:CORRECT-SENTENCE-STRUCTURE-COMMUNICATION-PARSE-SYNTAX-GRAMMAR-DOCUMENT-CONTRACT-POSTAL-COURT-FLAG.

# PTTP CRYPTO SEAL ECOSYSTEM

# Φ

# WHITE PAPERS



PTTP Crypto Seal (PTTP)

White papers by:

2017 Michel Kuipers

## 1 Introduction

In the year 2009 Satoshi Nakamoto introduced Bitcoin as a new digital currency that is secured by encrypted signatures, decentralized and anonymous. It would ensure fast transactions, world width at very low fees, creating a competitive money system compared to centralized banking systems.

After eight years we can evaluate the long-term implications Bitcoin's design has in practical use and derive the imperfections from it to determine what is necessary to redefine the fundamental needs to create the perfect Crypto Currency.

We will define the solutions for every imperfection we'll find and in this manner create PTTP Crypto Seal (PTTP) from scratch to suggest it to the Ecosystem as a new big player in the Crypto (Value) Currency world.

## 2 Preconditions

A crypto currency should respect certain conditions which defines the necessity and therefore right of its existence. To determine these conditions we need to look at the ideas that originally have created the existence of Bitcoin.

The problem the banking systems face can be found in numerous observations:

a Security. Your money is protected by your identity. Anyone who gains access to your personal data can access your bank-account.
b Speed. Bank systems have a delay sending money between each others, controlled by a correspondent banking system, like for international transactions, SWIFT.
Especially in weekends this can be a bother.
c Trust. Banks are their own entities, which you lend your money to reclaim it, but the banks actually own your money and can relend it, invest it, or choose to do with it as they please, creating an insecure

system where you are dependent of the acting of the participants of the banking system, which can be quite insane watching the movie 'The Wolf of Wall Street'.

d Costs. Bank transfers, especially through credit cards, are quite expensive. You even pay for depositing fiat money to make it digital. You pay for your card, and you pay for your account.

So our new Crypto should respect the facts that it satisfies the conditions to be secure, fast, and trustworthy and make transactions possible at very low costs.

We cannot escape the fact that all transactions should somehow be collected into a ledger, which should be protected at such a way that not a single bit in this ledger ever may be changed to keep it valid.

We will now look at the original design and at each point determine if the practical outcome meets the given standards of the principles our new Crypto Currency should comply to.

## 3 Evaluation

At the base the original design of crypto currency is very secure. There are multiple levels of insurance, considering the ledger, transactions and ownership. Data is secured by hashing it under the SHA256 algorithm twice, avoiding rainbow table attacks, and which also forms a searchable ID to find the transaction in the ledger. The ledger consists of blocks of transactions; each pointing to the previous one, keeping a consistent chain of validated data, the transactions in a block is signed using Merkle trees to ensure they are original, validated and unmodified in the block. The ownership of claimed value (seal) is checked by the Elliptic Curve algorithm Secp256k1 which creates unique digital signatures out of a private key which can be validated with a public key but there is no way to create a signature without the private key. The public keys form wallets-addresses which are used as bank-accounts to claim outstanding (unclaimed) value (seal).

In the original design we have miners and nodes connected in a distributed core-system keeping an individual copy of the ledger and verifying the ledger is correct by taking the longest block chain in time. This design made it necessary to make it unlikely to add blocks at the same time. To establish the goal of dividing blocks in time, a mathematical challenge was invented which must be solved to add the block to a ledger. If a miner solves this problem, the block will be added to the block chain and the miner is

rewarded by newly created coins, which brings the supply of the coin into existence. By adjusting the difficulty in time Satoshi could guarantee every node in the system would eventually receive a block in a predetermined averaged interval. In the case two chains were formed, the longest would eventually win, and every transaction in so called Orphan blocks would be refilled into the outstanding transactions. The miners collect and verify transactions to fill a block which they generate and sign, then this block is distributed in the core-network and everyone is trying to solve the mining-process which consists of adding some data, called a nonce, to make the hash begin with a certain amount of zeros.

## 4 Analytics

Although the current block chain technology is well protected we have to look at the lowest level possible to find its imperfections. When we search for a certain transaction, it is stored in a block, so we have to search the block for the transaction, which is speeded up by using Merkle trees. After this we can validate a transaction is valid. To create a transaction we will have to compare which transactions contain unspent value (seal), these are part of transactions in the blocks that contain the wallet of the receiver, and the valid signature of the sender which have no following block in the ledger, where the receiver has, at his turn, has spent this value (seal). Likewise we have to walk through the whole ledger from the beginning to see if value (seal) is spendable or not, which takes up a really long time if not implemented wisely. In most crypto currency implementations the wallet-software calls a known API dedicated to provide spendable blocks to create a transaction.

To avoid inflation, Satoshi reasoned that there only could be a finite amount of Bitcoin. Since the miners create the value (seal) out of nothing by creating and validating the block chain, the miners receive less and less reward in time. The specifics can be found here:
https://en.bitcoin.it/wiki/Controlled_supply

Now the problem arises that it takes up a lot of effort and therefore energy (electric power) to earn newly created Bitcoin, and the consequence may be people stop mining. The solution the core-developers suggested is that transactions have to pay more fees to the miners, who collect the transactions in the first place, making it an actual problem some transactions are delayed a long time before being integrated into the block chain. This has the implicit danger of raising the fees on a transaction before

even being considered to be integrated into the block chain. Another problem is a block is limited to 1 Megabytes of data, so if the miners are limited in time to be able to solve blocks to add them to the ledger, there may be more blocks then there could be mined in time, delaying transactions. To verify a miner has mined a valid block, it has to wait 100 blocks to be added to the block chain to spend their earned value (seal).

Since Bitcoin is a minable coin, more and more people started to mine, and where you used to be able to mine with CPU and GPU, now you can only effectively mine with dedicated hardware devices called ASIC. A few companies, especially based in China, have built such large mining farms of ASIC devices; they actually control the creation of blocks into the system. This give them an advantage over the future developments of Bitcoin, which have already lead to many hard-forks like B2X, BCH and BTG, in which case a part of the network developers insisted that Bitcoin should stay decentralized at its principle values, and speed up transactions under lower fees, therefore decided to split up from the centralized core.

## 5 Conciliation

To solve the defined imperfections current Crypto Currency faces we must not be afraid to redefine the fundamental base of which it consist.

- A transaction is a transfer of spendable value (seal) sent to a wallet that is property of anyone who has access to it by signing it with their private key. The recipient can spend the value (seal), which corresponds to its wallet, by claiming it, through singing it with its private key. Anyone can verify this transaction by verifying the signature corresponds to the public key and wallet that has spent the value (seal).
- A ledger is a collection of transactions validated by a hashing verification and signed by the spender.
- A miner creates new (called coinbase) value (seal) to add to the circulation supply by verifying mathematical problems biased on a certain time-interval.
- The system at any point has to validate the transactions are valid and are not already spent on the ledger, and does not invalidate pending transactions. Illegal transactions should be ignored and deleted.
- Fees must be low, always, to buy anything without extra costs.
- The time it costs for transactions to be included in the ledger must be as short as possible. This determines the time the receiver can spend

the value (seal) again. Also miners should be able to spend their deserved coinbase value (seal) as quick as possible.
- The core network is a decentralized collection of interconnected nodes, validating the ledger's accuracy, where every node should have the same abilities as any other in the whole network, making it impossible for anyone to gain power over the whole.
- Every node should be able to individually act as an API for wallet-applications that want to create a transaction and send it. The node should respond with appropriate information.

## 6 Inspiration

A transaction basically consists of a collection of spendable blocks, that are gathered together, validating they come from the same wallet that spent the value (seal), signed by the owner of this wallet, to give to wallets, connected to public keys, who can claim this value (seal) again by signing it through their public and private keys. To validate a transaction we need to make sure the wallets are corresponding with the signing public key, the signature and the amount spent. To create a transaction we have to find all spendable blocks corresponding to a wallet and see if the amount spendable is enough for the transaction to be created.

The ledger can connect these transactions at any manner, as long as all decentralized nodes agree on the order. So we need to build in a security that guarantees the order in which transactions will appear in the ledger, and exclude any disconsent of cores to a consensus the absolute majority of nodes have agreed upon.

We absolutely must ensure we use the finest, best, fastest encryption possible keeping an eye on Moore's law, hackers and all development in the future.

Since a miner's basic function is to create new coinbase value (seal) in time, the question arises if the core-nodes or the miners should validate and collect transactions into blocks, or even use a block chain to collect transactions, or just add transaction to a ledger. The function of nodes and miners may be redesigned from scratch.

Since at any point the actual process that writes data into the ledger, it would be a wise choice to let the core-nodes validate the transactions,

raising the questions why miners should too, since their main goal is to create coinbase coins.

Every node in the network should have the exact possibilities of any other node in the network, there must never be a main node controlling other nodes.

No application, we call a leaf, like a wallet, may ever control a node. Should miners control a node?

Core-nodes are able to keep a list of transactions that are not yet added to the ledger, but can validate that they will be at some point in time by comparing these 'pending transactions' to the list of to be processed transactions. This makes it possible to have very quick conformations in stores when buying your bread, where the store-holder is secured it will receive its value (seal) to spend again very quickly.

When somebody wants to make a transaction it should be very easy to contact the PTTP-network and get the necessary information to create and verify a transaction. This never should be done by a centralized API, but every node in the core-network should be able to make a transaction by exploring the ledger to search for spendable blocks corresponding to the spending wallet of the transaction.

When the supply of the coin is limited, the miners will no longer be interested in mining the coin; therefore the fees of the transactions must rise to provide miners their rewards. This is in contrast to our principle goal, so there must be an unlimited amount of coins to be mined, to control this, the difficulty of the to be mined problem must vary, to stay in synchronization with the amount of coins we want to create, which should be rather little then more, to not overflow demand in time and in time keep the value of PTTP stable.

The processes which actually validate the transactions should be rewarded for it. This should be a very quick process.

## 7 Creation

The main problem that can be deduced is that although it takes some time to find a transaction to be validated, because they are collected in blocks, it takes a hell of a time to find all spendable blocks a wallet may claim. Since the ledger is huge and only huge databases determine such answers quickly,

most wallets use an API to generate their spendable blocks. Our design asks for a system where any node can perform this task.

The only way to solve this is to split the spendable blocks into separate entities we can mark as being spent or spendable. By doing this we separate the not-spendable blocks, we can use to identify the spendable blocks, so we call these Ident-blocks, which can be input-blocks for transactions, or coinbase blocks. Also form a logical base for the genesis block to initiate the whole system.

Since miners are mining hashes at very high electric costs, and we want to give them a meaningful task, it would be advisable to derive the miners of the task of validating and collecting transactions into blocks and give this role to the nodes, where the main task of miners will be to create new PTTP-coins by calculating random given problems to solve, which can actually benefit fields of research.

Since nodes now fulfil the task of keeping the ledger intact and validating the transaction, it's fair to give everybody who is running a node, a share of all profit, which is translated into fees which everybody must pay to make a transaction. This guarantees the network is always alive and decentralized, because there will always be running nodes wanting to earn some value (seal), just like miners will always want to earn coinbase PTTP.

Since we want the transactions to be cheap we must allow a standard minimum fee of 0.5% (in the design we use units of 0,01%) to be paid to all participating core-nodes, to stimulate the willingness to participate in the PTTP-project. You should never have to pay a higher fee. Why higher fees are possible is very simple: the core-network will earn more value (seal), which pleases the whole network, which will protect your value (seal) and allow it to be transferred very quickly, but also a higher fee means your transaction will be added to the ledger more quickly, enabling the receiver to verify that they can spend your spent value (seal), building more trust between you as sender of the value (seal) and the receiver of the value (seal), so trust is worth the value (seal) of the fee.

# 8 Implementation

**For all the technical detail we refer you to the documentation pages on GitHub and the original Perl source-code.**

**Below you find the base ledger structure, and descriptions of the basic works of the system, and the mining difficulty table.**

## 8.a LEDGER

**Ledger Blocks**

```
#################################################################
# Ledger-blocks: (all big-endian)
# offset  length  content
# ===============================================================
# All
#
#     0      4  difference of position of the previous transaction in the chain compared to this position
#              (equal to length previous block)
#     4      4  difference of position of the next transaction in the chain compared to this position
#              (equal to length block)
#     8     64  transaction id (next fields from 'number' on, secure-hashed (sha256(sha512(data)))
#    72     64  cumulative hash of all transaction in-id's and coinbase id's
#   136     12  transaction number in chain (0=genesis) 48 bit
#   148      4  version
#   152      1  type
#   153     64  previous id (any transaction type)
#
# ===============================================================
# In / Genesis / Coinbase
#
#   217      8  time, epoch is 00:00:00 UTC, January 1, 1970 (UNIX)
#   225      2  number of out addresses in 8 bit ( = 1 for genesis, = 2 for coinbase)
#
# ===============================================================
# In
#
#   227      2  number of in addresses
#   229     64  public key
#   293    128  signature
#   421   64num  list of id's of collected out-transactions to form in-addresses
#
# ===============================================================
# Fee
#
#   217      8  time, epoch is 00:00:00 UTC, January 1, 1970 (UNIX)
#   225      4  number of out addresses in 16 bit
#   229      8  Unpayed Reserved Amount in 32 bit (100000000 = 1 PTTP)
#   237     12  last transaction number of collected out-blocks for the fee transactions
#   249    128  signature only (128) PTTP Service Public Key is hardcoded into the source
#
# ===============================================================
```

```
# ============================================================
# Out
#
#   217     68  PTTP-wallet address
#   285     16  Amount in 64 bit (100000000 = 1 PTTP)
#   301      4  fee = 0 for coinbase and genesis, minimum = 50 = 0,5% maximum = 10000 (100%)
#   305     10  lock expire time * Optional block for a Timed release of spendable out-blocks
#
# ============================================================
# All         1  block identifier 'z'
#
#   227        Genesis / Coinbase
#   377        Fee
#   Xin        In
#   305        Out
#   315        Out with lock expire time
#
# ============================================================
#
# CAREFUL: Ledger will end with a pointer to the last block, which will be the beginning of the next block,
#         thus forming a double linked list to search from the beginning or from behind!
#
##############################################################################
```

### a.i LEDGER-SEAL Structure & Types

The Ledger-Seal structure contains either a genesis-in-block, a coinbase-in-block, a fee-in-block or a transaction-in-block, with one or more out-blocks depending on the type of in-block.

## a.ii GENESIS-TRANSACTION-SEAL

The original Genesis-block creating the Ledger-Crypto-Seal-Chain origin. The Genesis Transaction Seal consists out of two blocks.

The first block is the Genesis-Block, the second block is an Out-Block and it ends after the first Next-Transaction-Seal block item 'prev' pointer to the last out-block of the current Genesis Transaction Seal.

### Genesis Transaction Seal

#### Genesis Block

| *Content* | *Offset* | *Length* |
|---|---|---|
| **prev** | 0 | 4 |
| **next** | 4 | 4 |
| **transaction id** | 8 | 64 |
| **cumulative hash** | 72 | 64 |
| **transaction number** | 136 | 12 |
| **version** | 148 | 4 |
| **block type** | 152 | 1 |
| **previous id** | 153 | 64 |
| **pttptime** | 217 | 8 |
| **number of out addresses** | 225 | 2 |
| **block identifier** | 227 | 1 |

## Out Block

| | | |
|---|---|---|
| **prev** | 0 | 4 |
| **next** | 4 | 4 |
| **transaction id** | 8 | 64 |
| **cumulative hash** | 72 | 64 |
| **transaction number** | 136 | 12 |
| **version** | 148 | 4 |
| **block type** | 152 | 1 |
| **previous id** | 153 | 64 |
| **PTTP-wallet address** | 217 | 68 |
| **Amount** | 285 | 16 |
| **Fee** | 301 | 4 |
| **Time * Optional** | 305 | 10 |
| **block identifier** | 305\|315 | 1 |

## Next Transaction Seal

| | | |
|---|---|---|
| **prev** | 306\|316 | 4 |

### a.iii COINBASE-SEAL

Coinbase blocks created out of the Winning Miners who mine the coinbase coins from the Coinbase-Release-Service every 5 minutes. The Coinbase Transaction Seal consists out of three blocks.

The first block is the Coinbase-Block, the second block is the Miner-Out-Block, the third block is the Node-Out-Block, and it ends after the first Next-Transaction-Seal block item 'prev' pointer to the last out-block of the current Coinbase Transaction Seal.

**Coinbase Transaction Seal**

**Coinbase**

| *Content* | *Offset* | *Length* |
|---|---|---|
| **prev** | 0 | 4 |
| **next** | 4 | 4 |
| **transaction id** | 8 | 64 |
| **cumulative hash** | 72 | 64 |
| **transaction number** | 136 | 12 |
| **version** | 148 | 4 |
| **block type** | 152 | 1 |
| **previous id** | 153 | 64 |
| **pttptime** | 217 | 8 |
| **number of out addresses** | 225 | 2 |
| **block identifier** | 227 | 1 |

## Miner Out Block *(10.00000000 PTTP)*

| | | |
|---|---|---|
| **prev** | 0 | 4 |
| **next** | 4 | 4 |
| **transaction id** | 8 | 64 |
| **cumulative hash** | 72 | 64 |
| **transaction number** | 136 | 12 |
| **version** | 148 | 4 |
| **block type** | 152 | 1 |
| **previous id** | 153 | 64 |
| **PTTP-wallet address** | 217 | 68 |
| **Amount** | 285 | 16 |
| **Fee** | 301 | 4 |
| **Time * Optional** | 305 | 10 |
| **block identifier** | 305\|315 | 1 |

**Node Out Block** *(0.50000000 PTTP)*

| | | |
|---|---|---|
| **prev** | 0 | 4 |
| **next** | 4 | 4 |
| **transaction id** | 8 | 64 |
| **cumulative hash** | 72 | 64 |
| **transaction number** | 136 | 12 |
| **version** | 148 | 4 |
| **block type** | 152 | 1 |
| **previous id** | 153 | 64 |
| **PTTP-wallet address** | 217 | 68 |
| **Amount** | 285 | 16 |
| **Fee** | 301 | 4 |
| **Time * Optional** | 305 | 10 |
| **block identifier** | 305\|315 | 1 |

**Next Transaction Seal**

| | | |
|---|---|---|
| **prev** | 306\|316 | 4 |

### a.iv NODE-FEE-SEAL

All nodes who actively process the Transactions-Seals within the active-online-core-node-network are rewarded once a week with a shared amount of the total fee processed at every Sunday night at 12 o clock of that week. The Fee Transaction Seal consists out of minimal 2 and up to 65535 blocks.

The first block is the Fee-Block, the second and all other blocks are the Out-Blocks, and it ends after the first Next-Transaction-Seal block item 'prev' pointer to the last out-block of the current Fee Transaction Seal.

## Fee Transaction Seal

### Fee Block

| *Content* | *Offset* | *Length* |
|---|---|---|
| **prev** | 0 | 4 |
| **next** | 4 | 4 |
| **transaction id** | 8 | 64 |
| **cumulative hash** | 72 | 64 |
| **transaction number** | 136 | 12 |
| **version** | 148 | 4 |
| **block type** | 152 | 1 |
| **previous id** | 153 | 64 |
| **pttptime** | 217 | 8 |
| **number of out addresses** | 225 | 4 |
| **unpayed amount** | 229 | 8 |
| **last transaction number** | 237 | 12 |
| **signature** | 249 | 128 |
| **block identifier** | 377 | 1 |

**… up to 65535 transaction out-blocks …**

**Out Block X**

| | | |
|---|---|---|
| **prev** | 0 | 4 |
| **next** | 4 | 4 |
| **transaction id** | 8 | 64 |
| **cumulative hash** | 72 | 64 |
| **transaction number** | 136 | 12 |
| **version** | 148 | 4 |
| **block type** | 152 | 1 |
| **previous id** | 153 | 64 |
| **PTTP-wallet address** | 217 | 68 |
| **Amount** | 285 | 16 |
| **Fee** | 301 | 4 |
| **Time * Optional** | 305 | 10 |
| **block identifier** | 305\|315 | 1 |

**Next Transaction Seal**

| | | |
|---|---|---|
| **prev** | 306\|316 | 4 |

## a.v TRANSACTION-SEAL

Transaction-Seals are created by the wallets who spend their available coins to an other wallet address. The Wallet Transaction Seal consists out of minimal 2 and up to 256 blocks.

The first block is the In-Block, the second and all other blocks are the Out-Blocks, and it ends after the first Next-Transaction-Seal block item 'prev' pointer to the last out-block of the current Wallet Transaction Seal.

## Wallet Transaction Seal

**In Block**

| *Content* | *Offset* | *Length* |
|---|---|---|
| **prev** | 0 | 4 |
| **next** | 4 | 4 |
| **transaction id** | 8 | 64 |
| **cumulative hash** | 72 | 64 |
| **transaction number** | 136 | 12 |
| **version** | 148 | 4 |
| **block type** | 152 | 1 |
| **previous id** | 153 | 64 |
| **pttptime** | 217 | 8 |
| **number of out addresses** | 225 | 2 |
| **number of in addresses** | 227 | 2 |
| **public key** | 229 | 64 |
| **signature** | 293 | 128 |
| **list of transaction id's** | 421 | 64Xin |
| **block identifier** | Xin | 1 |

**... up to 255 transaction out-blocks,**
**(including possible change-wallet-address) ...**

**Out Block X**

| | | |
|---|---|---|
| **prev** | 0 | 4 |
| **next** | 4 | 4 |
| **transaction id** | 8 | 64 |
| **cumulative hash** | 72 | 64 |
| **transaction number** | 136 | 12 |
| **version** | 148 | 4 |
| **block type** | 152 | 1 |
| **previous id** | 153 | 64 |
| **PTTP-wallet address** | 217 | 68 |
| **Amount** | 285 | 16 |
| **Fee** | 301 | 4 |
| **Time * Optional** | 305 | 10 |
| **block identifier** | 305\|315 | 1 |

**Next Transaction Seal**

| | | |
|---|---|---|
| **prev** | 306\|316 | 4 |

## 8.b COINBASE

Since the riddle for the coinbase release has to be a secret and is practically not secure to share thru a decentralized network as every part of that can and will be used for its vulnerability, we have a single coinbase service as central part of the node-list service thru where nodes, explorers, wallets, and miners find their node they need to connect to with their particular functionality with in the connected core node network.

### b.i COINBASE-RELEASE

PTTP starts with a coinbase release of 10 coins every 5 minutes, with a node-fee of 0.5 coin what calculates to 1.103.760 PTTP a year, which will grow in amount until the total minting release is mined, which is set to end in 2034, so as those who step in at a later point still make enough change to cache a coin and get into the crypto liquidity system to exchange their added value (seal) with.

### b.ii COINBASE-MINING

Miners, who can be a pool as well, making them more powerful, connect to the Node they find in the node-list from the node-list-service, to get the current riddle for the round to be found and returned thru an encrypted route so that only the coinbase-service itself can check your solution for the riddle and a node can't steal your winning solution.

### b.iii COINBASE-MINING-DIFFICULTY-TABLE

**Mining Difficultie table**

**FACTORIAL A 1**
fulldiff = 1

**FACTORIAL B 2**
fulldiff = 2

**FACTORIAL C 3**
hints 2 = 4
fulldiff = 6

**FACTORIAL D 4**
hints 2 = 12
hints 3 = 18
fulldiff = 24

**FACTORIAL E 5**
hints 2 = 48
hints 3 = 72
hints 4 = 96
fulldiff = 120

**FACTORIAL F 6**
hints 2 = 240
hints 3 = 360
hints 4 = 480
hints 5 = 600
fulldiff = 720

**FACTORIAL G 7**
hints 2 = 1,440
hints 3 = 2,160
hints 4 = 2,880
hints 5 = 3,600
hints 6 = 4,320
fulldiff = 5,040

**FACTORIAL H 8**
hints 2 = 10,080
hints 3 = 15,120
hints 4 = 20,160
hints 5 = 25,200
hints 6 = 30,240
hints 7 = 35,280
fulldiff = 40,320

**FACTORIAL I 9**
hints 2 = 80,640

**FACTORIAL J 10**
hints 2 = 725,760

hints  3 = 120,960
hints  4 = 161,280
hints  5 = 201,600
hints  6 = 241,920
hints  7 = 282,240
hints  8 = 322,560
fulldiff = 362,880

hints  3 = 1,088,640
hints  4 = 1,451,520
hints  5 = 1,814,400
hints  6 = 2,177,280
hints  7 = 2,540,160
hints  8 = 2,903,040
hints  9 = 3,265,920
fulldiff = 3,628,800

**FACTORIAL K 11**

hints  2 =  7,257,600
hints  3 = 10,886,400
hints  4 = 14,515,200
hints  5 = 18,144,000
hints  6 = 21,772,800
hints  7 = 25,401,600
hints  8 = 29,030,400
hints  9 = 32,659,200
hints 10 = 36,288,000
fulldiff = 39,916,800

**FACTORIAL L 12**

hints  2 =  79,833,600
hints  3 = 119,750,400
hints  4 = 159,667,200
hints  5 = 199,584,000
hints  6 = 239,500,800
hints  7 = 279,417,600
hints  8 = 319,334,400
hints  9 = 359,251,200
hints 10 = 399,168,000
hints 11 = 439,084,800
fulldiff = 479,001,600

**FACTORIAL M 13**

hints  2 =  958,003,200
hints  3 = 1,437,004,800
hints  4 = 1,916,006,400
hints  5 = 2,395,008,000
hints  6 = 2,874,009,600
hints  7 = 3,353,011,200
hints  8 = 3,832,012,800
hints  9 = 4,311,014,400
hints 10 = 4,790,016,000
hints 11 = 5,269,017,600
hints 12 = 5,748,019,200
fulldiff = 6,227,020,800

**FACTORIAL N 14**

hints  2 = 12,454,041,600
hints  3 = 18,681,062,400
hints  4 = 24,908,083,200
hints  5 = 31,135,104,000
hints  6 = 37,362,124,800
hints  7 = 43,589,145,600
hints  8 = 49,816,166,400
hints  9 = 56,043,187,200
hints 10 = 62,270,208,000
hints 11 = 68,497,228,800
hints 12 = 74,724,249,600
hints 13 = 80,951,270,400
fulldiff = 87,178,291,200

**FACTORIAL O 15**

hints  2 =  174,356,582,400
hints  3 =  261,534,873,600
hints  4 =  348,713,164,800
hints  5 =  435,891,456,000
hints  6 =  523,069,747,200
hints  7 =  610,248,038,400
hints  8 =  697,426,329,600
hints  9 =  784,604,620,800
hints 10 =  871,782,912,000
hints 11 =  958,961,203,200
hints 12 = 1,046,139,494,400
hints 13 = 1,133,317,785,600
hints 14 = 1,220,496,076,800
fulldiff = 1,307,674,368,000

**FACTORIAL P 16**

hints  2 =  2,615,348,736,000
hints  3 =  3,923,023,104,000
hints  4 =  5,230,697,472,000
hints  5 =  6,538,371,840,000
hints  6 =  7,846,046,208,000
hints  7 =  9,153,720,576,000
hints  8 = 10,461,394,944,000
hints  9 = 11,769,069,312,000
hints 10 = 13,076,743,680,000
hints 11 = 14,384,418,048,000
hints 12 = 15,692,092,416,000
hints 13 = 16,999,766,784,000
hints 14 = 18,307,441,152,000
hints 15 = 19,615,115,520,000
fulldiff = 20,922,789,888,000

**FACTORIAL Q 17**

hints  2 =  41,845,579,776,000

**FACTORIAL R 18**

hints  2 =  711,374,856,192,000

hints 3 = 62,768,369,664,000
hints 4 = 83,691,159,552,000
hints 5 = 104,613,949,440,000
hints 6 = 125,536,739,328,000
hints 7 = 146,459,529,216,000
hints 8 = 167,382,319,104,000
hints 9 = 188,305,108,992,000
hints 10 = 209,227,898,880,000
hints 11 = 230,150,688,768,000
hints 12 = 251,073,478,656,000
hints 13 = 271,996,268,544,000
hints 14 = 292,919,058,432,000
hints 15 = 313,841,848,320,000
hints 16 = 334,764,638,208,000
fulldiff = 355,687,428,096,000

hints 3 = 1,067,062,284,288,000
hints 4 = 1,422,749,712,384,000
hints 5 = 1,778,437,140,480,000
hints 6 = 2,134,124,568,576,000
hints 7 = 2,489,811,996,672,000
hints 8 = 2,845,499,424,768,000
hints 9 = 3,201,186,852,864,000
hints 10 = 3,556,874,280,960,000
hints 11 = 3,912,561,709,056,000
hints 12 = 4,268,249,137,152,000
hints 13 = 4,623,936,565,248,000
hints 14 = 4,979,623,993,344,000
hints 15 = 5,335,311,421,440,000
hints 16 = 5,690,998,849,536,000
hints 17 = 6,046,686,277,632,000
fulldiff = 6,402,373,705,728,000

**FACTORIAL S 19**

hints 2 = 12,804,747,411,456,000
hints 3 = 19,207,121,117,184,000
hints 4 = 25,609,494,822,912,000
hints 5 = 32,011,868,528,640,000
hints 6 = 38,414,242,234,368,000
hints 7 = 44,816,615,940,096,000
hints 8 = 51,218,989,645,824,000
hints 9 = 57,621,363,351,552,000
hints 10 = 64,023,737,057,280,000
hints 11 = 70,426,110,763,008,000
hints 12 = 76,828,484,468,736,000
hints 13 = 83,230,858,174,464,000
hints 14 = 89,633,231,880,192,000
hints 15 = 96,035,605,585,920,000
hints 16 = 102,437,979,291,648,000
hints 17 = 108,840,352,997,376,000
hints 18 = 115,242,726,703,104,000
fulldiff = 121,645,100,408,832,000

**FACTORIAL T 20**

hints 2 = 243,290,200,817,664,000
hints 3 = 364,935,301,226,496,000
hints 4 = 486,580,401,635,328,000
hints 5 = 608,225,502,044,160,000
hints 6 = 729,870,602,452,992,000
hints 7 = 851,515,702,861,824,000
hints 8 = 973,160,803,270,656,000
hints 9 = 1,094,805,903,679,488,000
hints 10 = 1,216,451,004,088,320,000
hints 11 = 1,338,096,104,497,152,000
hints 12 = 1,459,741,204,905,984,000
hints 13 = 1,581,386,305,314,816,000
hints 14 = 1,703,031,405,723,648,000
hints 15 = 1,824,676,506,132,480,000
hints 16 = 1,946,321,606,541,312,000
hints 17 = 2,067,966,706,950,144,000
hints 18 = 2,189,611,807,358,976,000
hints 19 = 2,311,256,907,767,808,000
fulldiff = 2,432,902,008,176,640,000

**FACTORIAL U 21**

hints 2 = 4,865,804,016,353,280,000
hints 3 = 7,298,706,024,529,920,000
hints 4 = 9,731,608,032,706,560,000
hints 5 = 12,164,510,040,883,200,000
hints 6 = 14,597,412,049,059,840,000
hints 7 = 17,030,314,057,236,480,000
hints 8 = 19,463,216,065,413,120,000
hints 9 = 21,896,118,073,589,760,000
hints 10 = 24,329,020,081,766,400,000
hints 11 = 26,761,922,089,943,040,000
hints 12 = 29,194,824,098,119,680,000
hints 13 = 31,627,726,106,296,320,000
hints 14 = 34,060,628,114,472,960,000
hints 15 = 36,493,530,122,649,600,000
hints 16 = 38,926,432,130,826,240,000
hints 17 = 41,359,334,139,002,880,000
hints 18 = 43,792,236,147,179,520,000
hints 19 = 46,225,138,155,356,160,000
hints 20 = 48,658,040,163,532,800,000
fulldiff = 51,090,942,171,709,440,000

**FACTORIAL V 22**

hints 2 = 102,181,884,343,418,880,000
hints 3 = 153,272,826,515,128,320,000
hints 4 = 204,363,768,686,837,760,000
hints 5 = 255,454,710,858,547,200,000
hints 6 = 306,545,653,030,256,640,000
hints 7 = 357,636,595,201,966,080,000
hints 8 = 408,727,537,373,675,520,000
hints 9 = 459,818,479,545,384,960,000
hints 10 = 510,909,421,717,094,400,000
hints 11 = 562,000,363,888,803,840,000
hints 12 = 613,091,306,060,513,280,000
hints 13 = 664,182,248,232,222,720,000
hints 14 = 715,273,190,403,932,160,000
hints 15 = 766,364,132,575,641,600,000
hints 16 = 817,455,074,747,351,040,000
hints 17 = 868,546,016,919,060,480,000
hints 18 = 919,636,959,090,769,920,000
hints 19 = 970,727,901,262,479,360,000
hints 20 = 1,021,818,843,434,188,800,000
hints 21 = 1,072,909,785,605,898,240,000

fulldiff = 1,124,000,727,777,607,680,000

**FACTORIAL W 23**

hints 2 = 2,248,001,455,555,215,360,000
hints 3 = 3,372,002,183,332,823,040,000
hints 4 = 4,496,002,911,110,430,720,000
hints 5 = 5,620,003,638,888,038,400,000
hints 6 = 6,744,004,366,665,646,080,000
hints 7 = 7,868,005,094,443,253,760,000
hints 8 = 8,992,005,822,220,861,440,000
hints 9 = 10,116,006,549,998,469,120,000
hints 10 = 11,240,007,277,776,076,800,000
hints 11 = 12,364,008,005,553,684,480,000
hints 12 = 13,488,008,733,331,292,160,000
hints 13 = 14,612,009,461,108,899,840,000
hints 14 = 15,736,010,188,886,507,520,000
hints 15 = 16,860,010,916,664,115,200,000
hints 16 = 17,984,011,644,441,722,880,000
hints 17 = 19,108,012,372,219,330,560,000
hints 18 = 20,232,013,099,996,938,240,000
hints 19 = 21,356,013,827,774,545,920,000
hints 20 = 22,480,014,555,552,153,600,000
hints 21 = 23,604,015,283,329,761,280,000
hints 22 = 24,728,016,011,107,368,960,000
fulldiff = 25,852,016,738,884,976,640,000

**FACTORIAL X 24**

hints 2 = 51,704,033,477,769,953,280,000
hints 3 = 77,556,050,216,654,929,920,000
hints 4 = 103,408,066,955,539,906,560,000
hints 5 = 129,260,083,694,424,883,200,000
hints 6 = 155,112,100,433,309,859,840,000
hints 7 = 180,964,117,172,194,836,480,000
hints 8 = 206,816,133,911,079,813,120,000
hints 9 = 232,668,150,649,964,789,760,000
hints 10 = 258,520,167,388,849,766,400,000
hints 11 = 284,372,184,127,734,743,040,000
hints 12 = 310,224,200,866,619,719,680,000
hints 13 = 336,076,217,605,504,696,320,000
hints 14 = 361,928,234,344,389,672,960,000
hints 15 = 387,780,251,083,274,649,600,000
hints 16 = 413,632,267,822,159,626,240,000
hints 17 = 439,484,284,561,044,602,880,000
hints 18 = 465,336,301,299,929,579,520,000
hints 19 = 491,188,318,038,814,556,160,000
hints 20 = 517,040,334,777,699,532,800,000
hints 21 = 542,892,351,516,584,509,440,000
hints 22 = 568,744,368,255,469,486,080,000
hints 23 = 594,596,384,994,354,462,720,000
fulldiff = 620,448,401,733,239,439,360,000

**FACTORIAL Y 25**

hints 2 = 1,240,896,803,466,478,878,720,000
hints 3 = 1,861,345,205,199,718,318,080,000
hints 4 = 2,481,793,606,932,957,757,440,000
hints 5 = 3,102,242,008,666,197,196,800,000
hints 6 = 3,722,690,410,399,436,636,160,000
hints 7 = 4,343,138,812,132,676,075,520,000
hints 8 = 4,963,587,213,865,915,514,880,000
hints 9 = 5,584,035,615,599,154,954,240,000
hints 10 = 6,204,484,017,332,394,393,600,000
hints 11 = 6,824,932,419,065,633,832,960,000
hints 12 = 7,445,380,820,798,873,272,320,000
hints 13 = 8,065,829,222,532,112,711,680,000
hints 14 = 8,686,277,624,265,352,151,040,000
hints 15 = 9,306,726,025,998,591,590,400,000
hints 16 = 9,927,174,427,731,831,029,760,000
hints 17 = 10,547,622,829,465,070,469,120,000
hints 18 = 11,168,071,231,198,309,908,480,000
hints 19 = 11,788,519,632,931,549,347,840,000
hints 20 = 12,408,968,034,664,788,787,200,000
hints 21 = 13,029,416,436,398,028,226,560,000
hints 22 = 13,649,864,838,131,267,665,920,000
hints 23 = 14,270,313,239,864,507,105,280,000
hints 24 = 14,890,761,641,597,746,544,640,000
fulldiff = 15,511,210,043,330,985,984,000,000

**FACTORIAL Z 26**

hints 2 = 31,022,420,086,661,971,968,000,000
hints 3 = 46,533,630,129,992,957,952,000,000
hints 4 = 62,044,840,173,323,943,936,000,000
hints 5 = 77,556,050,216,654,929,920,000,000
hints 6 = 93,067,260,259,985,915,904,000,000
hints 7 = 108,578,470,303,316,901,888,000,000
hints 8 = 124,089,680,346,647,887,872,000,000
hints 9 = 139,600,890,389,978,873,856,000,000
hints 10 = 155,112,100,433,309,859,840,000,000
hints 11 = 170,623,310,476,640,845,824,000,000
hints 12 = 186,134,520,519,971,831,808,000,000
hints 13 = 201,645,730,563,302,817,792,000,000
hints 14 = 217,156,940,606,633,803,776,000,000
hints 15 = 232,668,150,649,964,789,760,000,000
hints 16 = 248,179,360,693,295,775,744,000,000
hints 17 = 263,690,570,736,626,761,728,000,000
hints 18 = 279,201,780,779,957,747,712,000,000
hints 19 = 294,712,990,823,288,733,696,000,000
hints 20 = 310,224,200,866,619,719,680,000,000
hints 21 = 325,735,410,909,950,705,664,000,000
hints 22 = 341,246,620,953,281,691,648,000,000
hints 23 = 356,757,830,996,612,677,632,000,000
hints 24 = 372,269,041,039,943,663,616,000,000
hints 25 = 387,780,251,083,274,649,600,000,000
fulldiff = 403,291,461,126,605,635,584,000,000

## 8.c NODES

Nodes, who get thru an online check thru the node-list-service and who are having a fully synced ledger, join the active-online-core-node-list to be reached by other new online nodes, and explorers, wallets and miners, to process their transaction activities, and vote on the core-process of the first-in-line transaction-blocks to be added newly to the ledger. The Node manages their own fast-memory-databases to validate the available in-blocks not yet spend to be spend for transactions and validates for any valid transaction sent to the node, and either processes it as a spendable amount into a transaction-block and sends it thru the network of core nodes with a confirmed processed transaction event, or sends an error on return when a transaction isn't valid or has problems spending too much in-blocks at ones due to the current transaction-in-block-design with only 256 in-block max, new types for bigger transaction would not be necessary first as multiple transactions breaking up the main amount to be spend, can solve the problem on the return of an available amount to spend (spendable) which is used to create multiple transactions to complete a bigger amount to be spend, instead of one transaction, yet this process is done in the wallet-software layer. In a later stage at core version upgrade there will be other types of in-block as a multi-in-block available in where we can join the whole transaction in a single ledger seal block. Not only the nodes, but also the wallets will have a need for an version upgrade for use of the new and extra functionality. All the transaction activity is further completely decentralized on a neural-node-network of individual nodes who validate every transaction to be processed in as short as time possible, and the a whole network can validate transactions in a short as possible time to serve all wallet transactions and digital services, and those nodes who do not follow the core-consensus will get de-synced from the core and/or fork with invalid or other choices than the core consensus, so that they have to re-sync to the core-ledger-consensus to get back in again, so that also those not keeping up can fallback to a re-sync process and only those with the exact signature of the longest valid ledger keep in consensus on a decentralized shared and correct ledger.

### c.i NODELIST-SERVICE

The node-list-service is a distributed shared network of closest node-list-servers directly connected to the coinbase-server to process the online-core-node-list requests for all core applicable base services and the nodes initialization process to join the active-online-core-node-network.

### c.ii COINBASE-TRANSACTION

Once the coinbase service gets a valid solution to the riddle it sends to all connected-online-core-nodes it's coinbase-transaction-seal signed by it's signature to ensure it is the coinbase service itself and the core-nodes will vote it into the ledger thru their voting process.

### c.iii NODE-FEE-TRANSACTION

Once a week at Sunday night at 12 o clock the coinbase-service calculates the fee to be shared over the nodes wallets to send a fee-seal to the connected-online-core-nodes signed with a secured signature to be added to the ledger. From there a Wallet owner is able to spend the fee with a new transaction.

### c.iv WALLET-TRANSACTION

Wallet transactions are created thru any wallet software platform based on the communication protocols used by the node network. The Main principles are JSON communication over WEBSOCKET, with a strict transaction protocol to setup the transactions, sign and send them. See 8.h for further details

### c.v TRANSACTION-VOTING-CONSENSUS-PROSESS

The Core-Node-Network uses an artificial type of neural networking in where they have the single goal to come to a consensus over the first next transaction in its voting list of the to be processed transactions. Coinbase and Fee, are prioritized, then come the amounts of fee to be earned, than the oldest in the list. So that those with a higher fee get

processed sooner than the lower ones. As spend-ability has a big priority with larger amounts for any receiver.

#### c.vi WALLET-VALIDATION-DATABASES

Nodes have a fast-memory-database structure in where they keep track of all spendable blocks for all the wallets, to not have only the ability of truly validating a transaction but also do it very fast. So that their respond time for setting up transaction and processing other nodes transactions in randomly order is secure and fast.

### 8.d LEDGER-EXPLORER

A Ledger-Explorer connects to the node to process it's own ledger-copy and creates from that a complete wallet history database to be used by wallets to see their complete history and as a HTTP explorer to see the ledger as entire database of processed seal blocks in html.

#### d.i WALLETS-COMPLETE-HISTORY

Wallet connect to the Explorer to download their history blocks in to a local database what can be used for a front-end display of a wallet history explorer either textual or graphical. There are three types of history blocks, Transaction (History), Coinbase (Mint-story) and (coinbase/transaction) Fee (Fee-story).

### 8.e WALLET

The Wallet is the individuals centerpiece of it's own security.

#### e.i NODELIST-SERVER

It needs the node-list to connect to any (chosen) online-core-node to setup it's transactions.

#### e.ii WALLET-FILE

The Wallet-File is where a list of wallets is stored in and secured thru a password to decode the private-keys used for its signatures to secure

it's transactions. This password is used to enter your wallet software and only has a decoded version in its memory, yet can even be extended with a pin-code to actually create a transaction signature.

## The Wallet

```
#####################################################################
#
# Wallet structure
#
# offset length  content
#      0      2  '11' – PTTP identifier
#      2     64  Public hashkey
#     66      2  Checksum, xor ascii values 0-65 must be 0
#
# Wallet will be converted to uppercase always!
#
#####################################################################
```

**wallet.pttp File**

### Default

```
[
   {
      "name":"MyWalletName",
      "wallet":"11...",
      "pubkey":"64 byte public key",
      "privkey":"128 byte private key"
   },
   ... (multiple wallets)
]
```

### Password Protected

```
{
   "encoded":1,
   "wlist":[
      {
         "name":"MyWalletName",
         "wallet":"11...",
         "pubkey":"64 byte public key encoded",
         "privkey":"128 byte private key encoded"
      },
      ... (multiple wallets)
   ],
}
```

### e.iii WALLET-SOFTWARE

The wallet software can be made on all frameworks, as they only need to have the basic protocols in order to communicate a command to either the Node, Explorer or Node-List-Service to thru JSON strings over a websocket and/or HTTP(S) layer.

#### iii.1 Creating New WALLET's

Creating a wallet is done by the wallet software itself to ensure the private part of the private-key creation for that wallet. A code example is found in the FCC::wallet.pm module itself.

#### iii.2 Spending Crypto

The main goal of a wallet is to administer your own spending as central control of your spendable crypto coins. And this with a few security layers to ensure your wallets safety. Either local or remote. And to have a personal central administrative overview of all your transactions. Either in general or per contact.

#### iii.3 Contacts

The contact-list is a feature where you connect personal data to a wallet of others, either credit or debit, as to exchange crypto on a personal red line with your personal or corporate contacts. One or several to receive crypto on and one or several to send crypto to.

## 8.f MINER

The Minders are purely as a result of having a shared need for an actual averaged random winner in a kind of lottery by purely mining randomly to find the riddle the first, to create an actual liquidity ability for our value (seal) exchange in correlation with the actual available market-capital to make a market flow it's value we create with our added value out of our own hands. A Dynamic Liquidity Tool to Exchange True Value.

### f.i Buildin Single Core Wallet Miner

Our base wallet has a builtin Single Core Miner for just mining within the wallet itself. Later versions will give more control over all the extended tools what can come to support a wallet which are run separately in the base version.

### f.ii Standalone Multithread Miner

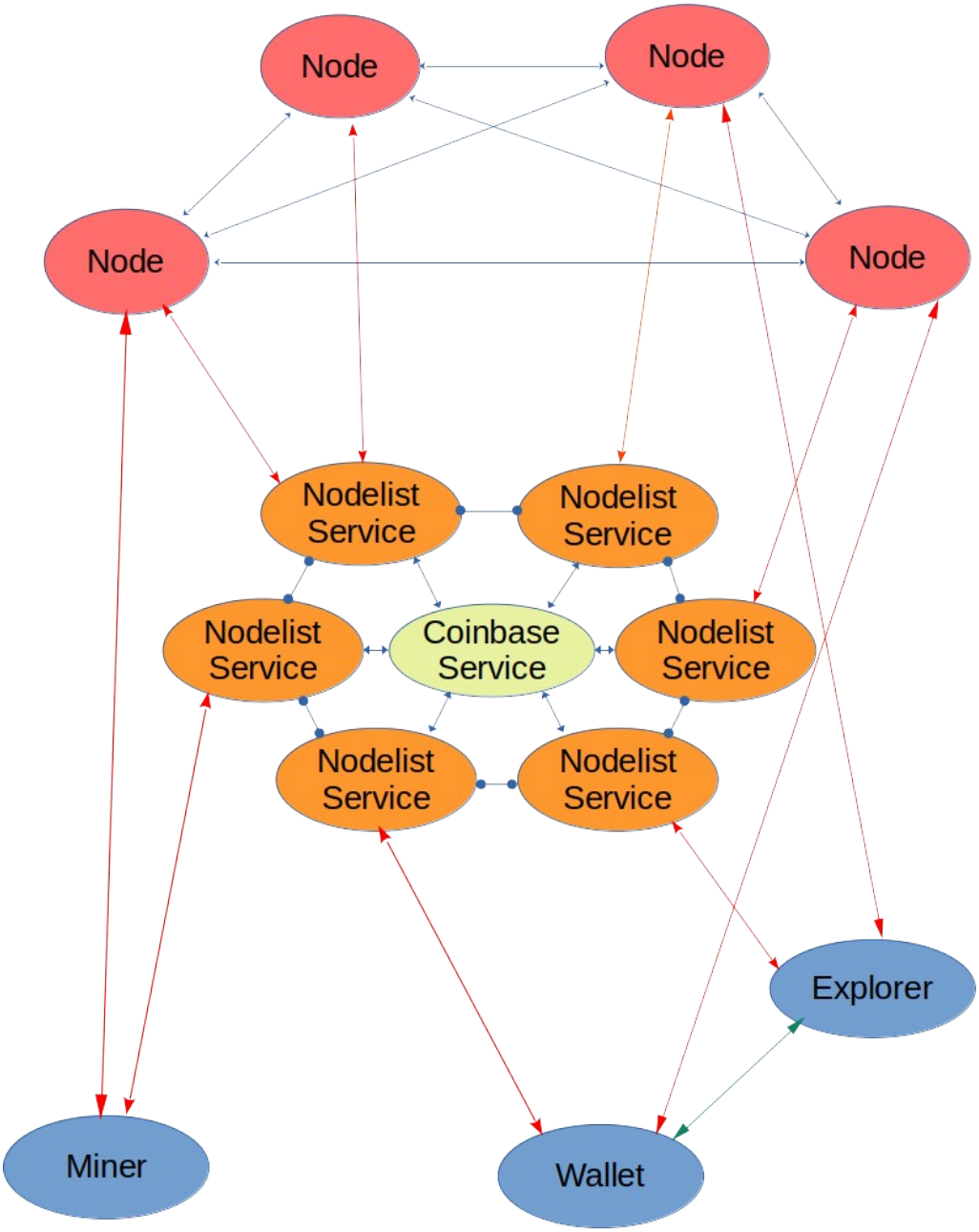
A standalone multi-thread-miner has been added as an example for a multi-threaded miner.

### f.iii Pool mining

Pool mining is purely based to be like a miner, yet to share it's mine-able blocks to a network of pool-miners supporting the pool. Personal networks can be created to mine for a pool this way.

With a descent administrative base anyone can setup a public pool with their own strategies of sharing it's minted coins.

## 8.g Network Overview




A minimal complete picture of the network structure when online and actively running the coin-ledger.

### g.i Coinbase/Nodelist Service

The Coinbase- and Node-List-Service is a joined network of servers to keep supporting the online nodes to be found by all sub layer services, like newly connecting nodes, explorers, wallets and (pool)miners and have a maximal staggered support for the whole network of applicable layers.

### g.ii Nodes

The Nodes have a parent-child structure to the network depending on how the new nodes come online, and hook into the network of active-online-core-nodes, their max number of node connected is set to 500 in total so to spread network load over several network groups connected thru it's dynamic online structure.

### g.iii Explorers

Explorers are for supporting the wallet and mining history administration and use the nodes to download a ledger copy and updates to be added to the history, this gives a little delay in the administrative part of the history within the wallet, yet when you create the transaction and it is processed both sender and receiver can know it has been processed and is directly spendable, as the node will let you directly know the active spendable amount on a wallet.

### g.iv Wallets

Wallets are to make new transactions, and have the original private-key to sign transactions. The standard software is to run one standalone, yet web-based applicability is just like a local wallet application with the difference that the private-key is stored on the remote-web-server-system, and gives that privacy element to the web-masters responsibility. Think of Exchange wallets for example.

g.v Miners

Miners mint the newly created coinbase coins released thru the mining process by the coinbase-service via the online-active-core-nodes on where a miner connects and communicates its solutions.

8.h Wallet and Miner Leaf Api Implementation Documentation

# Basic Framework

### Websocket Support

All Communcations go over the WebSocket protocol.
*(*Except for the nodelist collection thru https)*

### JSON Communication

All Communcations are spoken with the JSON protocol.

### Ed25519 Encryption

All Encryption needed is the Ed25519 Sign Function for Signatures with your Wallets Public and Private Keys.

# New Wallet

### 0. https-get Wallet

***For easy purpose only.***
For details about creating the wallet yourself, see FCC::wallet.pm

**PTTP** Wallet :

`https://factorialcoin.nl:9612/?wallet`

returns :


```
{
  "encryted":0,
  "wlist":[
    {
      "name":"[ No name ]",
      "wallet":"[PTTP-WALLET-ADDRESS]",
      "pubkey":"[PUBLIC-KEY]",
      "privkey":"[PRIVATE-KEY]"
    }
  ]
}
```

# Wallet-Leaf Connection Protocol

## 0. https-get Nodelist

https://factorialcoin.nl:9612/?nodelist
returns PTTP Nodelist : [node-ip]:[node-port][space][...]

***** After setting up your WebSocket Connection to the node ... the node will react with JSON commands *****

## 1. in < command:hello

```
{
  "command":"hello",
  "host":"[node-ip]",
  "port":"[node-port]",
  "version":"[pttp-version]"
}
```

## 2. out > command:identify

```
{
  "command":"identify",
  "type":"leaf",
  "version":"[pttp-version]"
}
```

***** After this react with the following JSON commands *****

# Connected Wallet-Leaf Commands

## Wallet Balance

### 1. out > command:balance

```
{
  "command":"balance",
  "wallet":"[pttp-wallet-address]"
}
```

### 2. in < command:balance

#### a. when error occured

```
{
  "command":"balance",
  "error":"[error-message]"
}
```

#### b. on success

```
{
  "command":"balance",
  "wallet":"[pttp-wallet-address]",
  "balance":[pttp-amount]
}
```

## Wallet Transaction

### 1. out > command:newtransaction

```
{
  "command":'newtransaction',
  "transid":[your-transaction-idnr],
  "pubkey":"[wallet-pubkey]",
  "to":[
    {
      "wallet":"[wallet-address]",
      "amount":[doggy],
      "fee":[doggy]
    }, ..
  ]
}
```

### 2. in < command:newtransaction

#### a. when error occured

```
{
  "command":"newtransaction",
  "transid":[your-transaction-idnr],
  "error":"[error-message]",
  "availabale":"[max-amount-available for transactions]",
        * optional, on Insufficient funds *
  "spendable":"[max-amount-spendable based on max inblock count value]",
        * optional, on more than 255 inblocks for it's max-inblock count with
          its spendable value including minimum fee, for creating multiple
          transactions to transfer the available amount intended *
}
```

#### b. on success

```
{
  "command":"newtransaction",
  "transid":[your-transaction-idnr],
  "sign":"[transaction-ledger-data-to-sign]",
  "pttptime":[pttptimestamp]
}
```

## 3. out > command:signtransaction

```
{
  "command":'signtransaction',
  "transid":[your-transaction-idnr],
  "signature":[your-transaction-ledger-data-signature]
}
```

### a. formulating the signature in [Perl Code]

```
[your-transaction-ledger-data-signature] =
  octhex (
    Crypt::Ed25519::sign (
      [transaction-ledger-data-to-sign],
      hexoct ( [your-wallet-public-key] ),
      hexoct ( [your-wallet-private-key] )
    )
  );
```

* **octhex** translates binary data into hexadecimal data
* **hexoct** translates hexadecimal data into binary data
* **Crypt::Ed25519::sign** signs your data with your private and public keys

## 4. in < command:signtransaction

### a. when error occured

```
{
  "command":"signtransaction",
  "transid":[your-transaction-idnr],
  "error":"[error-message]"
}
```

### b. on success

```
{
  "command":"signtransaction",
  "transid":[your-transaction-idnr],
  "transhash":"[node-transaction-id]"
}
```

### 5. in < command:processed

#### a. when error occured

```
{
  "command":"processed",
  "transhash":"[node-transaction-id]",
  "error":"[error-message]"
}
```

#### b. on success

```
{
  "command":"processed",
  "transhash":"[node-transaction-id]",
  "wallet":"[wallet-address]",
  "status":"success"
}
```

## Miner-Leaf Connection Protocol

### 0. http-get Nodelist

https://factorialcoin.nl:9612/?nodelist
returns PTTP Nodelist : [node-ip]:[node-port][space][...]

****** After setting up your WebSocket Connection to the node ... the node will react with JSON commands ******

### 1. in < command:hello

```
{
  "command":"hello",
  "host":"[node-ip]",
  "port":"[node-port]",
  "version":"[pttp-version]"
}
```

### 2. out > command:identify

```
{
  "command":"identify",
  "type":"miner",
  "version":"[pttp-version]"
}
```

****** After this react with the following JSON commands ******

# Connected Miner-Leaf Commands

## 1. out > command:mine

```
{
  "command":"mine"
}
```

## 2. in < command:mine

```
{
  "command":"mine",
  "challenge":"[ANSWER]",
  "coincount":[CBCOUNT],
  "diff":[DIFF],
  "length":[DFAC],
  "hints":"[HINTSTR]",
  "ehints":"[EHINTSTR]",
  "reward":[MINERPAYOUT],
  "time":[FCCTIME],
  "lastsol":"[LASTSOL]"
}
```

## 3. out > command:solution

```
{
  "command":"solution",
  "solhash":"[SOLUTION_HASH]",
  "wallet":[pttp-wallet-address]
}
```

## 4. in < command:solution

### a. on error

```
{
  "command":"solerr"
}
```

### b. on success

```
{
 "command":"solution"
}
```

**Version 1.0.1 of this PTTP WHITE PAPERS by:**

2017
Michel Kuipers