SeguraChain – Technologie de cryptomonnaie décentralisée
(Version expérimentale)

Version: 1.0
Écrit par: Sam Segura
Pays : France
Début du développement : 06/09/2020
Date: 14/10/2021
Dernière mise à jour : 03/06/2021

→ **Titres:**

I. Description du processus de mining PoW&C.
II. Description du système de gestion de la mémoire.
III. Description du système de communication P2P entre les nodes (P2P).
IV. Description du système de confirmation interne des transactions.
V. Description du système de mise à jour souveraine.
VI. Contraintes & problèmes potentiels.
VII. Présentations des outils disponibles.

Sommaire:

SeguraChain (Secured Chain ou Chaîne Sécurisée traduit de l'espagnol) est une technologie développée essentiellement en C#, elle permet de facilement créer, comprendre un processus de blockchain décentralisée, elle peut être configurée pour supporter un système de mise à jour souveraine, permettant d'inclure des mises à jour sur celle-ci sans éditer le contenu de la Blockchain. Elle a été développé pour mettre à jour **Xenophyte**, qui est une cryptomonnaie centralisé.

Ce système permet de certifier un node de synchronisation publique comme étant de confiance, de mettre à jour la méthode de processus de mining sans avoir la nécessité d'avoir un point réseau fixe et centraliser, les mises à jour sont échangées entre les nodes, ces mises à jour disposent d'une hauteur de bloc d'activation, permettant ainsi d'assurer le broadcast de celles-ci plus facilement avant leurs activations par les nodes/utilisateurs qui disposent de celles-ci en mémoire.

La méthode de processus de mining **PoW&C** (Preuve de travail et de compatibilité), repose sur différents processus reproductibles en fonction des données émises par le miner, ce processus utilise des informations du bloc précédent à celui actuellement ciblé mais aussi une partie du bloc de celui-ci.
Le contenu de ce travail détient l'identité du portefeuille du miner, permettant d'éviter l'absorption du travail par un node de synchronisation édité. **Une édition du travail, change radicalement la valeur finale du travail, ce qui empêche un tiers de s'en approprier.**

Les transactions envoyées entre les utilisateurs sont bien évidemment signées par leurs clés privées, leurs représentations sont signées, Les représentations respectent une structure précise pour simplifier, accélérer leurs recherches si nécessaire.
Cette technologie, dispose d'un système de gestion de la mémoire, permettant de ne pas conserver la totalité de la blockchain en mémoire, de travailler directement si nécessaire avec le système de cache, mais aussi de faire du **High Scalling** des données à travers différentes machines, afin d'obtenir un système suffisamment rapide et stable et fonctionnel sur le long terme.

Les données synchronisées à partir des nodes publiques, sont toujours celles émises à l'identique à travers la majorité des nodes contactés, bien évidemment une grille de vérifications interne permet de s'assurer que l'ensemble des données récupérées sont fonctionnel et toujours compatible avec les données précédentes détenus par l'utilisateur/node de la blockchain.

La majorité des processus exécutés sont des tâches dites asynchrones, pouvant être annulées en fonction de certains résultats, problèmes obtenues.
Une multitude de connections sont ainsi ouvertes pendant la synchronisation par exemple, ou pendant le broadcast de données, permettant de manière asynchrone, de recevoir/d'envoyer des données à différents nodes.
Il est tout à fait possible avec un peu de connaissance d'éditer le projet, de pouvoir centraliser un réseau en autorisant seulement les nodes certifiés par mise à jour souveraine comme étant ceux à avoir autorité à délivrer des données de synchronisation.

Voici une liste de quelques détails techniques, ainsi que la configuration par défaut que vous pouvez retrouver dans les codes sources de ce projet, vous pouvez ainsi apporter vos modifications à votre convenance:

Détails techniques:

→ Langage de développement: C#
→ Framework: **Net 5 (Retrocompatibilité : NetFramework 4.8)**
→ CrossPlatform: Oui via DotNetCore
→ Support: Windows/Linux/BSD, ARM (Raspberry PI 3 testé)
→ Technologie de signature utilisée: ECDSA avec SHA3-512, Curve: SECT571R1
→ Blockchain version: 01

Note: La méthode d'hashing ***SHA3-512*** *est utilisé dans l'ensemble des processus qui en ont besoin pour réduire au maximum de potentiels* ***attaques d'extension de longueur****, le* ***SHA2-512*** *est plus vénérable, même si une certaine puissance significative est nécessaire pour tenter ce genre d'attaque.*

Configuration par défaut:

→ Nom de la cryptomonnaie (par défaut): **Xenophyte**
→ Cygle: **XENOP**
→ Format adresse: Base58
→ Taille de checksum de l'encodage Base58: 16
→ Longueur adresse portefeuille WIF: 109
→ Longueur adresse publique WIF: 219
→ Longueur adresse privée WIF: 119
→ Maximum de décimales: 8
→ Maximum de coins pouvant être distribué: 26,000,000`.00000000`
→ Coût d'une taxe de transactions minimal: 0.000002
→ Coût par kb par rapport à la taille d'une transaction: 0.000002
→ Nombre de confirmations d'un bloc reward: 10
→ Nombre de confirmations minimal par transactions: 2
→ Nombre de blocs height cible minimal par défaut: 5
→ Maximum de transactions par blocs: 100 000
→ Échantillons de blocs utilisés pour jauger la valeur d'une taxe de transaction: 1440 blocs
→ Dev Fee: activée
→ Dev Fee pourcentage: 0,005%
→ Montant récompense d'un bloc: 10`.00000000 -` Dev Fee si activée`.`
→ Coin Halving: activée
→ Coin Halving intervalle de blocs: tout les 100 000 blocs débloqués
→ Support de mise à jour souveraine: activée
→ Bloc Time: 60 secondes.
→ Nombre de blocs attendu débloqués par jour: 1440
→ Échantillons de blocs utilisés pour calculé la prochaine difficulté: Tout les 480 blocs précédent.
→ Précision décimal de calcul de la difficulté: 100000
→ Difficulté minimale: 1000
→ Montant par défaut pré-miné (Genesis Bloc): 2,973,370`.00000000`
→ Nombre de transaction du Genesis Bloc: 1
→ Nombre de confirmation par réseau d'un bloc confirmé: 2
→ Délai maximum d'un share débloquant un bloc: +/- 60 secondes
→ Clé de marquage réseau: activée, permet de différencier différents réseaux en la modifiant.

I. Description du processus de mining PoW&C

→ Sommaire:

1. Contenu d'un share.
2. Processus effectués sur un nonce sélectionné.
3. Processus de cryptage effectué du share.
4. Calcul de la valeur du share.
5. Vérifications effectués sur le share.
6. Schéma de représentation du processus de mining.

Sommaire:

PoW&C – Preuve de travail et de compatibilité, est un mélange du PoW (Preuve de travail) et d'une preuve de compatibilité.
En réutilisant une partie du bloc précédent à celui cible, et en utilisant des informations du bloc cible, une preuve de compatibilité est donc soumise à la preuve de travail qui suit le processus, cela permet d'ignorer toute incohérence, incompatibilités par des shares invalides plus facilement.
Une partie du contenu du share doit contenir l'adresse de portefeuille du miner décodée, contournant des potentiels problèmes d'absorption du travail par un node de synchronisation publique qui aurait été éditée et qui souhaiterait récupérer le travail effectué.
Tout changements dans le contenu, le nonce, le cryptage, mais aussi sur les données utilisées changent directement la valeur finale d'un share.
Il est ainsi normal qu'un share ne soit plus fonctionnel sur un nouveau bloc cible et si édité, que sa valeur finale ne soit plus la même que celle du travail original.

1. Contenu d'un share.

Un miner génère tout d'abord un contenu PoC (preuve de compatibilité) avec des parties aléatoires à l'intérieur. La taille de ce contenu doit être égale à 129 bytes. Toutes différences seront considérées comme invalides.

Structure PoC d'un share:

Ce schéma montre la structure de donnée du contenu d'un share, plus précisément du PoC généré.

→ [0-4][0-8] = Premier nombre | Second Nombre = le nombre de transactions du bloc précédent.
→ [8-16] = Le timestamp du share.
→ [16-48] = Donnée de checksum aléatoire, si le miner utilise un Pool, c'est le Pool qui le communique.
→ [48-113] = Adresse de portefeuille décodée, si le miner utilise un Pool, c'est l'adresse du pool qui sera communiqué et utilisé.
→ [113-121] = La hauteur du bloc cible.
→ [121-129] = Le nonce utilisé.
Total: 129 Bytes, représentant le contenu total du PoC.

Spécifications:

Les nombres contenu dans le PoC, sont utilisés pour retourner le nombre de transactions du bloc précédent, seulement certains opérateurs mathématiques sont utilisés: **+ * % -**
Le timestamp contenu dans le PoC, est utilisé dans le cas que la valeur finale du share soit suffisante pour débloqué le bloc cible, soit suffisamment proche du Bloc Time, pour l'accepter, il convient que dans les processus établie, que toutes triches du timestamp, ne soit toléré.
Par exemple, un timestamp beaucoup trop en avance par rapport à celui actuel, ne sera pas valide, comme pour un timestamp beaucoup trop en retard.
Également le timestamp contenu doit être indiqué dans la requête réseau servant au broadcast, dans le cas d'une validité absolue, le/les nodes cibles exécuteront un broadcast du travail fourni en ciblant les autres nodes.

2. Processus effectués sur un nonce sélectionné.

Lorsque le PoC est généré correctement, le miner sélectionne un nombre appelé Nonce, entre un intervalle minimal et maximal accepté dans la configuration du processus de mining.
Ce nonce généré, retravaillé à travers différents processus, est utilisé dans la méthode finale de cryptage des données du share en tant que IV.
Lorsqu'un node de synchronisation est appelé à vérifier un share afin de le validé et de débloqué le bloc cible ainsi que d'effectuer un broadcast, tout les processus dédiés à cette partie sont utilisés pour vérifier si le Nonce source retourne bien le IV généré.

Configuration par défaut d'un nonce valide:

→ Minimum: 1
→ Maximum: 4294967295

1. Premièrement le processus génère le nonce entre intervalle valide, via une méthode de génération aléatoire sécurisée.
2. Le nombre généré est convertit en tableau de bytes pour être ainsi travaillé avec les processus suivant.
3. Un nombre précisément établie dans la configuration de mining de computations SHA3-512 sont appliquées sur le Nonce.
4. Ensuite, un processus XOR, effectue des changements sur le résultat des computations.
5. Une méthode, génère des points, retournant des angles, permettant de générer un pseudo carré virtuel, un certain nombres de computations sont effectuées, si un carré est trouvé, le processus ignore les autres computations à effectuées, sinon un autre nombres de computations SHA3-512 sera effectuées, ce processus donne une dose de "chance" aléatoire.
6. Les deux premiers processus de computations sont effectuées à nouveau.
7. Compression du travail par le biais de l'algorithme de compression LZ4.
8. Pour finir, génération du IV de cryptage par dérivation RFC2898.

3. Processus de cryptage effectué du share.

Pour crypter le travail du miner, la représentation des transactions du bloc précédent (final transaction hash) est utilisé comme clé de cryptage, pour compléter le cryptage l'IV généré à travers le nonce sélectionné par le biais des processus de travail effectués est utilisé.

Pour générer la clé de cryptage, la représentation des transactions est soumise à une computation SHA3-512, le résultat obtenu retourne un tableau de 64 bytes il est ainsi redimensionné à la taille de 32 bytes pour être fonctionnel en tant que clé de cryptage.
L'outil de mining proposé est optimisé pour garder en mémoire cette clé générée tant que le bloc cible n'est toujours pas débloqué.

Configuration AES:

→ Key size: 256bits.
→ Block size: 128bits.
→ Mode: CFB.
→ Padding: PCKS7.

Un certain nombre établie dans la configuration de mining de boucles de cryptages sont effectuées.
Lorsque que cette tâche est effectuée, une donnée convertit en hexadécimal du share est égale à **352 caractères**. Cette donnée est importante pour le calcul de la valeur du share généré.

4. Calcul de la valeur du share.

Le processus de calcul de la valeur d'un share est simple, une computation SHA3-512 est exécuté sur la share crypté, la représentation généré est ensuite convertit en BigInteger.
Pour finir un calcul mathématique entre la difficulté du bloc cible, ainsi que le maximum de possibilité de représentation d'un SHA 512 peut offrir est effectué.

Formule mathématique:

valeur_share = BigInteger(SHA3-512(share_crypté))

valeur_share_final = (2^512 / difficulté_bloc) / (valeur_share / difficulté_bloc).

Exemple, en assumant que la valeur du share est environ égale à 2^494 et que la difficulté du bloc cible est de 300000, le calcul final est:

(2^512 / 300000) / (2^494 / 300000) = 262144

Dans ce cas précis, la valeur du share est inférieur à celle de la difficulté du bloc, le share ne sera donc pas accepté pour débloquer celui-ci.
Si elle est supérieur ou égale à celle du bloc, alors il débloquera le bloc actuel.

5. Vérifications effectuées sur le share.

Une vérification effectuée sur un share s'effectue de la même manière que la génération d'un share, sauf que celle-ci est effectuée à l'envers, avec diverses méthodes au préalable pour vérifier son contenu tant sur la requête reçu qu'au contenu de celle-ci, tant sur le format du contenu, longueurs par exemple.

Comme précédemment expliqué, le timestamp du PoC contenu dans le share crypté doit être identique à la requête envoyée par le miner.
Il en est de même que l'adresse de portefeuille fournis dans cette requête, elle doit être identique à celle contenu dans le PoC.

La requête doit indiquer avec exactitude certaines informations du bloc cible, sa hauteur, sa représentation hexadécimal (block hash).

La requête doit également indiquer le nonce utilisé, la représentation hexadécimal du IV généré, le share crypté au format hexadécimal, ainsi que la valeur du share qui doit être retrouvé durant la vérification.

- Structure d'un share envoyé au format JSON:

```
{
	WalletAddress:"adresse_portefeuille_miner",
	BlockHeight:"hauteur_du_bloc_cible",
	BlockHash:"représentation_du_bloc_cible",
	PoCShare:"le_share_crypter_en_hexadécimal",
	Nonce:"le_nonce_sélectionné",
	NonceComputedHexString:"le_iv_généré_à_partir_du_nonce",
	PocShareDifficulty:"la_valeur_du_share_calculer",
	Timestamp:"le_timestamp_contenu_dans_le_poc_du_share"
}
```

- Réception:

Lorsqu'un share est reçu, un node acceptera seulement les shares qui dispose d'un timestamp suffisamment récent ou peu en retard pour le prendre en compte.

→ Vérifications effectuées:

1. Vérification du format de l'adresse de portefeuille contenu dans la requête.
2. Vérification du bloc hash, pour assurer qu'il soit identique à celui ciblé et valide.
3. Vérification du nonce sélectionné.
4. Vérification du contenu du share crypté fourni, vide, format/longueur invalide par exemple.
5. Vérification de la valeur du share en fonction du share crypté fourni.
6. Vérification par décryptage du share crypté.
7. Vérification du IV fournis en générant par le biais du nonce fournis que celui-ci est le même.
8. Vérification de la preuve de compatibilité.
9. Vérification du timestamp contenu dans le PoC.
10. Comparaison de la valeur du share comme étant valide pour débloquer le bloc cible.

→ Broadcast du share:

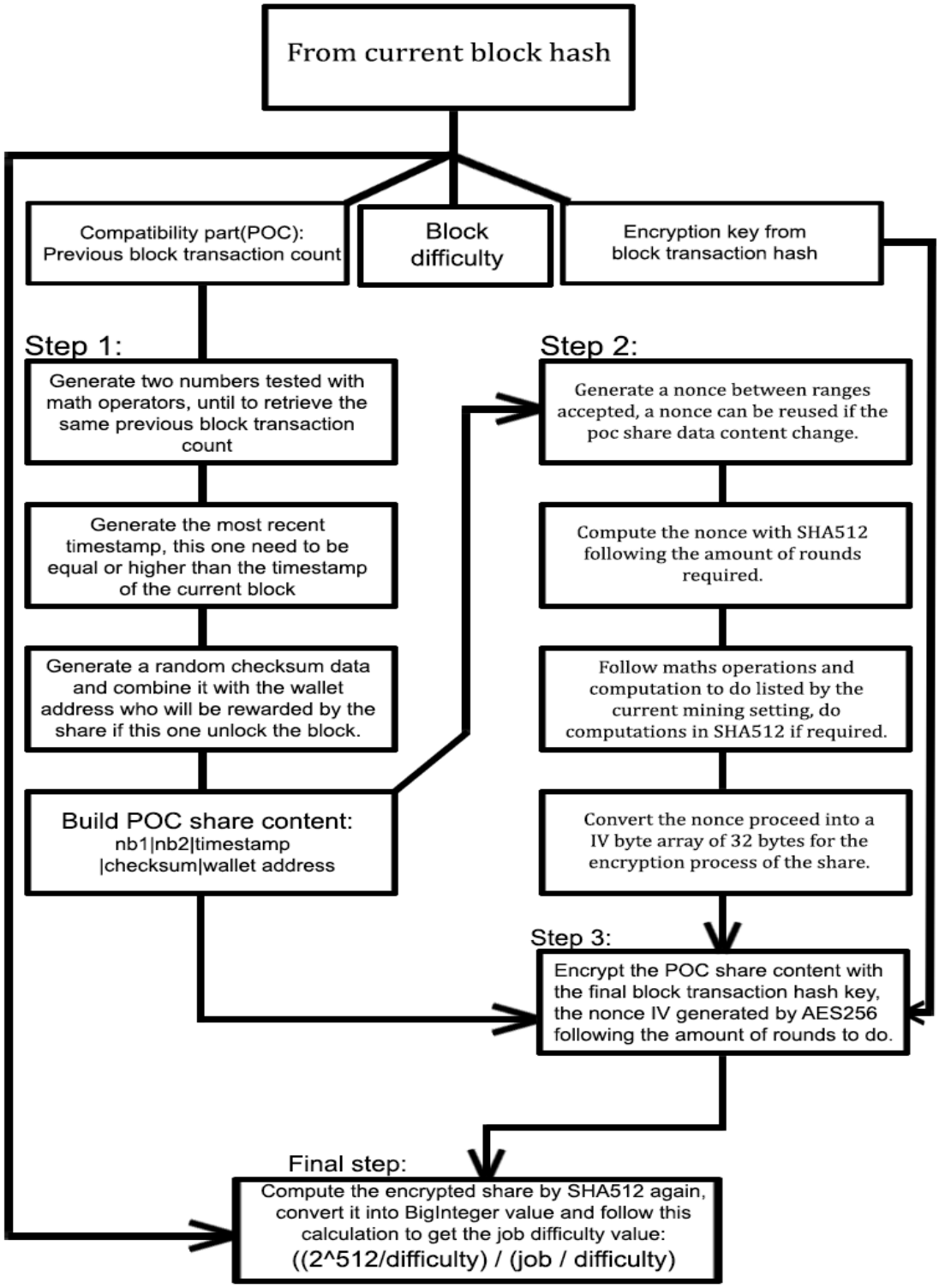
Lorsque le share fournir débloque le bloc cible, le ou les nodes ciblés effectue un broadcast de celui-ci à tout les nodes listés dans leurs mémoire pour s'assurer que celui-ci soit prise en compte par la majorité des nodes du réseau.

→ Étape de confirmation réseau:

Cette étape est exécuté dans le système de synchronisation des nodes, le principe est simple, appelé les différents nodes en mémoire à retourner les metadata du bloc débloqué,
si tout est identique une compteur de confirmation réseaux est incrémenté dans la mémoire du node, lorsque la limite est atteinte, celui-ci est considéré comme étant disponible dans la majorité du réseau décentralisée.

6. Schéma de représentation du processus de mining.

Pour tenter de simplifier l'ensemble des informations, voici un schéma représentant les processus effectuées.



II. Description du système de gestion de la mémoire.

Pour donner la possibilité de rester fonctionnel sur le moyen/long terme, un système de cache a été conçu pour ce problème, permettant aux nodes mais aussi au Desktop Wallet des utilisateurs de fonctionner dans le temps quelque soit la charge de données qu'impose la Blockchain synchronisé.

Ce système de cache est conçu pour fonctionner qu'avec ce qui est nécessaire d'avoir, afin d'éviter tout problèmes, de perte de performances, de compatibilité, et autres, aucune technologie externe n'a été utilisé pour la concevoir, cela permet d'avoir un système indépendant, fonctionnant qu'avec les principes et processus établies.

Par défaut le système de cache est configuré en mode **DISK**, un certain nombres d'index de fichiers seront alors créer en fonction de la blockchain.

Les flux sont maintenus, rendant l'exécution d'une instruction lecture/écriture plus rapide, les flux sont également ouvert en mode asynchrone avec accès aléatoire, donnant une indication supplémentaire au système pour la mise en cache des changements sur les fichiers.

Chaque éléments mis à jours, sont écrits à la fin de leurs index de fichier, ainsi la longueur totale, ainsi que leurs positions sont sauvegardés en mémoire afin de pouvoir les relire rapidement sur demande sans devoir relire tout l'index de fichier dédié aux données stockées en cache.

Une tâche asynchrone dans la gestion de mémoire est exécuté sur un intervalle de temps reconfigurable pour réécrire les contenus des fichiers de cache afin de sauvegarder de l'espace disque.

Lorsqu'un élément est appelé plusieurs fois sur un intervalle de temps court qui lui aussi est reconfigurable, celui-ci est stocké en mémoire si la mémoire maximale alloué le permet.

Lorsque la mémoire maximale alloué est atteinte, le système de cache, va tenter de sauvegarder les données en mémoire mise à jours sur les index de fichiers et les enlever de la mémoire, afin d'obtenir suffisamment de mémoire disponible pour conserver l'élément.

Si l'élément en question ne peut pas être stocké en mémoire, il sera réécrit dans le cache si celui-ci a été mise à jour, sinon il sera simplement ignoré.

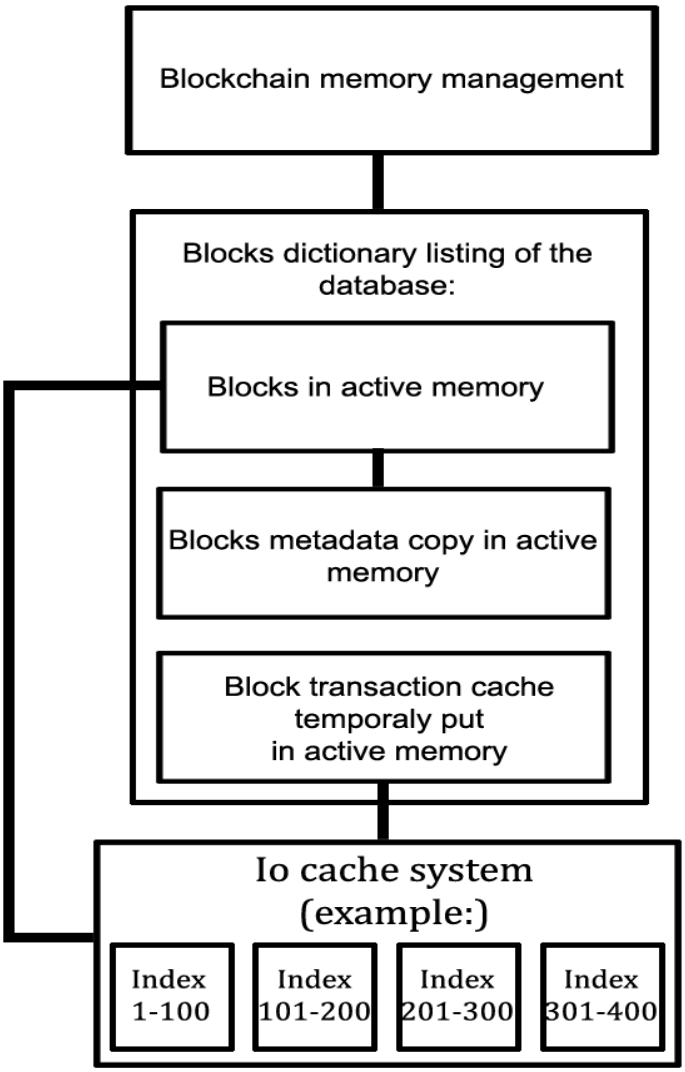
Les éléments supprimé de la mémoire active non mises à jours, seront également ignorés, et ne seront donc pas réécrit puisque leurs versions actuelle est déjà écrite dans leurs index de fichiers de cache.

Pour assurer le fonctionnement de ce système, chaque index dispose d'un Semaphore, permettant de bloquer l'accès multithreading lorsqu'une tâche est déjà en cours, il sera libéré lorsque la tâche est complétée, annulée.

Un index de fichier peut contenir plusieurs blocs synchronisé, ce nombre est également reconfigurable dans la configuration du système de cache.

Pour le mode **NETWORK** de ce système, l'ensemble des processus décrits sont également effectués, mais à la différence, sur l'hôte ou les hôtes réseaux qui détiennent les données, un flux réseau est alors ouvert pour exécuter les différents ordres équivalent au système en mode DISK.

Pour finir, voici un schéma simplifié du système de gestion de la mémoire de la Blockchain:



III. Description du système de communication P2P entre les nodes.

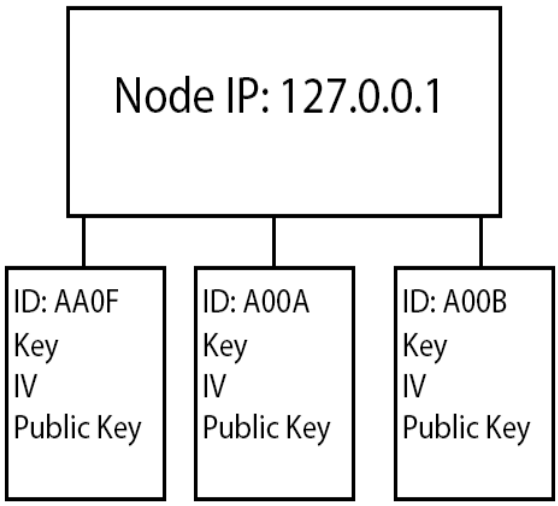
1. Structure d'un node listés:

Les nodes listés sont répertoriés par IP mais aussi par un UniqueID généré et communiqué par le node qui a échangé différentes informations.
Ce système de listing permet de listé plusieurs nodes publique avec différentes informations, mais aussi de configuration réseaux, tels que le port de communication ouvert au publique.
En donnant cette possibilité, un hébergeur de nodes publique peut aisément arrêté l'un d'entre eux tout en gardant au moins ou plusieurs nodes accessible.

Les clés de cryptages et la clé publique permettant de vérifier la signature des paquets envoyés/reçu peuvent être identique d'un node à l'autre ou être complètement différents, par défaut un node contactant un autre node, générera des clés différentes pour la communication.

Le cryptage des communications n'est pas réellement nécessaire mais apporte une étape supplémentaire de compatibilité à travers tout le réseau décentralisée.

Voici un simple schéma de listing de plusieurs nodes possédant tous la même IP publique:



Configuration par défaut d'un node:

→ Longueur Unique ID hash: 128 (Non configurable)
→ Port P2P par défaut: 2400
→ Port P2P minimale: 1
→ Port P2P maximale: 65535
→ Max connections P2P par IP: 1000
→ Max connections API par IP: 1000
→ Semaphore max délai d'attente par entrée de la même IP: 5 secondes.
→ Synchronisation par rangée: Activé.
→ Max transactions synchronisées par rangée: 5
→ Max blocs synchronisés par rangée: 5

D'autres options sont configurables, et permettent d'optimiser, d'ajuster la configuration réseau d'un node.

2. Description du système de synchronisation:

Le système de synchronisation d'un node exécute plusieurs tâches asynchrones dans une instance de synchronisation, elles sont réparties en plusieurs types de tâches, permettant de ne pas effectuées celle-ci une par une.

Chaque tâche de synchronisation exécuté, dispose d'une liste de nodes générée, mise à jour, et garde le plus longtemps possible les connections ouvertes avec ceux-ci pour accélérer les communications.

Le système de mise à jour des listes de nodes, peut également ajouter de nouveaux nodes si en court de route, de nouveaux ont été récupérés, et de supprimer ceux qui seraient inaccessibles, ou ayant retournés trop d'éléments invalides.

Liste des tâches de synchronisation:

→ Tâche de demande de liste de nodes:

Cette tâche demande aux nodes déjà listés en mémoire, des nodes potentiellement non connu, permettant d'agrandir le nombre de node contact connu.

→ Tâche de synchronisation des mises à jour souveraines:

Cette tâche demande aux nodes connus s'ils possèdent des mises à jour souveraines, elles sont récupérées, vérifiées et installées si elles sont valides et précédemment inexistante sur le node.

→ Tâche de synchronisation de la progression de la Blockchain:

Cette tâche demande aux nodes connus, la progression actuelle de la Blockchain, la majorité retournant la même progression, sera prise en compte.
Une vérification interne est effectuées pour s'assurer que les données sont bien valide.

→ Tâche de synchronisation des blocs:

Cette tâche demande aux nodes connus en fonction de la progression actuel de la Blockchain récupérée dans la tâche de synchronisation dédiée à celle-ci, les métadonnées des blocs.

En fonction de la configuration du node, un ou plusieurs métadonnées de blocs sont récupérées.
Les métadonnées récupérées sont ceux étant retournées à l'identique via la majorité, des vérifications internes sont effectuées avant de les prendre en compte.

Une fois la récupération des métadonnées effectuées, une ou plusieurs tâches de synchronisation de leurs transactions sont effectuées, de la même manière, les données utilisées sont celle émises par la majorité et des vérifications internes sont effectuées.

Pour finaliser, la synchronisation d'un bloc, le final transaction hash du bloc récupéré par synchronisation de la métadonnée du bloc en question est comparée avec la représentation hash des transactions synchronisées, si cette représentation est identique et que toute les vérifications précédentes sont valides, le node estime que le bloc est valide.

→ Tâche de vérification des blocs non confirmés par réseaux:

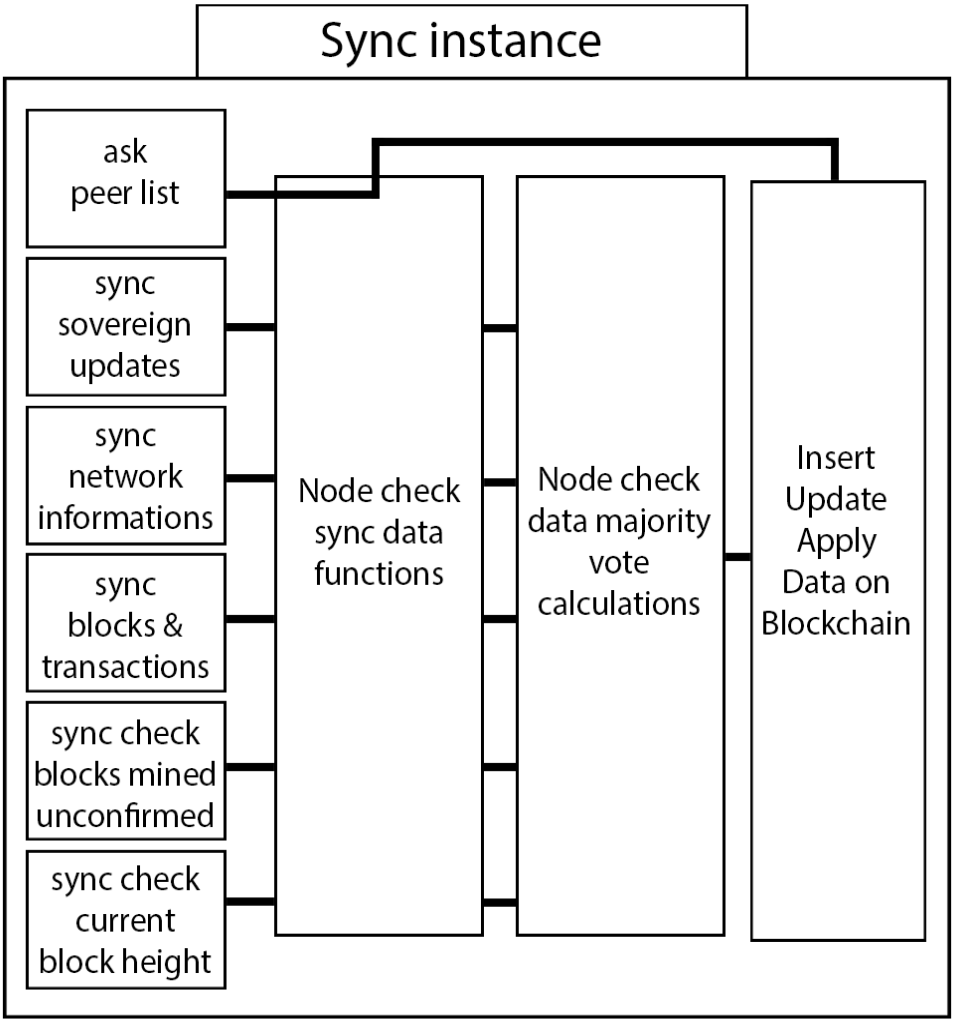
Cette tâche demande aux nodes connus en fonction des blocs contenu par le node, de vérifier si la majorité retourne avec exactitude les mêmes métadonnées des blocs débloqués qui n'ont pas été confirmés par réseaux.

Si un ou plusieurs blocs ne sont pas identiques à la majorité, ils sont resynchronisés.
Une fois que cette étape est effectuées, les blocs confirmés par réseaux pourront être prises en compte dans la tâche de confirmation des transactions qui s'effectue en interne.

→ Tâche de vérification de l'état du bloc height actuel:

Cette tâche demande aux nodes connu si le bloc height actuel est débloqué, cette tâche porte assistance aux nodes non publique ou à ceux publique n'ayant pas pu recevoir avec succès le broadcast d'un share débloquant celui-ci.
Cette vérification prend toujours en compte le résultat de la majorité et des vérifications internes sont effectuées.

Voici un schéma simple décrivant le processus de l'instance de synchronisation d'un node:



Les paquets envoyés/reçus, sont au format **JSON** et formatés en **Base64** séparés par un caractère séparateur unique pour assurer leurs transmissions. Le caractère séparateur indique la fin d'un paquet, si durant la réception la somme des données dépasse une certaine limite, l'ensemble des données reçus seront supprimés et la tâche de réception sera annulée.
Comme énoncé précédemment, les communications entre les nodes sont cryptées, il est ainsi le cas durant les tâches de synchronisations.
Chaque connections ouvertes ont chacune un objet de connections pour envoyer une requête et recevoir les données attendus, cela assure la bonne réception en parallèle de chaque requêtes.

3. Description du système de broadcast:

Le système de broadcast est seulement utilisés pour transmettre des transactions, mining shares aux autres autres nodes publiques, qui eux même feront le broadcast si les données transmises sont nouvelles et valides.
Ces tâches ce font de manière asynchrones et donc de manière indépendante à tout les autres tâches effectuées par les autres systèmes d'un node de synchronisation.
Un **Desktop Wallet** ou un **RPC Wallet**, effectue également le broadcast des données qu'ils envoient, cependant par défaut, ils ne sont pas disponible au publique pour recevoir des données de l'extérieur par le biais de ce système.
Ils peuvent synchroniser les transactions en attente d'être prises en charge par un bloc depuis les nodes publiques.

Deux tâches de broadcast sont dédiés pour les transactions :

Deux listes d'instances sont générées de manière continue, tant que des nodes sont disponibles, ces instances sont ensuite exécutées et leurs états surveillées.

Les instances ouvrent des connexions vers les nodes ciblés, tout en essayant de les maintenir le plus longtemps possible en y envoyant des requêtes de **Keep-Alive**.

- La première liste contient des tâches asynchrones qui consiste à envoyer les transactions à chaque nodes listé, de manière continu **si le node est configuré en mode publique seulement**:

Cette tâche tourne en boucle tant que le **Node** cible est accessible, l'instance envoie toute les transactions en **MemPool** et conserve la progression pour ne pas les renvoyées au node ciblé.

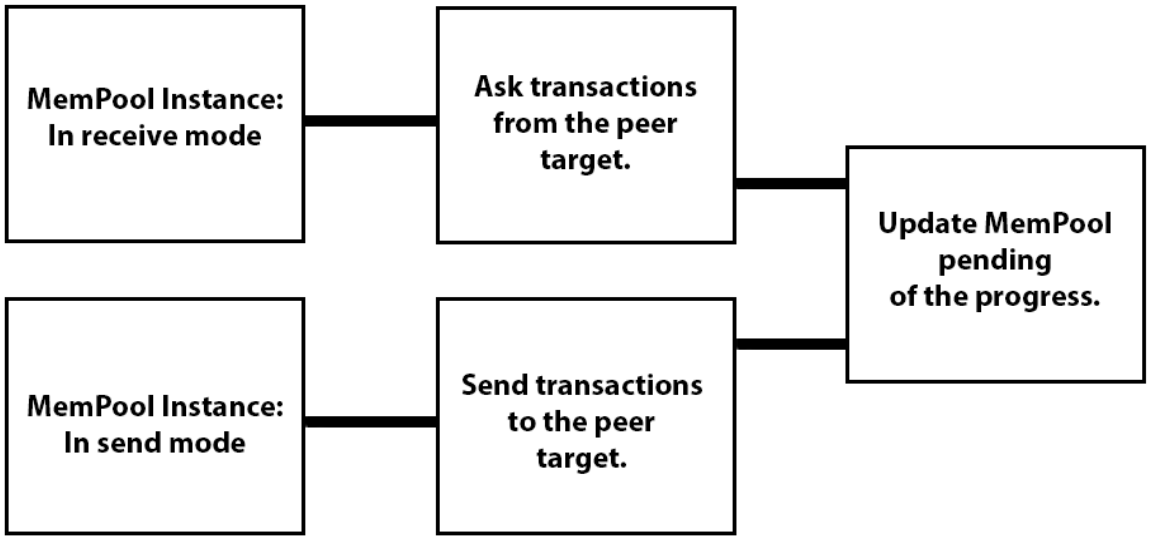
L'instance récupère une réponse du **Node** cible pour prendre en compte la réception de la transaction envoyée.

- La deuxième liste contient des tâches asynchrones qui consiste à recevoir des transactions depuis chaque nodes listés de manière continu :

Ces tâches consistes à recevoir une liste de hauteur de blocs contenu dans leurs MemPool de chaque nodes ciblés, une fois leurs listes reçus, si elles contiennent un nombre positif de transactions, des demandes de réceptions des transactions sont effectuées à partir de chacune des instances.

La progression de chaque réceptions sont sauvegardées pour ne pas les redemandés par la suite, à chaque nouvelle demandes, si une hauteur de bloc contient plus de transactions, les demandes sont relancées en indiquant le nombre déjà réceptionné, pour ne pas recevoir à nouveau toute les transactions déjà synchronisées.

Voici un schéma simple des deux instances de broadcast MemPool ciblant un node publique :



4. Description du système de vote sur les contenus reçus:

Quand un élément de synchronisation ou d'une demande de validation par broadcast sont effectuées, Le node effectue un vote après la vérification du contenu ou de la réponse à une validation reçu en listant les éléments identiques dans une liste, la majorité est prise en compte, sauf si un nombre de réponse de nodes est inférieur à une certaine limite reconfigurable dans le fichier de paramétrage du node.

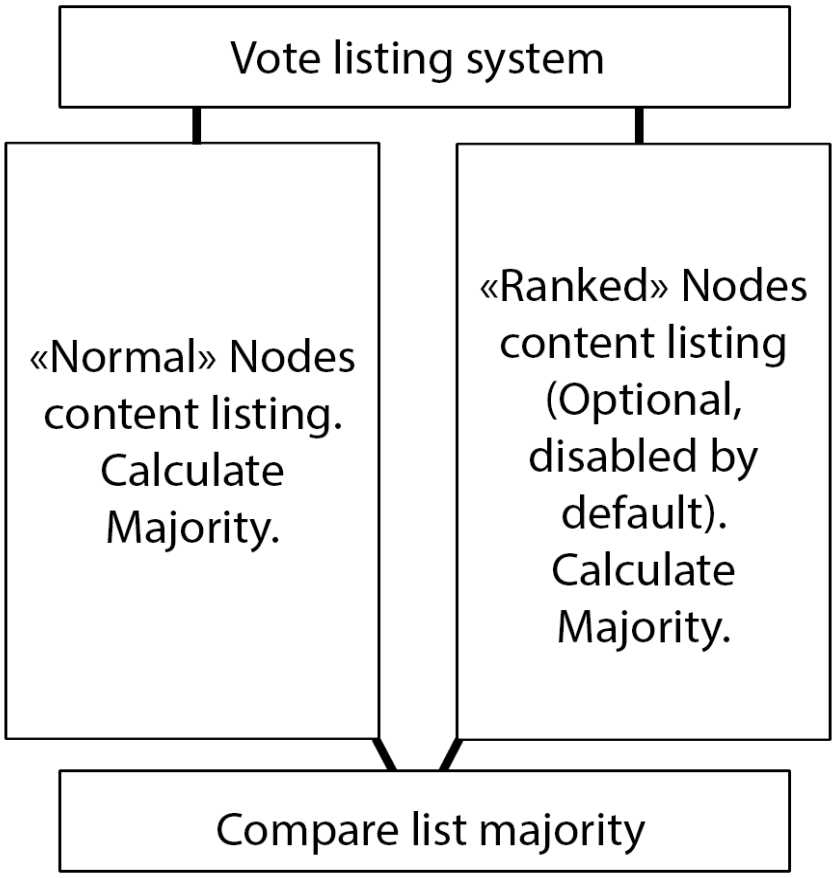
Il existe également une autre liste, regroupant les éléments reçus de nodes certifiés par une ou plusieurs mises à jour souveraine appliqué.
Une vérification des signatures par le biais des clés publiques référencés par ces mises à jour souveraines des contenus reçus de la part de ces nodes est effectuées.

Autre chose, même si plusieurs nodes d'une même IP envoi une réponse, seule un des nodes de la même IP sera prise en compte dans le vote.
Il est en de même pour les nodes certifiés par mise à jour souveraine, évitant ainsi toute triche.

Une comparaison entre les divers listes de contenu est effectuée, ensuite une comparaison entre les listes de contenu majoritaire entre les Nodes "Normaux" et les Nodes "Ranked" est effectuée.

Le listing de node certifiés est optionnel, même si les mises à jour souveraines sont faites automatiquement lors de la synchronisation.
Dans le cas où cette option reste désactivée, les nodes listés sont considérés comme des Nodes "normaux".

Voici un schéma simple de description du système de vote:



IV. Description du système de confirmation interne des transactions.

Une tâche dédiée à la confirmation des transactions s'exécute en interne du node de synchronisation,

1. Vérification.

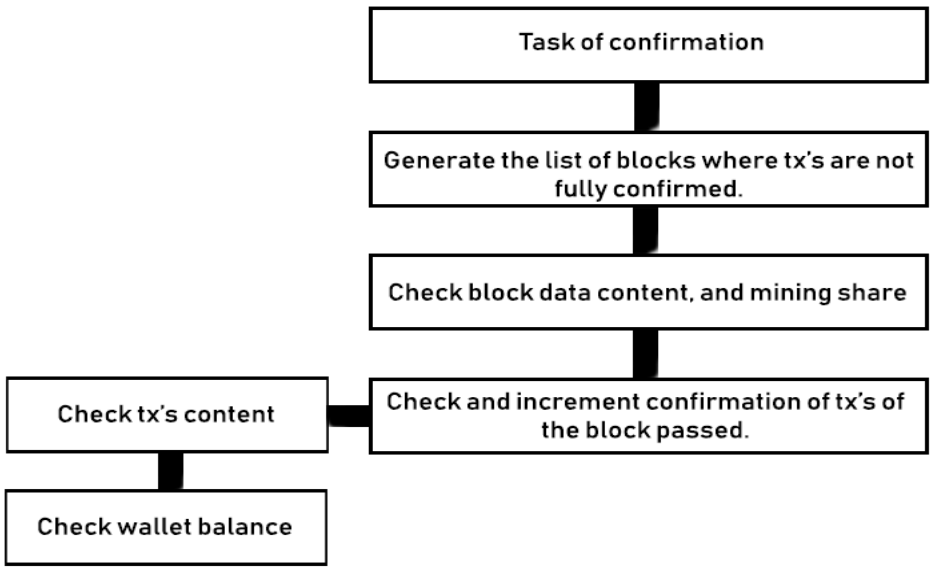
Pour commencer une vérification du contenu, formats, valeurs est effectuées.
Une vérification du contenu et les signatures associées à ce même contenu d'une transaction est effectuée, cette vérification intervient qu'une seule fois, c'est à dire au moment de procéder à la première confirmation.
Une représentation « légère » contenu dans la transaction est régénérée pour comparer que la même représentation fournis est exacte, puis une vérification de la signature associée à cette représentation est testée.
Par la suite un **big hash**, une représentation est régénérée car non fournie dans la transaction pour alléger son envoie, stockage. Seule la signature de cette représentation est contenu dans la transaction, elle est ensuite testé avec cette représentation.
Cette représentation est composée des éléments de la transaction découpée tout les **1024 bytes**, pour empêcher l'utilisation d'un exploit telle que l'**extension length attack.**

2. Wallet Balance.

A chaque confirmation nécessaire, une balance est calculée par rapport aux transactions précédentes, afin de s'assurer que le montant envoyé est disponible jusqu'au moment de la dernière confirmation.

Des **checkpoints** sont générées afin de ne pas repartir du premier bloc jusqu'au dernier, cela permet d'accélérer le calcul de cette balance.

Les **checkpoints** sont ensuite enregistrées dans un cache contenant les adresses de portefeuille ayant eu au moins une transaction, ce cache tant à évoluée en utilisant le disque dur afin de ne pas utiliser trop de mémoire vive.



V. Description du système de mise à jour souveraine.

Le système de mise à jour souveraine utilise la clé privé développeur, pour signer les mises à jour souveraine.
Elles permettent de faire des changements sur la Blockchain de manière contrôler, sans modifier le contenu, ni les transactions.
Elles permettent cependant d'effectuer les changements suivant :

- D'assigner un rang « **SEED** » sur un node synchronisation, permettant de lui conférer plus de pouvoir sur les votes de synchronisation, il est seulement pris en compte si un node en question active le mode suivant, ce « pouvoir » est tout simplement ignoré.

- De supprimer le rang « **SEED** » sur un node de synchronisation.

- De pouvoir changer la signature du développeur assigné par défaut dans la configuration de la class **BlockchainSetting.cs** sans éditer le code.

- De pouvoir effectuer des changements sur la configuration d'algorithme de **mining**.

Pour garantir au maximum la propagation d'une mise à jour souveraine, une hauteur de bloc est demandée, elle permet de s'activer à partir de la hauteur programmé.

Tout un système de suivis des mises à jour souveraine permet de s'assurer de la progression de tout les changements effectuer, étant donné qu'un seul changement sur la configuration de mining, change tout les résultats nécessaires et à venir.

De même pour le changement de signature de développeur.

VI. Contraintes & problèmes potentiels.

Contrairement aux autres cryptomonnaie qui n'ont pas choisi d'exposer les contraintes et problèmes potentiels de la décentralisation, ceux-ci sont exposés ici.

Il existe différentes manières de compromettre un système décentralisé, toute cryptomonnaie décentralisé peuvent y faire face, tout dépend des moyens à disposition pour les mettre en place.

Les problèmes et contraintes exposés ne permettent en rien d'éditer, de soumettre des transactions invalides, ni de pouvoir modifier les balances des portefeuilles, étant donné que contrairement aux autres cryptomonnaie décentralisées, les vérifications ce font interne dans une tâche parallèle après déblocage des blocs, une attaque de 51% du hashrate total, ne pourra donc pas permettre le « double spending ».

Seulement, si une cryptomonnaie y fait face, cela peut avoir des conséquences assez grande sur son fonctionnement, pour être honnête, il valait mieux les décrire pour ne pas avoir de mauvaises surprises.

**Le système de mise à jour souveraine, permettant de certifier des nodes, peut limiter voir bloquer ce genre d'attaque, sans porter atteinte à la décentralisation, étant donné que comme les nodes « normaux », sont également affectés comme ceux-ci avec les mêmes règles de fonctionnement.
Étant donné le scepticisme de la part des utilisateurs, le système de vote séparé des nodes certifiés est désactivé par défaut.**

1. Compromettre le système de broadcast.

Dans le cas où le réseau ne serait pas assez grand, c'est à dire avec un nombre réduit de nodes publique disponible qui interagisse entre eux, ou si une personne mal intentionné disposerait des moyens financier et technique pour compromettre le système de broadcast, cela pourrait affecter légèrement voir grandement celui-ci.

Admettons que le réseau en question ce compose de seulement 5 nodes publiques, détenu par 5 individus différent, si une personne mal intentionné souhaiterait influencer le broadcast, il pourrait bien évidemment exécuter un nombre plus important de nodes publique et ignorer toute transactions en broadcast.

Ensuite pour finaliser cette attaque, il suffirait d'un hashrate plus important que la moyenne pour compromettre le broadcast des transactions en attente et donc de les « annuler » de la prise en compte de celle-ci sur leurs blocs height cible.

Il n'en reste pas moins que l'attaque en question n'est pas infaillible, une tentative peut résulter d'un Fork (copie) virtuelle et les nodes en questions détenu par la personne mal intentionné ce retrouverait en dehors du réseau.

Cette attaque peut également compromettre le broadcast des mining shares, si la majorité détenu par cette personne les ignores, et ne prend en compte que les siens, cela pourrait influencer sur la priorité de progression de la blockchain, évidemment, il faut que cela soit fait suffisamment rapidement avec un hashrate conséquent pour y arriver.

2. Compromettre le système de synchronisation.

Dans le même cas précédemment expliqué, un réseau pas assez grand pourrait être la cible de ce genre d'attaque.
Si une personne mal intentionné souhaite compromettre la synchronisation des nouveaux venu, c'est à dire à partir d'un point inférieur à celui actuellement disponible sur le réseau, il est possible de faire synchroniser une autre blockchain, étant donné qu'il s'agit de la nouvelle majorité disponible.

Cela ne fonctionne que si l'utilisateur ou le node en synchronisation ce retrouve à un point de synchronisation bien inférieur à celui actuelle.

VII. Présentations des outils disponibles.

Voici la liste des outils disponibles, ils sont développés en utilisant la référence **SeguraChain-Lib.dl**l.

Cette référence contient un ensemble de class, permettant de générer la base de donnée nécessaire, exécuter la gestion mémoire, d'exécuter une instance de node synchronisation en interne et bien d'autres class.

Notez que les outils présentés évolueront, que leurs apparences peuvent changer au cours du développement, des mises à jour dans l'avenir.
Tout les outils comporte le nom de Xenophyte, étant donné qu'elle sera mise à jour par l'ensemble de tout ces outils présenté.

XENOPHYTE - Desktop Wallet :

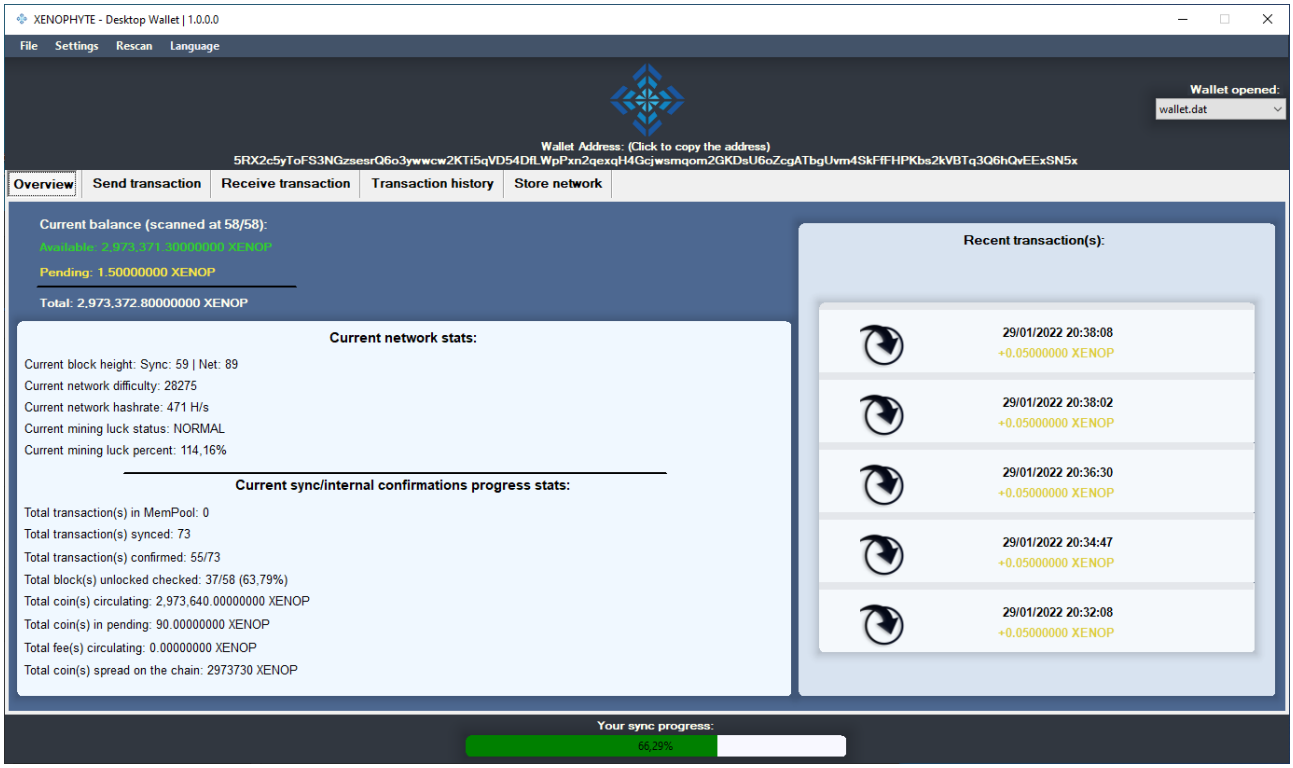
C'est le Desktop Wallet par défaut fournis aux utilisateurs, il exécute en son sein, une instance d'un node de synchronisation, configuré comme étant fermé de l'extérieur, sans participation donc au réseau.

Il est possible de le configurer pour rendre son instance de synchronisation accessible au publique pour participer au réseau, si la bande passante de l'utilisateur le permet évidemment.

Il est également possible de configurer le Desktop Wallet pour ce connecter à un Node externe ou hébergé en local, cependant il est recommandé de toujours utiliser son propre Node de synchronisation.

Le design a été programmé avec des controls **Winform** customisés, certains effets graphiques comme les bords arrondie sur les panels, bitmaps affichés, ainsi que les effets d'ombres ont été programmé et affiché avec le moteur graphique par défaut de **Winform**, **GDI+**.

Les historiques de transactions affichés, sont également des instances virtuelles créer totalement avec des **Rectangles**, **Bitmap**, **Graphics** et des calculs de maths.
Les instances des historiques de transactions sont contenu en mémoire et mises à jour en tâche de fond.



XENOPHYTE – Peer :

Il s'agit d'un outil exécutant l'instance de node de synchronisation, il permet de synchroniser la blockchain, faire du solo mining, lié des outils externes tels que le Desktop/RPC Wallet, participer au réseau en le configurant en tant que node publique.
C'est l'outil indispensable pour faire fonctionner le réseau décentralisé.

XENOPHYTE – Solo Miner :

Le Solo Miner, permet de participer au réseau en effectuant le processus de mining pour débloquer des blocs, chaque blocs débloqué avec succès donne un certains nombre de cryptomonnaie, suivant la configuration établie et la progression en cours de la Blockchain.
Cette outil est indispensable pour faire progresser la Blockchain, prendre en charge les transactions en attente, plus un nombre de miners différents participent, plus la quantité de hashrate estimé augmente, plus le réseau sera solide contre de potentiels attaques énoncé dans le **chapitre VI.**