SeguraChain - Decentralized Cryptocurrency Technology
(Experimental version)

Version: 1.0
Written by: Sam Segura
Start of development: 06/09/2020
Date: 02/28/2021
Last updated: 19/03/2021

→ **Titles:**

Summary:
SeguraChain (Secured Chain translated from Spanish) is a technology developed in C #. It allows users to easily create and understand a decentralized blockchain process. It can be configured to support a sovereign update system, allowing updates to be included on it without editing the content of the Blockchain. It was developed to update Xiropht, which is a centralized cryptocurrency.

It will also make it possible to further minimalize cryptocurrencies, because this technology is easy to implement. This will increase the difficulty for those who wish to create a coin with this algo to differentiate themselves Given the simplicity of creating a coin.
When a task to be accomplished becomes simple, the final value becomes generally much lower over time. Users will be aware of the ease to create one for themselves.

This system makes it possible to certify a public synchronization node as being trusted, to update the mining process method without having the need to have a fixed network point and centralize the updates are exchanged between nodes, these updates have an activation block height, thus allowing to ensure the broadcast of these more easily before their activations by the nodes / users who have these in memory.

The PoW & C (Proof of Work and Compatibility) mining process method is based on different reproducible process according to the data emitted by the miner, this process uses information from the block previous to the one currently targeted but also part of the block thereof.
The content of this work holds the identity of the miner's portfolio, avoiding the absorption of the work by an edited synchronization node. An edit of the work radically changes the final value of the work, this which prevents a third party from appropriating it.

Transactions sent between users are obviously signed by their private keys, their representations are signed, The representations respect a precise structure to simplify and accelerate

their research if necessary.

This technology has a memory management system, making it possible not to keep all of the memory, blockchain in memory, to work directly if necessary with the cache system, but also to make **High Scaling** of data across different machines, in order to obtain a sufficiently fast and stable system and functional in the long term.

The data synchronized from the public nodes are always those sent identically through the majority of the nodes contacted, of course an internal checklist makes it possible to ensure that all data recovered is functional and still compatible with previous data held by the user / node of the blockchain.

The majority of the processes executed are so-called asynchronous tasks, which can be canceled depending on some results, problems obtained.

A multitude of connections are thus open during synchronization for example, or during broadcast data, allowing asynchronously to receive / send data to different nodes.
It is quite possible with a little knowledge to edit the project, to be able to centralize a network in only authorizing nodes certified by sovereign update as being those with authority to issue synchronization data.

Here is a list of some technical details, as well as the default configuration that you can find in the source codes of this project, you can thus make your modifications to your convenience:

Technical details:
→ Development language: C #
→ Framework: **Net 5 (Retrocompatibility: NetFramework 4.8)**
→ CrossPlatform: Yes via DotNetCore
→ Support: Windows / Linux / BSD, ARM (Raspberry PI 3 tested)
→ Signature technology used: ECDSA with SHA3-512, Curve: SECT571R1
→ Blockchain version: 01

Note: The **SHA3-512** hashing method is used in all processes that need it to minimize potential length extension attacks, SHA2-512 is more **venerable,??**? even if some significant power is needed to attempt this kind of attack.

Default Configuration:
→ Name of the cryptocurrency (default): XIROPHT
→ Cygle: XIRO
→ Address format: Base58
→ Base58 encoding checksum size: 16
→ WIF wallet address length: 109
→ WIF public address length: 219
→ WIF private address length: 119
→ Maximum of decimal places: 8
→ Maximum corners that can be distributed: 26,000,000.00000000
→ Cost of a minimum transaction tax: 0.000002
→ Cost per kb compared to the size of a transaction: 0.000002

→ Number of confirmations of a reward block: 10
→ Minimum number of confirmations per transaction: 2
→ Default number of minimum target height blocks: 5
→ Maximum transactions per block: 100,000
→ Sample blocks used to gauge the value of a transaction tax: 1440 blocks
→ Dev Fee: activated
→ Dev Fee percentage: 0.005%
→ Amount reward of a block: 10.00000000 - Dev Fee if activated.
→ Coin Halving: activated
→ Coin Halving block interval: every 100,000 blocks unlocked
→ Sovereign update support: enabled
→ Block Time: 60 seconds.
→ Expected number of blocks unlocked per day: 1440
→ Sample blocks used to calculate the next difficulty: All previous 480 blocks.
→ Decimal precision of difficulty calculation: 100,000
→ Minimum difficulty: 1000
→ Pre-mined default amount (Genesis Bloc): 2,973,370.00000000
→ Number of Genesis Block transactions: 1
→ Number of confirmations per network of a confirmed block: 2
→ Maximum delay of a share unlocking a block: +/- 60 seconds
→ Network tagging key: activated, allows different networks to be distinguished by modifying it.

I. Description of the PoW & C mining process

→ Summary:

1. Content of a share.
2. Processes performed on a selected nonce.
2. Encryption process performed on the share.
3. Calculation of the share value.
4. Checks carried out on the share.
5. Diagram showing the mining process.

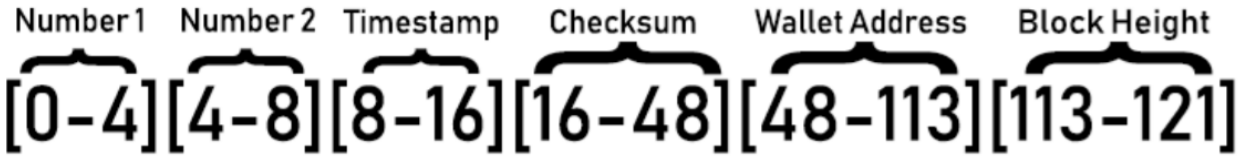
Summary:
PoW & C - Proof of Work and Compatibility, is a mixture of PoW (Proof of Work) and a proof of compatibility.

By reusing part of the previous block to the target one, and using information from the target block, a proof of compatibility is therefore subject to the proof of work that follows the process, this allows to ignore any inconsistency, incompatibilities by invalid shares more easily.
Part of the content of the share must contain the decoded miner's wallet address, bypassing potential problems of absorption of work by a public synchronization node that would have been edited and who would like to recover the work done.
Any changes in the content, the nonce, the encryption, but also on the data used change directly the final value of a share.
It is therefore normal that a share is no longer functional on a new target block and if edited, that its

value final is no longer the same as that of the original work.

1. Content of a share.
A miner first generates PoC content (proof of compatibility) with random parts to inside. The size of this content must be equal to 121 bytes. Any differences will be considered as invalid.

PoC Structure of a Share:



This diagram shows the data structure of the content of a share, more precisely of the generated PoC.

→ [0-4] [0-8] = First number | Second Number = the number of transactions in the previous block.
→ [8-16] = The timestamp of the share.
→ [16-48] = Random checksum data, if the miner uses a Pool, the Pool communicates it.
→ [48-113] = Decoded wallet address, if the miner uses a Pool, the pool address will be communicated and used.
→ [113-121] = The height of the target block.

Total: 121 Bytes, representing the total content of the PoC.

Specifications:
The numbers contained in the PoC, are used to return the number of transactions of the previous block, only certain mathematical operators are used: + *% -
The timestamp contained in the PoC, is used in the case that the final value of the share is sufficient for unlocked the target block, which is close enough to the Time Block, to accept it, it is appropriate that in the established process, that all timestamp cheats are not tolerated.
For example, a timestamp much too far ahead of the current one will not be valid, like for a much too late timestamp.
Also the contained timestamp must be indicated in the network request used for the broadcast, in the case of absolute validity, the target node (s) will broadcast the work provided by targeting the other nodes.

2. **Processes performed on a selected nonce.**

When the PoC is generated correctly, the miner selects a number called Nonce, between a minimum and maximum interval accepted in the configuration of the mining process.
This generated nonce, reworked through different processes, is used in the final method of encryption of share data as IV.
When a synchronization node is called to verify a share in order to validate it and unblock the target block as well as performing a broadcast, all the processes dedicated to this part are used to check if the source Nonce returns the generated IV.

Default configuration of a valid nonce:
→ Minimum: 1
→ Maximum: 4294967295

1. First the process generates the nonce between valid interval, via a method of secure random generation.
2. The generated number is converted into an array of bytes to be used in this way with the next process.
3. A number precisely established in the mining configuration of SHA3-512 computations are applied on the Nonce.
4. Next, an XOR process makes changes to the result of the computations.
5. A method, generates points, returning angles, allowing to generate a pseudo virtual square, a number of computations are performed, if a square is found, the process ignores other computations to be performed, otherwise another number of SHA3-512 computations will be performed, this process gives a dose of random "luck".
6. The first two processes of computations are performed again.
7. Work compression using the LZ4 compression algorithm.
8. Finally, generate the RFC2898 Bypass Encryption IV.

3. Encryption process performed on the share.

To encrypt the work of the miner, the representation of transactions from the previous block (final transaction hash) is used as an encryption key, to complete the encryption the IV generated at through the selected nonce through performed work processes is used.

To generate the encryption key, the representation of transactions is subjected to a computation SHA3-512, the result obtained returns an array of 64 bytes so it is resized to the size of 32 bytes to be functional as an encryption key.
The proposed mining tool is optimized to keep in memory this generated key as long as the block target is still not unlocked.

AES configuration:

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

A number of established in the mining configuration of encryption loops are performed.
When this task is performed, a data converted into hexadecimal of the share is equal to 288 characters. This data is important for calculating the value of the share generated.

4. Calculation of the share value.

The process of calculating the value of a share is simple, a SHA3-512 computation is performed on the encrypted share, the generated representation is then converted into BigInteger.
To finish a mathematical calculation between the difficulty of the target block, as well as the maximum of possibility of representation of a SHA 512 can offer is performed.

Mathematical formula:

share_value = BigInteger (SHA3-512 (share_crypter))

share_value_final = (2 ^ 512 / block difficulty) / (share_value / block difficulty).

Example, assuming that the value of the share is approximately equal to 2 ^ 494 and that the difficulty of the block target is 300,000, the final calculation is:

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

In this specific case, the value of the share is lower than that of the difficulty of the block, the share will not be therefore not accepted to unlock this one.
If it is greater than or equal to that of the block, then it will unlock the current block.

5. Checks carried out on the share.

A check performed on a share is carried out in the same way as the generation of a share, except that this one is carried out in reverse, with various methods beforehand to check its content both on the request received and the content thereof, both on the format of the content, lengths for example.

As previously explained, the PoC timestamp contained in the encrypted share must be identical to the request sent by the miner.
It is the same as the wallet address provided in this request, it must be identical to that contained in the PoC.

The request must accurately indicate certain information of the target block, its height, its hexadecimal representation (block hash).
The request must also indicate the nonce used, the hexadecimal representation of the generated IV, the share encrypted in hexadecimal format, as well as the share value that must be found during the verification.

- Structure of a share sent in JSON format:

```
{
        WalletAddress: "wallet_address_miner",
        BlockHeight: "target_block_height",
        BlockHash: "target_block_hash_representation",
        PoCShare: "encrypted_share_hex_format",
        Nonce: "the_selected_nonce",
        NonceComputedHexString: "the_iv_generated_from_the_nonce",
        PocShareDifficulty: "share_value",
        Timestamp: "timestamp_poc_share"
}
```

- Reception:
When a share is received, a node will only accept shares that have a timestamp recent enough or little late to take it into account.

→ Checks carried out:

1. Checking the format of the wallet address contained in the request.
2. Checking the hash block, to ensure that it is identical to the targeted one and valid.
3. Verification of the selected nonce.
4. Verification of the content of the encrypted share provided, empty, invalid format / length for example.
5. Checking the share value according to the encrypted share provided.
6. Verification by decryption of the encrypted share.
7. Verification of the IV provided by generating through the nonce provided that it is the even.
8. Verification of the proof of compatibility.
9. Verification of the timestamp contained in the PoC.
10. Comparison of the share value as being valid to unlock the target block.

→ Share broadcast:

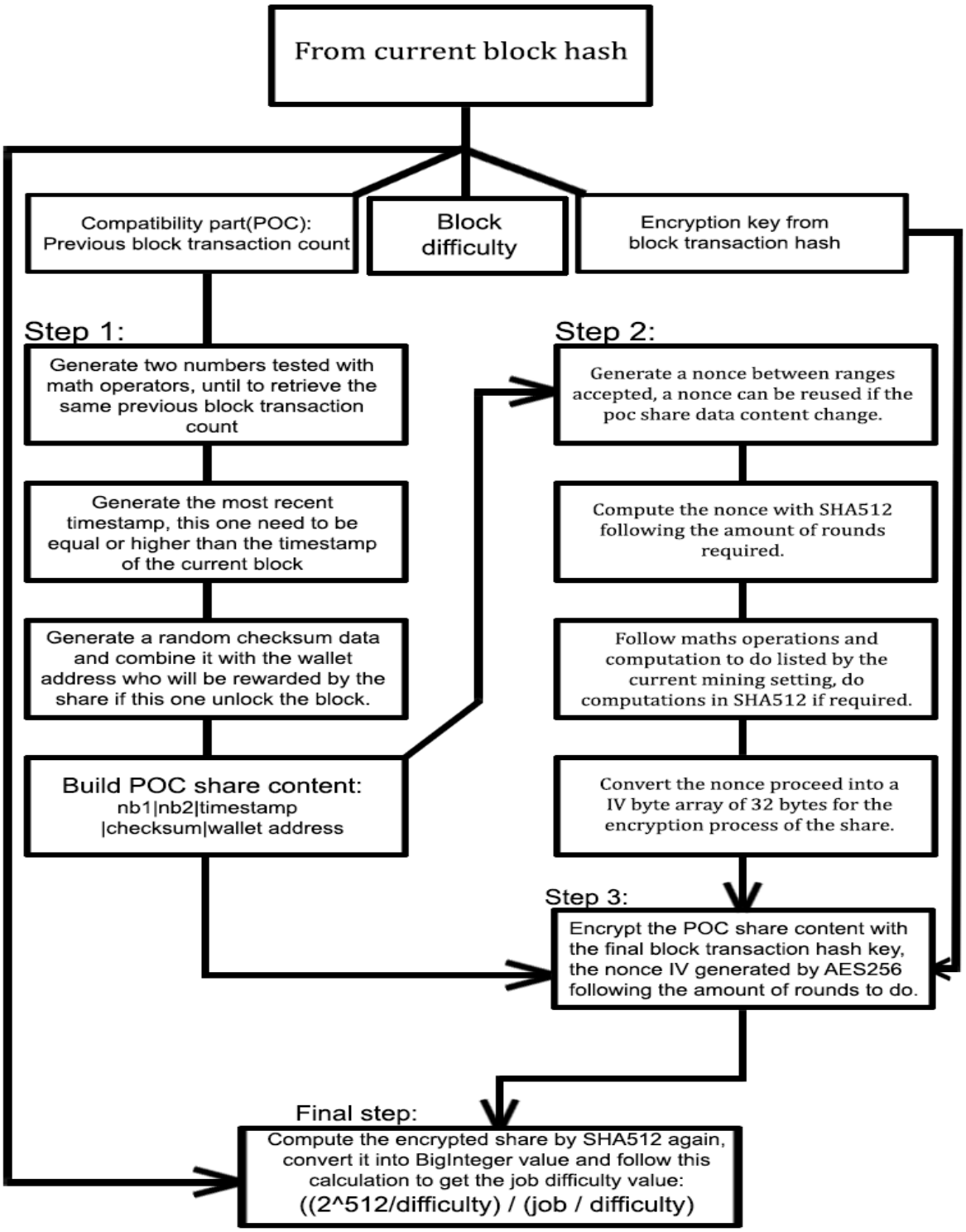
When the share provided unblocks the target block, the target node (s) performs a broadcast of this to all the nodes listed in their memory to ensure that it is taken into account by the majority of the nodes of the network.

→ Network confirmation step:

This step is performed in the node synchronization system, the principle is simple, called the various nodes in memory to return the metadata of the unlocked block, if everything is identical, a network confirmation counter is incremented in the memory of the node, when the limit is reached, it is considered to be available in the majority of the decentralized network.

6. **Diagram showing the mining process.**

To try to simplify all the information, here is a diagram representing the processes carried out.



II. Description of the memory management system.

To give the possibility of remaining functional in the medium / long term, a caching system has been designed for this problem, allowing nodes but also the Desktop Wallet of users of operate over time regardless of the data load imposed by the synchronized Blockchain.

This cache system is designed to work only with what is necessary to have, in order to avoid any problems, loss of performance, compatibility, and others, no external technology was used to design it, this allows to have an independent system, functioning only with established principles and processes.

By default the cache system is configured in DISK mode, a certain number of file indexes will then be created according to the blockchain.

The streams are maintained, making the execution of a read / write instruction faster, the streams are also opened in asynchronous mode with random access, giving an indication additional to the system for caching file changes.

Each updated elements are written at the end of their file index, thus the total length, as well as their positions are saved in memory so that they can be read quickly on request without having to re-read the entire file index dedicated to cached data.

An asynchronous task in memory management is executed on a time interval reconfigurable to rewrite the contents of cache files to save space disk.

When an element is called several times over a short time interval which is also reconfigurable, it is stored in memory if the maximum allocated memory allows it.

When the maximum allocated memory is reached, the cache system will attempt to save the update data in memory on file indexes and remove them from memory, so get enough free memory to keep the item.

If the item in question cannot be stored in memory, it will be written back to the cache if it has been updated, otherwise it will just be ignored.

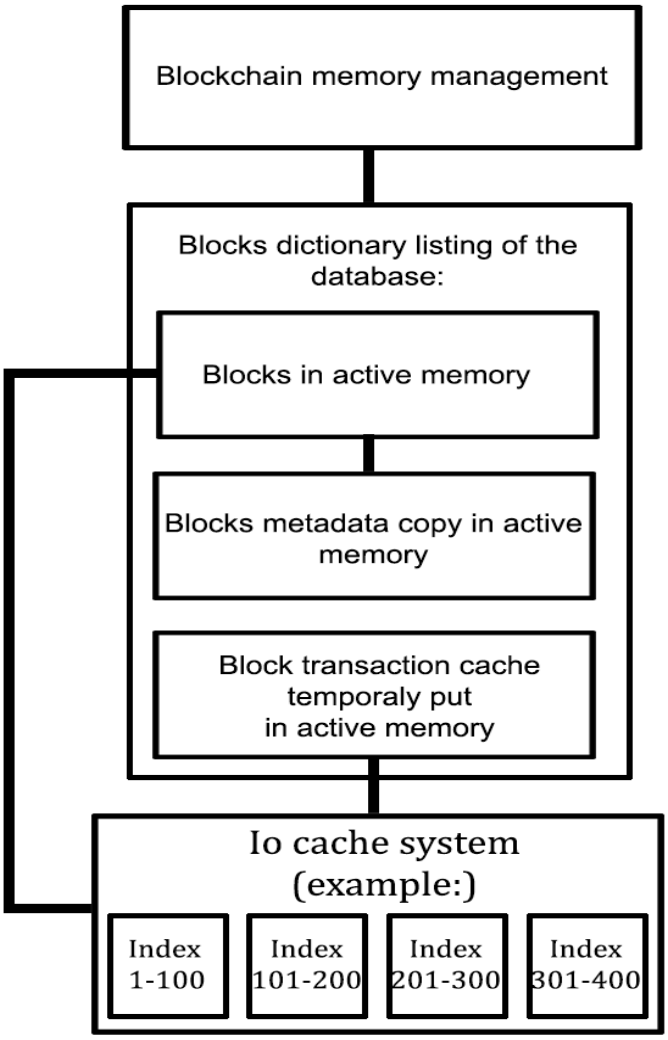
Items deleted from active memory that are not updated will also be ignored, and will not be will therefore not be rewritten since their current versions are already written to their file indexes. Hidden.

To ensure the functioning of this system, each index has a Semaphore, allowing to block multithreading access when a task is already in progress, it will be released when the task is completed, canceled.

A file index can contain several synchronized blocks, this number is also reconfigurable in the cache system configuration.

For the **NETWORK** mode of this system, all the processes described are also performed, but with the difference, on the host or the network hosts which hold the data, a flow network is then open to execute the different commands equivalent to the system in DISK mode.

Finally, here is a simplified diagram of the Blockchain memory management system:



III. Description of the P2P communication system between the nodes.

1. Structure of a listed node:

The listed nodes are listed by IP but also by a UniqueID generated and communicated by the node which exchanged different information.

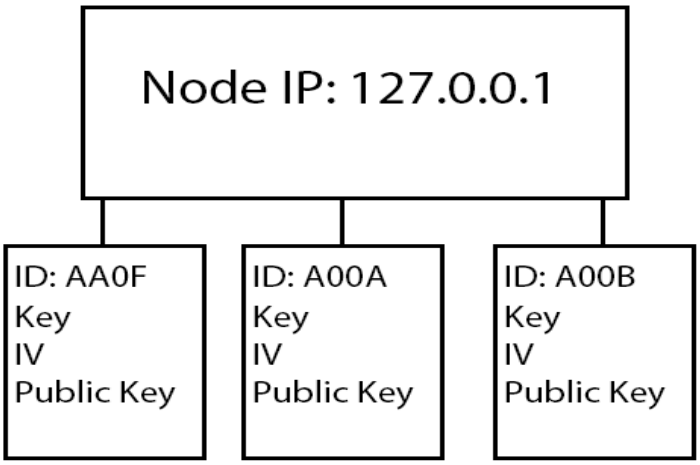
This listing system allows you to list several public nodes with different information, but also network configuration, such as the communication port open to the public.
By giving this possibility, a public node host can easily shut down one of them while keeping at least or more nodes accessible.

Encryption keys and the public key to verify the signature of sent / received packets

can be identical from one node to another or be completely different, by default a node contacting another node, will generate different keys for communication.

Encryption of communications is not really necessary but brings a step additional compatibility throughout the decentralized network.

Here is a simple listing diagram of several nodes all having the same public IP:



Default configuration of a node:

→ *Unique ID hash length: 128 (Not configurable)*
→ *Default P2P port: 2400*
→ *Minimum P2P port: 1*
→ *Maximum P2P port: 65535*
→ *Max P2P connections per IP: 1000*
→ *Max API connections per IP: 1000*
→ *Semaphore max waiting time per entry of the same IP: 5 seconds.*
→ *Row synchronization: Enabled.*
→ *Max synchronized transactions per row: 5*
→ *Max synchronized blocks per row: 5*

Other options are configurable, and make it possible to optimize, adjust the network configuration of a node.

2. Description of the synchronization system:

The synchronization system of a node performs several asynchronous tasks in an instance of synchronization, they are divided into several types of tasks, making it possible not to perform this one by one.

Each synchronization task executed, has a list of nodes generated, updated, and kept as long as possible the open connections with them to accelerate communications.
The node list update system, can also add new nodes if in short road, new ones were recovered, and to delete those which would be inaccessible, or having returned too many invalid items.

List of synchronization tasks:

→ Node list request task:

This task asks nodes already listed in memory, nodes that are potentially unknown, allowing to increase the number of known contact nodes.

→ Sovereign updates synchronization task:

This task asks known nodes if they have sovereign updates, they are retrieved, checked and installed if they are valid and previously non-existent on the node.

→ Blockchain progress synchronization task:

This task asks known nodes, the current progress of the Blockchain, the majority returning the same progression, will be taken into account.
An internal check is performed to ensure that the data is valid.

→ Block synchronization task:

This task requests the known nodes according to the current progress of the Blockchain retrieved in the synchronization task dedicated to this, the metadata of the blocks.

Depending on the configuration of the node, one or more block metadata is retrieved.
The metadata retrieved are those returned identically via the majority, internal checks are performed before taking them into account.

Once the metadata has been retrieved, one or more synchronization tasks for their transactions are carried out, in the same way, the data used are those issued by the majority and internal audits are carried out.

To finalize, the synchronization of a block, the final transaction hash of the block retrieved by synchronization of the metadata of the block in question is compared with the hash representation of the synchronized transactions, if this representation is identical and that all the previous checks are valid, the node considers that
the block is valid.

→ Checking task for unconfirmed blocks by networks:

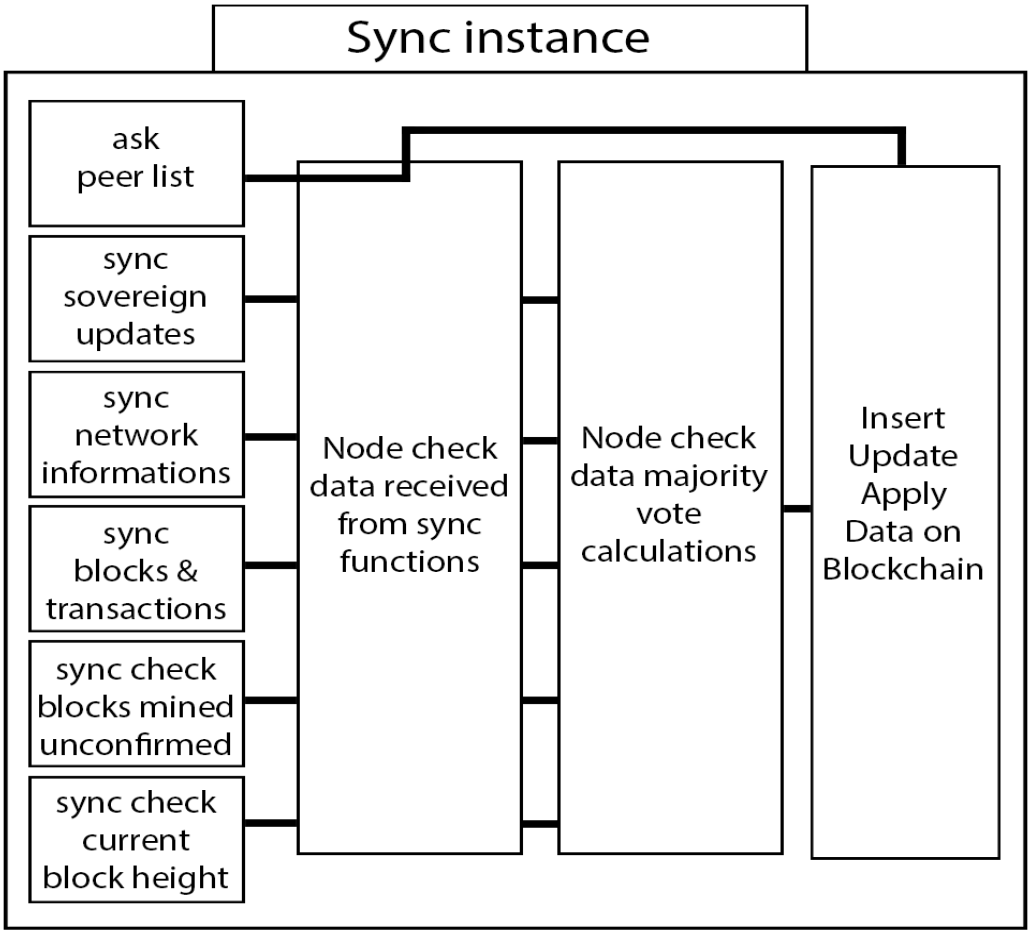
This task asks the nodes known according to the blocks contained by the node, to check if the majority accurately returns the same metadata of unlocked blocks that have not been confirmed by networks.

If one or more blocks are not identical to the majority, they are resynchronized. Once this step is carried out, the blocks confirmed by networks can be taken into account in the internal transaction confirmation task.

→ Task to check the state of the current height block:

This task asks known nodes if the current height block is unlocked, this task supports non-public nodes or to those public nodes that could not successfully receive the broadcast of a share unlocking this one.
This verification always takes into account the result of the majority and internal verifications are carried out.

Here is a simple diagram describing the process of a node sync instance:



The packets sent / received, are in **JSON** format and formatted in **Base64** separated by a character single separator to ensure their transmissions. The separator character indicates the end of a packet, if during reception the sum of the data exceeds a certain limit, the set of received data will be deleted and the receive job will be canceled.

As stated previously, the communications between the nodes are encrypted, it is thus the case during synchronization tasks.

Each open connections each have a connection object to send a request and receive the expected data, this ensures the correct reception in parallel of each request.

Two broadcast tasks are dedicated for transactions:

Two lists of instances are generated continuously, as long as nodes are available, these instances are then executed and their states monitored.

Instances open connections to targeted nodes, while trying to maintain them the most long possible by sending **Keep-Alive** requests to it.

- The first list contains asynchronous tasks which consists of sending transactions to each nodes listed, continuously if the **node** is **configured in public mode only:**

This task loops as long as the target **Node** is accessible, the instance sends all the transactions in **MemPool** and keep the progress so as not to send them back to the targeted node.

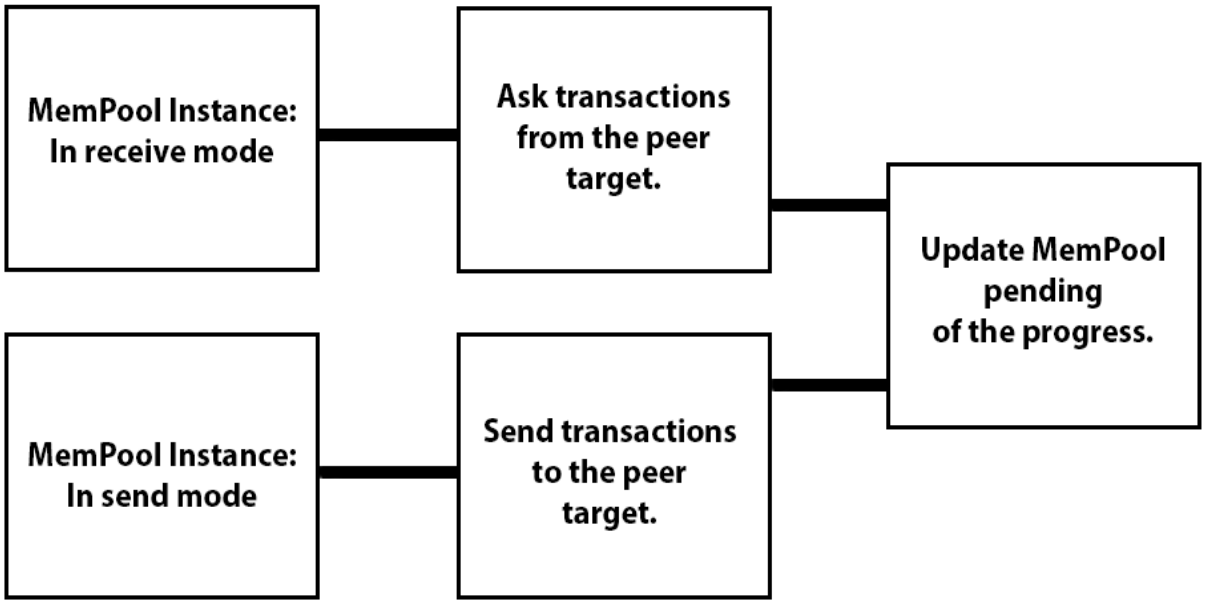
The instance retrieves a response from the target node to take into account the reception of the transaction sent.

- The second list contains asynchronous tasks which consists of receiving transactions from each nodes listed continuously:

These tasks consist of receiving a list of block heights contained in their MemPools of each target nodes, once their lists have been received, if they contain a positive number of transactions, requests to receive transactions are made from each of the instances.

The progress of each reception is saved so that they are not requested again later, each new request, if a block height contains more transactions, the requests are relaunched by indicating the number already received, so as not to receive all the transactions already synchronized.

Here is a simple diagram of the two MemPool broadcast instances targeting a public node:



4. **Description of the voting system on the content received:**

When a synchronization element or a broadcast validation request are made,
The node performs a vote after checking the content or the response to a validation received in listing the identical elements in a list, the majority is taken into account, unless a number of response from nodes is less than a certain reconfigurable limit in the configuration file of the node.

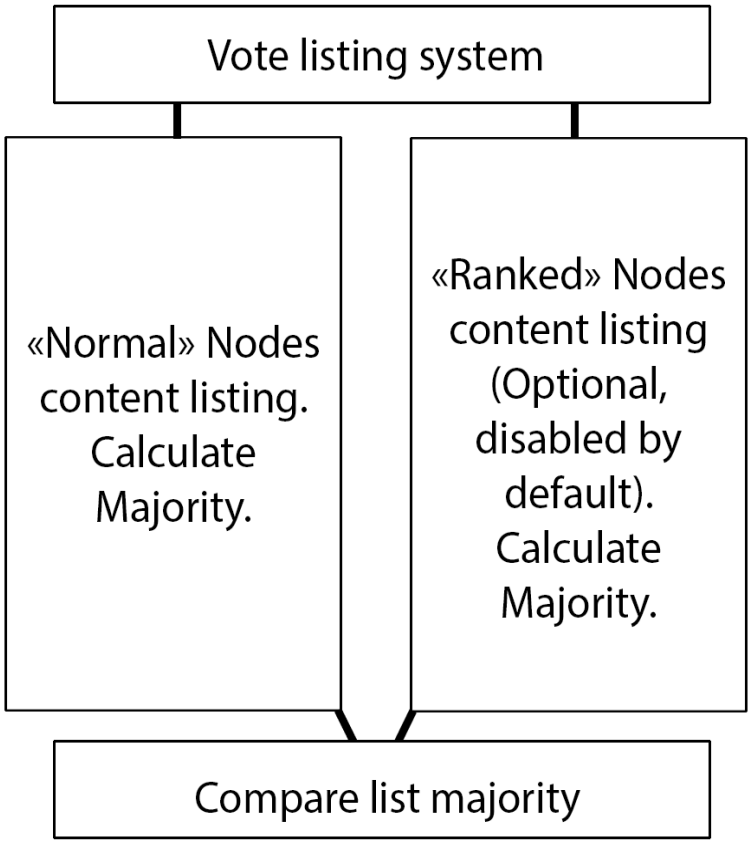
There is also another list, regrouping the elements received from nodes certified by one or several sovereign updates applied. A verification of the signatures through the public keys referenced by these updates sovereign content received from these nodes is performed.

Another thing, even if several nodes of the same IP sends a response, only one of the nodes of the same IP will be taken into account in the vote.
It is the same for the nodes certified by sovereign update, thus avoiding any cheating.

A comparison between the various content lists is made, then a comparison between the Majority content lists between "Normal" Nodes and "Ranked" Nodes are made.

Listing of certified nodes is optional, even if sovereign updates are made automatically during synchronization.
If this option remains disabled, the listed nodes are considered as Nodes "Normal".

Here is a simple diagram describing the voting system:



Description of the internal transaction confirmation system.

Uncomplete part.

VI. Constraints & potential problems.

Unlike other cryptocurrencies which have not chosen to expose the constraints and problems potentials of decentralization, these are presented here.

There are different ways to compromise a decentralized system, any decentralized cryptocurrency can cope, it all depends on the means available to put them in place.

The problems and constraints exposed do not allow editing, submitting invalid transactions, nor to be able to modify the balances of the portfolios, since unlike the other decentralized cryptocurrency, the checks are done internally in a parallel task after the blocks, an attack of 51% of the total hashrate, will therefore not allow “double spending”.

Only, if a cryptocurrency faces it, it can have rather big consequences on its operation, to be honest, it was better to describe them so as not to have unpleasant surprises.

The sovereign update system, allowing nodes to be certified, can limit or block this kind of attack, without undermining decentralization, given that like nodes "Normal", are also assigned like these with the same operating rules. Given the skepticism on the part of users, the separate voting system for certified nodes is disabled by default.

1. **<u>Compromise the broadcast system.</u>**

In the event that the network is not large enough, i.e. with a reduced number of public nodes available who interacts with each other, or if an ill-intentioned person would have the financial means and technique to compromise the broadcast system, it might affect slightly or even greatly
this one.

Let us admit that the network in question consists of only 5 public nodes, owned by 5 individuals different, if an ill-intentioned person would like to influence the broadcast, he could of course run a larger number of public nodes and ignore any broadcast transactions.

Then to finalize this attack, it would suffice a hashrate larger than the average to compromise the broadcast of pending transactions and therefore “cancel” them from the taking into account of this on their target height blocks.
The fact remains that the attack in question is not infallible, an attempt can result from a Fork (virtual copy) and the nodes in question held by the ill-intentioned person would be found outside the network. This attack can also compromise the broadcast of mining shares, if the majority owned by this no one ignores them, and only takes into account their own, this could influence the priority of
progress of the blockchain, obviously, this must be done quickly enough with a consequent hashrate to get there.

2. **<u>Compromise the synchronization system.</u>**

In the same case previously explained, a network that is not large enough could be the target of this kind of attack.
If a malicious person wishes to compromise the synchronization of newcomers, that is to say to From a point lower than that currently available on the network, it is possible to synchronize another blockchain, given that this is the new majority available.

This only works if the user or the node in synchronization is found at a point of synchronization much lower than the current one.

VII.Presentations of available tools.

Here is the list of available tools, they are developed using the SeguraChain-Lib.dll reference.

This reference contains a set of class, allowing to generate the necessary database, to execute memory management, running an internal synchronization node instance and many other class.

Note that the tools presented will evolve, that their appearances may change over time, development, updates in the future.
All tools carry the name of Xiropht, since it will be updated by all of everything these tools presented.

Xiropht - Desktop Wallet:

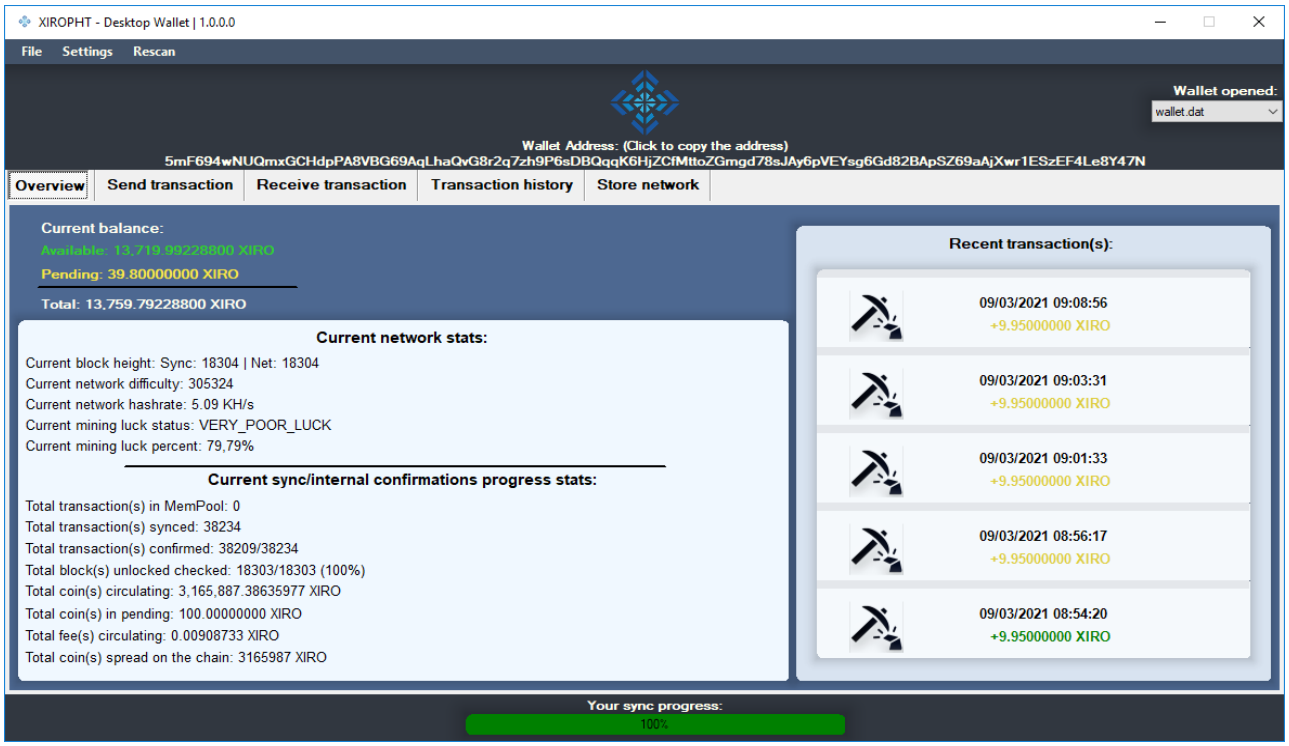
This is the default Desktop Wallet provided to users, it runs within it, an instance of a node of synchronization, configured as being closed from the outside, therefore without participation in the network.

It is possible to configure it to make its synchronization instance accessible to the public for participate in the network, if the user's bandwidth obviously allows it.

It is also possible to configure the Desktop Wallet to connect to an external or hosted Node in local, however it is recommended to always use your own Synchronization Node.

The design has been programmed with customized **Winform** controls, some graphic effects such as rounded edges on the panels, displayed bitmaps, as well as shadow effects have been programmed and displayed with **Winform's Default Graphics Engine, GDI +.**

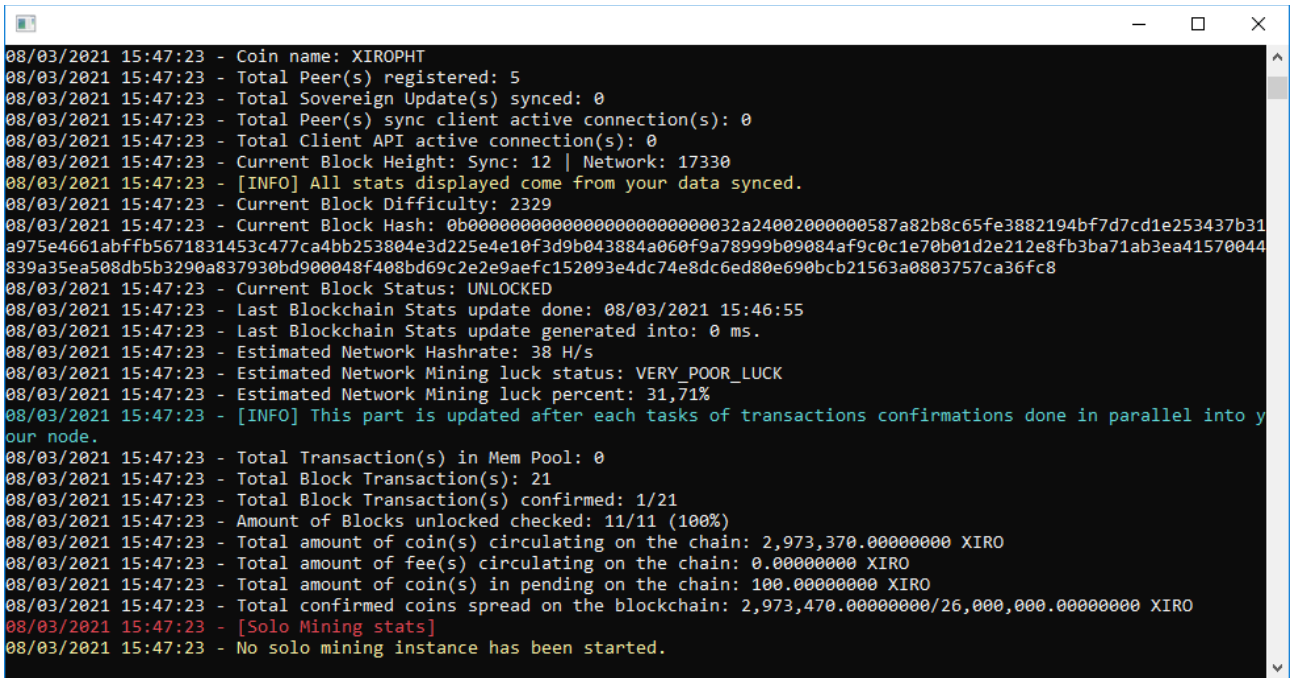
The transaction histories displayed are also virtual instances created completely with Rectangles, Bitmap, Graphics and math calculations.
Instances of transaction histories are stored in memory and updated in the background.



Xiropht – Peer :

This is a tool running the synchronization node instance, it allows you to synchronize the blockchain, solo mining, linked to external tools such as Desktop / RPC Wallet, participate to the network by configuring it as a public node.

It is the essential tool to operate the decentralized network.



Xiropht – Solo Miner :

The Solo Miner, allows to participate in the network by carrying out the mining process to unlock blocks, each blocks successfully unlocked gives a certain number of cryptocurrency, depending on the configuration established and the
on-going progress of the Blockchain.

This tool is essential to advance the Blockchain, support pending transactions, plus a number of different miners participate, the more the amount of estimated hashrate increases, the stronger the network will be against potential attacks set out in Chapter VI.