REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR

ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE SAAD DAHLAB DE BLIDA 1

FACULTE DES SCIENCES

DEPARTEMENT D'INFORMATIQUE

PROJET DE FIN D'ETUDE POUR L'OBTENTION DU DIPLOME DE MASTER EN SECURITE DES SYSTEMES D'INFORMATION

Mémoire réalisé par : BENNANNI Sid Ahmed

Implémentation d'un *Smart contract* sous la plateforme Ethereum : vote électronique

Organisme d'accueil : CERIST

Promotrice : Mlle N.Boustia **Encadreur :** Mr S.Hadjar

Soutenue le 16/07/2019 devant le jury composé de :

Mlle Ghebghoub **Présidente**

Mlle Arkam **Examinatrice**

Juillet 2019

Résumé

Le système *Blockchain* a introduit de nouveau concepts et idées dans le domaine de recherches de sécurité, proposant des nouvelles approches basées sur la cryptographie pour éliminer la notion de centralisation et l'autorité intermédiaire, c'est la couche de confiance dans l'internet. La *Blockchain* est un réseau p2p de nœuds qui communiquent entre eux, effectue des calculs cryptographique pour garantir la sécurité et l'immutabilité des données, ces données sont structurées dans des blocs chainées et partagées entre les nœuds du réseau. La deuxième génération *Blockchain* comme la plateforme Ethereum nous permet d'écrire du code immutable et infalsifiable qui représente une alliance s'appelle *Smart contract.*

Dans notre travail, nous avons étudié le système *Blockchain,* les *Smart contract* ainsi que la plateforme Ethereum. Nous avons abordé la problématique du vote électronique qui a besoin de confiance et de transparence des données, nous avons proposé une solution *Blockchain* basé sur les *Smart contract* pour développer une application décentralisée de vote. Pour valider notre étude nous avons développé un *Smart contract* dans la plateforme Ethereum avec une interface web pour simuler le scénario de vote et les bienfaits de la *Blockchain* et les *Smart contract* en termes de sécurité, transparence, immutabilité de processus et des données du vote électronique.

Mots clés : *Blockchain*, Ethereum, *Smart contract*, *p2p*, nœud.

Abstract

The Blockchain system has introduced new concepts and ideas in the field of security research, proposing new approaches based on cryptography to eliminate the notion of centralization and intermediate authority, it is the layer of trust in the Internet. The Blockchain is a p2p network of nodes that communicate with each other, perform cryptographic calculations to ensure the security and immutability of the data, these data are structured in chained blocks and shared between the nodes of the network. The second generation Blockchain like the Ethereum platform allows us to write immutable and tamper-proof code that represents an alliance called Smart Contract.

In our work, we studied the Blockchain system, the Smart contract and the Ethereum platform. We tackled the issue of electronic voting that needs confidence and transparency of data, we proposed a Blockchain solution based on Smart contract to develop a decentralized voting application. To validate our study we have developed a Smart contract in the Ethereum platform with a web interface to simulate the voting scenario and the benefits of Blockchain and Smart contracts in terms of security, transparency, immutability of electronic voting process and data.

Keywords: Blockchain, Ethereum, Smart contract, p2p, node.

Remerciements

Je remercie d'abord ALLAH le tout puissant qui m'a guidé et m'a donné la force et la volonté de réaliser ce mémoire.

Mes pensées vont vers mes parents, qui ont toujours cru en moi. C'est grâce à leur soutient et prières que j'ai accomplie ce travail, ils savent déjà combien je leur dois.

Comme je remercie ma promotrice Mme Narhimene Boustia de m'avoir pris en charge et aidé tout au long du projet. Ainsi que mon encadreur Mr Samir Hadjar de m'avoir orienté avec ces précieux conseils et remarques.

Mes remerciements les plus sincères à toutes les personnes qui auront contribué de près ou de loin à l'élaboration de ce mémoire ainsi qu'à la réussite de cette formidable année universitaire.

Enfin, je tiens aussi à remercier les jurés pour avoir accepté d'examiner et de juger mon travail.

Sommaire

Liste des figures

Liste des tableaux

Introduction générale

Introduction:

La *Blockchain* fondamentalement est une base de données décentralisée mais aussi un réseau décentralisé où tous les nœuds et dispositifs communiquent entre eux sans intermédiaire, contrairement au web traditionnel qui est basé sur des serveurs centralisés. C'est pour ces deux avantages majeurs que nous avons opté de développer notre application de vote sur une *Blockchain,* ainsi nous pouvons assurer l'indépendance de toute entité centrale qui peut influencer les résultats d'un scrutin électronique, et assurer la transparence, la sécurité du déroulement de l'opération, et surtout l'immutabilité des résultats.

Problématique:

Les élections traditionnelles nécessitent des moyens matériels, humains et financiers importants (les bureaux dans chaque mairie, des milliers de bénévoles pour trier les voix, des centaines de contrôleurs, de la paperasse, centralisation des voix), et beaucoup de temps, sans oublier que ni la traçabilité ni l'intégrité des calculs des voix n'est garanti, car ce processus nécessite des autorités intermédiaires qui peuvent modifier les calculs et le résultat.

Les élections électroniques dans le web traditionnelle minimisent les procédures du vote mais ne garantit pas la transparence et l'intégrité des données, car tout le processus est centralisé dans un serveur qui est gérer par une autorité intermédiaire, les données peuvent être altérer facilement sans aucun ne le sache.

Malgré la technologie du web 2.0 qui a réduit les procédures, le temps, les ressources humaines et financières, le problème de l'intégrité des données et la transparence des résultats n'est pas garanti.

Objectif:

La *Blockchain* est un réseau *p2p* de nœuds qui communiquent entre eux, chaque machine connectée à la *Blockchain* est un nœud qui communique avec les autres et partage une partie de responsabilité que d'habitude un serveur web assume seul. Chaque nœud obtient une copie de toutes les données partagées à travers la

Blockchain, ces données sont structurées dans des blocs qui sont chainés l'un a l'autre pour former un grand livre public, tous les nœuds travaillent ensemble pour assurer la sécurité et l'immutabilité de ce grand livre, et c'est très important pour une application de vote électronique parce que :

- On peut toujours avoir et garder une trace de chaque compte qui a envoyé une transaction lors du vote.
- Chaque voix va au candidat choisi sans possibilité de corruption.
- Toute transaction est enregistrée de façon permanente et immutable pour toujours sur tous les nœuds du réseau *Blockchain*.

Le code qui régit toutes les règles du système du vote est sécurisé, immutable et accessible par tout le monde, c'est très important pour une application de vote parce que cela signifie que les règles ne changeront pas en plus en gardant la transparence des élections. Ce code est appelé un *Smart contract* car il représente une sorte d'alliance ou accord de notre vote. L'accord de notre *Smart contract* est présenté comme suit :

- Définir la période de chaque phase (d'inscription, du vote, du résultat final).
- Définir les contrôles et les vérifications des données lors l'inscription.
- Limiter le nombre d'inscription à 1 pour chaque utilisateur.
- Identifier les citoyens et les électeurs.
- Limiter le nombre de vote à 1 pour chaque électeur.
- Calculer les voix en temps réel.
- Délibérer le résultat final.

Notre application dans ce travail est une application décentralisée en tous sens :

- Un réseau décentralisé présenté en p2p.
- Les données sont décentralisées car elles sont partagées entre tous les nœuds.
- Le code est décentralisé car il est également partagé et exécuté sur tous les nœuds.

Chapitre I : Les *Blockchains*

1. Introduction :

Depuis l'introduction de Bitcoin en 2009 le monde a connu une nouvelle technologie qui assure les critères de sécurité des transactions monétaires avec un système totalement décentralisé qui se base sur la cryptographie et le consensus des utilisateurs.

De grandes institutions financières et de nombreuses entreprises de différents secteurs ont commencé à explorer la *Blockchain* pour diminuer les coûts de transaction, d'accélérer leur délai, de réduire le risque de fraude et éliminer les services d'intermédiaire. Certains tentent de réinventer les systèmes et services existants pour les amener à un nouveau niveau et proposer de nouveaux types de services.

2. Historique :

L'histoire de la *Blockchain* et de bitcoin sont directement attachée, Satoshi Nakamoto, pseudonyme sous lequel le monde le connaît, est un programmeur ou un groupe de travail inconnu qui a introduit le principe de bitcoin et la *Blockchain* en 2008 sous la forme d'un livre blanc qui a été publié en tant que logiciel open source en 2009.

Les systèmes monétaires n'ont pas été touchés par la révolution technologique depuis les années 1980. Les banques ont formé les institutions centralisées qui ont géré les enregistrements des transactions, régi les interactions, renforcé la confiance et la sécurité, et réglementé l'ensemble du système **[BegB].**

L'ensemble du commerce dépend de ces institutions financières, qui servent de tiers de confiance pour traiter les paiements. La médiation des institutions financières augmente les coûts et le temps nécessaire au règlement d'une transaction, tout en limitant la taille des transactions. Elle était indispensable pour régler les différends, mais cela signifiait qu'une transaction totalement irréversible n'était jamais possible. Cela a abouti à une situation où la confiance était nécessaire pour que quelqu'un puisse traiter avec un autre. Certes, ce système bureaucratique a dû être modifié pour

suivre la transformation numérique attendue de l'économie. Satoshi a donc proposé une crypto-monnaie appelée Bitcoin qui a été activée par la technologie sous-jacente, la *Blockchain* Bitcoin est juste une utilisation monétaire **[BegB]**.

La *Blockchain* ne s'est pas arrêtée au bitcoin, elle a connu plusieurs améliorations. Chacune d'elles a inclut une nouvelle désignation *Blockchain* 1.0 se base juste sur la crypto-monnaie comme le bitcoin, la seconde classe *Blockchain* 2.0 arrive avec les *Smart contracts* ayant la possibilité de stocker des données dans la *Blockchain* comme l'Ethereum qui a été proposée en 2013 par Vitalik Buterin **[AGTUB]**.

3. Définitions :

3.1. Blockchain :

A l'origine, le terme *Blockchain* est un terme informatique qui désigne ce qui permet de structurer et de partager des données dans un réseau d'individus indépendants sous forme d'un grand livre numérique. Elle constitue une nouvelle approche de la base de données distribuée, elle présente la cinquième évolution majeure de l'informatique. C'est la couche de confiance manquante précédemment **[LBPN]**.

- *Blockchain* est un système *peer-to-peer* de transaction de valeurs sans tiers de confiance.
- Il n'est pas nécessaire que des tiers de confiance servent d'intermédiaires pour vérifier, sécuriser et régler les transactions.
- Il s'agit d'un registre des transactions partagés, décentralisé et ouvert. Cette base de données est répliquée sur un grand nombre de nœuds.
- Cette base de données de grand livre est une base de données contenant uniquement des ajouts et ne peut être ni modifiée ni supprimée. Cela signifie que chaque entrée est une entrée permanente. Toute nouvelle entrée est répercutée sur toutes les copies des bases de données hébergées sur des nœuds différents.
- Il s'agit d'une couche supplémentaire au-dessus d'Internet et peut coexister avec d'autres technologies Internet.
- Tout comme TCP / IP a été conçu pour réaliser un système ouvert, la technologie de *Blockchain* a été conçue pour permettre une véritable décentralisation. Pour ce faire, les créateurs de Bitcoin l'ont ouvert à la source,

ce qui lui a permis d'inspirer de nombreuses applications décentralisées **[BegB]**.

3.2. Transaction :

La *Blockchain* permet le partage et l'échange d'informations entre nœuds .Cet échange s'effectue au moyen de fichiers contenant des informations de transfert d'un nœud à l'autre, générées par un nœud source et diffusées sur l'ensemble du réseau à des fins de validation. L'état actuel de *Blockchain* est représenté par ces transactions, qui sont générées en permanence par les nœuds, puis rassemblées en blocs. Dans le cas de bitcoin, chaque transaction représente le transfert de devise d'un nœud à l'autre. Tous les nœuds sont conscients du solde actuel à chaque adresse et conservent une copie de la *Blockchain* existante, qui est le journal contenant l'historique des transactions précédentes. L'état de la *Blockchain* change après chaque transaction. Avec un nombre considérable de transactions générées chaque seconde, il est très important de valider et de vérifier les transactions authentiques et d'éliminer le faux **[EYWB]**.

- **Signature numérique :**

Chaque utilisateur possède une paire de clés privée et publique. La clé privée est utilisée pour signer les transactions. Les transactions numériques signées sont réparties dans l'ensemble du réseau et sont accessibles par les clés publiques, qui sont visibles à tout le monde dans le réseau. La figure.I.1 montre un exemple de signature numérique utilisée dans les *Blockchain*. La signature numérique typique est impliquée dans deux phases: la phase de Signature et la phase de Vérification.

Lorsqu' Alice veut signer une transaction, elle génère d'abord une valeur de *hachage* dérivée de la transaction. Puis chiffre cette valeur en utilisant sa clé privée et l'envoie à Bob, le *hachage* chiffré avec les données d'origine. Bob vérifie la transaction reçue par la comparaison entre le *hachage* décryptée (en utilisant la clé publique d'Alice) et la valeur de hachage dérivée à partir des données reçues par la même fonction de hachage. Les algorithmes de signature numérique typiques utilisés dans les *Blockchains* comprennent *Elliptic Curve Digital Signature Algorithm* (ECDSA) **[BCO]**.

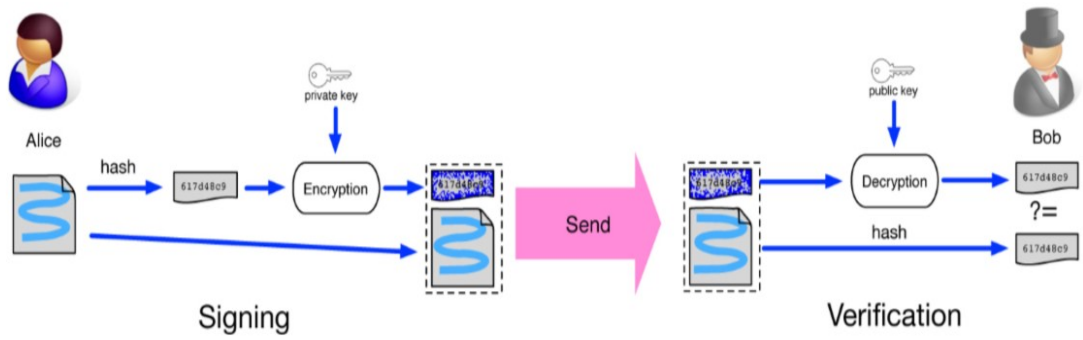

Figure.I.1: Signature numérique utilisée dans la *Blockchain* **[BCO]**

3.3. Bloc :

Un bloc contient un en-tête et un corps, affichés dans la figure.I.2 :

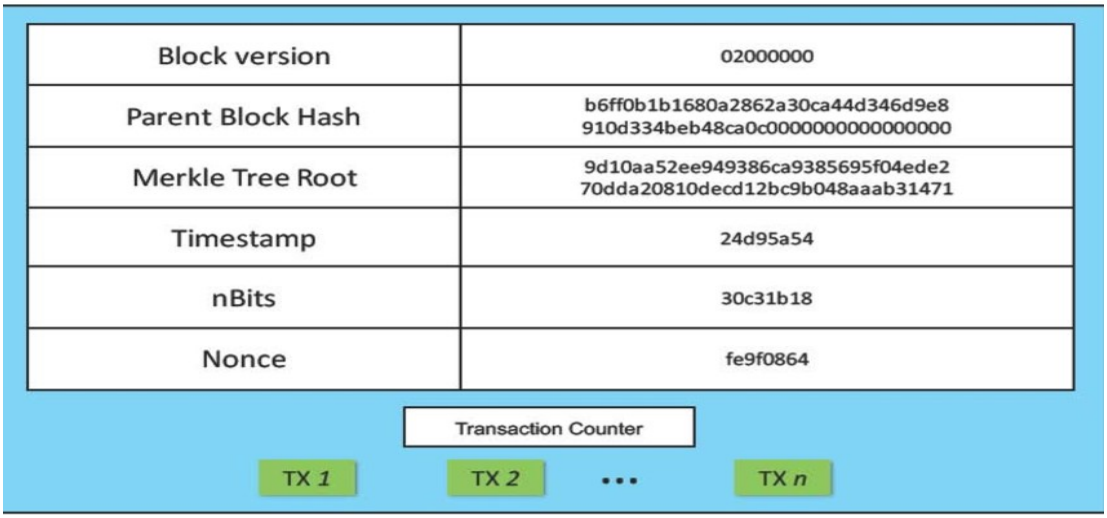

Figure.I.2 : Structure d'un bloc **[BCO]**

En particulier, l'en-tête de bloc comprend :

- Version de bloc : indique le jeu de règles de validation de bloc à suivre.
- Hachage de bloc parent : valeur de hachage de 256 bits qui pointe vers le bloc précédent. (Voir figure.I.3) :

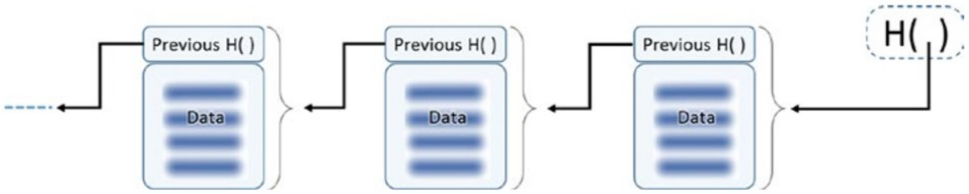

Figure.I.3 : Enchainement des blocs **[BegB]**

- Hachage racine *Merkle Tree* : la valeur de hachage de toutes les transactions du bloc.

Les arbres de Merkle constituent un moyen très efficace de vérifier si une transaction spécifique appartient à un bloc particulier. S'il existe «n» transactions dans une arborescence Merkle (éléments feuille), cette vérification prend alors juste le temps de connexion (n), comme illustré à la figure.I.4 **[BegB]**.

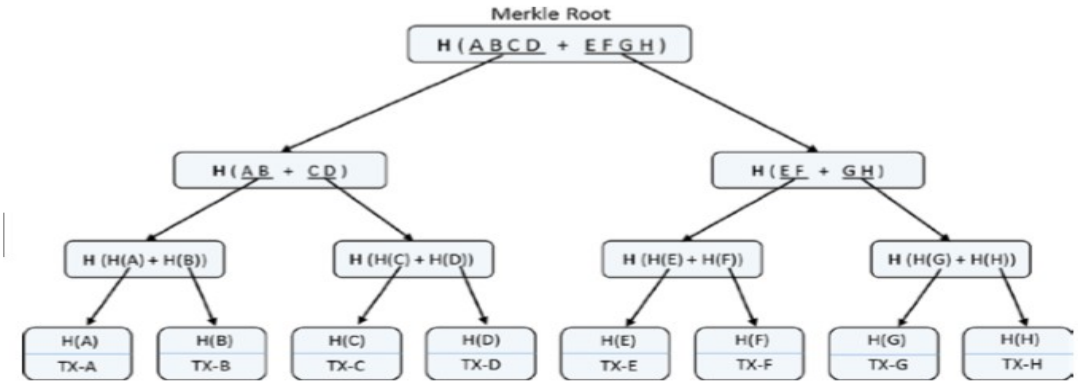

Figure.I.4: Arbre de Merkle **[BegB]**

Pour vérifier si une transaction ou tout autre élément feuille appartient à un arbre Merkle, nous n'avons pas besoin de toutes les transactions ni de l'arbre complet. Comme le montre le diagramme de la figure.I.5, seulement un sous-ensemble est nécessaire.

Pour garantir l'intégrité d'un bloc téléchargé par rapport à l'ensemble des données, il suffit de posséder les hashs des frères, les hashs des oncles et le hash-sommet. De plus, seul le hash-sommet doit être récupéré de manière sûre pour garantir l'intégrité de l'ensemble des données représentées par l'arbre **[BegB].**

Par exemple, si on veut vérifier l'intégrité du bloc TX-E, il suffit d'avoir récupéré le H-F (son frère), le hash (EF+GH) (son oncle) et le hash-sommet H(ABCD+EFGH).

Dans la figure.I.5, seuls les rectangles pleins sont nécessaires et les rectangles en pointillés peuvent être simplement calculés, à condition que les données du rectangle solide soient respectées **[BegB]**.

Comme il y a huit éléments de transaction (n = 8), seuls trois calculs (log2 8 = 3) seraient nécessaires pour la vérification.

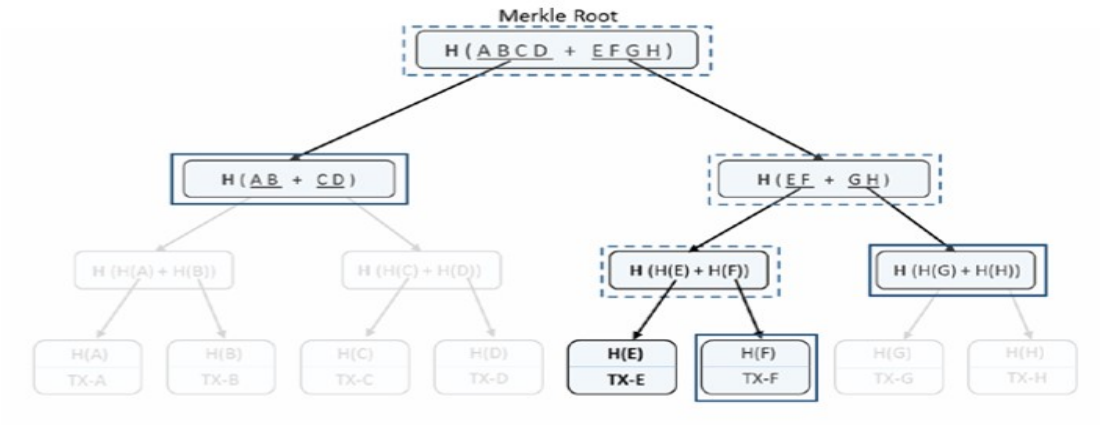

Figure.I.5: Arbre de Merkle (Vérification des transactions)**[BegB]**

- Horodatage : horodatage actuel en secondes depuis le 1970-01-01T00: 00 UTC.
- nBits : cible de hachage actuelle dans un format compact.
- Nonce : un champ de 4 octets, qui commence généralement par 0 et augmente pour chaque calcul de hachage **[BCO]**.

Le corps du bloc est composé d'un compteur de transactions et un ensemble de transactions. Le nombre maximal de transactions qu'un bloc peut contenir dépend de la taille du bloc et de la taille de chaque transaction. *Blockchain* utilise un mécanisme de cryptographie asymétrique pour valider l'authentification des transactions **[BCO]**.

Le premier bloc de la *Blockchain* est un bloc sans parent appelé ; Genesis bloc dans le cas de bitcoin le bloc *Genesis* contient une seule transaction de 50 BTC et une valeur de hash de bloc précédent égale à 0 **[BegB].**

3.4. Monnaie Electronique :

La monnaie électronique est un substitut à l'argent liquide (pièces et billets), stocké dans un dispositif électronique, magnétique ou sur un serveur distant **[LE]**.

Une monnaie cryptographique, ou crypto monnaie, est une monnaie électronique sur un réseau décentralisé (chaque client appelé nœud, est également un serveur). Afin de sécuriser cette monnaie, le système de transaction repose sur le concept de *Blockchain*, basé en partie sur des procédés de cryptographie **[MFE]**.

3.5. Nœud :

C'est un ordinateur relié au réseau *Blockchain* qui utilise un programme relayant les transactions. Les nœuds conservent une copie du grand registre *Blockchain* et sont répartis partout dans le monde. Il existe trois types de nœud **[BF]** :

- **Un nœud complet :** c'est un nœud qui contient l'ensemble des transactions et des blocs. Il participe à la sécurité du réseau *Blockchain*.

- **Un nœud mineur :** c'est un nœud complet qui fait des calculs de *hash* pour la sécurité de la *Blockchain*, il est difficile et couteux de sorte que les gens ne font pas fonctionner un nœud mineur gratuitement d'où l'algorithme de *Blockchain* les récompense par un *token* ou une crypto monnaie pour leur service.

- **Un nœud simple :** c'est un nœud qui contient les derniers blocs valides complets, ainsi que l'empreinte (hash) des transactions et blocs plus anciens **[LBPN] [MFE]**.

3.6. Mineurs :

Le minage c'est le moyen par lequel les transactions Bitcoin sont sécurisées. Les mineurs effectuent avec leur matériel informatique des calculs mathématiques pour la sécurité du réseau. Comme récompense pour leurs services, ils collectent les bitcoins nouvellement créés ainsi que les frais des transactions qu'ils confirment.

Actuellement cette récompense est de 12,5 bitcoins par bloc. Elle est divisée par deux environ tous les quatre ans. Ethereum récompense de la même manière tous les ans l'ensemble des mineurs ayant participé à l'ajout de bloc, et ceci pour toujours (15,6 millions d'éther sont générés tous les ans) **[MFE]**.

Les mineurs effectuent des *hashs* cryptographiques (deux *SHA256* successifs) sur ce qu'on appelle un entête de bloc. Pour chaque nouveau hash, le logiciel de minage

utilise un nombre aléatoire différent qu'on appelle le *nonce*. Le nonce est une valeur entière avec 32 *bits* de mémoire qui lui sont alloués. Cela signifie qu'il est limité à environ 4 milliards de possibilités seulement, ce qui, avec la puissance de *hachage* actuel du réseau, est largement insuffisant. On ajoute donc dans le hash l'horodatage du bloc en heure Posix, constamment actualisée **[Bitc]**.

Autre élément variable sur lequel les mineurs peuvent jouer : la configuration des transactions. Intégrant le numéro de bloc, l'horodatage, le *nonce*, les données du bloc et le hash du bloc précédent, le hash produit ressemblera à ceci :

```
93ef6f358fbb998c60802496863052290d4c63735b7fe5bdaac821de96a53a9a
```

Ce hash peut être converti dans un très long nombre. (Il s'agit d'un nombre hexadécimal, ce qui signifie que les lettres A-F sont les nombres 10-15). Pour rendre le minage difficile, il y a ce qu'on appelle la difficulté cible. Afin de créer un bloc valide, un mineur doit trouver un *hash* qui est inférieur à la difficulté cible **[Bitc].** Par exemple, si la difficulté est de :

```
100000000000000000000000000000000000000000000000000000000000000
```

N'importe quel nombre qui débute par un zéro serait accepté et considéré comme inférieur à la cible **[Bitc]**.

4. Consensus :

Lorsque les nœuds commencent le partage et l'échange de données via une plate-forme *Blockchain*, ils n'ont ni une partie centralisée pour régler et résoudre les litiges ou se protéger des violations de la sécurité, ni de mécanisme permettant de suivre le flux de fonds et d'assurer un échange inattaquable et éviter les fraudes, telles que les attaques par double dépense **[EYWB]**.

Tous les nœuds devraient convenir d'un protocole commun de mise à jour du contenu du registre afin de le maintenir dans un état cohérent. Les blocs ne devraient pas simplement être acceptés comme faisant partie de la *Blockchain*, sans le consentement de la majorité, c'est ce qu'on appelle un mécanisme de consensus par lequel les blocs sont créés et ajoutés au grand livre existant pour une utilisation future. Dans le cas de bitcoin, les destinataires peuvent, après la vérification de la signature,

utiliser plusieurs fois les sorties pour les utiliser dans des transactions ultérieures car elles sembleraient valables pour les destinataires individuels **[EYWB]**.

Ainsi, uniquement pour éviter les doubles dépenses, Satoshi a été le premier à proposer une crypto-monnaie décentralisée basée sur le consensus parmi des nœuds non fiables. Ce consensus est un accord entre les nœuds, qui implique une extraction de blocs, dans laquelle les mineurs se font concurrencer pour trouver le prochain bloc valide en calculant un *hachage* de bloc cryptographique. Les nœuds qui trouvent la solution sont récompensés par des bitcoins, générant ainsi une nouvelle monnaie. Cette valeur de *hachage* est appelée "preuve de travail" et si toutes les transactions et toutes les preuves de travail sont valides, les nœuds l'acceptent en mettant à jour leur copie **[EYWB]**.

Les étapes nécessaires à la validation de bloc sont résumées comme suit:

- Toutes les transactions contenues dans le bloc actuel sont vérifiées. Après la vérification individuelle, l'ordre chronologique des transactions en fonction de leurs occurrences et références est confirmé.
- Le hachage du bloc précédent référencé par le bloc actuel existe et est valide. Ceci est généralement vérifié à partir du bloc de *genesis*.
- L'exactitude de l'horodatage est vérifiée.
- La preuve de travail pour le bloc actuel est valide **[EYWB]**.

Bitcoin et Ethereum utilisent un mécanisme de consensus *Proof of Work* (PoW) pour sélectionner au hasard un nœud pouvant proposer un bloc. Une fois que ce bloc est proposé et propagé à tous les nœuds, ils vérifient s'il s'agit d'un bloc valide avec toutes les transactions légitimes et que le *puzzle* de la preuve de travail a été résolu correctement. Ils ajoutent ce bloc à leur propre copie de *Blockchain* et poursuivent la construction. Il existe de nombreuses variantes des protocoles de consensus tels que *Proof of Stake* (PoS), Délégué *PoS* (dPoS), *Practical byzantin Fault Tolerance* (PBFT) **[BegB]**.

4.1. Algorithme de tolérance aux pannes byzantines pratique (PBFT) :

L'algorithme PBFT a été proposé comme solution au problème des généraux byzantins, qui consiste à mener une attaque réussie contre une ville rivale par l'armée byzantine. Pour que l'armée byzantine gagne, tous les généraux fidèles doivent travailler sur le même plan et attaquer simultanément. En outre, quoi que fassent les traîtres, les généraux fidèles devraient s'en tenir au plan décidé et un petit nombre de traîtres ne devrait le ruiner.

De même, dans la *Blockchain*, PBFT s'efforce d'établir un consensus entre les nœuds participants. Ces derniers conservent un état actuel qui à la réception d'un nouveau message sera alimenté par celui-ci pour les calculs afin d'aider le nœud à prendre une décision. Cette décision est ensuite diffusée sur le réseau. La majorité des décisions déterminent un consensus pour le réseau. *Hyperlegder*, qui travaille au développement de systèmes de *Blockchains* en consortium pour les entreprises, utilise PBFT comme mécanisme de consensus sous-jacent. Il convient de noter que bon nombre des nouveaux développements sur la *Blokchain* découlent des travaux antérieurs sur les bases de données distribuées **[EYWB]**.

4.2. *Proof-of-Work* (POW) :

La preuve de travail a été le premier protocole consensuel décentralisé proposé par Satoshi, visant à assurer la cohérence et la sécurité dans le réseau Bitcoin. En Bitcoin, le transfert de la crypto-monnaie s'effectue de manière totalement décentralisée nécessitant ainsi un consensus pour l'authentification et la validation de bloc. Les nœuds du réseau bitcoin se font concurrencer pour calculer le hash du bloc suivant supposée être inférieure à une valeur cible variante de manière dynamique, déterminée par la règle de consensus **[EYWB]**.

Les nœuds réalisant la solution attendent la confirmation mutuelle d'autres nœuds avant d'ajouter le bloc à la *Blockchain* existante. Plusieurs blocs valides peuvent être générés si plusieurs nœuds trouvent une solution appropriée provoquant une branche temporaire dans le réseau. Dans de tels scénarios, tous sont acceptables et les nœuds les plus proches des mineurs acceptent la solution qu'ils reçoivent et transmettent la même chose à d'autres pairs. Le conflit est évité à un stade ultérieur en acceptant la

«version la plus longue» de la chaîne disponible à tout moment **[EYWB]**. (Voir figure.I.6)

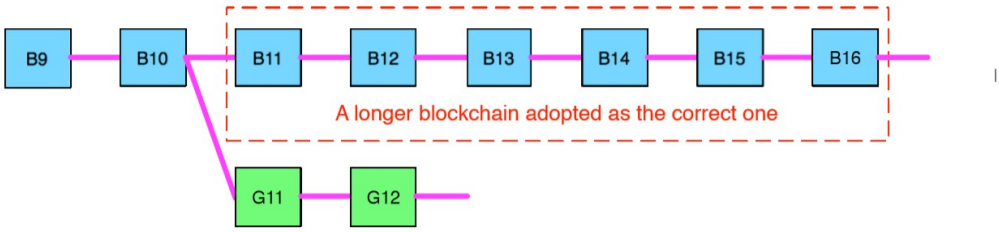

Figure.I.6: Scénario de branche *Blockchain* **[BCO]**

4.3 *Proof-of-Stake* (PoS) :

Une preuve d'enjeux a été proposée pour surmonter les inconvénients de la consommation d'énergie excessive de POW en bitcoin. Ethereum utilise la preuve d'enjeu pour atteindre un consensus. Au lieu d'investir dans des ressources pouvant effectuer des calculs rigoureux pour trouver le hash dans POW, POS propose d'acheter la crypto-monnaie et de l'utiliser comme participation au réseau. L'enjeu est directement proportionnel aux chances de devenir le validateur de bloc. Pour parvenir à un consensus, le validateur de bloc est sélectionné de manière aléatoire et n'est pas prédéterminé. Les nœuds produisant des blocs valides bénéficient d'incitations, mais si leur bloc n'est pas inclus dans la chaîne existante, ils perdent également une partie de leur participation. Différents modèles de consensus ont été différenciés en fonction de plusieurs facteurs **[EYWB]**.

5. Les types de *Blockchain* :

Les systèmes de *Blockchain* actuels peuvent être grossièrement classés en trois types:

5.1. Les *Blockchains* publiques : sont des grands réseaux distribués accessibles, ouvert à tous et à tous les niveaux, et ont un code source ouvert que leur communauté maintien à jour comme Bitcoin.

5.2. Les *Blockchains* consortium : sont des réseaux distribués qui contrôlent les rôles de chaque nœud dans les réseaux tels que Ripple, le code source peut-être ouvert ou non.

5.3. Les *Blockchains* privées : sont plus petites que les autres types, leur accès est complétement contrôlé.

Les trois types utilisent la cryptographie pour permettre à chaque nœud de participer à la gestion du grand registre de manière sécurisée sans autorité centrale **[LBPN]**.

Voici les principales différences entre les types de *Blockchain* :

• **Détermination du consensus :** Dans la *Blockchain* publique, chaque nœud peut prendre part au processus de consensus. En outre, un ensemble sélectionné de nœuds peut être utilisé pour valider le bloc dans la *Blockchain* du consortium. Quant à la *Blockchain* privée, elle est entièrement contrôlée par une organisation qui pourrait déterminer le consensus final **[BCO]**.

• **Autorisation de lecture :** Les transactions dans une *Blockchain* publique sont visibles par le public tandis que la tutelle de la direction dépend du secret de la *Blockchain* privée ou du consortium. Le consortium ou l'organisation pourrait décider si les informations stockées sont publiques ou restreintes **[BCO]**.

• **Immutabilité :** Étant donné que les transactions sont stockées dans différents nœuds du réseau distribué, il est donc pratiquement impossible de modifier la *Blockchain* publique. Toutefois, si la majorité du consortium ou l'organisation dominante souhaite altérer la *Blockchain*, celle du consortium ou privée peut être inversée ou altérée **[BCO]**.

• **Efficacité :** La propagation des transactions et des blocs prend beaucoup de temps car il existe un grand nombre de nœuds sur la *Blockchain* publique. Compte tenu de la sécurité du réseau, les restrictions sur la *Blockchain* publique seraient beaucoup plus strictes. En conséquence, le débit de transaction est limité et la latence est élevée. Avec moins de validateurs, la *Blockchain* consortium et privée pourraient être plus efficaces **[BCO]**.

• **Centralisé :** La principale différence entre les trois types de *Blockchain* réside dans le fait que la *Blockchain* publique est décentralisée, que la *Blockchain* du consortium est partiellement centralisée et que la *Blockchain* privé est entièrement centralisée car elle est contrôlé par un seul groupe **[BCO]**.

• **Processus de consensus :** Tout le monde dans peut rejoindre le processus de consensus de la *Blockchain* publique. Un nœud doit être certifié pour rejoindre le processus de consensus en consortium ou en *Blockchain* privé **[BCO]**.

Nous comparons ces trois types de *Blockchain* de différentes perspectives. La comparaison est listée dans le tableau.I.1.

	Blockchain publique	*Blockchain* autorisé	*Blockchain* privé
Détermination du consensus	Tous les mineurs	Ensemble sélectionné de nœuds	Une organisation
Autorisation de lecture	Publique	Peut-être public ou restreint	Peut-être public ou restreint
immutabilité	Presque impossible falsifié	Peut-être falsifié	Peut-être falsifié
Efficacité	Faible	Haute	Haute
Centralisé	Non	Partiel	Oui
Processus de consensus	Sans permission	Avec permission	Avec permission

Tableau.I.1: Les comparaisons entre *Blockchain* publique, consortium et privée **[BCO]**

6. Les Caractéristiques d'une *Blockchain* :

En résumé, *Blockchain* a les caractéristiques clés suivantes :

• **Décentralisation** : Une transaction dans le réseau *Blockchain* peut être effectuée sans l'authentification par l'organisme central entre deux pairs (*P2P*). De cette manière, *Blockchain* peut significativement réduire les coûts de serveur (y compris le coût de développement et le coût de fonctionnement) et atténuer les goulets d'étranglement au niveau du serveur central **[BCO]**.

• **Persistance** : Étant donné que chacune des opérations d'épandage à travers le réseau doit être confirmée et enregistrée dans des blocs répartis dans l'ensemble du réseau, il

est presque impossible de les falsifier. En outre, chaque bloc serait diffusé validé par d'autres nœuds et les transactions seraient vérifiées. Donc, la falsification pourrait être détectée facilement **[BCO]**.

• **Anonymat** : Chaque utilisateur peut interagir avec le réseau *Blockchain* avec une adresse générée. En outre, un utilisateur peut générer de nombreuses adresses pour éviter une exposition d'identité. Il n'y a plus de partie centrale qui garde les informations privées des utilisateurs. Ce mécanisme préserve une certaine confidentialité sur les transactions incluses dans le *Blockchain* **[BCO]**.

• **Auditabilité** : Étant donné que chacune des transactions sur la *Blockchain* est validée et enregistrée avec un horodatage, les utilisateurs peuvent facilement vérifier et retracer les enregistrements précédents. Dans Bitcoin chaque transaction pourrait être attribuée à des transactions antérieures itérativement. Il améliore la traçabilité et la transparence des données stockées dans la *Blockchain* **[BCO]**.

7. Utilisations :

Ces dernières années, la *Blockchain* a commencé à être reconnue par un public plus large, ce qui a entraîné une augmentation significative du nombre de services proposés et d'applications logicielles, qui seraient basés sur la *Blockchain*.

Malheureusement, comme cela est courant au début de l'adoption, il existe de nombreuses propositions et attentes irréalistes, c'est pourquoi il est très important de bien comprendre les limites de la *Blockchain*, ses applications et avantages possibles.

Il n'est pas surprenant que la plupart des nombreuses tentatives d'utilisation de la *Blockchain* se produisent dans le secteur financier. Cela est dû en grande partie à Bitcoin et à d'autres crypto-monnaies, qui ont présenté la *Blockchain* au reste du monde. Hileman & Rauchs ont estimé qu'environ 30% des cas d'utilisation de la *Blockchain* étaient liés à des services bancaires et financiers. Elle a également gagnée du terrain dans d'autres secteurs, tels que le gouvernement (13%), les assurances (12%) et le *healthcare* (8%). Cependant, comme l'indiquent plusieurs auteurs, la technologie *Blockchain* est encore au stade initial du cycle d'adaptation et les attentes sont actuellement très exagérées **[BUCTF]**.

En raison du développement rapide de la *Blockchain*, de nouvelles solutions apparaissent également chaque jour. Le tableau ci-dessous est destiné à donner un aperçu et une idée générale des solutions les plus populaires actuellement utilisées. Certains d'entre elles n'en sont encore qu'au stade des tests ainsi, toutes les fonctionnalités ne sont pas présentes. La classification des cas d'utilisation en catégories sert uniquement à faciliter la vue d'ensemble et ne doit pas être considérée comme une classification officielle **[BUCTF]**.

Catégorie	Cas d'utilisation	Applicateur
Gestion des données	Infrastructure réseaux	Eris, Mastercoin, Chromaway
	Contenu et ressource distribution	Swarm
	Stockage en ligne (*Cloud*)	Maidsaife, PeerNova
	Surveillance des données (Data monitoring)	Modum.io
	Gestion des identités de données	UniquID, SolidX, OneName, uPort Microsoft
	Gestion des contrats	Ethereum, Mirror, Symbiont
	Gestion des données inter-organisation	Mulitichain
	Journal et piste d'audit	Factom, Securechain
	Stockage méta données de system	Blockstack
	Réplication des données et protection contre la suppression	Securechain
	Edition de contenu numérique et la vente	Alexandria.io, Ascribe
	Achat des données de capteurs internet d'objet	DataBrokerDAO, Chimera, Filament

Vérification des données	Preuve des photos/vidéos	Uproov
	Notarisation des documents	BitCourt, Blocksign
	Vérification de l'historique du travail	APPII
	Certification académique	Sony Global Eduction
	Vérification de l'identité et gestion des clés	Microsoft, Authentichain, Everpass
	Vérification de la qualité des produits	Everledger, Veerisart, Bitshares
	Preuve de l'origine	Provenance, ArtPlus, Tierion
Finance	Financement du commerce	Barclays, Santander, BNP Paribas
	Paiement poste à poste	Codius, Bitbond, BTCjam
	Financement participatif	Waves, Starbase
	Assurance	Insurechain
	Part d'action et obligation d'émission	Chain
	Emission de l'argent des banques centrales	La Suède, La Russie (sur le niveau de l'idée)
	Transfert de la valeur	Bitcoin, Ripper, Litecoin, Monero
	Echange de devise et remise	Kraken, Bitstamp, Bitso, Coincheck
	Gestion des chaines logistiques	Eaterra, Profeth
	Système de vote social	ThanksCoin
	Enregistrement de don de domaine	Namecoin
	Enregistrement de dossiers de santé	Medicare, BitrHealth, DNA bits
	Validation du License de	IBM

Autre	logiciel	
	Horodatage du contenu ou d'un produit	Nexus Group
	Enregistrement du droit de propriété	Georgia Land Register, Ascribe, Chroma way, Bit Land
	Création / suivi de notation sociale	SOMA
	Le vote aux élections	Ballotchain, European Parliament
	L'enregistrement de mariage	Borderless.tech
	Les procédures judiciaires	PrecedentCoin
	Donations	BitGive
	Serrures électroniques	Slock.it
	Ventre d'électro énergie	TransActive Grid
	Jeu (*Gamming*)	PlayCoin, Deckbound
	Avis et approbation	TRST.im, Asimov, The World Table

Tableau.I.2 : Cas d'utilisations de la *Blockchain* **[BUCTF]**

La *Blockchain* dans la gestion d'actifs : Il s'agit de transférer en toute sécurité des actifs au sein d'un réseau professionnel. Un actif peut être physique, par exemple un serveur, un ordinateur, ou un actif immatériel tel qu'un logiciel ou un service. La *Blockchain* offre une fonctionnalité de grand livre partagé, ce qui signifie une visibilité complète d'un bout à l'autre du réseau de l'entreprise. Elle se concentre uniquement sur cinq événements clés, à savoir la fabrication en série d'actifs pour initier la *Blockchain*, la réception et la validation d'actifs, la capitalisation d'actifs, l'activation de la garantie et l'installation de l'actif **[EYWB]**.

La *Blockchain* dans le Finance : Les paiements transfrontaliers constituent un processus très important, qui devient assez coûteux et fastidieux en raison de la présence d'intermédiaires inutiles. Il faut plusieurs banques (et devises) avant que

l'argent puisse être collecté. Des services tels que *Western Union* peuvent être utilisés, ce qui est plus rapide mais aussi coûteux. La *Blockchain* peut accélérer et simplifier ce processus en éliminant les intermédiaires inutiles. Dans le même temps, cela rend les envois de fonds plus abordables. Jusqu'à présent, les coûts d'envoi de fonds étaient de 5 à 20%. La *Blockchain* réduit les coûts à 2-3% du montant total et permet des transactions transfrontalières garanties en temps réel **[EYWB]**.

La *Blockchain* dans l'*IOT* : Les solutions IOT (*Internet of things*) utilisant les *Blockchain* peuvent être construites pour maintenir une liste d'enregistrement de données croissante sans cesse et sécurisée par la cryptographie, protégée contre toute modification ou altération. Par exemple, lorsqu'un actif connecté à l'*IOT* (RFID, par exemple) avec des informations de localisation et de température sensible se déplace le long de divers points d'un entrepôt ou d'une maison intelligente, ces informations pourraient être mises à jour sur une *Blockchain*. Cela permet à toutes les parties concernées de partager des données et le statut du paquet lors de son déplacement entre différents rassemblements afin de garantir le respect des termes d'un accord **[EYWB]**.

La *Blockchain* dans la santé : Les antécédents médicaux du patient sont stockés dans un système décentralisé, accessible aux médecins traitants et aux prestataires d'assurance médicale **[EYWB]**.

8. Technologies :

Les technologies de *Blockchain* telles que Bitcoin et Ethereum ont connu une croissance rapide. Et chacune peut être perçue comme une étape distincte de l'évolution de cette technologie. Le bitcoin qui a jeté les bases des technologies de *Blockchain* en général peut être vu comme *Blockchain* 1.0. Cela a permis l'évolution d'autres crypto-monnaies.

L'introduction d'algorithmes à exécution autonome, ainsi que de Smart contracts par Ethereum peuvent être vu comme *Blockchain* 2.0, cette étape d'évolution a été la base de l'émergence d'une application décentralisée appelée *DApps*. Les deux étapes d'évolution ont quelques problèmes, comme l'évolutivité donc une troisième génération des *Blockchain* arrive avec des projets qui visent l'amélioration du principe de *Blockchain*.

8.1. *Blockchain* 2.0 :

Proposé par Vatalik Buterin en 2013, Ethereum est une plate-forme informatique distribuée basée sur une *Blockchain* publique. Contrairement à la *Blockchain* 1.0 telle que Bitcoin, elle peut fonctionner comme un ordinateur, même si les performances seront plus lentes que la plupart des ordinateurs actuels, car le temps de transaction est d'environ 12 secondes. Mais, comme il a son propre langage tel que Solidity ou Serpent, il permet aux développeurs d'écrire et de compiler un programme. Une fois compilé, il peut fonctionner sur la machine virtuelle Ethereum. C'est révolutionnaire parce que cela donne aux développeurs la possibilité d'écrire un code qui peut être exécuté sur une *Blockchain* **[MiotD]**.

8.2 *Blockchain* 3.0 :

- **ICON *:***

Le projet ICON vise « Hyper-connecter le monde » en « construisant l'un des plus grands réseaux décentralisés du monde ». Ll'équipe d'ICON est en train de construire une plate-forme massive qui permettra aux différentes *Blockchain* d'interagir entre elles par le biais de *Smart contracts*, les communautés se connecteront entre elles en utilisant la technologie loopchain d'ICON. Le but d'ICON est de fournir une plateforme où les acteurs des secteurs financiers, de la sécurité, de l'assurance, de la santé, de l'éducation et du commerce et au-delà peuvent coexister et interagir sur un seul réseau **[TNGB]**.

- **IOTA *:***

L'année passée, la technologie de la *Blockchain* a fait l'objet d'une recherche et d'un développement scientifiques de plus en plus poussés.

Cependant, des enquêtes sur des cas d'utilisation pratiques ont révélé que des caractéristiques spécifiques de Bitcoin ou d'Ethereum empêchent la technologie de trouver une application générale au sein de l'industrie 4.0 qui correspond à une nouvelle façon d'organiser les moyens de production. Cette nouvelle industrie s'affirme comme la convergence du monde virtuel, de la conception numérique, de la gestion (opérations, finance et *marketing*) avec les produits et objets du monde réel **[TNGB]**.

- **Cardano :**

Le projet Cardano a débuté en 2015 et a été publié en 2017. C'est plus qu'une crypto-monnaie, Cardano est une plate-forme technologique plus importante. C'est la maison de la crypto-monnaie Ada. Cette monnaie est utilisée pour envoyer et recevoir des fonds numériques. La plate-forme Cardano elle-même est capable d'exécuter des applications financières. À l'heure actuelle, elle est utilisée par les gouvernements, les particuliers et les organisations **[TNGB]**.

La plate-forme est construite en couches, cela donne au système la possibilité d'être plus facilement maintenable et les *hard forks* sont autorisées. La crypto-monnaie ADA de Caradno a une capitalisation boursière d'environ 3 milliards d'euros. Cela en fait le 8ème endroit pour la plus grande capitalisation boursière des crypto-monnaies **[TNGB]**.

- **GOLEM :**

Une autre branche remarquable des projets de *Blockchain* 3.0 est le développement d'applications décentralisées dans une boutique de type *App Store*. Des plates-formes Internet telles que "*State of the DApps*" regroupent plus de 1 400 applications prêtes à être utilisées par le public, l'une de ces applications décentralisées est Golem, une plate-forme *peer-to-peer* pour profiter de la puissance de calcul inutilisée **[TNGB].**

Golem utilise des contrats intelligents basés sur Ethereum et a été l'un des tout premiers projets de chaînes de télévision financés par ICO.

Dans les médias, l'application est souvent appelée *Airbnb for Computers*, Golem a pour objectif principal de combiner et d'utiliser la puissance de calcul inutilisée d'un poste à l'autre **[TNGB]**.

Tous les projets présentés ont été ciblés pour résoudre les problèmes actuels de la technologie de diffusion distribuée. Par conséquent, les projets de plate-forme tels qu'ICON et Cardanostrive permettent de résoudre le problème d'interopérabilité en jouant le rôle d'intermédiaire et de gouverneur. Ainsi, non seulement l'échange entre *Blockchains* est rendu possible, mais une entité de contrôle standardisée est également définie, ce qui accroît la sécurité. De plus, IOTA et sa technologie Tangle se sont concentrés sur l'un des défis les plus grands, atteignant une grande évolutivité en termes de transactions gérées par seconde. Enfin, Golem, en tant que représentant du

développement d'applications décentralisées, aide à mieux utiliser les ressources informatiques **[TNGB]**.

10. Conclusion :

La *Blockchain* est une innovation informatique qui permet d'organiser les échanges de données sur un réseau distribué, assurant une sécurisation des données par chiffrement et faisant participer les nœuds du réseau pour la vérification des transactions et la création de nouveaux blocs de la chaîne.

Blockchain a montré son potentiel de transformation de l'industrie traditionnelle avec ses principales caractéristiques : la décentralisation, consistance, anonymat et auditabilité.

La technologie des *Blockchain* continue d'affecter tous les secteurs d'activité, aussi bien de manière positive que négative, de sorte que nous devons faire preuve de vigilance quant à ses implications en termes de sécurité. Il y a beaucoup de projets en cours de développement qui vont améliorer encore plus cette technologie en la rendant plus utile, plus flexible et plus sécurisée.

Chapitre II: Les Smart contracts

1. Introduction :

Après l'émergence et le succès de la *Blockchain* 1.0 qui traite juste la crypto-monnaie, une nouvelle génération est apparue *Blockchain* 2.0 avec de nouveaux concepts et différents cas d'utilisations Cette révolution a introduit la possibilité d'écrire des programmes en mode Turing-complet qui s'exécutent sur le système *Blockchain*, ces dernier s'appellent des *Smart contracts*.

Dans ce chapitre, nous allons présenter une étude détaillée à propos des *Smart contracts*, nous allons expliquer les notions de bases des *Smarts contracts* et les travaux en cours, les principes de fonctionnement de ces derniers et leurs limites, ces propriétés et la façon de déploiement, les différents secteurs d'application puis nous allons nous focaliser sur la sécurité d'internet des objets avec les *Blockchains* en général et les *Smarts contracts* plus précisément.

2. Historique:

Nick Szabo a théorisé pour la première fois les contrats intelligents à la fin des années 90, mais il a fallu presque 20 ans pour que leur potentiel et leurs avantages réels soient vraiment appréciés. Les *Smart contracts* sont décrits par Szabo comme suit:

"*Un Smart contract est un protocole de transaction informatisé qui exécute les termes d'un contrat. Les objectifs généraux sont de satisfaire les conditions contractuelles communes (telles que les conditions de paiement, les privilèges, la confidentialité et même l'application), de minimiser les exceptions, tant malveillantes qu'accidentelles, et la nécessité de disposer d'intermédiaires de confiance. Les objectifs économiques associés incluent la réduction des pertes liées à la fraude, les coûts d'arbitrage et d'exécution, ainsi que d'autres coûts de transaction*. "

L'idée de *Smart contract* a été mise en œuvre de manière limitée dans Bitcoin en 2009, où les transactions en Bitcoin peuvent être utilisées pour transférer la valeur entre les utilisateurs, sur un réseau *peer to peer* où les utilisateurs ne se font pas nécessairement de confiance et aucun intermédiaire de confiance n'est nécessaire **[IBMB].**

3. Présentation des *Smart contracts* et travaux en relation :

3.1. Un contrat :

Un contrat est une convention, accord de volontés ayant pour but d'engendrer une obligation d'une ou de plusieurs personnes envers une ou plusieurs autres. (Quatre conditions sont nécessaires pour la validité du contrat : le consentement des parties, la capacité de contracter, un objet certain, une cause licite.) **[DR]**.

3.2. Un *Smart Contract*:

Greenspan fournit cette définition: "*A Smart Contract is a piece of code which is stored on a Blockchain, triggered by Blockchain transactions, and which reads and writes data in that Blockchain's database*". (2016)

Un *Smart contract* est un code qui s'exécute sur la *Blockchain* pour appliquer les termes d'un accord. L'objectif principal d'un *Smart contract* est d'exécuter automatiquement les conditions d'un accord une fois que les conditions spécifiées sont remplies. Ainsi, les *Smart contracts* promettent un faible frais de transaction par rapport aux systèmes traditionnels qui nécessitent un tiers de confiance pour appliquer et exécuter les termes d'un accord (Voir figure.II.1) **[BBSCSM]**.

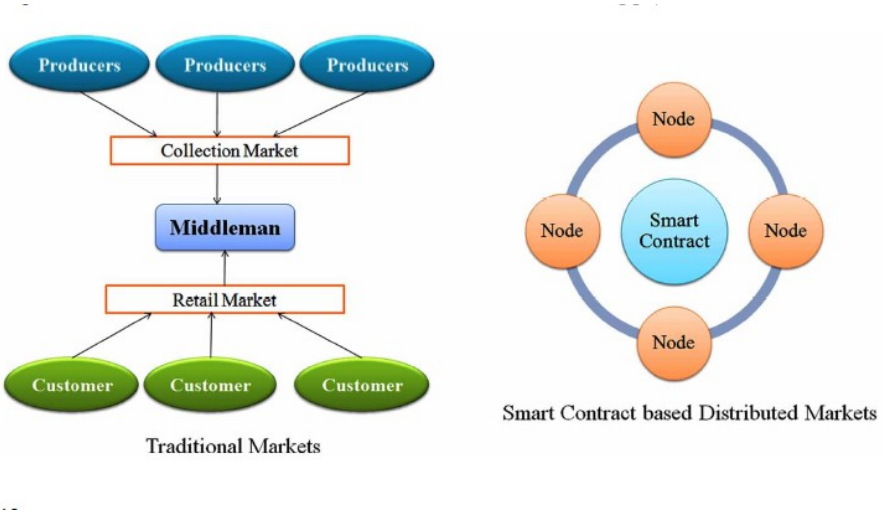

Figure.II.1: Comment un *Smart contract* élimine les intermédiaires dans la chaîne d'approvisionnement **[DCUBTS]**

Un *Smart contract* peut être considéré comme un système qui transmet des actifs numériques à toutes ou à certaines des parties impliquées une fois que des règles prédéfinies ont été respectées. Par exemple, Alice envoie X unités monétaires à Bob si elle reçoit Y unités monétaires de Carl 8**[BBSCSM]**.

Les *Smart contracts* héritent plusieurs caractéristiques de la *Blockchain* comme suite :

a. Suppression d'intermédiaires :

Tout comme la *Blockchain*, le *Smart contract* permet d'automatiser un processus, tout en supprimant les intermédiaires. Un grand nombre de domaines pourraient en bénéficier comme le domaine légal, dans la musique, les réseaux sociaux, la publicité et bien d'autres. Cette technologie permettant de réduire les coûts tout en repensant la manière de participer et d'être récompensé pour cela **[ASCB]**.

b. Inarrêtable :

L'intégralité du code est immuable, dans le sens que son code est publié et donc inscrit dans une transaction de la Blockchain., ce qui rend l'application inattaquable et inarrêtable **[ASCB]**.

c. Sans frontière :

L'avantage d'être inarrêtable et sans autorité de contrôle est que le contrat n'est plus dépendant des frontières, de sa localisation physique et de la juridiction de son pays **[ASCB]**.

d. Open source :

Accessible et vérifiable (au minimum sous forme compilée ce qui suffit pour être renversable par retro engineering et obtenir le code source). Il est donc préférable que le code source soit également fourni afin de permettre à la communauté de comprendre la logique du contrat sans avoir à passer par une étape de retro engineering. De plus, une fois rendu public, il est possible de profiter pleinement d'un système de primes en ligne permettant à la communauté de trouver des bugs lors des tests **[ASCB]**.

Malgré cela, certains préfèrent dissimuler leur savoir-faire et les failles non découvertes en rendant le code source difficile à accéder.

3.3. Un Contrat légal intelligent :

Il signifie un code pour compléter ou remplacer les contrats légaux. La capacité de ce contrat ne dépend pas de la technologie, mais plutôt sur les institutions juridiques, politiques et commerciales **[IBMB].**

3.4. Les contrats Ricardiens :

Ils ont été proposés à l'origine dans le document Cryptographie financière d'Ian Grigg à la fin des années 90. Ces contrats étaient initialement utilisés dans un système de négociation et de paiement appelé Ricardo. L'idée principale est d'écrire un document qui soit compréhensible et acceptable à la fois par un tribunal et par un logiciel. Les contrats Ricardiens relèvent le défi de l'émission de valeur sur Internet. Il identifie l'émetteur et englobe tous les termes et clauses du contrat dans un document afin de le rendre acceptable en tant que contrat juridiquement contraignant. Basé sur la définition originale d'Ian Grigg **[iang]** un contrat Ricardien est un document qui possède plusieurs des propriétés suivantes **[IBMB]** :

- Un contrat proposé par un émetteur à ses titulaires
- Un droit précieux détenu par les titulaires et géré par l'émetteur
- Facilement lisible par les gens (comme un contrat sur papier)
- Lisible par les programmes (analysable, comme une base de données)
- Signé numériquement
- Porte les clés et les informations du serveur
- Allié avec un identifiant unique et sécurisé

4. Principe de fonctionnement des *Smart Contracts* :

Un *Smart contract* a une adresse, un solde de compte, une mémoire privée et un code exécutable. L'état du contrat comprend le stockage et le solde du contrat. L'état est stocké sur la *Blockchain* et il est mis à jour chaque fois que le contrat est invoqué. La figure.II.2 illustre le système de contrat intelligent **[BBSCSM]**.

Chaque contrat sera attribué à une adresse unique de 20 octets. Une fois le contrat déployé dans la *Blockchain*, le code du contrat ne peut être modifié. Pour exécuter un contrat, les utilisateurs peuvent simplement envoyer une transaction à l'adresse du

contrat. Cette transaction sera ensuite exécutée par tous les nœuds de consensus (appelés mineurs) du réseau afin de parvenir à un consensus sur sa sortie **[BBSCSM]**.

L'état du contrat sera ensuite mis à jour, en conséquence le contrat peut sur la base de la transaction, recevoir, lire/écrire à son stockage privé, stocker de l'argent dans son compte, envoyer/recevoir des messages ou de l'argent des utilisateurs/autres **[BBSCSM].**

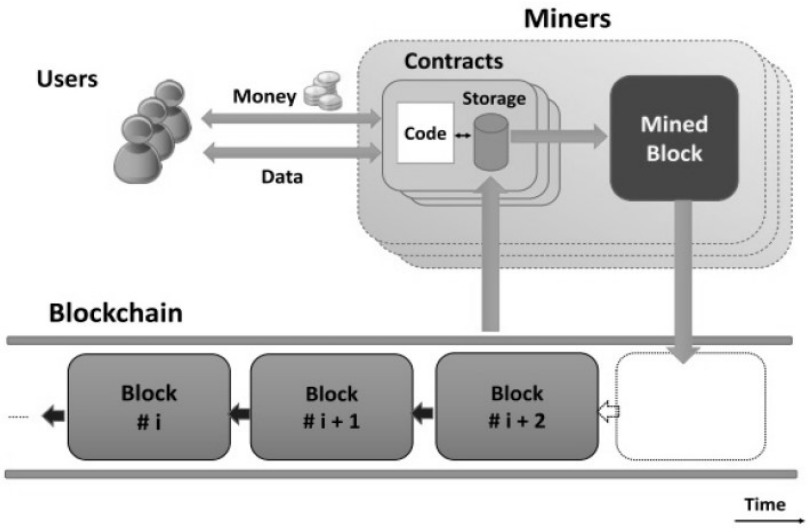

Figure.II.2: Système d'un *Smart Contract* **[BBSCSM]**

5. Types des *Smart Contracts* :

Il existe deux types de *Smart contract*, déterministes et non déterministes.

Un *Smart contract* déterministe : est un contrat intelligent qui, lorsqu'il est exécuté, ne nécessite aucune information émanant d'une partie externe (extérieure à la *Blockchain*) **[BBSCSM]**.

Un *Smart contract* non déterministe : est un contrat intelligent qui dépend d'informations (appelées oracles ou flux de données) provenant d'une partie externe. Par exemple, un contrat nécessitant l'exécution des informations météo actuelles, qui n'est pas disponible dans la *Blockchain* **[BBSCSM].**

6. Les plateformes des Smart Contracts :

Les *Smart contracts* peuvent être développés et déployés dans différentes plateformes de chaîne de blocs (p. ex., Ethereum, Bitcoin et NXT). Certaines plateformes soutiennent des langages de programmation de haut niveau pour

développer des *Smart contracts*. Nous nous concentrerons uniquement sur trois plateformes publiques.

6.1. Bicoin :

Bitcoin est une plate-forme *Blockchain* publique qui peut être utilisée pour traiter les transactions crypto-monnaie, mais avec une capacité de calcul très limitée. Bitcoin utilise un *stack-based bytecode scripting language*. La capacité de créer un contrat intelligent avec une logique riche en utilisant Bitcoin script est très limitée. Dans Bitcoin, une logique simple qui nécessite plusieurs signatures pour signer une seule transaction avant de confirmer le paiement est possible. Toutefois, il n'est pas possible de rédiger des contrats ayant une logique complexe en raison de langage de script Bitcoin, par exemple, il ne supporte pas les boucles **[BBSCSM]**.

6.2. NXT :

NXT est une plate-forme *Blockchain* publique qui inclut des *Smart contracts* intégrés comme modèles (*Templates*). NXT permet seulement de développer des *Smart contracts* à l'aide de ces modèles. Cependant il ne permet pas de réaliser des *Smart contrats* personnalisés en raison du manque d'exhaustivité *Turing* dans son langage de script **[BBSCSM]**.

6.3. Ethereum :

Ethereum est une plate-forme *Blockchain* publique qui peut soutenir des *Smart contracts* personnalisés et avancés à l'aide du langage de programmation *Turing-complet*. La plateforme Ethereum peut soutenir les limites de retrait, les boucles, les contrats financiers. Le code des *Smart contracts* Ethereum est écrit dans un langage de *bytecode* basé sur la pile (*stack-based bytecode language*) et exécuté en Ethereum Virtual Machine (EVM) **[BBSCSM]**.

Plusieurs langages (Solidité, Serpent et LLL,..) peuvent être utilisés pour écrire un *Smart contract*. Le code de ces langages peut ensuite être compilé en EVM *bytecodes* pour être exécuté. Ethereum est actuellement la plate-forme la plus commune pour développer des contrats intelligents **[BBSCSM]**.

7. Exécution des *Smart Contracts* :

Les *Smart contracts* doivent par nature être déterministes. Cette propriété permettra l'exécution d'un *Smart contract* par n'importe quel nœud d'un réseau et produira le

même résultat. Si le résultat diffère même légèrement entre les nœuds, le consensus ne peut pas être atteint et tout un paradigme de consensus distribué sur la *Blockchain* peut échouer **[IBMB]**.

Une fonction déterministe garantit que les *Smart contract* produisent toujours le même résultat pour un intrant spécifique. En d'autres termes, les programmes une fois compilés génèrent une logique précise, parfaitement conforme aux exigences programmées dans le code de haut niveau **[IBMB]**.

8. Oracle :

Les Oracles sont un élément important de l'écosystème des *Smart contracts*. La limite de ces derniers est qu'ils ne peuvent pas accéder aux données externes (hors *Blockchain*) qui pourraient être nécessaires pour contrôler l'exécution de la logique applicative.

Un Oracle est une interface qui fournit des données d'une source externe à des *Smart contracts* selon le secteur et les exigences, l'Oracle peut fournir différents types de données, allant des bulletins météo aux actualités du monde réel, en passant par les opérations sur titres, aux données provenant des dispositifs de l'Internet des objets (IoT). Les Oracles sont des entités de confiance qui utilisent un canal sécurisé pour transférer des données vers un *Smart contract* **[IBMB]**.

Les Oracles sont également capables de signer numériquement les données, prouvant ainsi que la source des données est authentique. Les *Smart contracts* peuvent ensuite être abonnés aux Oracles, et peuvent extraire les données ou Oracles peut les transférer vers les *Smart contracts* **[IBMB]**.

8.1. Oracles standards :

Il est également nécessaire qu'un Oracle ne puisse pas manipuler les données qu'il fournit et qu'il soit capable de fournir des données authentiques. Même si on fait confiance aux oracles, il peut toujours être possible que les données soient incorrectes en raison de manipulation. Il serait peut-être acceptable pour un concepteur d'un *Smart contract* d'accepter des données pour un oracle fourni par un grand tiers de confiance, mais le problème de la centralisation reste. Ces types d'oracles peuvent être appelés oracles standards ou simples **[IBMB].**

8.2. Oracle décentralisé :

Un autre type d'Oracle, qui a essentiellement vu le jour en raison des exigences de décentralisation, peut être appelé Oracle décentralisé. Ces types d'Oracles peuvent être construits à partir d'un mécanisme distribué. On peut également envisager que les Oracles puissent eux-mêmes s'approvisionner en données auprès d'une autre *Blockchain* pilotée par consensus, garantissant ainsi l'authenticité des données. Par exemple, une institution exécutant sa propre *Blockchain* privée peut publier ses données alimentées via un Oracle qui peut ensuite être consommé par d'autres *Blockchains* **[IBMB].**

8.3. Le concept Oracle matériels :

Les chercheurs ont également introduit un autre concept de matériel Oracles, qui requièrent des données réelles provenant de périphériques physiques. Par exemple, cela peut être utilisé en télémétrie (Procédé technique permettant de mesurer la distance d'un objet lointain par utilisation d'éléments optiques) et en IOT. Cependant, cette approche nécessite toutefois un mécanisme dans lequel les périphériques matériels ne peuvent pas être altérés. **[IBMB]**.

8.4. L'écosystème d'Oracle et les Smart contracts :

Il existe actuellement des plates-formes disponibles pour permettre à un *Smart contract* d'obtenir des données externes à l'aide d'Oracle. Il existe différentes méthodes utilisées par Oracle pour écrire des données dans la *Blockchain* en fonction du type de *Blockchain* utilisé.

Par exemple, dans bitcoin, un oracle peut écrire des données dans une transaction spécifique, et un *Smart contract* peut surveiller cette transaction et lire les données. Divers services Oracle en ligne sont disponibles ici **[Oracle].**

En outre, un autre service disponible sur **[Sm]** fournit des données externes et permet d'effectuer des paiements à l'aide de *Smart contract*. Le but de tous ces services est de permettre à ce dernier d'obtenir les données dont il a besoin pour s'exécuter et prendre des décisions **[IBMB].**

Le diagramme suivant illustre un modèle générique d'écosystème Oracle et *Smart contract* :

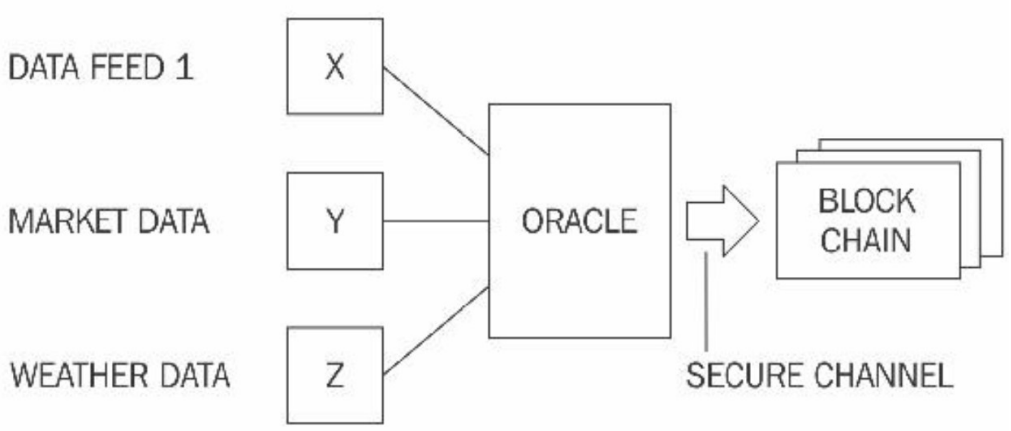

Figure.II.3: Un modèle générique d'oracle en interaction avec un *Smart contract* sur la *Blockchain* **[IBMB]**

9. Déploiement des *Smart contracts* :

Ethereum est un exemple de *Blockchain* qui prend en charge de manière native le développement et le déploiement des *Smart contracts*. À titre de comparaison, dans la *Blockchain* bitcoin, le champ lock_time de la transaction bitcoin peut être considéré comme un outil permettant d'activer une version de base d'un contrat intelligent. Le champ lock_time permet de verrouiller une transaction jusqu'à une heure spécifiée ou après un certain nombre de blocs, imposant ainsi un contrat de base selon lequel une transaction donnée ne peut être ouverte que si certaines conditions (durée écoulée ou nombre de blocs) sont remplies. Toutefois, il s'agit d'un projet très limité qui ne devrait être considéré que comme un exemple de *Smart contract* de base **[IBMB].**

Dans le cas d'Ethereum les fonctions codées dans un *Smart Contract* sont appelables (si celles-ci sont publiques ce qui est le cas par défaut). Chaque appel prend la forme d'une transaction qui, une fois ajoutée à un bloc, est exécutée par tous les nœuds du réseau, car cela fait partie intégrante de la transition d'état et de la validation du bloc. (Vitalik, 2013b)

Il est également possible de stocker des valeurs, introduire des variables et donc de faire évoluer le *Smart Contract***,** gérer nativement le temps grâce à des mots clés simples en Solidity, qui de manière transparente va se référer à la durée moyenne d'ajouts de blocs d'Ethereum et aux *blockhashs* pour gérer le temps. Il est donc

possible de coder des conditions temporelles. Il est possible d'utiliser des sources de données externes pour obtenir des informations nécessaires provenant de bases de données, d'utilisateurs ou d'autres *Smart Contracts* **[ASCB].**

Finalement, il est possible de procéder à des tests avant la publication officielle du *Smart Contract*. La *Blockchain* Ethereum officielle est appelée *mainNet*, il existe d'autres réseaux alternatifs (appelés *testNet*) où il est possible de faire des tests sans dépenser de vrais Ethers **[ASCB].**

10. Le DAO :

Le DAO (*Decentralised Autonomous Organisation*) est l'un des projets les plus cofinancés et a démarré en avril 2016. Il s'agissait essentiellement d'un ensemble de *Smart contracts* destinés à fournir une plateforme d'investissement. En raison d'un bug dans le code, celui-ci a été piraté en juin 2016 et un montant équivalent à 50 millions de dollars a été détourné de DAO vers un autre compte. Cela a abouti à un *hard fork* sur Ethereum afin de récupérer le montant perdu.

Il convient de noter que la notion de code est la loi, ou des *Smart contracts* imparables, doivent être considérés avec un certain scepticisme, car la mise en œuvre de ces concepts n'est pas assez mature pour justifier une confiance totale et incontestable. Cela ressort clairement des récents événements où la fondation Ethereum a pu s'arrêter et modifier l'exécution de 'The DAO' en introduisant un *hard fork*. Bien que ce *hard fork* été introduit pour de véritables raisons, il va à l'encontre du véritable esprit de décentralisation et la notion de « code est loi ».

D'autre part, une résistance contre cet évènement par certains mineurs qui ont décidé de continuer à exploiter l'activité minière sur la chaîne d'origine ce qu'a abouti à la création d'Ethereum Classic. C'est la *Blockchain* Ethereum originale, sans *hard fork*, où le code est toujours la loi **[IBMB]**.

Cette attaque met en évidence les dangers des *Smart contracts* et le besoin absolu de développer un langage formel pour eux. L'attaque a également souligné l'importance de tests approfondis. Ethereum a récemment découvert diverses vulnérabilités autour du langage de développement des *Smart contracts* **[IBMB]**.

11. Utilisation des *Smart contracts* :

Il existe diverses utilisations possibles des *Smart contracts*, certains d'entre eux sont les suivantes:

11.1. Internet des objets et propriété intelligentes :

Il y a des milliards de nœuds qui partagent des données entre eux par le biais d'Internet. Un cas d'utilisation potentiel de *Blockchain*-basée sur les *Smart contracts* est de permettre à ces nœuds de partager ou d'accéder à différentes propriétés numériques sans un tiers de confiance. Il y'a plusieurs entreprises qui enquêtent sur ce cas d'utilisation par exemple, Slock.it est une entreprise allemande qui utilise des contrats intelligents basés sur Ethereum pour louer, vendre ou partager quoi que ce soit (par exemple, vendre une voiture) sans la participation d'un tiers de confiance **[BBSCSM].**

11.2. Gestion des droits de la musique :

Un cas d'utilisation potentielle consiste à enregistrer les droits de propriété d'une musique dans la *Blockchain*. Un *Smart contracts* peut forcer le paiement pour les propriétaires de musique, une fois qu'une musique est utilisée à des fins commerciales. Il veille également à ce que le paiement est distribué entre les propriétaires de la musique. Ujo est une entreprise qui enquête sur l'utilisation de *Blockchain*-basée sur les *Smart contracts* dans l'industrie de la musique **[BBSCSM].**

11.3. Commerce électronique:

Un cas d'utilisation potentiel consiste à faciliter les échanges entre des parties non fiables (par exemple, vendeur et acheteur) sans tiers de confiance. Cela entraînerait une réduction de frais des échanges. Les *Smart contracts* ne peuvent remettre le paiement au vendeur que lorsque l'acheteur est satisfait du produit ou du service reçu **[BBSCSM].**

11.4. Coder des conditions de paiement divers :

À travers un *Smart Contract* il est possible de définir ses propres règles, entre autres d'écrire les conditions pour déverrouiller des *assets* cryptographiques stockés dans la *Blockchain*. Un exemple d'application est la création de mécanismes de *wallets* multi signatures qui, au lieu de nécessiter la signature de tous ses membres, peut être

redéfini selon la volonté du programmeur. Ainsi, il est possible de demander la signature de deux personnes sur trois, 60% des membres ou ce qui semble pertinent. (Szabo, 2016)

À cela s'ajoute la possibilité d'y intégrer des conditions temporelles, grâce à l'accès aux dates des blocs visibles de manière transparente en Solidity. Par exemple, il pourrait être possible de retirer 20% des fonds tous les mois, ou de recevoir de l'Ether pendant 10 jours puis d'arrêter le *crowdfunding* près de cette période. **[ASCB]**

Le *crowdfunding* est un financement participatif décrivant tous les outils et méthodes de transactions financières qui font appel à un grand nombre de personnes afin de financer un projet.

11.5. KickStarter :

En règle général, un *Smart Contract "KickStarter"* a pour but de demander des donations afin d'aider une organisation à concrétiser un projet, il permet de recevoir des fonds de plusieurs donneurs et de les stocker dans un conteneur dans la *Blockchain*. Il permet donc de connaître le montant total reçu lors de ce levé de fond initial, et de le distribuer aux bénéficiaires. De la même manière que le système de *crowdfunding*, si le contrat expire sans avoir atteint le seuil minimal, les fonds seront retournés aux donneurs **[ASCB]**.

Il existe d'autres applications possibles telles que le vote électronique, le paiement hypothécaire, gestion des droits numériques, assurance automobile, stockage de fichiers distribué, gestion de l'identité et chaîne d'approvisionnement **[BBSCSM].**

12. Conclusion :

Dans ce chapitre nous avons présenté une étude à propos des *Smart contracts*, ses propriétés et ses caractéristiques héritées de la *Blockchain* qui ramènes plus de sécurité dans plusieurs domaines à travers l'immutabilité, la transparence ainsi que la suppression d'intermédiaire. Nous avons étudier la capacité des langages de programmation des *Smart contracts* , ses limites et l'entité de confiance Oracle qui rajoute plus de liberté pour les développeurs en terme d'extraction de données externes.

Chapitre III : Etude de la plateforme Ethereum

1. Introduction :

Dans ce chapitre nous allons présenter une étude sur la structure de la *Blockchain* Ethereum et ses principes de fonctionnement, puis nous allons aborder les principaux éléments et mécanismes qui constituent cet écosystème. Une étude sur le développement des *Smart contracts* sous la plateforme Ethereum sera présentée dans la dernière partie.

Ethereum a été conceptualisé par Vitalik Buterin en novembre 2013. L'idée principale proposée était le développement d'un langage de *Turing complet* permettant le développement de programmes arbitraires (*Smart contract*) pour des applications *Blockchain* décentralisées. Cela contraste avec bitcoin, où le langage de script est très limité et ne permet que des opérations de base et nécessaires.

2. Généralités sur l'Ethereum :

Divers clients Ethereum ont été développés dans différentes langues et sont actuellement les plus populaires go-Ethereum et parity. Go-Ethereum a été développé avec le langage Go, alors que parity a été construit avec le langage Rus. D'autres clients sont également disponibles, mais généralement, le client go-Ethereum appelé geth est suffisant à toutes fins. Mist est un portefeuille convivial doté d'une interface graphique qui exécute geth en arrière-plan pour se synchroniser avec le réseau **[IBMB].**

Chaque version d'Ethereum porte un nom, la première s'appelait *Frontier*, l'actuel est *Homestead*, et la prochaine sera *Metropolis, c*ette dernière se concentre sur la simplification des protocoles actuelle et l'amélioration des performances. La version finale est nommée *Serenity*, ce qui devrait implémenter un algorithme de preuve de participation (Casper). Les autres domaines de recherche ciblés avec *Serenity* comprennent l'évolutivité, la confidentialité et la mise à niveau de la machine virtuelle Ethereum (EVM) **[IBMB].**

2.1. Web 3 :

Web 3.0 est un concept qui propose essentiellement un Web sémantique et intelligent comme une évolution de la technologie Web 2.0 existante. Telle est la

vision d'un écosystème dans lequel les personnes, les applications, les données et le Web sont tous connectés et peuvent interagir de manière intelligente. Avec l'avènement de la technologie *Blockchain*, une idée du Web décentralisé a également émergé, ce qui était en fait la vision originale d'Internet. L'idée centrale est que tous les principaux services, tels que le DNS, les moteurs de recherche et l'identité sur Internet, seront décentralisés dans le Web 3.0. C'est là qu'Ethereum est envisagé comme une plate-forme pouvant aider à concrétiser cette vision **[IBMB].**

2.2 La pile Ethereum (*Ethereum Stack*) :

La pile Ethereum est composée de divers composants, au cœur il y a la *Blockchain* Ethereum fonctionnant sur un réseau *P2P,* et un client Ethereum (généralement geth) s'exécute sur les nœuds et se connecte au réseau à partir duquel la *Blockchain* est téléchargée et stockée localement, il fournit diverses fonctions, telles que le minage et la gestion de compte. La copie locale de la *Blockchain* est régulièrement synchronisée avec le réseau. Un autre composant est la bibliothèque web3.js qui permet une interaction avec geth via l'interface RPC (*Remote Procedure Call*) **[IBMB].**

3. La *Blockchain* Ethereum :

3.1. La crypto-monnaie (ETH-ETC) :

Comme un incitatif pour les mineurs, Ethereum récompense également sa monnaie native appelée Ether, abrégé comme ETH. Après le piratage DAO, un *hard fork* a été proposé afin d'atténuer le problème, par conséquent, il y a maintenant deux *Blockchains* Ethereum, l'un est appelé Ethereum classique et sa monnaie est représentée par ETC, alors que la version du *hard fork* est l'ETH, qui continue à se développer et sur laquelle se réalise un développement actif **[IBMB].**

Ether est considéré par les mineurs comme une récompense monétaire pour l'effort informatique qu'ils déploient afin de sécuriser le réseau en vérifiant et en validant des transactions et des blocs. L'ether est utilisé dans la *Blockchain* Ethereum pour payer l'exécution de contrats sur l'EVM, aussi pour acheter du gaz sous forme de carburant crypté qui est nécessaire pour effectuer des calculs sur la *Blockchain* Ethereum **[IBMB].**

3.2. Le GAS :

Un autre concept clé d'Ethereum est celui du gaz, toutes les transactions sur la chaîne de blocs Ethereum doivent couvrir le coût des calculs qu'elles effectuent, ce dernier est couvert par ce que l'on appelle le gaz ou le crypto-carburant (*crypto-fuel*). Ce gaz est payé d'avance par les initiateurs de la transaction en tant que frais d'exécution, le carburant est consommé à chaque opération **[IBMB]**.

Chaque opération est associée à une quantité de gaz prédéfinie, chaque transaction spécifie la quantité de gaz qu'il est prêt à consommer pour son exécution. S'il manque de gaz avant la fin de l'exécution, toute opération effectuée par la transaction jusque-là est annulée. Si la transaction est exécutée avec succès, le gaz restant est restitué à son auteur **[IBMB].**

Ce concept ne doit pas être confondu avec les redevances minières, qui sont un concept différent qui est utilisé pour payer le gaz à titre de redevance aux mineurs.

3.3. Le Hard forks :

En raison d'importantes mises à niveau de protocole Ethereum un *Hard Fork* est nécessaire. Le protocole a été mis à niveau au numéro de bloc 1 150 000, ce qui a entraîné la migration de la première version d'Ethereum, appelée *Frontier*, vers la deuxième version d'Ethereum appelée *homestead.* **[IBMB]**

Un *fork* récent et involontaire survenu le 24 novembre 2016 à 14:12:07 UTC était dû à un bug dans le mécanisme de journalisation du client geth, la connexion réseau s'est produite au numéro de bloc 2 686 351. En raison de ce bug, geth ne parvenait pas à annuler les suppressions de compte vides dans le cas d'une exception *out-of-gas*. Ce n'était pas un problème de parity. Cela signifie qu'à partir du bloc 2686351, la *Blockchain* Ethereum est divisée en deux, l'une fonctionnant avec des clients parity et l'autre avec geth. Ce problème a été résolu avec la publication de la version 1.5.3 de geth **[IBMB].**

3.4. L'état du monde :

L'état du monde dans Ethereum représente l'état global de la *Blockchain* Ethereum. Il s'agit essentiellement d'une correspondance entre les adresses et les états des comptes. Les adresses ont une longueur de 20 octets. Ce mappage est une structure de données qui est sérialisée à l'aide du préfixe de longueur récursive (RLP) **[IBMB].**

RLP est un schéma d'encodage qui est utilisé dans Ethereum pour sérialiser les données binaires pour le stockage ou la transmission sur le réseau, et aussi pour sauvegarder l'état dans un arbre Patricia (*Practical Algorithm To Retrieve Information Coded In Alphanumeric*) qui est une structure de données compacte permettant de représenter un ensemble de mots adaptée pour la recherche **[IBMB].**

3.5. Les transactions :

Une transaction dans Ethereum est un paquet de données qui contient les instructions, signé numériquement en utilisant une clé privée, qui donne lieu soit à un appel de message, soit à la création d'un contrat. Les transactions peuvent être divisées en deux types en fonction de ce qu'elles produisent **[IBMB]** :

a. Transactions d'appels de messages : Cette transaction produit simplement un appel de message qui est utilisé pour transférer des messages d'un compte à un autre.
b. Transactions de création de contrats : Comme leur nom l'indique, ces transactions donnent lieu à la création d'un nouveau contrat. Cela signifie que lorsque cette transaction est exécutée avec succès, elle crée un compte avec le code associé**.**

Ces deux types sont composés d'un certain nombre d'attributs communs, qui sont :

a. ***Nonce* :** est un nombre qui est incrémenté d'une unité, chaque fois qu'une transaction est envoyée par l'expéditeur. Il doit être égal au nombre d'opérations envoyées et sert d'identificateur unique pour l'opération. Une valeur nonce ne peut être utilisée qu'une seule fois.
b. ***Gas Price* :** Le champ *gasPrice* représente le montant de Wei nécessaire à l'exécution de la transaction.
c. ***Gas limit* :** Le champ *gasLimit* contient la valeur qui représente la quantité maximale de gaz pouvant être consommée pour exécuter la transaction.
d. ***To*** : Comme son nom l'indique, le champ à une valeur qui représente l'adresse du destinataire de la transaction.
e. ***Value* :** La valeur représente le nombre total de Wei à transférer au bénéficiaire, dans le cas d'un compte de contrat, elle représente le solde que le contrat détiendra.

f. ***Signature*** **:** La signature est composée de trois champs, à savoir V, R et S. Ces valeurs représentent la signature numérique et certaines informations qui peuvent être utilisées pour récupérer la clé publique. La signature est basée sur le schéma ECDSA (*Elliptic Curve Digital Signature Algorithm*) qui est un algorithme de signature numérique à clé publique.

   V est une valeur d'un octet qui représente la taille et le signe du point de la courbe elliptique et peut être 27 ou 28. V est utilisé dans le contrat de récupération ECDSA comme ID de récupération. Cette valeur est utilisée pour récupérer (dériver) la clé publique de la clé privée.

   Pour signer une transaction, on utilise la fonction ECDSASIGN, qui prend en entrée le message à signer et la clé privée et produit une valeur d'octet unique(V), (R) une valeur de 32 octets, et (S) une autre de 32 octets. L'équation est la suivante : ECDSASIGN (Message, Clé privée) = (V, R, S)

g. ***Init*** **:** La zone *Init* n'est utilisée que dans les transactions destinées à créer des contrats. Ceci représente un tableau d'octets de longueur illimitée qui spécifie le code EVM à utiliser dans le processus d'initialisation du compte. Le code contenu dans ce champ n'est exécuté qu'une seule fois, lors de la création du compte pour la première fois, et est détruit immédiatement après.

   Init retourne également une autre section de code appelée body, qui persiste et fonctionne en réponse aux appels de message que le compte contrat peut recevoir. Ces appels de message peuvent être envoyés via une transaction ou une exécution de code interne.

h. ***DATA*** **:** Si la transaction est un appel de message, le champ *DATA* est utilisé à la place de *init*, qui représente les données d'entrée de l'appel de message. Il est également illimité en taille et organisé comme un tableau d'octets **[IBMB].**

Les transactions peuvent être trouvées soit dans des pools de transactions, soit dans des blocs de transactions. Lorsqu'un nœud mineur commence à vérifier les blocs, il commence par les transactions les plus payantes du pool de transactions et les exécute une à une. Lorsque la limite de gaz est atteinte ou qu'il ne reste plus de transactions à traiter dans le pool de transactions, l'exploitation commence. Dans ce processus, le bloc est haché à plusieurs reprises jusqu'à ce qu'un *nonce* valide soit trouvé qui, une fois haché avec le bloc, donne une valeur inférieure à la cible de difficulté. Une fois le bloc extrait avec succès, il sera diffusé immédiatement sur le réseau, déclarant le

succès, et sera vérifié et accepté par le réseau. Ce processus est similaire au processus minage de Bitcoin discuté dans le chapitre précédent. La seule différence est que l'algorithme de preuve du travail d'Ethereum est résistant aux ASIC où trouver un *nonce* nécessite une grande mémoire, connu sous le nom d'Ethash **[IBMB]**.

3.6. Les blocs Ethereum :

Les blocs sont les principaux éléments constitutifs d'une *Blockchain*. Les blocs d'Ethereum se composent de divers composants, qui sont décrits comme suit :

- L'en-tête du bloc
- La liste des transactions
- La liste des en-têtes d'*Ommers* ou d'oncles

La liste des transactions est simplement une liste de toutes les transactions incluses dans le bloc. En outre, la liste des en-têtes des oncles est également incluse dans le bloc. La partie la plus importante et la plus complexe est l'en-tête de bloc, qui contient les champs suivants **[IBMB]** :

- **Parent hash :** Il s'agit du hachage Keccak 256 bits de l'en-tête du bloc parent (précédent)**.**
- **Ommers hash :** Il s'agit du hachage Keccak 256 bits de la liste des blocs *Ommers* inclus dans le bloc**.**
- **Bénéficiaire :** Le champ Bénéficiaire contient l'adresse de 160 bits du destinataire qui recevra la récompense d'extraction une fois que le bloc aura été miné avec succès.
- **Racine d'état (*state root*) :** Le champ racine d'état contient le *hachage* Keccak 256 bits du nœud racine de l'arbre d'état (*State trie*). Il est calculé une fois que toutes les transactions ont été traitées et finalisées.
- **Racine des transactions (*transactions root*) :** La racine de la transaction est le hachage Keccak 256 bits du nœud racine de l'arbre des transactions. *Transaction trie* représente la liste des transactions incluses dans le bloc.
- **Recette racine (*receipts root*) :** La racine des recettes est le hachage keccak de 256 bits du nœud racine du processus de réception de la transaction. Ce contrat est composé des recettes de toutes les transactions incluses dans le

bloc. Les reçus de transaction sont générés après le traitement de chaque transaction et contiennent des informations utiles après la transaction.

- **Difficulté :** Le niveau de difficulté du bloc actuel.

La difficulté du bloc augmente si le temps entre deux blocs diminue, alors qu'elle diminue si le temps entre deux blocs augmente. Ceci est nécessaire pour maintenir un temps de génération de bloc à peu près constant.

Si la différence de temps entre la génération du bloc parent et le bloc courant est inférieure à 10 secondes, la difficulté augmente. Si le décalage horaire est de 10 à 19 secondes, le niveau de difficulté reste le même. Enfin, si le décalage horaire est de 20 secondes ou plus, le niveau difficile diminue, cette diminution est proportionnelle au décalage horaire.

En plus de l'ajustement de difficulté basé sur l'horodatage, il y a aussi une autre partie qui augmente exponentiellement la difficulté après chaque 100 000 blocs. C'est ce qu'on appelle la bombe à retardement de difficulté qui rendra très difficile l'exploitation minière sur la *Blockchain* Ethereum à un moment donné dans le futur. Cela encouragera les utilisateurs à passer à PoS, car l'exploitation minière PoW finira par devenir prohibitive **[IBMB].**

- **Nombre :** Le nombre total de tous les blocs précédents (le bloc de *genèse* est le bloc zéro).
- **Limite de gaz :** Le champ contient la valeur qui représente la limite définie pour la consommation de gaz par bloc.
- **Gaz utilisé :** Le champ contient le total de gaz consommé par les transactions incluses dans le bloc.
- **Horodatage :** L'horodatage est l'heure Unix de l'époque de l'initialisation du bloc.
- **Données supplémentaires :** Un champ de données supplémentaire peut être utilisé pour stocker des données arbitraires liées au bloc.
- ***Mixhash* :** Le champ *Mixhash* contient un *hachage* de 256 bits qui, une fois combiné avec le *nonce,* est utilisé pour prouver qu'un effort de calcul adéquat a été déployé afin de créer ce bloc.

- ***Nonce* :** *Nonce* est un nombre 64 bits utilisé pour prouver, en combinaison avec le champ *mixhash*, que des efforts de calcul suffisants ont été déployés pour créer ce bloc.

3.7. Consensus :

Le mécanisme de consensus d'Ethereum est basé sur le protocole GHOST initialement proposé par Zohar et Sompolinsky en décembre 2013.

Greedy Heaviest Observed Subtree (*GHOST*) a été introduit pour la première fois comme un mécanisme permettant d'atténuer les problèmes posés par les temps de génération de blocs rapides qui ont conduit à des blocs périmés ou orphelins. Dans GHOST, les blocs périmés sont ajoutés dans les calculs pour déterminer la *Blockchain* la plus longue et la plus lourde. Les blocs périmés sont appelés oncles ou *Ommers* dans Ethereum **[IBMB].**

3.8. Le mécanisme du Gas :

Le gaz doit être payé pour chaque opération effectuée sur la *Blockchain* Ethereum. Il s'agit d'un mécanisme qui garantit que des boucles infinies ne peuvent pas faire décrocher toute la *Blockchain* en raison de la nature *Turing complet* de l'EVM. Des frais de transaction sont facturés comme une certaine quantité d'éther et sont prélevés sur le solde du compte de l'initiateur de la transaction. Des frais sont payés pour les transactions qui doivent être incluses par les mineurs pour l'exploitation minière. Si ces frais sont trop bas, il se peut que la transaction ne soit jamais minée, plus les frais sont élevées, plus les chances que les transactions soient choisies par les mineurs pour être incluses dans le bloc.

Inversement, si la transaction pour laquelle des frais appropriés ont été payés est incluse dans le bloc par les mineurs mais qu'elle comporte trop d'opérations complexes à effectuer, elle peut donner lieu à une exception de dégazage si le coût du gaz n'est pas suffisant. Dans ce cas, la transaction échouera mais sera quand même intégrée au bloc et le donneur d'ordre de la transaction ne sera pas remboursé **[IBMB].**

Les coûts de transaction peuvent être estimés à l'aide de la formule suivante :

*Total cost = gasUsed * gasPrice*

gasUsed est le gaz total qui est censé être utilisé par la transaction pendant l'exécution et *gasPrice* est spécifié par l'initiateur de la transaction comme une incitation pour les mineurs à inclure la transaction dans le bloc suivant. Des frais sont attribués à chaque code d'opération EVM. Il s'agit d'une estimation car le gaz utilisé peut être supérieur ou inférieur à la valeur spécifiée à l'origine par l'initiateur de la transaction.

Par exemple, si le calcul prend trop de temps ou si le comportement du *Smart contract* change en réponse à d'autres facteurs, la transaction peut alors exécuter plus ou moins d'opérations que prévu et peut entraîner une consommation plus ou moins importante de gaz. Si l'exécution manque du gaz, tout est immédiatement annulé sinon, si l'exécution est réussie et qu'il reste du gaz, il est renvoyé à l'expéditeur de la transaction **[IBMB]**.

Chaque opération coûte du gaz, un barème de frais généraux de quelques opérations est présenté à titre d'exemple (Voire figure.III.1).

Operation Name	Gas Cost
step	1
stop	0
suicide	0
sha3	30
sload	20
txdata	5
transaction	500
contract creation	53000

Figure.III.1: Grille de taxes *gas-operation* **[IBMB]**

Sur la base de la grille de taxes précédente et de la formule évoquée précédemment, un exemple de calcul de l'opération contract creation peut être calculé comme suit :

Le prix actuel de gaz est de 20 GWei, soit 0,000000025 Ether.

Gas total : 0.00000002 * 53000 = 0.000001046 Ether

Au total 0.000001046 Ether correspondent au gaz total qui sera chargé **[IBMB].**

3.9. Validation et exécution des transactions :

Les transactions sont exécutées après vérification de leur validité. Voici les tests initiaux de la vérification **[IBMB]** :

- Une transaction doit être bien formée et codée RLP sans octets supplémentaires.
- La signature numérique utilisée pour signer la transaction est valide.
- Le *nonce* de transaction doit être égal au *nonce* courant du compte de l'expéditeur.
- La limite de gaz ne doit pas être inférieure au gaz utilisé par la transaction.
- Le compte de l'expéditeur contient un solde suffisant pour couvrir les frais d'exécution.

3.10. Le mécanisme de validation des blocs :

Un bloc Ethereum est considéré comme valide s'il réussit les contrôles suivants **[IBMB]** :

- Cohérent avec les oncles et les transactions. Cela signifie que tous les Oncles satisfait la propriété qu'ils sont effectivement des oncles et aussi leurs preuve de travail est valide.
- Si le bloc précédent (parent) existe et est valide.
- Si l'horodatage du bloc est valide. Cela signifie essentiellement que l'horodatage du bloc courant doit être supérieur à celui du bloc parent. Tous les temps de bloc sont calculés en temps Unix.

Si l'un de ces contrôles échoue, le bloc sera rejeté.

3.11. La machine virtuelle Ethereum (EVM) :

EVM est une simple machine d'exécution basée sur des piles qui exécute des instructions *bytecode* afin de transformer l'état du système d'un état à l'autre. La taille de mot de la machine virtuelle est définie sur 256 bits. La taille de la pile est limitée à 1024 éléments et elle est basée sur la file d'attente LIFO (dernier entré, premier sorti).

L'EVM est une machine de *Turing complet*, mais limitée par la quantité de gaz nécessaire pour exécuter toute instruction. Cela signifie que les boucles infinies et les attaques par déni de service ne sont pas possibles en raison des besoins en gaz. EVM

prend également en charge la gestion des exceptions, telles que le manque de gaz ou d'instructions non valides, auquel cas la machine s'arrête immédiatement et renvoie l'erreur à l'agent d'exécution **[IBMB].** EVM est un environnement d'exécution entièrement isolé et le code qui s'exécute sur elle n'a accès à aucune ressource externe.

3.11.1. Type de stockage :

Il existe deux types de stockage disponibles pour les contrats. Le premier s'appelle *memory*, qui est un tableau d'octets. Lorsqu'un contrat termine l'exécution du code, la mémoire est effacée. Cela ressemble au concept de RAM. L'autre type, appelé *storage*, est stocké de manière permanente dans la *Blockchain* **[IBMB].**

Le code du programme est stocké dans une mémoire morte virtuelle (ROM virtuelle) accessible via l'instruction CODECOPY. L'instruction CODECOPY est utilisée pour copier le code du programme dans la mémoire principale. Initialement, tout le stockage et la mémoire sont mis à zéro dans l'EVM **[IBMB].**

La machine virtuelle peut également s'arrêter dans des conditions normales si des opcodes STOP ou SUICIDE ou RETURN sont rencontrés au cours du cycle d'exécution. Le code écrit dans un langage de haut niveau tel que serpent, LLL ou Solidity est converti en code d'octet compris par EVM pour être exécuté **[IBMB].**

3.11.2. Les opcodes:

Ces des suites d'instructions qui représentent le langage natif de l'EVM. Les opcodes sont divisés en plusieurs catégories en fonction de l'opération qu'ils effectuent **[IBMB]** :

- Operateurs arithmétiques.
- Operateurs logiques et cryptographique.
- Instructions d'information environnementales.
- Instructions d'information de bloc.

3.12. Les comptes Ethereum :

Les comptes sont l'un des principaux éléments constitutifs de la *Blockchain* Ethereum. L'état est créé ou mis à jour à la suite de l'interaction entre les comptes. Les opérations effectuées entre et sur les comptes représentent des transitions d'état. La

transition d'état est réalisée à l'aide de la fonction de transition d'état Ethereum, qui fonctionne comme suit **[IBMB]** :

- Confirmez la validité de la transaction en vérifiant la syntaxe, la validité de la signature et le *nonce*.
- Les frais de transaction sont calculés et l'adresse d'envoi est résolue. De plus, le solde du compte de l'expéditeur est vérifié et soustrait en conséquence et le *nonce* est incrémenté. Une erreur est renvoyée si le solde du compte n'est pas suffisant.
- Fournissez suffisamment d'éther (prix du gaz) pour couvrir le coût de la transaction.
- Dans cette étape, le transfert de valeur réel se produit. Le flux va du compte de l'expéditeur au compte du destinataire. Le compte est créé automatiquement si le compte de destination spécifié dans la transaction n'existe pas encore. De plus, si le compte de destination est un contrat, le code du contrat est exécuté. Cela dépend aussi de la quantité de gaz disponible. Si suffisamment de gaz est disponible, le code du contrat sera exécuté intégralement, sinon il fonctionnera jusqu'au point de manquer de gaz.
- En cas d'échec de la transaction en raison de gaz insuffisant, tous les changements d'état sont annulés, à l'exception du paiement de la redevance, qui est versée aux mineurs.
- Enfin, le reste (le cas échéant) des frais est renvoyé à l'expéditeur au fur et à mesure du changement et une redevance est versée aux mineurs en conséquence. À ce stade, la fonction renvoie l'état résultant.

- **Types des comptes :**
  - Comptes détenus en externe
  - Comptes de contrats

Les comptes détenus en externe sont similaires aux comptes qui sont contrôlés par une clé privée dans bitcoin. Les comptes de contrat sont les comptes auxquels est associé le code ainsi que la clé public seulement **[IBMB].**

Un compte détenu en externe a un solde d'éther, est capable d'envoyer des transactions et n'a pas de code associé, alors qu'un compte de contrat a un solde

d'éther, un code associé et la capacité de se déclencher et d'exécuter un code en réponse à une transaction ou un message. Il est à noter qu'en raison de la propriété de *Turing complet* d'Ethereum, le code dans les comptes contractuels peut être de n'importe quel niveau de complexité, ce dernier est exécuté par EVM par chaque nœud mineur du réseau Ethereum. En outre, les comptes de contrats sont en mesure de maintenir leur propre état permanent et peuvent appeler d'autres contrats **[IBMB].**

4. La plateforme Ethereum :

4.1. Clients et portefeuilles :

Comme Ethereum est en plein développement et évolution, de nombreux composants, clients et outils ont été développés et introduits au cours des dernières années. Vous trouverez ci-dessous une liste de tous les composants principaux, du logiciel client et des outils disponibles avec Ethereum. Cette liste est fournie afin de réduire l'ambiguïté autour des nombreux outils et clients disponibles pour Ethereum **[IBMB]**.

- **Geth :** est l'implémentation Go du client Ethereum.
- **Eth :** est l'implémentation C ++ du client Ethereum.
- **Pyethapp :** est l'implémentation Python du client Ethereum.
- **Parity :** cette implémentation est construite avec Rust et développée par EthCore qui est une entreprise qui travaille au développement du client à parity.
- **Clients légers :** Les clients légers ne téléchargent qu'un petit sous-ensemble de la *Blockchain*. Cela permet aux périphériques à faibles ressources, tels que les téléphones mobiles, les périphériques intégrés ou les tablettes, de pouvoir vérifier les transactions **[IBMB].**
- **Mist browser :** Mist browser est une interface utilisateur graphique riche en fonctionnalités, utilisée pour accéder aux applications décentralisées (Dapps), ainsi que pour la gestion des comptes et des contrats. Lorsque Mist est lancé pour la première fois, il initialise geth en arrière-plan et se synchronise avec le réseau. La synchronisation complète avec le réseau peut prendre des heures et/ou jours, selon la vitesse et le type du réseau. Si TestNet est utilisé, la synchronisation se termine relativement vite, car la taille de TestNet n'est pas aussi grande que celle de MainNet **[IBMB].**

4.2. Le Réseau Ethereum :

Le réseau Ethereum est un réseau *peer to peer* des nœuds auquel ces derniers participent afin de maintenir la *Blockchain* et de contribuer au mécanisme de consensus. Le réseau peut être divisé en trois types, en fonction des besoins et de l'utilisation **[IBMB] :**

1. **Main net :**

MainNet est le réseau live actuel d'Ethereum. La version actuelle de MainNet est homestead.

2. **Test Net :**

Pour l'instant il 'y a 3 réseaux de test Ropsten, Kovan, Rinkeby, ils sont utilisées pour tester les *Smart contracts* et les DApp (*Decentralized application*) avant d'être déployée dans le MainNet. De plus, étant un réseau de test, il permet l'expérimentation et la recherche **[IBMB]**.

3. **Private Net :**

Comme son nom l'indique, il s'agit du réseau privé qui peut être créé en générant un nouveau bloc de genèse, où un groupe privé d'entités lance sa propre *Blockchain*. Private Net d'Ethereum, ce qui permet de créer un réseau privé indépendant pouvant servir de grand livre distribué entre les entités participantes et pour le développement et le test de *Smart contracts* **[IBMB]**.

4.3. Protocoles de support :

Divers protocoles de prise en charge sont en cours d'élaboration afin de prendre en charge l'ensemble de l'écosystème décentralisé (Voir figure.III.2).

- **Whisper :** Whisper fournit des capacités de messagerie décentralisées *P2P* au réseau Ethereum, c'est un protocole de communication utilisé par les nœuds pour communiquer entre eux, les données et le routage des messages sont cryptés dans des communications silencieuses. Whisper est également conçu pour fournir une couche de communication qui ne peut pas être tracée et fournit une "communication sombre" entre les parties. La *Blockchain* peut être utilisée pour la communication, mais cela coûte cher et un consensus n'est pas vraiment nécessaire pour les messages échangés entre les nœuds **[IBMB]**.

- **Swarm :** Swarm est en cours de développement en tant que plate-forme de stockage de fichiers. Les fichiers de ce réseau sont adressés par le hachage de leur contenu. Swarm est prévu pour fournir une couche de stockage distribué résistante aux défaillances DDOS (*Distributed Denial of Service*) pour Ethereum **[IBMB]**.

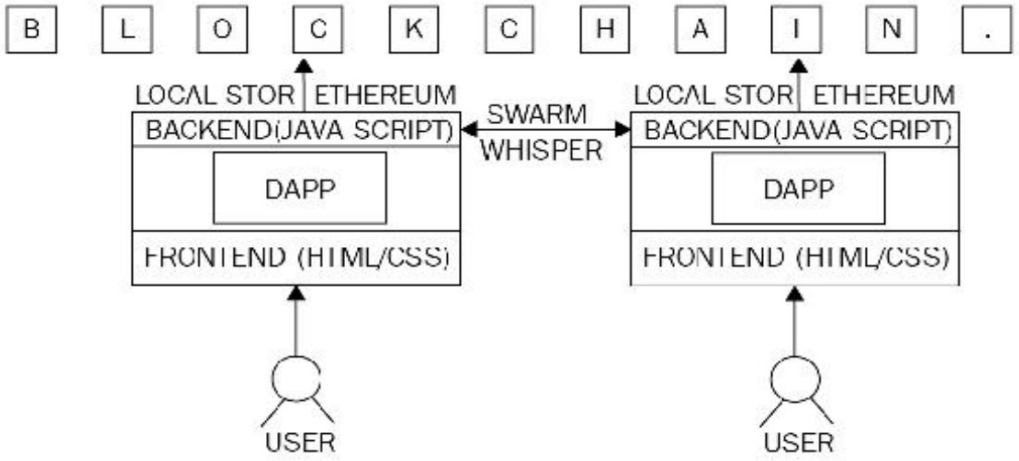

Figure.III.2: Swarm et whisper dans la *Blockchain* **[IBMB]**

4.4. Environnement de développement :

4.4.1. Langages de programmation des *Smart contracts* :

Les contrats peuvent être programmés dans une variété de langages. Quatre langages peuvent être utilisés pour rédiger des contrats **[IBMB]** :

- **Mutan:** Il s'agit d'un langage de style Go, qui a été déconseillé au début de 2015 et qui n'est plus utilisé.
- **Serpent:** Un langage simple et propre, semblable à celui de Python, il est activement utilisé pour le développement de *Smart contract*.
- **Solidity:** Ce langage est devenu presque un standard pour la rédaction des *Smart contracts* Ethereum. Ce langage est l'objet de ce chapitre.

4.4.2. Compilateur :

Les compilateurs sont utilisés pour convertir le code source des *Smart contracts* de haut niveau au format compris par l'environnement d'exécution Ethereum. **Solc** est le compilateur de Solidity qui convertit un code de haut niveau en un *bytecode* Ethereum

Virtual Machine (EVM) afin qu'il puisse être exécuté sur la *Blockchain* par l'EVM **[IBMB]**.

5. Développement et déploiement d'un *Smart contrat* :

Dans cette partie nous allons voir les outils de développement des *Smart contract* sur la plateforme Ethereum, quand et comment les utiliser.

Diverses étapes doivent être franchies pour développer et déployer des *Smart contracts*. En gros, elles peuvent être divisées en quatre étapes : écriture, test, vérification et déploiement **[IBMB].**

L'étape d'écriture consiste à écrire le code source du contrat. Cela peut être fait dans n'importe quel éditeur de texte. Les tests sont généralement effectués par des moyens automatisés comme Truffle, qui utilise le cadre Mocha pour tester les contrats. Cependant, des tests manuels peuvent également être effectués. Une fois que le contrat est vérifié, fonctionne et testé sur un environnement simulé (Private Net), il peut être déployé sur le Test Net Ropsten ou Kovana et enfin, sur le Main Net (*Homestead*) **[IBMB]**.

5.1. Solidity :

C'est un langage à typage statique, ce qui signifie que la vérification du type de variable en Solidity est effectuée au moment de la compilation. Chaque variable, qu'elle soit d'état ou locale, doit être spécifiée avec un type lors de la compilation.

C'est un avantage aux sens que toute validation et vérification est terminée au moment de la compilation et que certains types de bugs, tels que l'interprétation des types de données, qu'elle peuvent être détectés dans le cycle de développement plutôt qu'au moment de l'exécution, ce qui pourrait être coûteux, les autres fonctionnalités du langage incluent l'héritage, les bibliothèques et la possibilité de définir des types de données composites **[IBMB].**

La syntaxe de Solidity est très similaire à C et JavaScript, et il est assez facile à programmer avec le concept orienté objet avec des notions et des mots clés spéciaux, compréhensible par la machine virtuelle Ethereum.

5.2. Installation des outils de développement :

Tout d'abord, nous allons installer et configurer une *Blockchain* privée pour développer un *Smart contract* localement.

5.2.1. La *Blockchain* personnelle Ganache :

Ganche est une *Blockchain* personnelle de développement local. Cela nous permettra de déployer des *Smart contracts*, de développer des applications et d'exécuter des tests**.** Nous avons choisi Ganache car il nous fournit 10 comptes Ethereum avec une balance de 100 ether (du faux ether) pour chaque compte, ainsi une interface graphique qui nous permet d'examiner tout ce qui ce passe dans cette *Blockchain*,

5.2.2. Node.JS :

Nous devons configurer notre environnement pour développer des *Smart contracts*. La première dépendance dont nous aurons besoin est *Node Package Manager*, ou NPM, fourni avec Node.js.

5.2.3. *Framework* Truffe :

Truffle est un *Framework* qui fournit une suite d'outils permettant de développer des *Smart contracts* Ethereum avec le langage de programmation Solidity.

Nous avons choisi Truffle pour le développement de notre application car c'est un *Framework* puissant qui nous facilite l'interaction avec notre *Smart contract* et il nous permet d'effectuer des tests, développer une interface coté client de notre *Smart contract*, la déployé dans n'importe quel réseau Ethereum.

Voici un aperçu de toutes les fonctionnalités du *Framework* Truffle **[DAPP]**:

- **Gestion des *Smart contracts* :** rédiger des *Smart contracts* avec Solidity et les compiler en *bytecode* pour exécuter sur l'Ethereum Virtual Machine (EVM).

- **Tests automatisés :** écrire des tests sur les *Smart contracts* pour assurer qu'ils se comportent comme convenu. Ces tests peuvent être écrits en JavaScript ou Solidity, et peuvent être exécutés sur n'importe quel réseau configuré par Truffle.

- **Deployment et Migration :** écrire des scripts pour migrer et déployer des *Smart contracts* sur n'importe quel réseau Ethereum.

- **Gestion du réseau:** se connecter à n'importe quel réseau public Ethereum, ainsi qu'à tout réseau *Blockchain* personnel pour des fins de développement.

- **Console de développement :** interagir avec des *Smart contracts* dans un environnement d'exécution JavaScript avec la console Truffle. Pour ce faire, il suffit de se connecter à n'importe quel réseau de *Blockchain* spécifié dans la configuration du réseau.

- ***Script Runner* :** écrire des *scripts* personnalisés pouvant être exécutés sur un réseau Ethereum public avec JavaScript. On peut écrire du code arbitraire dans ce fichier et l'exécuter dans notre projet.

- **Développement côté client** : configurer le projet Truffle pour héberger des applications côté client qui communiquent avec les *Smart contracts* déployés dans la *Blockchain* **[DAPP].**

Pour installer le *Framework* Truffle :

```
npm install truffle –g
```

5.2.4. Le portefeuille Metamask :

La plupart des principaux navigateurs Web ne se connectent pas aux réseaux décentralisés, Le plugin Metamask vous permet de transformer un navigateur Web en un navigateur *Blockchain*,

Nous avons choisi Metamask car il permet également la gestion des comptes *Blockchain*, ainsi que les fonds Ether pour payer les transactions **[DAPP].**

5.3. Configuration d'environnement :

Nous allons d'abord créer un répertoire qui va contenir les fichiers de notre projet comme ceci :

$ mkdir MonProjet

$ cd MonProjet

Maintenant, nous initialisons un nouveau projet de truffe pour développer notre projet comme ceci : $ truffle init

Après il faut installer les dépendances par cette commande : $ npm install

Maintenant que les dépendances sont installées, examinons la structure de répertoire de projet que nous venons de créer **[DAPP]** :

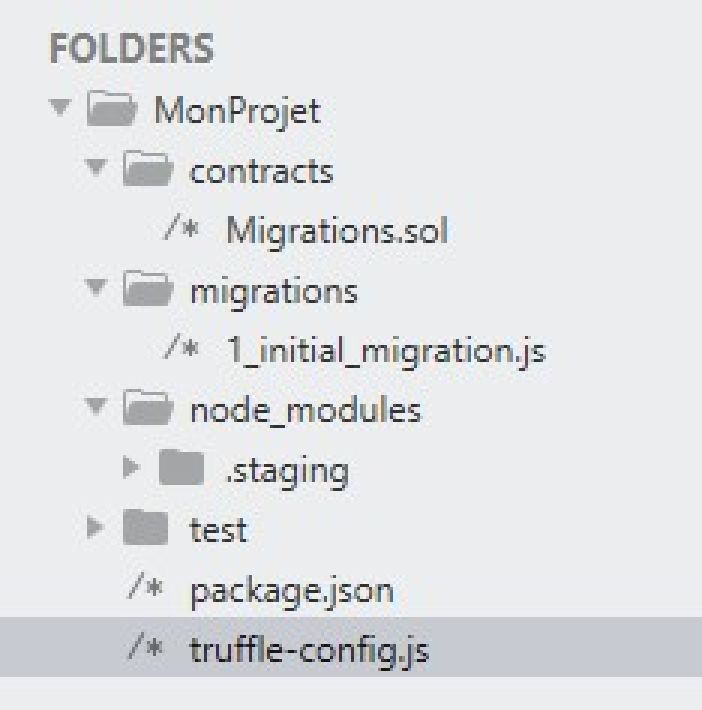

Figure.III.3: Structure de répertoire MonProjet

- **Répertoire des contrats :** c'est là que se trouve tous les *Smart contracts*. Nous avons déjà un contrat de migration qui gère nos migrations vers la *Blockchain*.

- **Répertoire migrations :** c'est là que résident tous les fichiers de migration. Ces migrations sont similaires aux autres infrastructures de développement Web qui nécessitent des migrations pour modifier l'état d'une base de données. Chaque fois que nous déployons des *Smart contracts* sur la *Blockchain*, nous mettons à jour l'état de la *Blockchain* et nous avons donc besoin d'une migration.

- **Répertoire *node_modules* :** c'est le répertoire de toutes nos dépendances de *node* que nous venons d'installer.

- **Répertoire de test :** c'est là que nous allons écrire nos tests pour notre *Smart contract*.

- **Fichier truffle-config.js :** il s'agit du fichier de configuration principal de notre projet Truffle, dans lequel nous allons gérer des tâches telles que la configuration réseau.

Maintenant on peut créer notre *Smart contract* par la création d'un fichier MonContrat.sol dans le répertoire '*contracts*'. On doit spécifier la version de notre compilateur Solidity comme suite :

```
pragma solidity ^0.5.0;
```

Voici un exemple de base d'un contrat :

```
pragma solidity ^0.5.0;
contract Moncontrat {
  uint var = 0;
}
```

La variable var est un type spécial de variable appelé "variable d'état". Toutes les données que nous stockons dans cette variable d'état sont écrites dans la *Blockchain*. Elles modifient l'état du *Smart contract* par opposition aux variables locales qui ne concernent que les fonctions **[DAPP]**.

Nous pouvons maintenant créer un moyen d'accéder à la valeur de cette variable d'état en dehors du contrat. Nous pouvons le faire avec un mot-clé spécial appelé public dans Solidity. Ce faisant, Solidity créera une fonction var () afin que nous puissions accéder à la valeur de cette variable en dehors du *Smart contract*. Cela sera utile lorsque nous interagissons avec dans la console, à partir de l'application côté client et dans les fichiers de test **[DAPP]**.

Maintenant, compilons le *Smart contract* et vérifions qu'il n'y ait pas d'erreur :

```
$ truffle compile
```

Un nouveau fichier est généré après chaque compilation d'un *Smart contract*, à l'emplacement suivant: `./Build/contracts/Moncontrat.json`. Ce fichier est le fichier ABI (*Abstract Binary Interface*) du *Smart contract*, ce dossier a beaucoup de responsabilité, voici les principaux **[DAPP]** :

- Il contient la version compilée en *bytecode* du *Smart contrat* qui peut être exécuté sur une machine virtuelle Ethereum (EVM).
- Il contient une représentation JSON des fonctions de *Smart contrat* pouvant être exposées à des clients externes, telles que des applications JavaScript côté client.

Pour accéder au *Smart contract* sur Ganache à l'intérieur de la console Truffle, nous devons effectuer quelques opérations **[DAPP]** :

- Mettre à jour le fichier de configuration de notre projet pour spécifier le réseau de *Blockchain* personnelle auquel nous voulons nous connecter (Ganache).
- Créer un script de migration qui indique à Truffle comment déployer le *Smart contract* sur le réseau de *Blockchain*.
- Exécutez le script de migration, en déployant le *Smart contract* sur la *Blockchain.*

Pour spécifier le réseau *Blockchain* Ganache que nous souhaitons configurer on met à jour le fichier truffle-config.js comme suite :

```
module.exports = {
 networks: {
  development: {
   host: "127.0.0.1",
   port: 7545,
   network_id: "*" // Match any network id
  } },
 }
```

Ensuite, nous allons créer un script de migration dans le répertoire des migrations pour déployer le *Smart contract* sur le réseau de *Blockchain* personnel, on crée un fichier 2_deploy_contract.js dans le répertoire migration puis on crée le script de migration **[dapp].**

contract :

```
var MonContrat = artifacts.require("./MonContrat.sol");
module.exports = function(deployer) {
  deployer.deploy(MonContrat);
};
```

Premièrement, nous avons besoin du contrat que nous avons créé et l'affectons à une variable appelée "MonContrat". Ensuite, nous l'ajoutons au manifeste des contrats déployés pour nous assurer qu'il est déployé lors de la migration. Ensuite nous exécutons ce script de migration à partir de la ligne de commande, comme suit :

```
$ truffle migrate
```

Maintenant que nous avons migré notre *Smart contract* vers le réseau *Blockchain*. Nous ouvrons la console de Truffe à partir de la ligne de commande comme ceci :

```
$ truffle console
```

Maintenant que nous sommes dans la console, nous obtenons une instance de notre *Smart contracts* déployé et voyons si nous pouvons lire notre variable 'var' du contrat depuis la console, avec ce code :

```
moncontrat = await Moncontrat.deployed()
```

Ici, Moncontrat est le nom de la variable que nous avons créé dans le fichier de migration, nous avons récupéré une instance déployée du contrat avec la fonction deployed(). Notons également l'utilisation du mot-clé await car nous devons interagir avec la *Blockchain* de manière asynchrone. JavaScript est donc un excellent choix pour les interactions côté client avec les *Smart contracts* [**dapp]**.

Nous pouvons obtenir l'adresse du *Smart contracts* qui a été déployé dans la *Blockchain* comme suit:

```
moncontrat.address // => '0xABC123...'
```

Et on peut lire la valeur de **var** comme suite :

```
v = await app.var() // => 0
```

Jusqu'à présent nous avons :

- Configuré un pc pour le développement sur *Blockchain.*
- Créé un nouveau projet Truffle.
- Créé un simple contrat.
- Interagi avec le *Smart contract* sur une *Blockchain* en direct.

5.4. Les tests de *Smart contract* avec Truffle :

L'importance des tests est d'assurer le bon fonctionnement du Smart contrat car **[dapp]** :

- Tout le code de la *Blockchain* Ethereum est immuable, ne change pas. Si le contrat contient des bugs, nous devons le désactiver et déployer une nouvelle copie, cette nouvelle copie n'aura pas le même état que l'ancien contrat et son adresse sera différente.
- Le déploiement des contrats coûte du gaz car il crée une transaction et écrit des données dans la *Blockchain*. Cela coûte de l'ether et nous voulons minimiser la quantité d'Ether à payer.
- Si l'une de nos fonctions contractuelles qui écrit dans la *Blockchain* contient des bugs, le compte qui appelle cette fonction peut potentiellement gaspiller l'Ether et il risque de ne pas se comporter comme prévu.

Pour effectuer des tests il faut d'abord créer un fichier de test dans le répertoire des tests comme suite 'Moncontrat.test.js'

On écrit les tests avec java script pour simuler l'interaction coté client avec notre *Smart contract*, puis on l'exécute pour voir le comportement de notre contrat comme suite :

```
$ truffle test
```

6. Déploiement du *Smart contract* sur le Test Net :

Cette étape consiste à faire une migration de notre *Smart contract* a une *Blockchain* public réel de test (ou les transactions sont validées et les blocs sont minés par des vrais mineurs), afin qu'on puisse tester le comportement de notre *Smart contract* avant de le déployer dans le Main Net.

Ce n'est pas évident de tester les *Smart contracts* dans le Main Net car les transactions et l'écriture dans la *Blockchain* coutent du gas qui coute aussi de l'ether,

contrairement au Test Net qui contient que du faux ether facile à collecter. Actuellement il y'a trois fameux Test Net :

- **Rinkbery et Kovan** : sont deux *Blockchains* public qui utilisent le consensus *Proof of Authorit*y (PoA), créés et maintenus par un consortium de développeurs Ethereum.

Dans les réseaux PoA, les transactions et les blocs sont validés par des comptes approuvés, connus sous le nom de validateurs. En attachant une réputation à une identité, les validateurs sont incités à maintenir le processus de transaction, car ils ne souhaitent pas que leurs identités soient associées à une réputation négative **[PAR] [POA] [KOV]**. La différence entre ces deux Test Net est la taille de chaque *Blokchain* et les clients supportés par chaque une (Kovan supporté par parity, Rinkeby supporté par geth)

- **Ropsten** : c'est une *Blockchain* public de test qui utilise *proof of work* comme consensus. Ropsten est supporté par parity et geth clients **[ETHS]**.

On peut voir tout ce qui ce passe dans un réseau Ethereum à l'aide d'un navigateur *Blockchain* comme Etherscan, qui nous permet d'explorer des réseaux Ethereum (Main Net ou Test Net). Nous avons choisi le Test Net Ropsten pour déployé notre *Smart contract* car il est le plus décentralisé et le plus similaire au Main Net.

Remix : est un IDE (Environnement de développement) qui est accessible par navigateur à l'adresse suivante **[REMIX]**. Nous avons choisi Remix car il est très pratique et très pertinent pour plusieurs raisons :

- On y accède juste par navigateur et il n'y a rien à installer.
- On dispose automatiquement des dernières versions de Solidity.
- Il y a un débogueur intégré : il permet de compiler et d'exécuter les *Smart contracts* instantanément, dans toutes sortes de *Blockchains*, c'est à dire qu'on peut déployer dans la vrai *Blockchain* Ethereum un *Smart contract* directement depuis Remix, mais il peut aussi se connecter à la *Blockchain* de test (qui s'appelle Ropsten), ou locale comme Ganache. Il est donc très souple et flexible **[BLKF]**.

L'interface de l'IDE Remix est constitué de :

- La barre d'outils de gestion des fichiers (pour créer un fichier, en charger un, et même accéder à un dossier local avec vos fichiers solidity).
- Le navigateur de fichiers ou vous retrouverez la liste des fichiers .sol (l'extension officielle pour Solidity).
- Le code.
- La console ou toutes les transactions et actions du contrat sont affichées.
- La barre d'outils pour compiler, débuguer, se connecter à la *Bockchain*, entrer des variables et plein d'autres choses que nous détaillerons au prochain chapitre.

Pour déployé notre *Smart contract* sur le Test Net Ropsten nous allons :

- Créer un compte dans ce réseau à l'aide de Metamask.
- Récolter du *Fake* ether, soit par minage, soit par des sites appeler *faucet* qui nous envoie de l'ether en introduisant notre adresse **[FAU]**.
- Ouvrir Remix.
- Configurer le web3 provider pour Ropsten.
- Se connecter avec notre compte.
- Compiler notre *Smart contract.*
- Emmètre une transaction de création du contrat (Migration).

Après la validation de cette transaction et le minage du bloc qui contient cette dernière le compte de notre *Smart contract* sera créé, et le code sera fonctionnel. Dans cette étape commence nos tests par des enjeux de données et simulation des transactions.

Les avantages d'Ethereum :

- Ethereum est la première *Blockchain* au monde à proposer la création des *Smart contracts* et des applications décentralisées **[AV]**.
- La *Blockchain* Ethereum permet d'écrire du code exécutable en mode *Turing-complète* et offre des possibilités quasi-infinies aux développeurs.
- La notion du gaz qui a éliminé le problème des boucles infinies et autrement à remédier à l'attaque DOS.

- Souplesse de l'algorithme preuve de travail (Ethash) en termes de consommation de ressources et du temps, sans oublier l'algorithme preuve d'enjeux qui va remplacer Ethash dans l'avenir avec plus de robustesse et moins de consommation de ressources.
- La plateforme a très vite rencontré un franc succès auprès des porteurs de projets et bénéficie d'une avance considérable par rapport à ses challengers **[AV].**
- Ethereum a déjà reçu les projets de grandes institutions économiques et financières, comme la Commonwealth Bank of Australia ou le géant Axa **[AV]**.

Les limites d'Ethereum :

- Le réseau Ethereum est en proie à de nombreux problèmes de scalabilité, malgré des efforts évidents comme l'amélioration de l'EVM **[AV]**.
- Le réseau Ethereum fait régulièrement face à des failles de sécurité **[AV].**
- Le langage Solidity est complexe et nécessite un nouvel apprentissage de la part des développeurs.
- Les tests des *Smart contracts* sont fait par du code Java Script.

7. Conclusion :

Dans ce chapitre nous avons présenté une étude sur la *Blockchain* Ethereum et nous avons abordé les éléments et mécanismes de cette plateforme, puis nous avons détaillé les outils de développement que nous allons utiliser dans l'implémentation de notre solution du vote électronique.

Chapitre IV : Conception et implémentation d'un *Smart contract*

1. Introduction :

Dans ce chapitre nous allons concevoir et développer une application décentralisée de vote électronique qui s'appuie sur le système *Blockchain.* Pour assurer l'indépendance de toute entité centrale qui peut influencer sur le résultat du scrutin, éliminer le *sinlge point of faillure*, et assurer la transparence et la sécurité du déroulement de l'opération, et surtout l'immutabilité des résultats.

2. La problématique du vote :

Le vote désigne une méthode permettant à un groupe une prise de décision commune. Les organisations formelles ou informelles ont recours à cette pratique, de toute nature (économiques, politiques, associatives, etc.). La pratique du vote vise à donner une légitimité à la décision en montrant qu'elle ne vient pas d'un individu isolé.

Sachant que beaucoup de ressources humaines, matérielles et financières sont investies dans les élections traditionnelles avec des procédures gênantes et désagréables pour les citoyens en termes de temps et d'effort. En plus ni la traçabilité ni l'intégrité ni la transparence des calculs des voix n'est garanti vu que ce processus impose des autorités intermédiaires qui peuvent falsifier le résultat.

Le vote électronique dans le web traditionnelle facilite les procédures du vote, réduit la consommation des ressources et du temps, mais ni la transparence ni l'intégrité des données est garanti, car tout le système du vote électronique est centralisé dans un serveur contrôlé par une autorité intermédiaire. Les données peuvent être altérées facilement en interne sachant que le code de l'application est une boite noire. Donc le problème de confiance et de la transparence est toujours présent, de plus les applications web du vote ont le problème de saturation des serveurs lors la phase du vote (point unique de défaillance).

3. L'objectif du travail :

Blockchain est un système *peer-to-peer* de transaction de valeurs sans tiers de confiance, il s'agit d'un registre des transactions partagés, décentralisé, ouvert et répliqué sur un grand nombre de nœuds. Ce grand livre est une base de données contenant uniquement des ajouts et ne peut être ni modifiée ni supprimé. Cela signifie que chaque entrée est une entrée permanente. Toute nouvelle entrée est répercutée sur toutes les copies des bases de données hébergées sur des nœuds différents, ces derniers travaillent ensemble pour assurer la sécurité et l'immutabilité de ce grand livre. Ce qui est très utile pour une application de vote électronique parce que la traçabilité des voix pour chaque compte est garantie. Chaque transaction écrite dans la *Blockchain* est immutable et public, impossible de détourner des voix dédiées a un candidat ou saturer notre Dapp (les transactions sont asynchrones), de plus on élimine le point unique de défaillance.

La plateforme Ethereum nous permet d'écrire des codes exécutables en mode *Turing complet* dans une *Blockchain* qui est sécurisée, immutable, public. Le code sera exécuté dans la machine virtuelle de manière décentralisée et se chargera de la lecture, de l'écriture des données, du transfert de la valeur, de la prise des décisions. C'est là qu'on définit la logique de cette application. C'est très important pour une application de vote parce que cela signifie que les règles ne changeront pas toute en gardant la transparence du vote, ce code est appelé un *Smart contract* car il représente une sorte d'alliance ou accord de notre vote.

L'accord de notre *Smart contract* est présenté comme suit :

- Définir la période de chaque phase (d'inscription, du vote, du résultat final)
- Définir les contrôles et les vérifications des données lors de l'inscription.
- Limiter le nombre d'inscription à 1 pour chaque utilisateur.
- Identifier les citoyens et les électeurs.
- Limiter le nombre de vote à 1 pour chaque électeur.
- Calculer les voix en temps réel.
- Délibérer le résultat final.

Notre solution dans ce travail est une application décentralisée en tous sens :

- Un réseau décentralisé présenté en p2p.

- Les données sont décentralisées car elles sont partagées entre tous les nœuds.
- Le code est décentralisé car il est également partagé et exécuté sur tous les nœuds.

Nous allons aussi concevoir une application coté client sous forme d'un site web (html, java script, css) et au lieu d'une connexion a un serveur web, on aura une connexion à une *Blockchain* public. Nous nous sommes inspirées d'une application web dynamique disponible sur **[SH]**.

On a choisi de déployer notre *Smart contract* dans une *Blockchain* public car n'importe quelle nœud peut lire le grand registre et peut participer à la validation des blocs ce qui rend l'élection purement transparente et infalsifiable et incontrôlable par un ou un ensemble défini de nœuds, contrairement à une *Blockchain* de consortium ou privé qui sont contrôlées par un nombre restreint et choisi de nœuds.

La preuve de travail Ethash est le type de minage actuelle de la *Blockchain* publique Ethereum (Main Net et Test Net Ropsten), qui s'appuie sur des calculs de hash successives d'un bloc pour atteindre une valeur de hash qui correspond à la difficulté du bloc, c'est couteux en terme de ressources mais garanti un niveau de sécurité élevée. En bref le travail doit être difficile a réalisé par le mineur mais facile à vérifier par les nœuds. Le temps moyen de la validation d'un bloc est de 12 secondes avec Ethash.

3. Les outils de développement :

La première étape est le développement d'une application décentralisé (*Smart contract et* une application coté client) de vote en local, nous allons utiliser la *Blockchain* personnel ganache, le *Framework* Truffle, le portefeuille Metamask que nous avons défini précédemment. Après cette étape nous allons déployer notre *Smart contract* au réseau *Blockchain* de test Ropsten à travers Remix IDE.

4. Définition du travail :

- **La phase d'inscription:**

La première phase dans notre application est l'inscription des candidats et des électeurs. Une base de données hashé des citoyens est écrite à travers un *mapping* dans la *Blockchain* précédemment pour vérifier l'identité.

- **Les candidats :**

L'inscription des candidats se fait dans une période défini, (une attestation d'acceptation sera délivré pour chaque candidat lors la validation des conditions de candidature à la présidence, le code de l'attestation sera hashé et enregistrer avec les hashs des informations de cet personne dans un *mapping* dans la *Blockchain*) , chaque candidat qui dispose d'une attestation d'inscription à l'élection agréée par l'état, peut s'inscrire avec le code d'attestation et les informations de sa carte d'identité dans l'interface de notre application, notre *Smart contract* vérifie la correspondance des entrées avec les informations de cet personne qui sont déjà enregistrer dans la *Blockchain*, après la vérification des données le *Smart contract* décide s'il sera inscrit ou non.

- **Les électeurs :**

L'inscription des électeurs se fait dans une période défini aussi, chaque électeur peut s'inscrire avec sa carte d'identité dans l'interface de notre application, notre *Smart contract* vérifie la correspondance des entrées à travers le *mapping* des citoyens, après la vérification des données notre *Smart contract* décide s'il sera inscrit ou non.

Après chaque inscription d'un citoyen (électeur ou candidat) son adresse et ses informations sont stocké dans une table clé-valeur (*mapping* adresse→électeur) qui sera écrite dans la *Blockchain* donc les données des électeurs sont infalsifiables aussi. A travers cette table nous pouvons authentifier les électeurs inscrits pour les autorisés à voter et limité le nombre de vote pour chaque électeur a 1.

L'interface du vote n'est pas accessible même pour les électeurs inscrits jusqu'à la période du vote.

- **Le vote :**

La deuxième phase du processus électoral est le vote, cela se fait après la fin de la phase d'inscription et à un temps définit à travers une interface qui contient la liste des candidats et un bouton de validation. Cette interface n'apparait qu'à ceux qui sont déjà inscrit et qui n'ont pas encore voté, des vérifications sont faites par le *Smart contract* lors la tentative d'accès d'un utilisateur à l'interface.

La vérification du *mapping* se fait :

- À chaque accès à l'interface du vote pour ne pas lui afficher la liste des candidats et le bouton de validation s'il a déjà voté ou bien n'est pas inscrit.
- À chaque tentation du vote aussi pour plus de sécurité (en cas où il appelle la fonction vote(id) à travers l'adresse du *Smart contract* sans passer par l'interface).

Si l'électeur choisi un candidat et valide son choix, une transaction d'appel de fonction vote(id) est initialisé (vers l'adresse du *Smart contract*), l'émission de cette transaction nécessite une confirmation à travers le portefeuille Ethereum. Après l'exécution de cette transaction l'électeur est interdit de voter, le nombre de voix du candidat choisi est incrémenter.

- **Le résultat final :**

C'est la dernière phase du processus électoral, après la fin du vote a un temps précis apparait une interface de résultat.

5. Diagrammes de conception :

Dans cette section nous allons définir l'analyse des besoins de notre application à travers les diagrammes de cas d'utilisation et nous allons décrire le comportement de notre application à travers les diagrammes de séquence.

5.1. Diagrammes de cas d'utilisation :

5.1.1. Diagramme de cas d'utilisation générale :

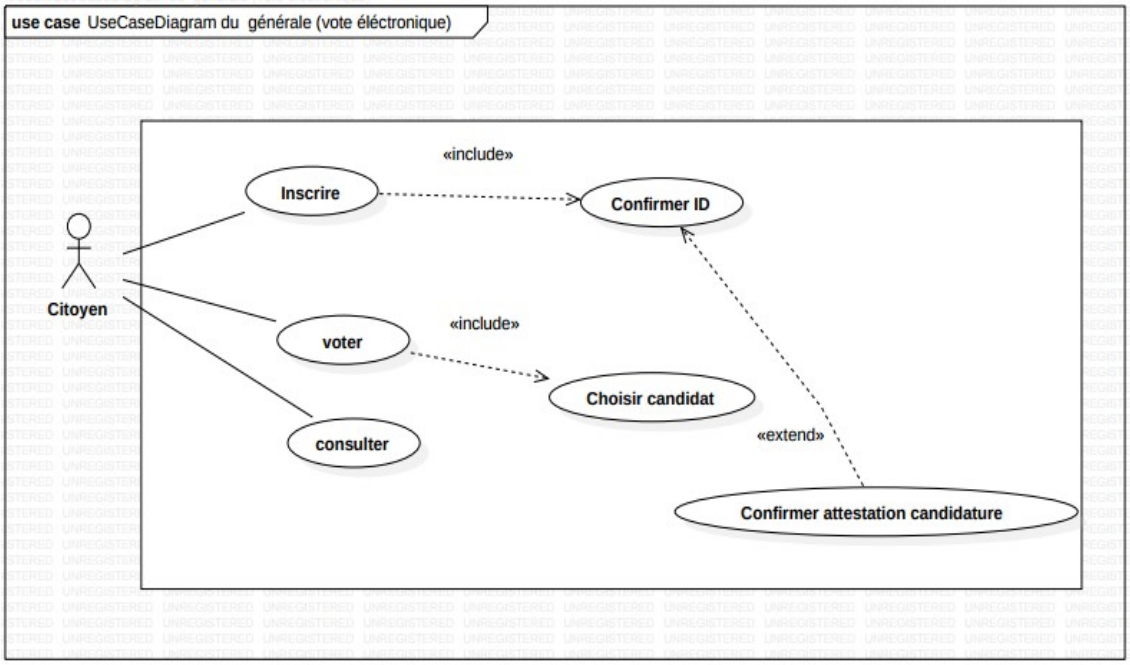

Figure.IV.1: Diagramme de cas d'utilisation générale

Pour qu'un citoyen ou un candidat peut accéder à l'interface de l'application il faut qu'il s'authentifie à son compte Ethereum à travers le réseau *Blockchain* par le portefeuille Metamask.

La première phase est l'inscription à notre application pour qu'on authentifie chaque candidat et électeur, l'inscription se fait dans un temps limité en remplissant un formulaire d'inscription avec les données correctes de la carte identité pour les électeurs, la même chose pour les candidats sauf que il faudra confirmer aussi l'attestation de candidature avec un code secret valide.

Pendant la période du vote, une interface contenant la liste des candidats est chargé dans le navigateur décentralisé de chaque citoyen déjà inscrit, il choisit un candidat puis valide son vote en émettant une transaction d'appel de fonction avec argument à l'adresse du *Smart contract*. L'interface dépend de l'application coté client qui dépend aussi des résultats retourné du *Smart contract*.

5.1.2. Diagramme cas d'utilisation (Inscription des candidats) :

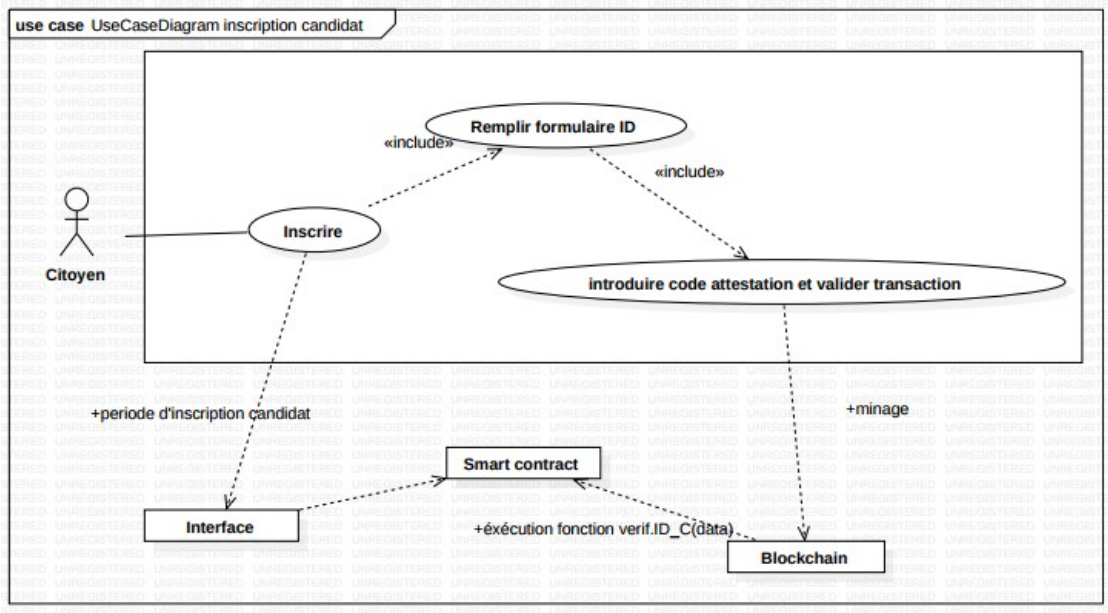

Figure.IV.2: Diagramme de cas d'utilisation (Inscription des candidats)

L'inscription des candidats se fait dans une période limité par l'accès du candidat a notre interface d'inscription en remplissant un formulaire de confirmation d'identité et le code d'attestation. Si tout est correct il sera inscrit et son nom sera afficher dans la liste des candidats lors la période du vote, il peut ultérieurement voter aussi.

5.1.3. Diagramme cas d'utilisation (Inscription des électeurs) :

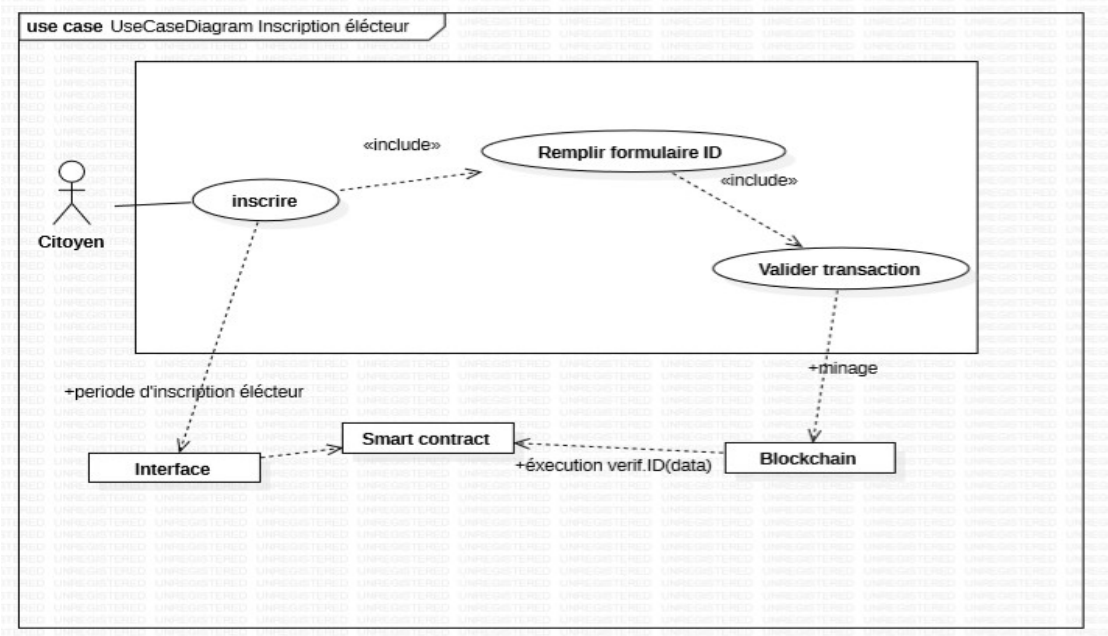

Figure.IV.3: Diagramme de cas d'utilisation (Inscription des électeurs)

L’inscription des électeurs se fait aussi dans une période limitée avant la phase du vote, chaque citoyen peut s’inscrire en accédant à notre interface d’inscription puis remplir le formulaire de confirmation d’identité (numéro carte d’identité et les coordonnées personnels), si tout est correct, il sera inscrit et peut voter ultérieurement lors la période du vote.

5.1.4. Diagramme cas d'utilisation (Vote) :

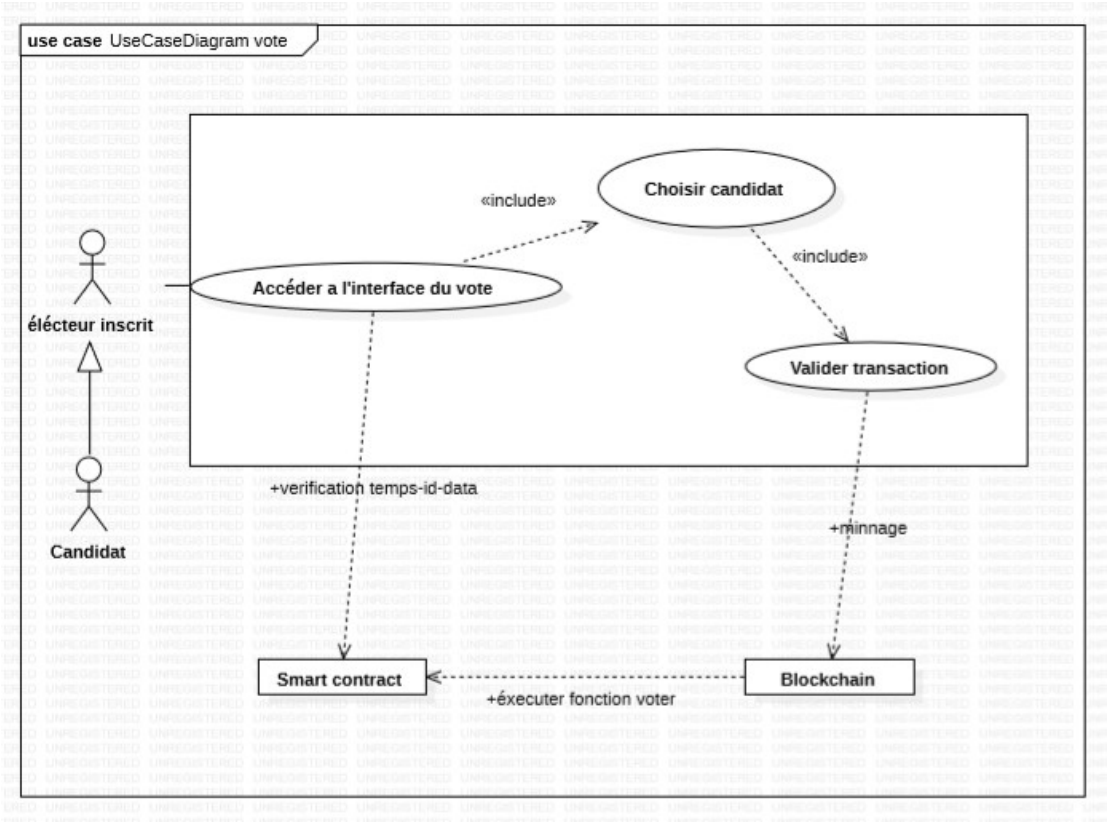

Figure.IV.4: Diagramme de cas d'utilisation (Vote)

La deuxième phase est la phase du vote. Il commence dans un temps défini par notre *Smart contract* après la fin de la phase d’inscription et se termine après une période données.

5.2. Diagrammes de séquence :

5.2.1. Diagramme de séquence (Inscription d'un électeur) :

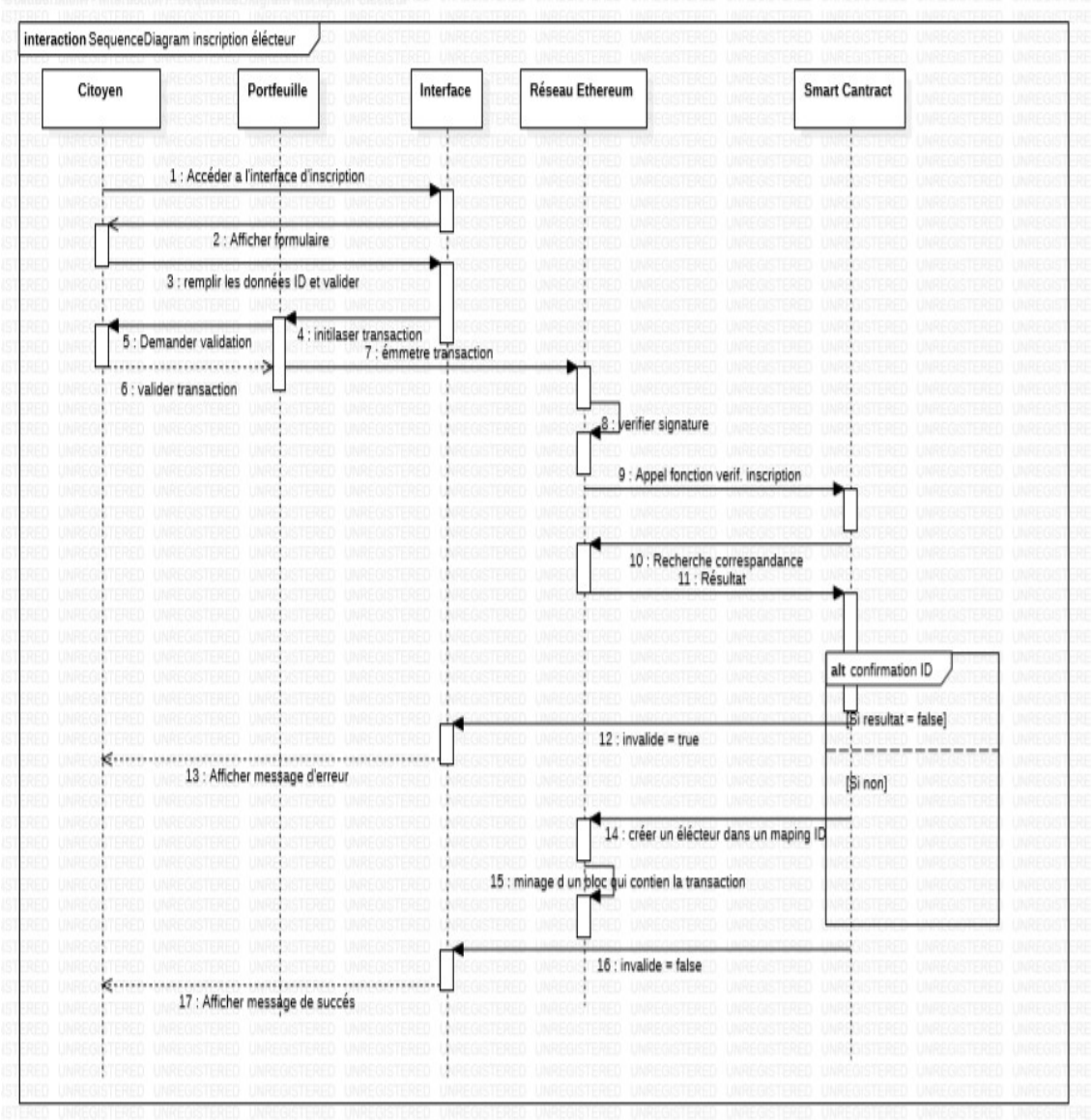

Figure.IV.5: Diagramme de séquence (Inscription d'un électeur)

- Citoyen : c'est un utilisateur qui dispose d'un navigateur et d'un compte Ethereum.
- Portefeuille Ethereum : dans notre cas, c'est Metamask qu'utilise le citoyen pour s'authentifier à son compte et accède au réseau *Blockchain* et peut émettre des transactions.
- Interface utilisateur web3 : c'est l'interface coté client du *Smart contract* accessible à travers le web sémantique.

- Reseaux Ethereum : c'est le système *Blockchain* à travers lequel se fait la validation des transactions, le minage des blocs et l'exécution du code par les nœuds.
- *Smart contract* : c'est un compte qui dispose de la clé public, contient un espace de stockage et un code source qui est écrit dans la *Blockchain* et partagé entre les nœuds.

5.2.2. Diagramme de séquence d'inscription d'un candidat :

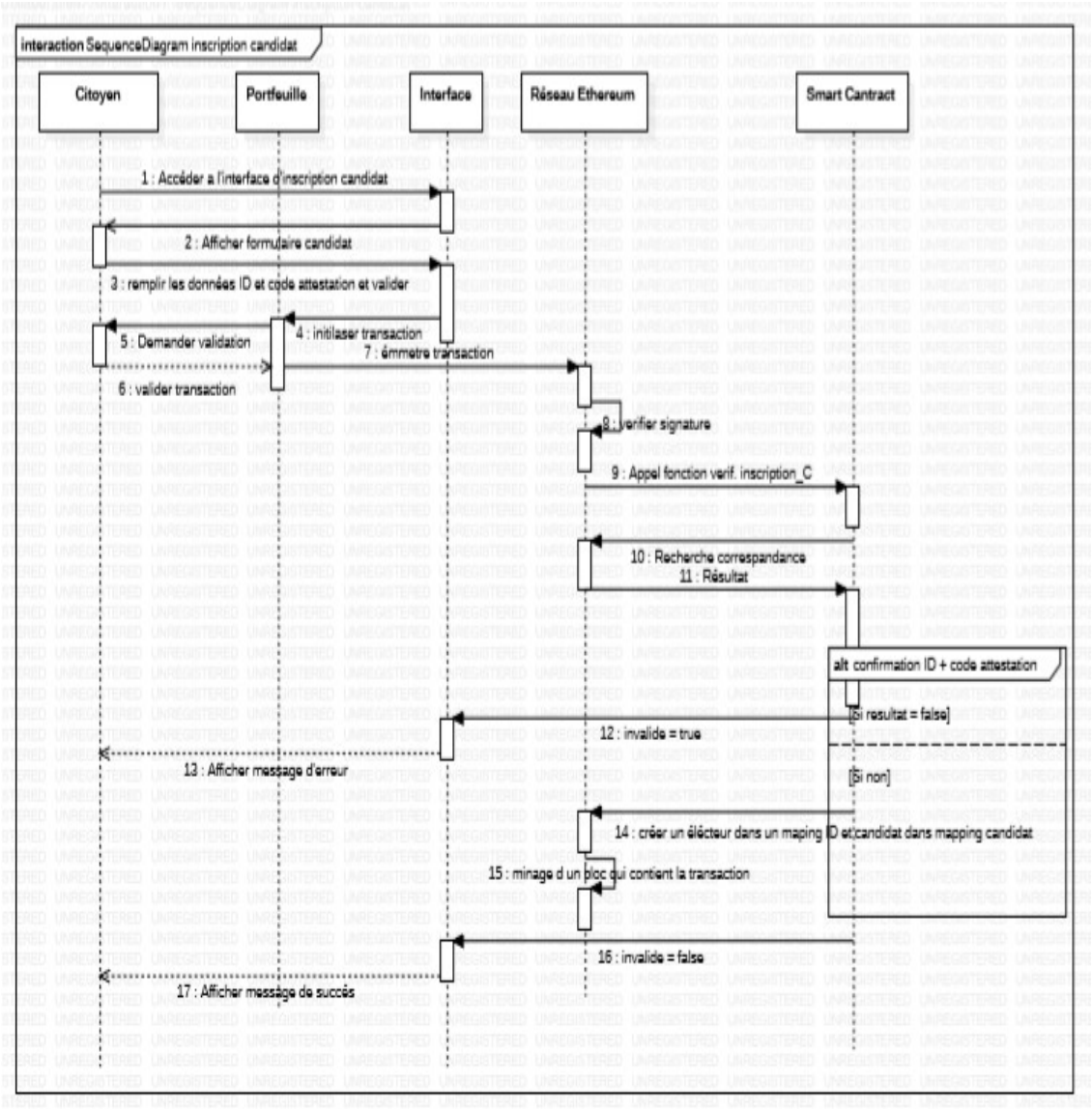

Figure.IV.6: Diagramme de séquence (Inscription d'un candidat)

5.2.3. Diagramme de séquence du vote :

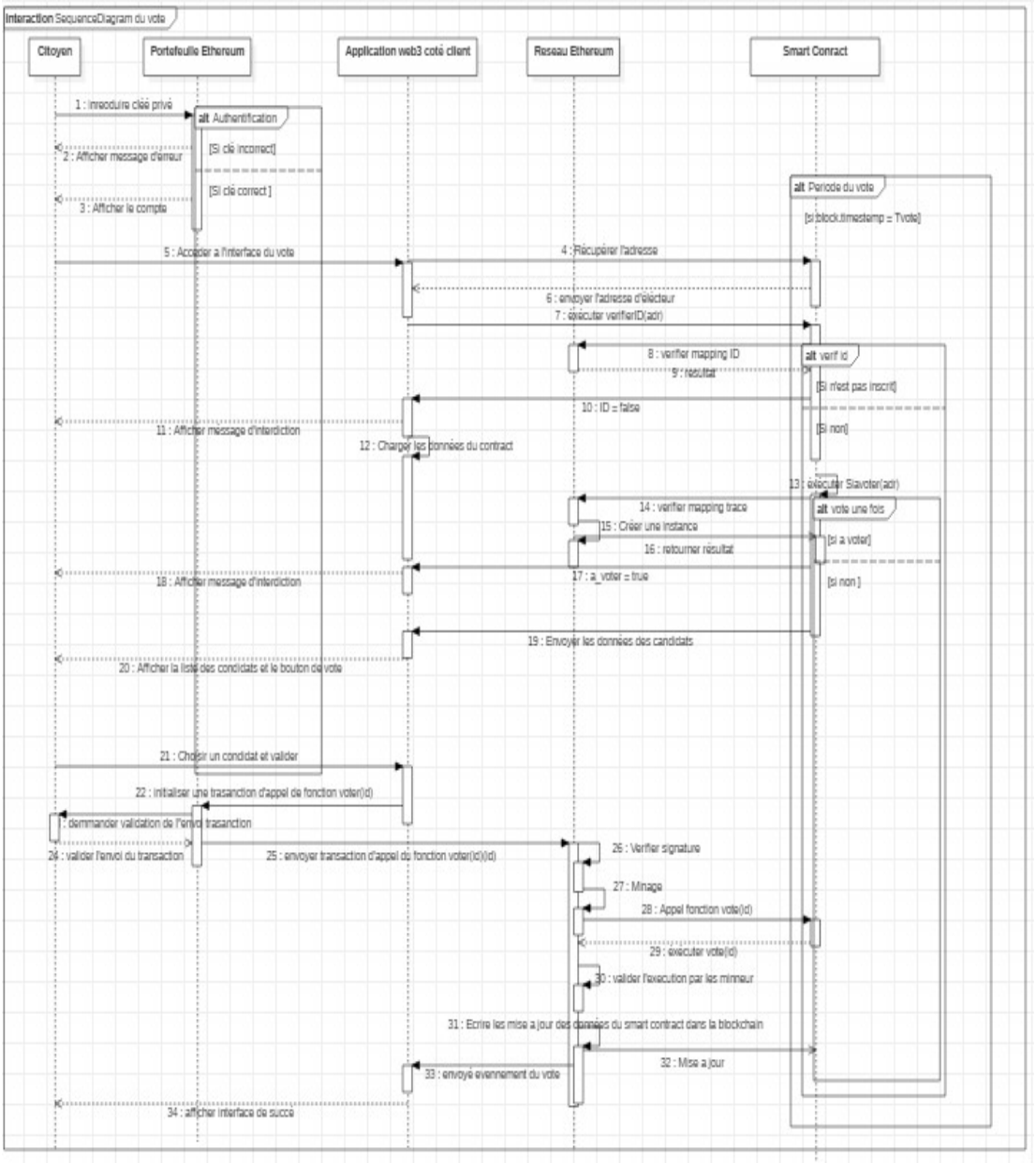

Figure.IV.7: Diagramme de séquence du vote

5.2.4. Diagramme de séquence (calcul des voix) :

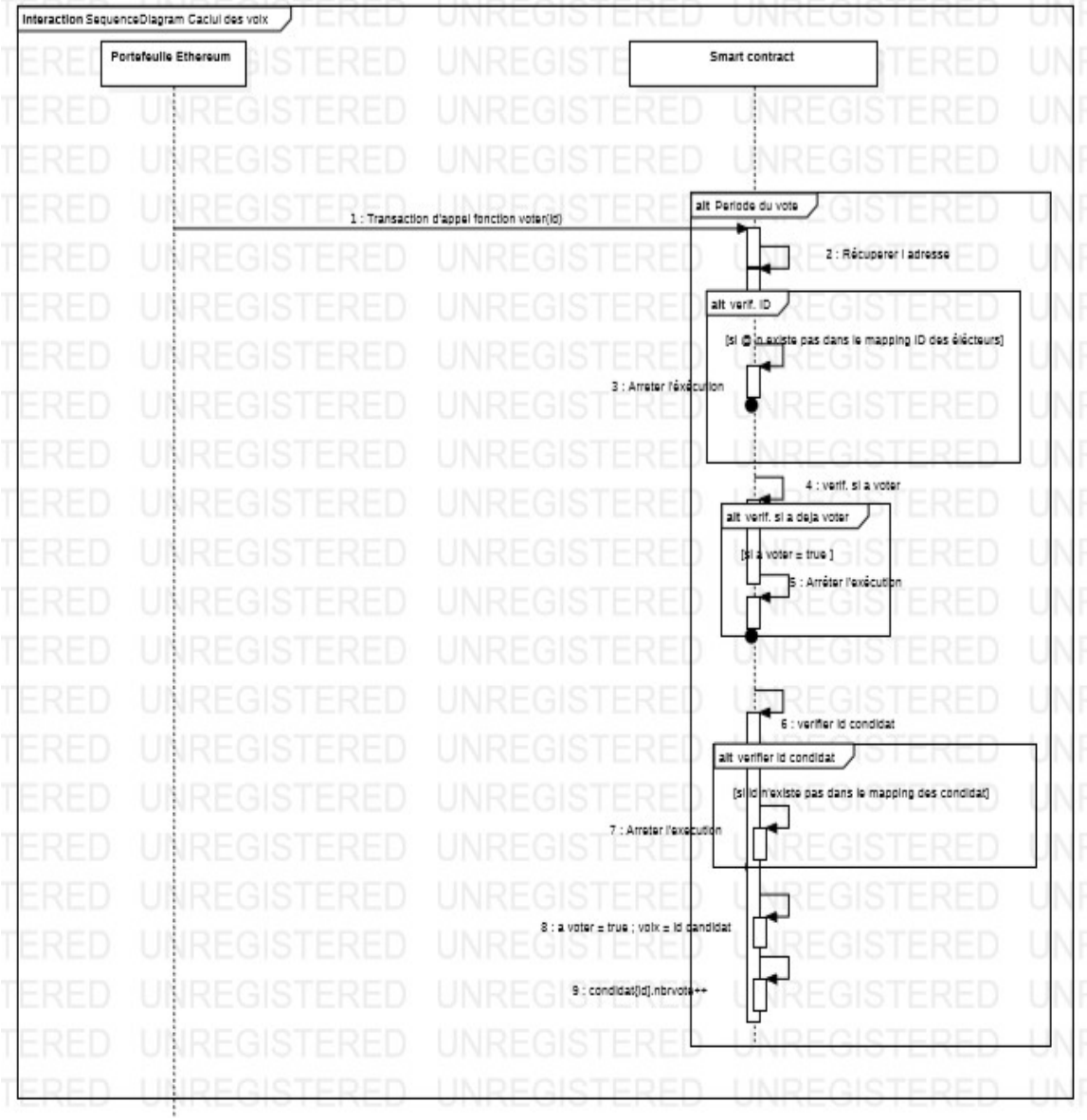

Figure.IV.8: Diagramme de séquence (calcul des voix)

6. La structure des données de notre application :

Les *Smart contracts* n'ont aucun accès aux données externes, nous avons mentionné déjà dans le chapitre 2, le seul moyen est d'utilisé l'entité Oraclize ou bien d'écrire les données dans la *Blockchain*.

Dans notre cas, nous avons besoin des données des citoyens et des candidats pour l'authentification, pour cela nous avons utilisé notre *Smart contract* pour écrire les données de vérification d'identité dans la *Blockchain* sous forme d'un *mapping* (table

de hashage), c'est très important en terme de sécurité, du temps et de complexité de l'algorithme de recherche, car on va accéder à nos données directement par ID. En écrivant le code d'un *Smart contract,* il faut toujours tenir au compte de la notion du gaz car l'utilisateur va payer l'exécution de chaque instruction, donc l'objectif est de trouver la solution la plus simple et la plus sécurisé.

Nous avons utilisé un tableau d'objet (candidats) dans notre *Smart contract* pour calculer le nombre des voix en temps réel.

Voici les *mappings* que nous avons utilisés :

Citoyen (ID → Nom, Prénom, Adresse, Code_attestation), pour la vérification d'identité.

Electeur (Adresse → ID, aVoter, Voix, Authentifier), pour gérer les règles du vote.

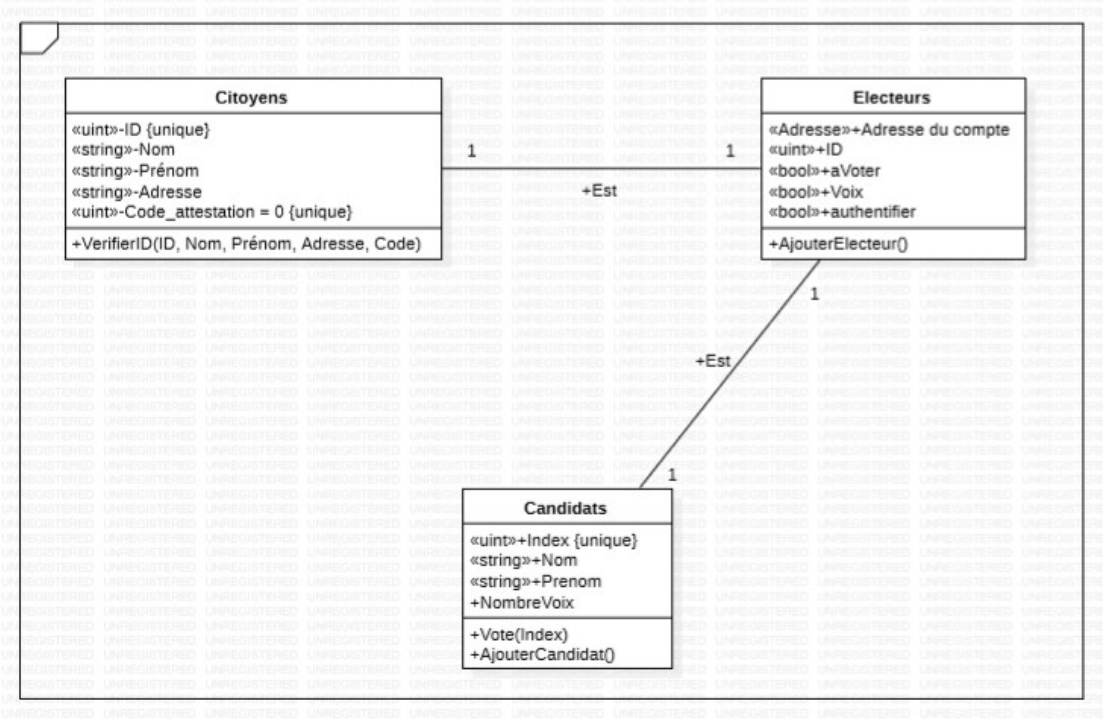

Figure.IV.9: Diagramme de classe

Un électeur est un citoyen authentifié, un candidat est un électeur qui dispose d'un code d'attestation de candidature et inscrit avec.

Pour gérer le processus du vote et le calcul des voix, nous avons utilisé aussi un tableau dynamique pour les candidats authentifié.

7. Les fonctions principales de notre *Smart contract :*

7.1. La fonction d'inscription :

Nous avons défini deux fonctions d'inscriptions, une pour les candidats et l'autre pour les électeurs, la différence est dans les entrées (code attestation en plus pour les candidats), dans l'intervalle du temps et dans la création d'un objet de type candidat. On prend comme exemple l'inscription des candidats :

```
function addCandidate(uint  _id,string memory _name,string memory _first_name,string memory _wilaya,uint  _code)  public {

  require(timeSub1(),"Phase d'inscription terminée");
  require(testStr2(_id,_name,_first_name,_wilaya),"Syntaxe incorrecte");
  require(!voters[msg.sender].ins,"Vous êtes inscrit déja");
  require(!citizens[_id].insc,"Vous êtes inscrit déja avec une autre adresse");
  require(citizens[_id].nameCa == keccak256(abi.encode(_name)),"Votre nom ou votre ID est invalide");
  require(citizens[_id].first_nameCa == keccak256(abi.encode( _first_name)),"Votre prénom est invalide");
  require(citizens[_id].wilayaCa == keccak256(abi.encode(_wilaya)),"Votre adresse est invalide");
  require(citizens[_id].code == keccak256(abi.encode(_code)),"Votre code est invalide");

    citizens[_id].insc= true;
    voters[msg.sender].ins= true;
    candidates.push(Candidate(_name, 0, _first_name));
    emit NewCandidate(_name);

}

function addVoter(uint  _id,string memory _name,string memory _first_name,string  memory _wilaya)  public  {

   require(timeSub2(),"Phase d'inscription terminée");
   require(testStr2(_id,_name,_first_name,_wilaya),"Erreur syntaxe");
   require(!voters[msg.sender].ins,"Vous etes déja inscrit ");
   require(!citizens[_id].insc,"Vous avez deja incrit avec une autre adresse");
   require(citizens[_id].nameCa == keccak256(abi.encode(_name)),"Votre nom ou votre ID est invalide");
   require(citizens[_id].first_nameCa == keccak256(abi.encode( _first_name)),"Votre prénom est invalide");
   require(citizens[_id].wilayaCa == keccak256(abi.encode(_wilaya)),"Votre code est invalide");
   require(!voters[msg.sender].voted);

    citizens[_id].insc= true;
    voters[msg.sender].ins= true;
    emit Newvoter(_name);

}
```

Figure.IV.10: fonctions d'inscription

La fonction require(x) : est l'équivalent de if (!x) { throw ; }, si l'argument x est faux l'exécution est arrêter immédiatement et un message defini s'affiche dans la console, sinon l'exécution continue. Nous avons utilisé require () pour :

- Contrôler le temps : chaque phase a un intervalle du temps défini, nous avons défini une fonction time1() que nous allons discuter prochainement, elle retourne vrai si le temps actuel appartient à la phase d'inscription.
- Vérifier la validité des entrées syntaxiquement avec la fonction teststr1()

- Vérifier l'authenticité de l'appelant de cette fonction, s'il est inscrit déjà avec ce compte ou avec un autre compte, donc impossible qu'un candidat soit inscrit deux fois.
- Comparaison des hashs des entrées avec les données du mapping de vérification.
- Si toutes les conditions sont vraies le candidat va être ajouté.

L'ordre des instructions est très important dans notre solution car on ne veut pas faire perdre les citoyens du gaz sans accomplir l'inscription. Nous avons utilisé l'algorithme keccak256 pour la comparaison des valeurs car une simple comparaison entre deux chaines de caractères coute du gaz beaucoup plus que la comparaison des *hashes*.

7.2. Fonction du vote :

La fonction du vote fonctionne comme suite :

- Interrompre l'exécution si ce n'est pas la période du vote.
- Interrompre l'exécution si l'index du candidat est négatif.
- Interrompre l'exécution si l'utilisateur n'est pas authentifier.
- Interrompre l'exécution si le citoyen a déjà voté.
- Interrompre l'exécution si l'index n'est valide.
- Mise à jour du *mapping* des électeurs.
- Incrémenter le nombre de voix pour ce candidat et le nombre total des voix
- Emission d'un évènement qui contient les nouvelles valeurs des voix pour qu'on puisse interagir en temps réel avec l'application coté client.

```
function vote(uint _candidate) public {

    require(timeVote(),"Phase du vote terminée");
    require(_candidate >= 0);
    require(voters[msg.sender].ins, "Vous n'etes pas autorisé a voter");
    require(!voters[msg.sender].voted, "Vous avez déja voter");
    require(_candidate < candidates.length, "Candidat invalide");

    voters[msg.sender].vote = _candidate;
    voters[msg.sender].voted = true;
    candidates[_candidate].voteCount += 1;
    totalVotes += 1;
    emit Vote(candidates[_candidate].name, candidates[_candidate].voteCount);
}
```

Figure.IV.11: fonction du vote

7.3. Les fonctions du temps :

Dans le langage Solidity il n'y a pas d'objet pour le temps ou la date, il y'a juste le mot clé now qui veut dire le *Block.timestemp,* il va récupérer le temps Unix du dernier bloc miné en milliseconde. Nous avons défini trois variables d'état qui vont être initialisé dans le constructeur par des valeurs défini. Nous avons déclaré ces fonctions comme *View* car ces dernières ne vont pas modifier l'état de la *Blockchain.*

8. Déroulement de l'application :

La première chose est la migration de notre *Smart contract* vers la *Blockchain* local. Voici le résultat :

```
Deploying 'Election'
--------------------
> transaction hash:    0x6666abf4dffb6610a61edb6c50ae9fe4bd2c74ce6baf1909d0e947741911de10
> Blocks: 0            Seconds: 0
> contract address:    0x6C81fE35d9561B69834dfA984e9DEf7097ba8e42
> block number:        46
> block timestamp:     1562614842
> account:             0x4c51BaA1CCd0AEf73B381A6CbEF863E72b1245FC
> balance:             99.6088502
> gas used:            4526854
> gas price:           20 gwei
> value sent:          0 ETH
> total cost:          0.09053708 ETH


> Saving migration to chain.
> Saving artifacts
-------------------------------------
> Total cost:          0.09053708 ETH
```

Figure.IV.12: Migration

Comme montre le résultat de migration, le cout total de cette transaction est de 0.09053708 ether, ce qui est l'équivalent de 28.23 $ et 3374,33 DA. Cette conversion est faite le 09/07/2019. Nous avons défini le prix de gas à 20 Gwei qui est égale à 0.000000002 ether, car en ce moment la moyenne du prix du gas dans le MainNet pour une transaction qui ne va pas dépasser une minute pour être minée est de 20 Gwei. Comme on a dit précédemment le marché du gas est variable selon les mineurs et la demande.

Après la migration de notre *Smart contract,* nous allons lancer un serveur virtuel en local à l'aide du *framework* Truffle qui contient l'application coté client que nous avons développé avec du javascript et du css relier avec notre *Smart contract* à travers

la bibliothèque web3js. On doit se connecter à notre portefeuille Metamask pour qu'on puisse connecter à notre réseau *Blockchain.* Voici le premier aperçu de notre application :

Figure.IV.13: Interface (Inscription des candidats)

On remarque qu'il 'y a des messages pour informer l'utilisateur de son état vers notre application et la phase actuelle. Ceci est fait en temps réel par la lecture des données de notre *Smart contract.* Si l'utilisateur est un candidat il peut s'inscrire directement, sinon il doit accéder au formulaire des électeurs comme suite :

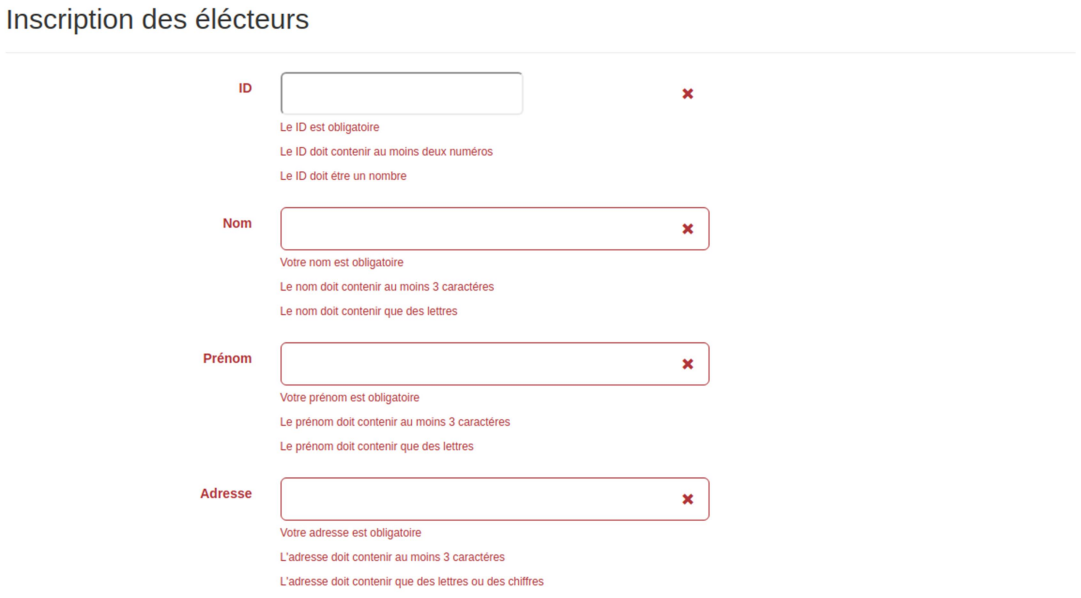

Figure.IV.14: Interface (Inscription des électeurs)

Lorsqu'un candidat valide son inscription en remplissant le formulaire et en cliquant sur le bouton d'inscription, une transaction est initialisée vers la fonction addCandidat de notre *Smart contract* avec ces informations (Voir la figure.IV.15)

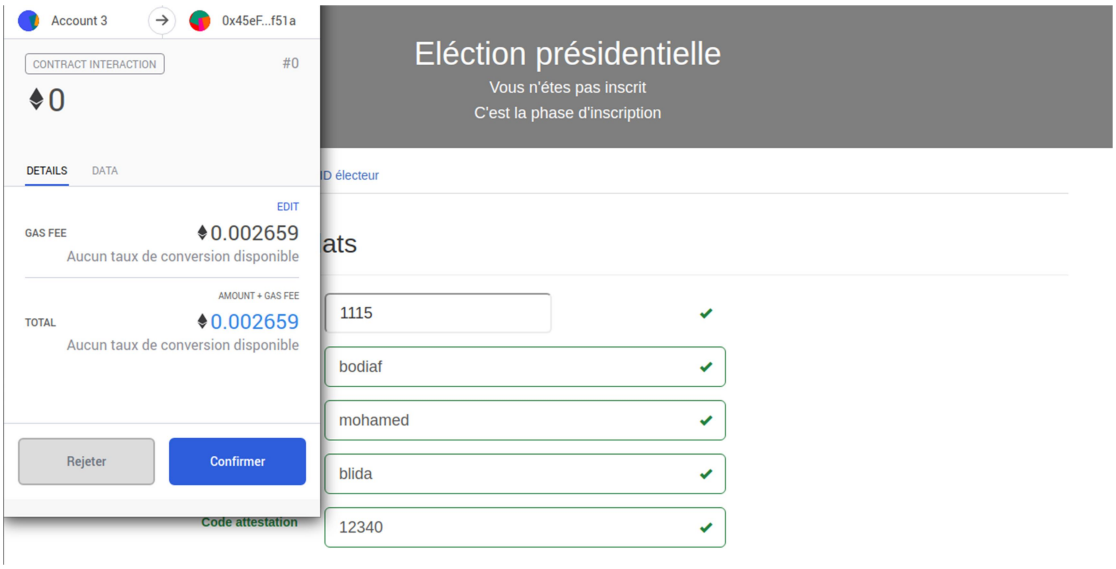

Figure.IV.15 : Interface (validation de la transaction d'inscription)

Comme le montre notre portefeuille Metamask le cout nécessaire pour l'inscription d'un candidat est de 0.002659 ether avec 20Gwei comme prix d'une unité de gas ce qui est la moyenne de prix de gas dans le MinNet.

0.002659 ether = 0.83 $ = 99.21 DA le 09/07/2019.

La période d'inscription des candidats dans notre application sera terminée avant la fin d'inscription des électeurs vu que les électeurs sont beaucoup plus que les candidats. Voici à quoi ressemble notre interface lors de cette période (formulaire d'inscription des candidats inaccessible) :

Figure.IV.16: Interface (Inscription des électeurs 2)

Les formulaires d'inscriptions sont accessibles (visibles) seulement lors de la phase d'inscription et si le citoyen n'es pas encore inscrit.

Si un citoyen est inscrit et ce n'est pas encore la phase du vote, un message sera affiché pour l'informer et les boutons du vote seront cachés (Voir figure.IV.17).

Figure.IV.17: Interface (Après l'inscription)

Sinon si c'est la période du vote, voici à quoi ressemble l'interface :

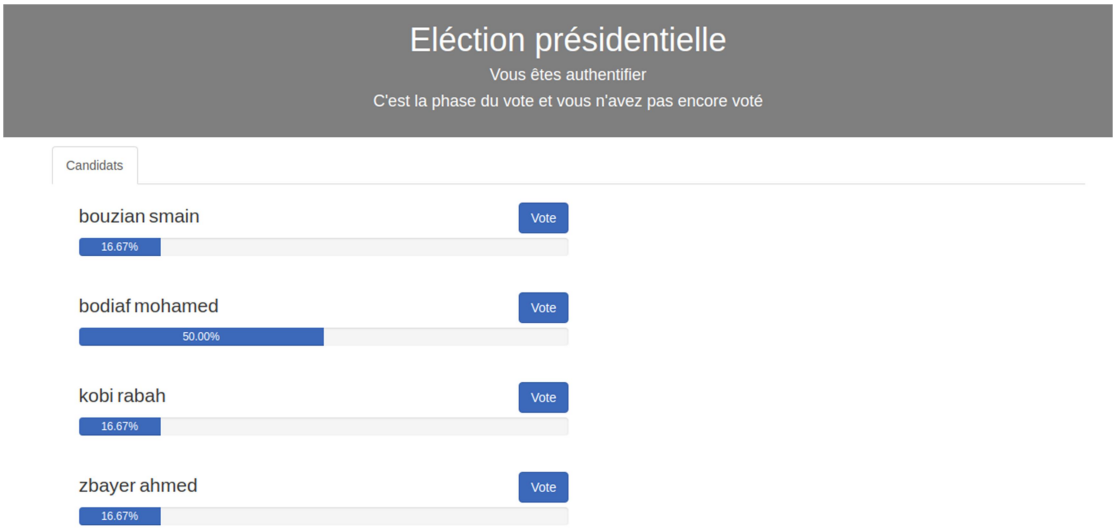

Figure.IV.18: Interface (vote)

Lorsqu'un électeur choisi son candidat en cliquant sur le bouton du vote, une transaction est initialisé vers la fonction vote de notre *Smart contract.*

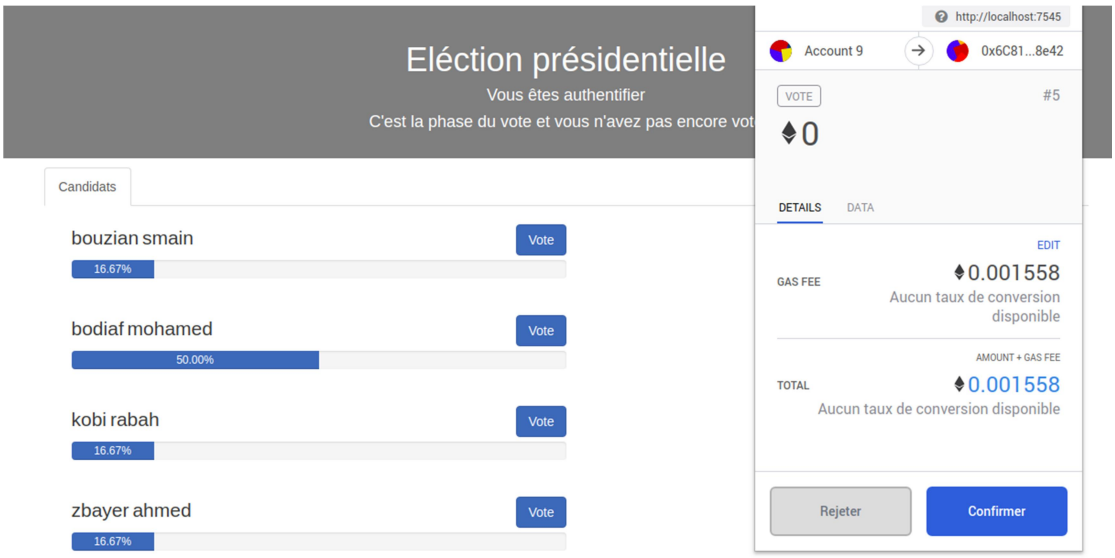

Figure.IV.19 : Interface (validation de la transaction du vote)

La transaction du vote de notre solution coute 0.001558 ether ce qui est l'équivalent de 0.49 $ et de 58.57 DA le 09/07/2019.

Après que l'électeur a voté les boutons sont cachés et sa voix est affichée, il peut voir le pourcentage des voix pour chaque candidat en temps réel, voici à quoi ressemble l'interface :

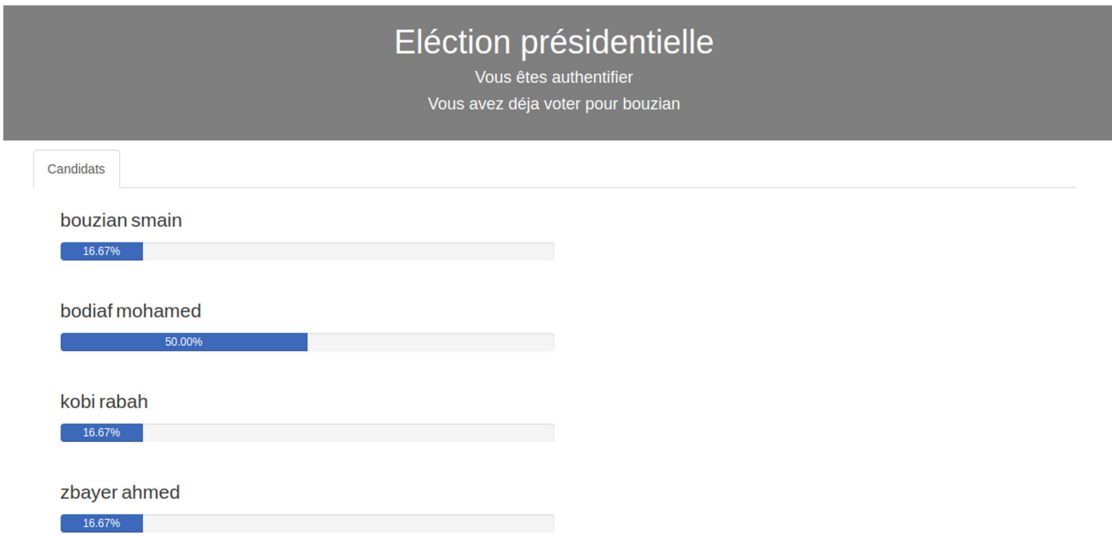

Figure.IV.20 : Interface (Après le vote)

Si la période du vote est écoulée, l'interface du résultat apparait pour tout le monde comme suite :

Figure.IV.21 : Interface (Résultat)

Chaque inscription ou vote est sous forme de transaction, chaque transaction est écrite dans la *Blockchain* de façon permanente et immutable, voici à quoi ressemble les transactions du déroulement précédent :

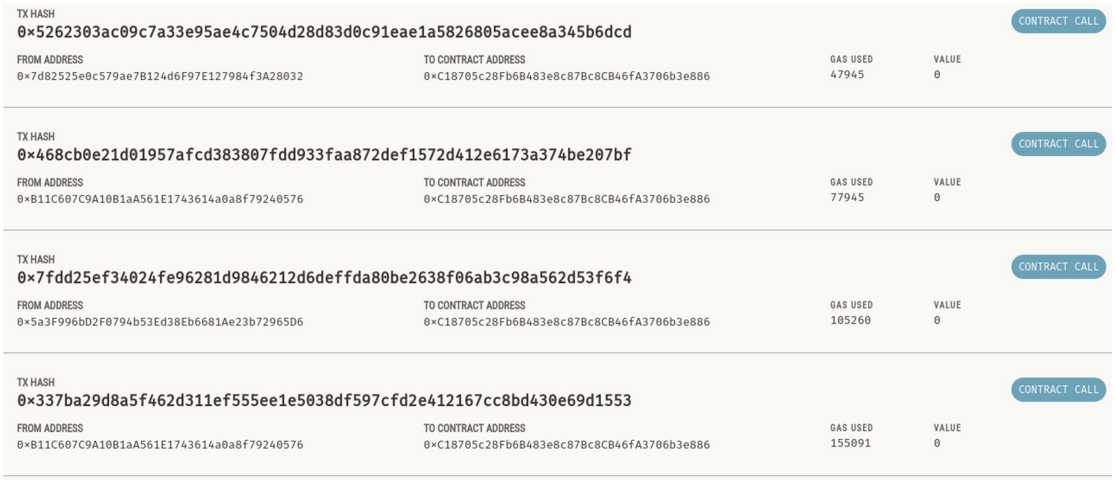

Figure.IV.22 : Transactions (Ganache)

9. Tests et résultats :

Nous avons testé notre *Smart contract* avec deux méthodes, la première avec le *Framework* Truffle en local par des jeux de données, la deuxième méthode avec l'IDE Remix en déployant note *Smart contract* a la fameuse *Blockchain* de test Ropsten.

- **Jeux de données en local :**

Les résultats de ces tests sont des interruptions d'exécution des transactions déclenchées par nos conditions dans le *Smart contract*, chaque transaction est minée malgré que l'exécution a échoué, de cela la traçabilité et la vérification d'identité sont garanti.

Si on introduit un ID ou un nom qui n'existe pas dans le *mapping* de vérification des données, voici le résultat :

```
_sendRawTransaction"} Error: VM Exception while
processing transaction: revert Votre nom ou votre
IDest invalide
  {code: -32603, message: "Error: Error: [ethjs-r
▶ pc] rpc error with payload {…saction: revert Vo
  tre nom ou votre IDest invalide"}
```

Si on introduit de vrais ou de fausses informations d'inscription lors d'une période hors la période d'inscription voici le résultat :

```
],"method":"eth_sendRawTransaction"} Error: VM
Exception while processing transaction: revert
Phase d'inscription terminé
  {code: -32603, message: "Error: Error: [ethjs-r
▶ pc] rpc error with payload {…g transaction: rev
  ert Phase d'inscription terminé"}
```

Si on introduit toutes les informations correctes d'un candidat sauf le code d'attestation qui est incorrecte voici le résultat :

```
],"method":"eth_sendRawTransaction"} Error: VM
Exception while processing transaction: revert
Votre code est invalide
  {code: -32603, message: "Error: Error: [ethjs-r
▶ pc] rpc error with payload {…ssing transaction:
  revert Votre code est invalide"}
```

Dans le cas où toutes les informations sont valides sauf l'adresse, voici le résultat :

```
],"method":"eth_sendRawTransaction"} Error: VM
Exception while processing transaction: revert
Votre adresse est invalide ▶Object
```

- **Déploiement et test du *Smart contract* sur le Test Net :**

Nous avons déployé notre *Smart contract* à la *Blockchain* de test Ropsten a travers l'IDE Remix, voici le cout du gas nécessaire pour la migration :

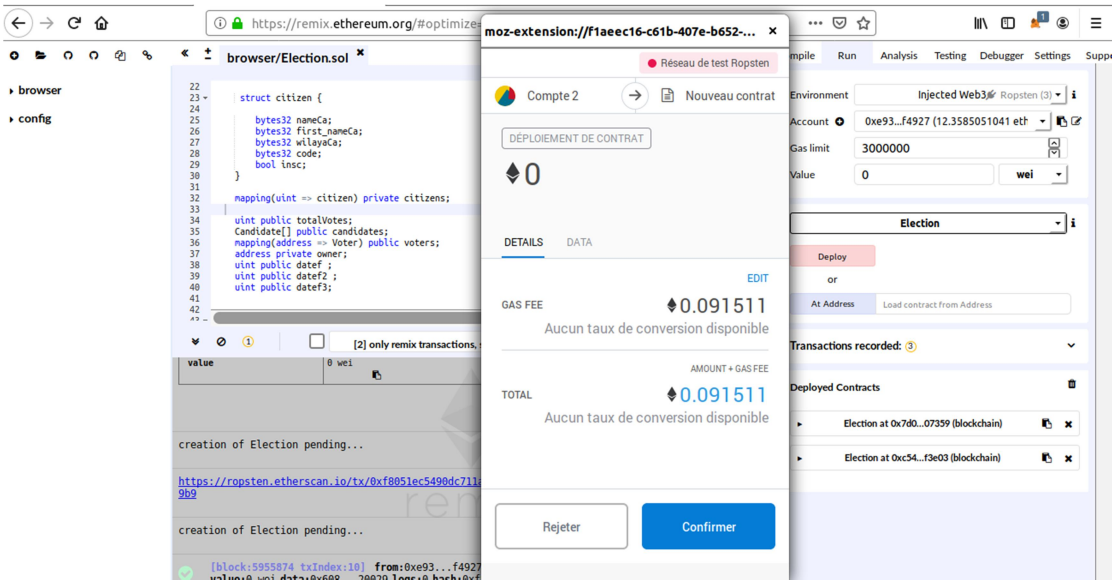

Figure.IV.23 : Remix

A travers Etherscan (Moteur de recherche *Blockchain*) on peut accéder aux compte du *Smart contract*, tout est public (le solde du compte, les transactions émises et reçu, les évènements, le byte code (code source compilé), on peut aussi le décompiler (Voir figure.IV.24).

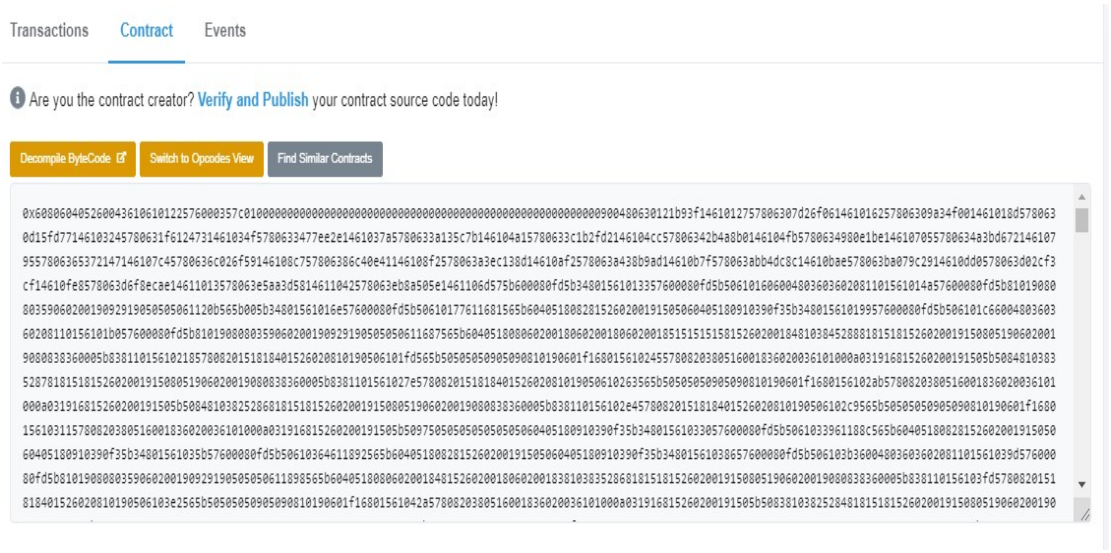

Figure.IV.24 : *bytecode* de notre *Smart contract*

Dans le compte du *Smart contract* le public trouve tous les évènements et les transactions (échoués et validés) en relation avec, tout est écrit de manière permanente et transparente depuis sa création. (Voir figure.IV.25)

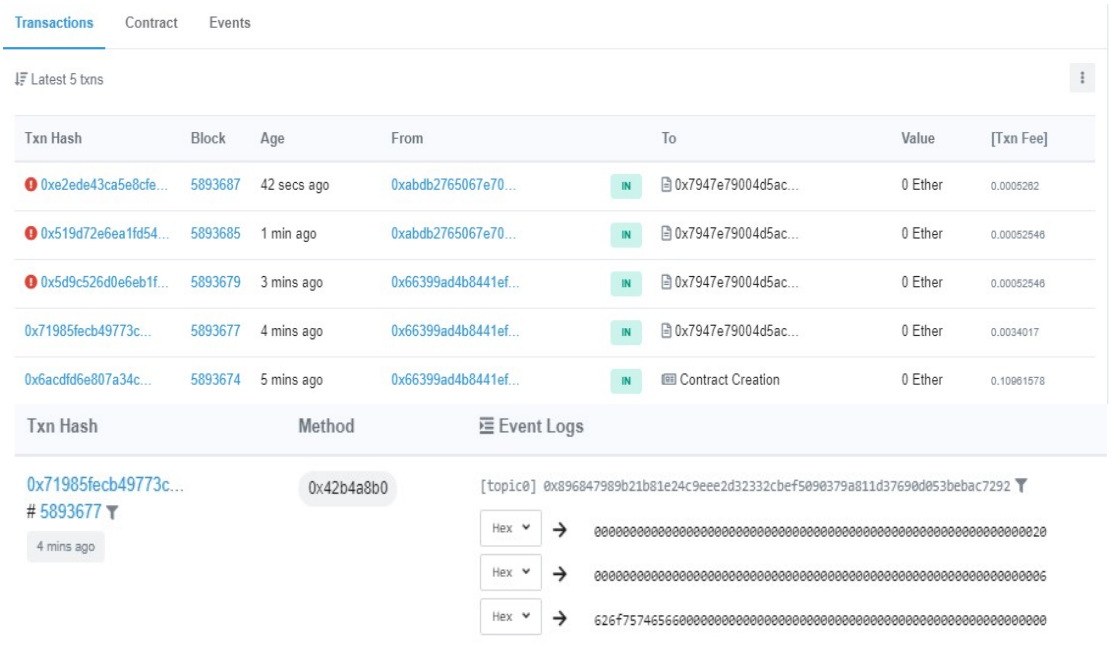

Figure.IV.25 : Le compte de notre *Smart contract*

Voici tous les fonctions principales publiques de notre *Smart contract* représenté par Remix après le déploiement de notre *Smart contract* :

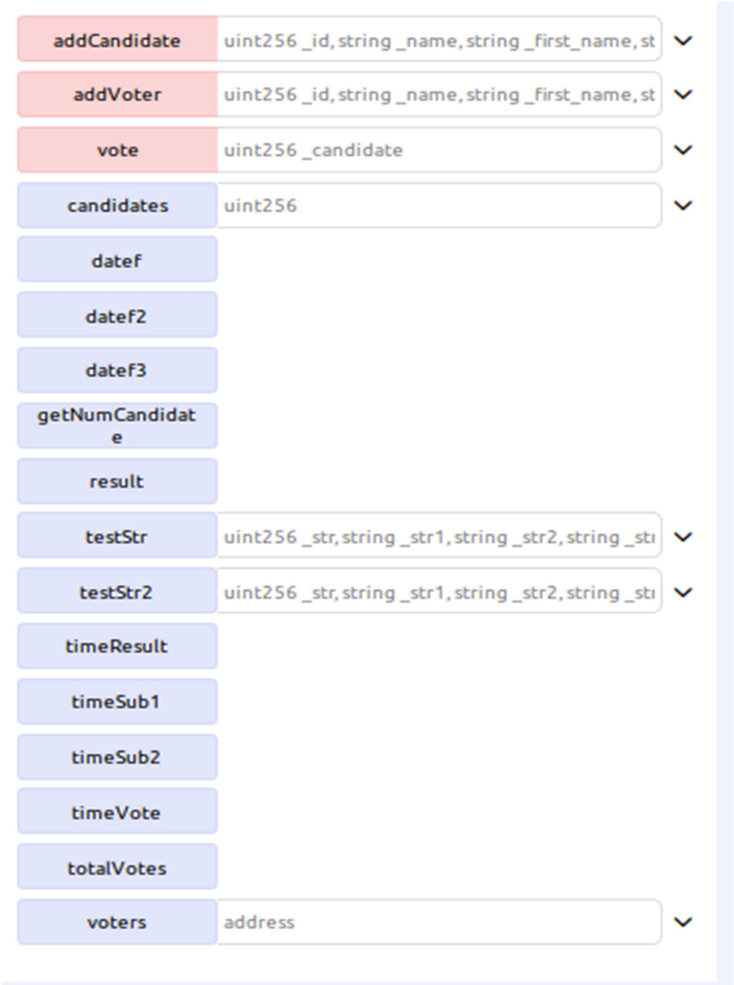

Figure.IV.26 : Les fonctions publiques de notre *Smart contract*

Les fonctions qui sont rouge nécessitent une transaction, les autres nécessitent juste un appel correct, toutes les fonctions sont accessible par le grand public.

Nous avons créé plusieurs comptes sur Ropsten pour tester le comportement de notre *Smart contract* depuis ces entrées, le résultat est :

- Les phases du vote sont infranchissables
- Chaque citoyen peut s'inscrit une fois seulement
- Chaque électeur a le droit de voter une seule fois
- Le calcul des voix correct
- Pas de défaillance ou anomalies dans le comportement du *Smart contract*

10. Comparaison des solutions du vote :

Nous avons trouvé d'autres solutions décentralisées pour le vote électronique sur Github. Voici le tableau de comparaison :

Solution / Fonctionnalité	Notre solution	La solution du Dappuniversity **[GDUV]**	La solution du Rajat Biswas **[GRBV]**	La solution de Samir Hussin **[SH]**
Interface d'interaction	Oui	Oui	Non	Oui
Inscription des citoyens	Oui	Non	Non	Non
Identification des citoyens	Oui	Non	Non	Oui (par adresse seulement)
Limiter le nombre de vote à 1	Oui	Oui	Oui	Oui
Interface de résultat dynamique	Oui	Non	Non	Non
Diviser le processus du vote en phases	Oui	Non	Non	Non
Gestion du temps	Oui	Non	Non	Non
Un candidat est un électeur et peut voter	Oui	Non	Non	Non

Tableau.IV.1 : Comparaison des solutions du vote existantes

11. Conclusion :

Dans ce chapitre nous avons abordé la problématique des élections et analysé les besoins d'une application décentralisé du vote électronique, nous avons proposé notre solution pour avoir un maximum de transparence, d'intégrité, d'immutabilité et de traçabilité des données sans autorité intermédiaire. Nous avons implémenté notre solution qui est un *Smart contract* et une application coté client en local puis nous avons déployé notre *Smart contract* dans un réseau de test pour mieux tester son comportement.

Suite aux tests effectués, nous avons prouvé que notre *Smart contract* est capable de gérer le processus du vote en toute transparence, intégrité, traçabilité et immutabilité du code et des données avec un cout abordable.

Conclusion générale

La *Blockchain* a prouvé son efficacité dans le domaine de la sécurité et la décentralisation dans diffèrent secteurs d'application dans le monde. Elle a apporté beaucoup de nouveaux concepts et d'idées dans le domaine de la recherches, proposant ainsi une nouvelle façon de concevoir les choses sans autorité intermédiaire, en s'appuyant sur les transactions et la cryptographie pour garder le système cohérant et sécurisé par l'ensemble des nœuds du réseaux qui disposent d'une copie de la *Blockchain* et communiquent entre eux.

Notre travail consiste à étudier la technologie *Blockchain* et les *Smart contract* ainsi que la plateforme Ethereum, et de concevoir et implémenter une solution du vote électronique à travers cette technologie pour garantir la transparence, l'intégrité, la traçabilité, l'immutabilité des données, des résultats et du code de notre application.

Nous avons développé et déployer un *Smart contract* dans la plateforme Ethereum qui assure l'exécution des règles du vote en toute sécurité, nous avons aussi développé une interface graphique pour l'interaction des citoyens avec notre *Smart contract*.

Nous avons effectué des tests en local à l'aide du *Framework* Truffle et aussi après le déploiement de notre *Smart contract* dans une *Blockchain* réel de test. Les règles du vote sont infranchissables et publiques, chaque citoyen inscrit vote une fois seulement, les calculs des voix sont effectués en temps réel, la traçabilité est garanti par le grand livre des transactions *Blockchain*, le cout des transactions est raisonnable.

Perspectives :

A titre d'amélioration de notre application décentralisée de vote, trois points ont été marqués :

- La création d'un autre *Smart contract* par le premier pour une phase finale de vote entre les deux candidats vainqueurs.
- L'affichage des statistiques des voix groupées par wilaya pour chaque candidat lors du résultat.
- L'utilisation de l'entité Oraclize pour fournir des données réelles des citoyens à notre *Smart contract*.

Bibliographie

- **[BegB]**: B. Singhal, G. Dhameja, P. Panda. *Beginning Blockchain*, 2018.
- **[AGTUB]**: S. Bennett. *Blockchain: A Guide to Understanding Blockchain*. 2018.
- **[LBPN]** : T. Laurence. *La Blockchain pour les nulls,* 2018
- **[EYWB]**: D. Puthal, N. Malik, P. Saraju Mohanty, E. Kougianos, and G. Gautam Das. *Everything You Wanted to Know About the Blockchain: Its Promise, Components, Processes, and Problems,* 2018.
- **[BCO]**: Z. Zheng et al. *Blockchain challenges and opportunities: a survey. International Journal of Web and Grid Services, Vol. 14, No. 4,.352-372,* 2018.
- **[LE}** : D. Bounie, et S. Soriano. *La monnaie Electronique. Les Cahiers du numérique*, Vol. 4, 71- 92, 2003.
- **[MFE]** : F. Godebarge, R. Rossat. *Mémoire fin d'étude,EM Lyon Business School*, 2016.
- **[BUCTF]**: K. Zīle, et R. Strazdiņa. *Blockchain Use Cases and Their Feasibility. Applied Computer Systems*, vol. 23, no. 1, 12–20, 2018.
- **[MiotD]** : S. Huh, S. Cho, S. Kim. *Managing iot devices using Blockchain platform.ICACT.464-467,* 2017.
- **[TNGB]**: J. Ackermann, M. Meier. *Blockchain 3.0 - The next generation of blockchain system, 2018.*
- **[RMAB]** : C. McFarland, T. Hux, E. Wuehler, S. Campbell. *Rapport sur les menaces associées aux blockchains.McAfee.* 2018.
- **[EF]** : J. Techey. *Nicehash : Attaquer une Blockchain, 2018.* Disponible : https://www.ethereum-france.com/nicehash-attaquer-une-blockchain/
- **[IBMB]**: I. Bashir. *Mastering Blockchain(Packet publishing).Birmingham B3 2PB, UK,* 2017.
- **[DR]** : Dictionnaire la rousse.
- **[ASCB]** : J. Paris. *Apports des Smart Contracts aux Blockchains et comment créer une nouvelle crypto-monnaie*, Haute École de Gestion de Genève (HEG-GE), 2017.

- **[DCUBTS]**: S. Asharaf, S. Adarsh. *Decentralized Computing Using BlockchainTechnologies and Smart Contracts*(IGI Global), United states of America, 2017
- **[BBSCSM]**: M. Alharby & A. Moorsel. *Blockchain Based Smart Contracts A Systematic Mapp. Computer Science & Information Technology* (CS & IT).125-140, 2017.
- **[ISRBS]**: M. A. Khan & K. Salah. *IoT security : Review , blockchain solutions , and open challenges* , *Future Generation Computer Systems* 82.395–411, 2018.

Webographie

- **[BF]** https://blockchainfrance.net/le-lexique-de-la-blockchain/ consulter le 14/02/2019
- **[Bitc]** https://bitcoin.fr/minage/ consulter le 14/02/2019
- **[Oracle]** http://www.oraclize.it/ et https://www.realitykeys.com
- **[SM]** https://smartcontract.com consulter le 02/03/2019
- **[IANG]** http://iang.org/papers/ricardian_contract.html consulter le 02/03/2019
- **[DAPP]** Gregory. *How to build Blockchain App-Ethereum, 2019*. Disponible ici : http://www.dappuniversity.com/articles/blockchain-app-tutorial consulter le 04/03/2019
- **[GIT]** https://github.com/ethereum/web3.js/issues/1783 consulter le 09/04/2019
- **[INF]** https://journalducoin.com/altcoins/ethereum-infura-dependance-afri-schoedon/ consulter le 17/05/2019
- **[SOL]** https://solidity-fr.readthedocs.io/fr/latest/introduction-to-smart-contracts.html consulter le 11/04/2019
- **[AV]** https://journalducoin.com/guides/debuter-cryptomonnaies/avis-ethereum/ consulter le 14/04/2019
- **[PAR]** https://github.com/paritytech/parity-ethereum consulter le 14/04/2019
- **[POA]** https://github.com/ethereum/guide/blob/master/poa.md consulter le 17/04/2019
- **[SH]** https://github.com/a-samir-hussien/full_dapp consulter le 20/05/2019
- **[KOV]** https://kovan-testnet.github.io/website consulter le 02/05/2019
- **[FAU]** https://faucet.ropsten.be consulter le 02/05/2019
- **[REMIX]** https://remix.ethereum.org
- **[ETHS]** https://ethereum.stackexchange.com/questions/27048/comparison-of-the-different-testnets consulter le 03/06/2019
- **[BLKF]** https://www.une-blockchain.fr/tutorial-developpement-solidity-remix/ consulter le 05/06/2019
- **[GDUV]** https://github.com/dappuniversity/election consulter le 11/06/2019
- **[GRBV]** https://github.com/rajatdiptabiswas/ethereum-dapp-vote consulter le 11/06/2019