A480F668>
        next
        A new transaction has been verified.
```

Après un moment :

```
Proof of work just achieved -> Last_proof was : 111004972 and current proof
is : 222009944
```

Les transactions ont toutes été traitées, on visualise la Blockchain :

```
New block : Block
<c998cfcc81f8ca2d85afee73fc42e25d9f8cffcd5feb9d4d63858fb9fb283d2d>: Genesis
Block
```

```
New block : Block
<9727f064ee31bb37379c83cbe5a845bd54e502934161473b7f828f5296d27653>:     From
:
Nom : Hafid
Prenom : Faycal
    To :
Nom : Maachou
Prenom : Marouane
    Amount sent : 1
```

```
New block : Block
<29e73c1e58917d7b3b3204db3556db806918ac04b1e079f919911b3fb8a0add3>:     From
:
Nom : Hafid
Prenom : Faycal
    To :
Nom : Maachou
Prenom : Marouane
    Amount sent : 0.2


New block : Block
<8318e7d669762d949929d02310d8c89f956451a7fb873f645c057e17e918da04>:     From
:
Nom : Dahoumane
Prenom : Mehdi
    To :
Nom : Dupraz
Prenom : Elsa
    Amount sent : 3
```

6- On souhaite retracer les Bitcoins que possède MarouaneMaachou :

```
Now let's trace Marouane's bitcoins :


Owners from most recent to least recent :

Nom : Maachou
Prenom : Marouane
Nom : Hafid
Prenom : Faycal


Owners from most recent to least recent :

Nom : Maachou
Prenom : Marouane
Nom : Hafid
Prenom : Faycal
```

On voit que Marouane possède deux blocs de Bitcoins (1Bitcoin issu de la transaction 1 et 0.2Bitcoin issues de la transaction 2), et que chacun provient de FaycalHafid.

Table des illustrations

1.1 Le hashage

1.1.2 Les fonctions de hachagecryptographiques

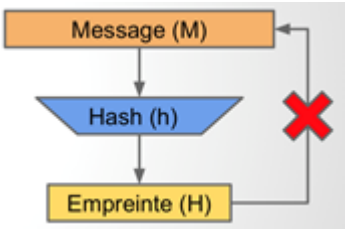

Figure 1

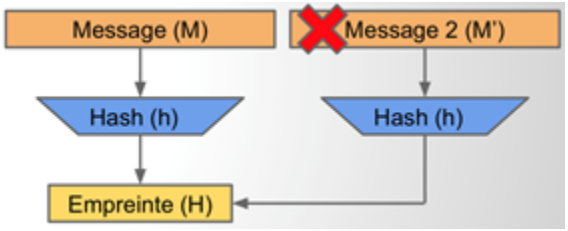

Figure 2

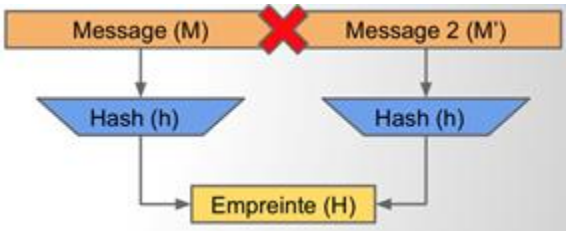

Figure 3

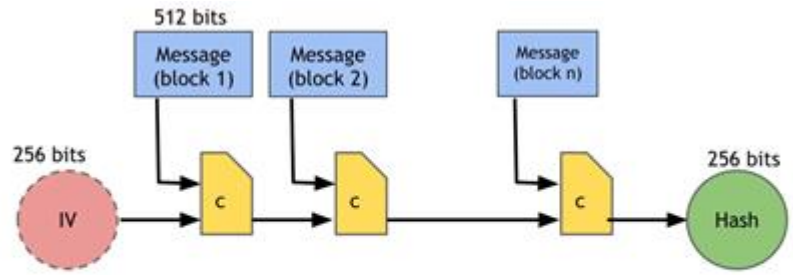

Figure 1.3: SHA-256 Hash Function (simplified). SHA-256 uses the Merkle-Damgard transform to turn a fixed-length collision-resistant compression function into a hash function that accepts arbitrary length inputs.

Figure 4 : Principe simplifié du fonctionnement de SHA-256

1.2 Méthodes de cryptographie et notions de clés

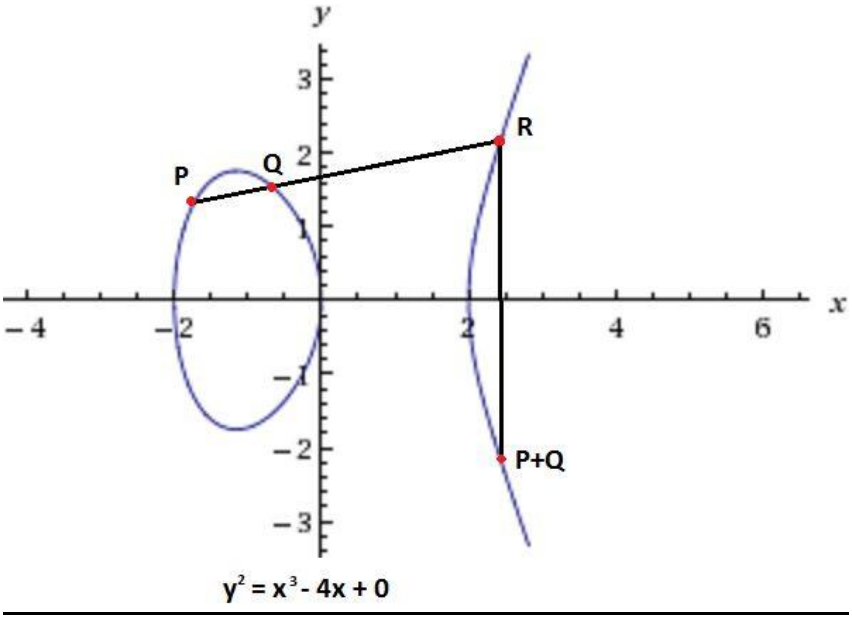

Figure 5 : Addition de points [2]

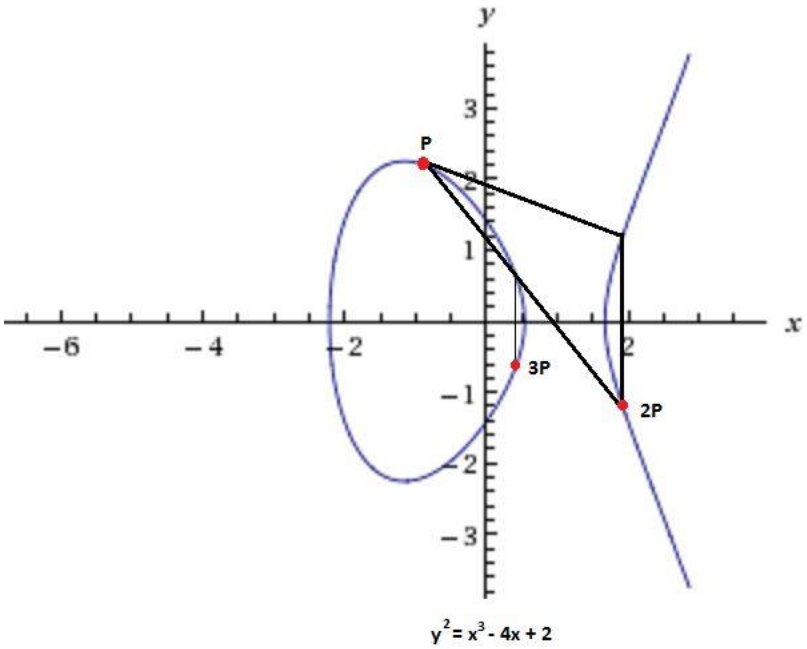

Figure 6 : Multiplication de points [2]

1.3 Principes du Bitcoin

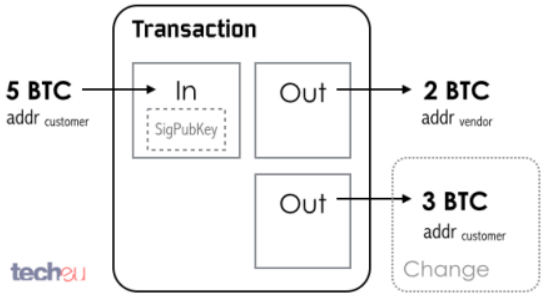

Figure 7: a single input-multiple output Bitcoin transaction
Source: http://tech.eu/features/808/bitcoin-part-one/
(Consulté le 22/05/2018)

1.4 Blockchain et lien avec le Bitcoin

```
# for i in {0..10000} ;do s=$(echo "$t+$i" | sha256sum -);echo "$s $i" ; [ "$(echo $s|cut -b -3)" == "000" ] && break ;done

99bf33aa466e1c01abe273e84cd161ee743aa4b214ffb25847345247008cb60b  - 0
19f2d870bd82c481813756df8905b1b9d69a364d7ab3ca30d2b3ea5503b1d647  - 1
b3cc684841c8f93527144821e5c2208dfd2879f607ac3855eb6cd7d1449cabd0  - 2
7703903253c2579da22b1b7e417891f62882d2ce3456d6bf5a7bffc151c16480  - 3
99f58bc59f5b49c4a044e66fec809155a26bced8532627c1cdf7ce7247485652  - 4
2d4420d4beea97475ce600daed7669fe936f0af50cfc265b6306b64229aff439  - 5
(…)
898ea41193317248a2cf21c12c13364b7c6897d7dca2da99309aedd48090efc6  - 3071
874fcdc16f84c493a1a3f8ab650093901f93b1e6a6f02bc269ef65da1ce056ad  - 3072
6ebb67f2e4936e178c36ecc029cd447b6245563efc8cc4c2a91f13a77fb1bf77  - 3073
f07b19296c246bcb2eaaba7a69c53c539f4292532273723b2119e58eb99d3768  - 3074
95c9ed201543e4a5ddbfb917ed9bd49626d5ed94218f94033d35881114dc3395  - 3075
000c4a032ca7d39734759da08c672cf50e1554a3671b5a3ddccd389c44bb5db8  - 3076
#
# echo 'Ceci est mon document !+3076' | sha256sum -
000c4a032ca7d39734759da08c672cf50e1554a3671b5a3ddccd389c44bb5db8  -
```

Figure 8 : Exemple de preuve par travail

2.2 Plateforme d'utilisation

```
Bienvenue dans l'application.
Donnez votre ID : Marouane

donnez votre MDP : 1
Vous êtes connectés. Bienvenue :)

Entrez le numéro de votre opération :
1- Consulter votre solde
2- Deconnexion
3-Consulter l'historique de vos transactions
4-Effectuer une transaction
```

Figure : Plateforme d'utilisation

2.2 Les classes de développement

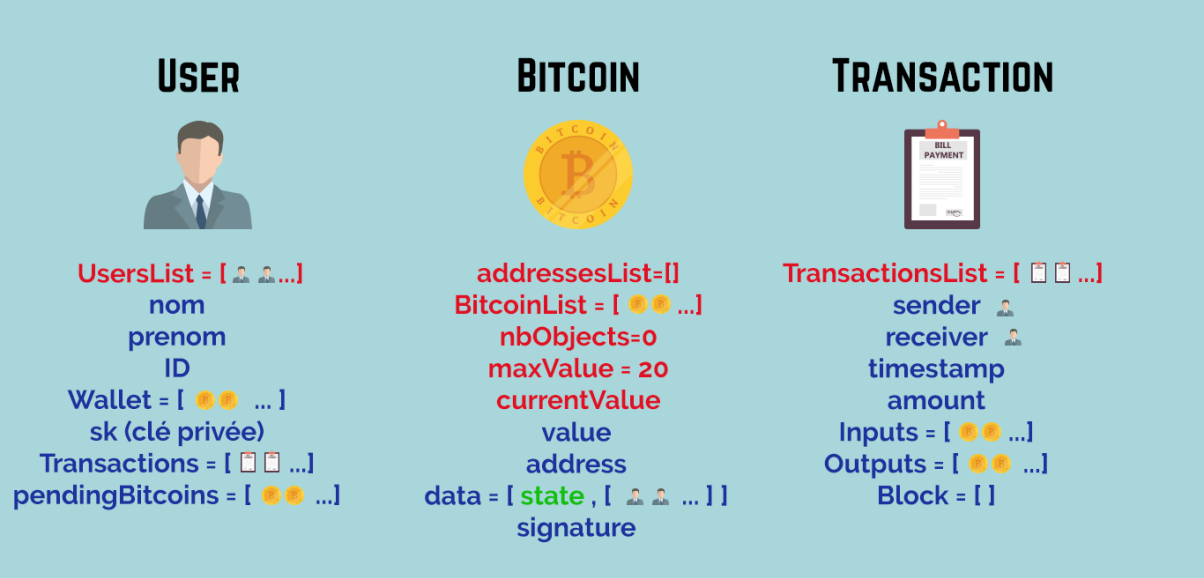

Figure 9 : Vision globale des classes
En rouge : attributs de classe, en bleu : attributs d'instance

Glossaire

Bitcoin : crypto-monnaie décentralisée, c'est la plus utilisée dans le monde.

Blockchain : littéralement « chaîne de blocs », technologie de stockage et de transmission d'informations, transparente, sécurisée, et décentralisée.

Clé privée : permet à l'utilisateur de signer cryptographiquement la transaction qu'il souhaite effectuer.

Clé publique : calculée depuis la clé privée, elle est connue de tous et permet de désigner le destinataire d'une transaction.

Crypto-monnaie : monnaie électronique basée sur la cryptographie pour vérifier les transactions et créant elle-même la monnaie.

ECDSA : 'EllipticCurve Digital Signature Algorithm' ; algorithme de cryptographie utilisé pour créer une signature digitale pour des données qui servira à vérifier l'authenticité de la signature sans pouvoir la falsifier : n'importe qui peut reconnaître une signature ECDSA mais ne peut la copier ou la trafiquer.

Fonction de hachage : Fonction qui, étant donnée une entrée, renvoie une empreinte permettant d'identifier rapidement et partiellement la donnée initiale.

Fonction de hachage cryptographique : fonction de hachage quasiment impossible à inverser ; fonction dite « à sens unique ».

Nœud : machine reliée au réseau et connectée à la blockchain.

O.S : ‘Operating System’ ou système d’exploitation, ensemble de programmes pilotant les composants et faisant marcher l’appareil électronique.

Programmation orientée objet : paradigme de programmation informatique basé sur la représentation des objets et leurs relations.

Proof of Work : « preuve de travail », mécanisme de sécurité qui ralentie la création de nouveaux blocs (10 minutes dans le cas des Bitcoins) et rendant difficile la modification de la blockchain.

Signature : mécanisme garantissant l’authenticité d’une donnée et d’authentifier son propriétaire ou son auteur.

Références bibliographiques

[1] Christina Boura. Analyse de fonctions de hachage cryptographiques. Cryptographie et sécurité [cs.CR]. Université Pierre et Marie Curie - Paris VI, 2012. Français. <tel-00767028> (https://tel.archives-ouvertes.fr/tel-00767028/document consulté le 22/05/2018)

[2] AvinashKak, Purdue University. Lecture Notes on “Computer and Network Security” written on March 26th, 2018 et 12:19am.
(https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture14.pdf consulté le 26/04/2018)

[3] Jean-Luc Parouty, Institut de Biologie Structurale (IBS), Jean-Luc.Parouty<at>ibs.fr.
Bitcoin : Principes généraux, limites et perspectives.

[4] Jean-Luc Parouty, Institut de Biologie Structurale (IBS), Jean-Luc.Parouty<at>ibs.fr.
Bitcoin : Éléments de compréhension technique.

[5] Cours en ligne : “Utiliser les sockets en python - réseau - cours débutant”
(http://apprendre-python.com/page-reseaux-sockets-python-port consulté le 22/05/2018)