Leaf

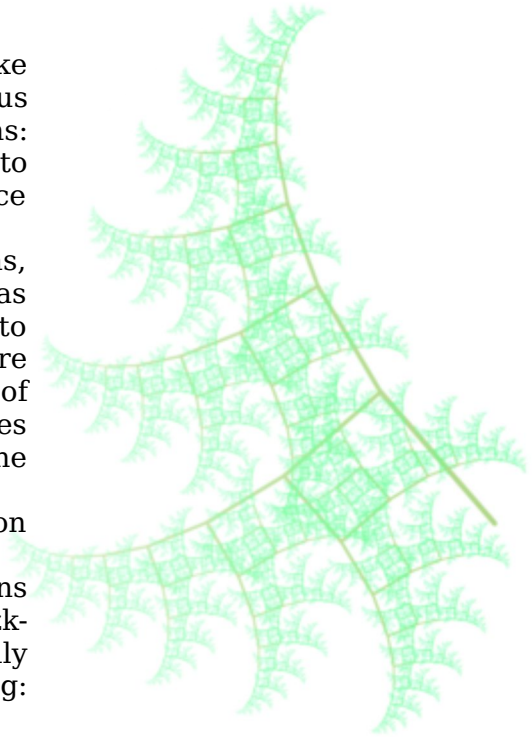
ECO FRIENDLY - SECURE - LIGHT - DECENTRALIZED - EGALITARIAN – ANONYMOUS CRYPTOCURRENCY INTEGRATED IN A DECENTRALIZED COMPUTATION AND STORAGE PLATFORM BASED EXCLUSIVELY ON POST-QUANTUM CRYPTOGRAPHY

Zhen Fen

Abstract:
This draft describes the building blocks of a green cryptocurrency based on post-quantum cryptography. We use a Proof of Work algorithm for interrupted mining in predefined time intervals, aiming to reduce the energy footprint of PoW. We combine the interrupted PoW model with a Proof of Burn scheme to enhance security. We use hash-based signatures and a sophisticated transaction mixing mechanism inspired by cryptonote to provide fungibility. We designed a synchronous anonymous network based on DC-nets and onion routing to achieve anonymity. We reorganized the blockchain to radically reduce its size. We move transaction and smart contract storage out of the blockchain: we only keep the hashes of the contracts on it to enforce their appliance, since anyone among the involved in one contract can reveal it to enforce it's usage in case it is violated. Last but not least, we propose to integrate this cryptocurrency with a decentralized storage and computing network, which could be used as a mechanism for price stabilization by setting coin supply according to the price of the provided services.

0. INTRODUCTION

Decentralization/Scalability issues – Environmental destruction
When Satoshi Nakomoto designed bitcoin, he couldn't predict all the parameters and consequences of his creative idea. Nowadays, cryptocurrencies, fail to become currencies, because of their unstable price. Instead, they are suitable only for investors.
Nakamoto designed bitcoin to be pseudonymous, but transactions on the blockchain are now being tracked by authorities. He invented “mining”, a method to distribute collective decision procedure, by the possession of computational power, but this turned into an industry of expensive specialized hardware which is controlled by a few, resulting in an energy hog, responsible for a relatively big portion of the industrial destruction of our planet.
There are also some technical problems. Most probably Satoshi couldn't predict the explosive adoption of his radical economy, and blockchain is now eating a lot of hard disk space, leading to the centralization of the public ledger in a few nodes. Furthermore, quantum computers which are able to break every blockchain, didn't seem realistic, back in 2008.



Partial solutions in altcoins

There have been multiple attempts to find solutions to these problems, and even more attempts to make money from these, resulting in thousands of cryptocurrencies available, consuming together enormous amounts of energy. Well, to summarize, proof of work mining is affected by severe problems: environmental destruction, mining pool centralization and centralization of the computational power to those who have the money to buy new specialized hardware (and those who have the money to produce the hardware).

Right now there are many coins which substitute Proof of Work with Proof of Stake minting algorithms, trying to solve the “nothing at stake” problem (a minter signs multiple blockchains, because he has nothing to lose) by bringing back centralization, since they use delegated users of the network to construct blocks and achieve consensus. But even if they manage to solve the problem of secure consensus, there is still a fundamental problem: what will substitute the decentralized distribution of money that mining provides? The answer is blowing in the wind... selling ICOs and making companies richer... The rich get richer both inside and outside PoS coins. This is totally in contrast with the decentralization spirit of bitcoin invention.

There are also some really interesting projects like IOTA or nanocoin, but they lack qualified security on their consensus algorithm, being vulnerable to attacks from adversaries with enough resources.

In terms of anonymity there were great advances. There are also many groundbreaking anonymous coins and anonymization technologies, like confidential transactions, coinjoin, cryptonote, Monero's RingCT, zk-SNARKs and Grin's Mimblewimble. All of them, are a source of inspiration for us, and they really contributed in braking governmental control over the money flow. But there is a huge problem remaining: when quantum computers will come to sunlight they will be admirable cryptosystems for the museums.

This problem is partially solved by zk-STARKs, beautiful crypto-structures which provide transaction anonymity in a post-quantum world. But they are very inefficient, so they are not yet suitable for blockchain use.

Our proposal

We are not going to reinvent the wheel. There are already beautiful projects out there to pick ideas and code from. Instead of using unexplored mathematics, we combine existing reliable technologies: Hash algorithms for blockchain verification, one-time Hash-based signatures for transactions, Pseudo-random Number Generators, DC-Nets combined with onion routing and Supersingular Isogeny Diffie-Hellman key exchange for an underlying anonymous communication network, Proof of Work used periodically and Proof of Burn to provide consensus. We reorganize blockchain structure in order to move store of transactions and contracts from the blockchain to wallets, leaving only a succinct proof (Hash) on it. Our invention is the timehash algorithm, a method for a network to verify that mining (re)started after a specific time. What we modify, is the general cryptocurrency architecture in order to **shield it against attacks from quantum computers** and to **radically reduce its environmental footprint.**

1. POST-QUANTUM SIGNATURES

Hash based signatures

Hash-based one-time signatures are known and well understood for decades. We will use this scheme because it is efficient, and we will shrink them in the blockchain with an interactive process.

In brief, if we want to sign a message of 256 bits, for every bit, we need 2 secret values, (a sum of 512 secret random values). We hash every one of them and we publish the hashes. The hashes are our public key and the random values are our private key. If we want to sign a message, for every bit of the message to be signed we chose one of the two private random values to expose publicly, and we expose 256 bit hashes. So only the owner of the private key could sign the message, and nobody else can reproduce the signature for a different message.

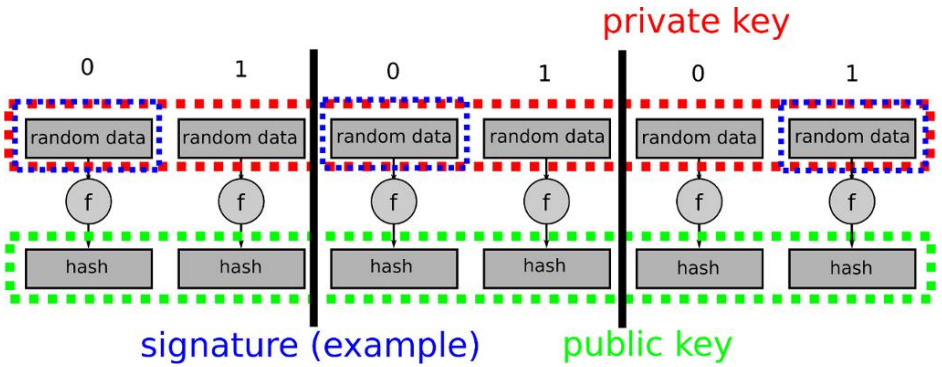


Figure 1:
Simple hash based signature

Of course this would require huge amounts of memory. Let us shrink it a bit. For shrinking the public key, we use Merkle trees, in order to derive one single hash from all the 256 hashes.

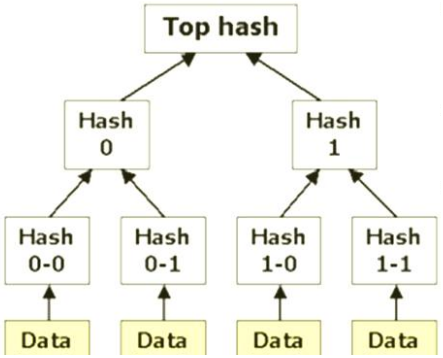


Figure 2:
Merkle hash tree

For shrinking private key and signature size, we use Winternitz's chains, as a method to represent 8 bits instead of 1 bit for every pair of hashes. This means that we construct two hash chains per pair, by hashing the hash of the random value repeatedly 256 times. From the last computed hashes on the chain we compute the much smaller Merkle tree of them, and their root hash is the public key. In order to sign the number 7 in a 8bit part of the message, the signer publishes the 7th hash in the first chain and the 255th hash in the second chain. Hence, nobody else can sign a value different from 7 on this position, since he does not know the previous hashes from which these hashes are derived. The verifier hashes repeatedly according to the signed message and he then computes the Merkle tree and the root hash. If the hash is the same with the public key provided, then the signature is valid.

In order to use this scheme in a cryptocurrency, we need a method to derive multiple signatures from one amount, while this amount is moving in the blockchain. Forget about accounts. Every amount is unique, and the owner of it is able to sign only one transaction with his amount. So the input amount of every transaction, signs the output amounts of it, which are the public keys of the above hash-based signature scheme.

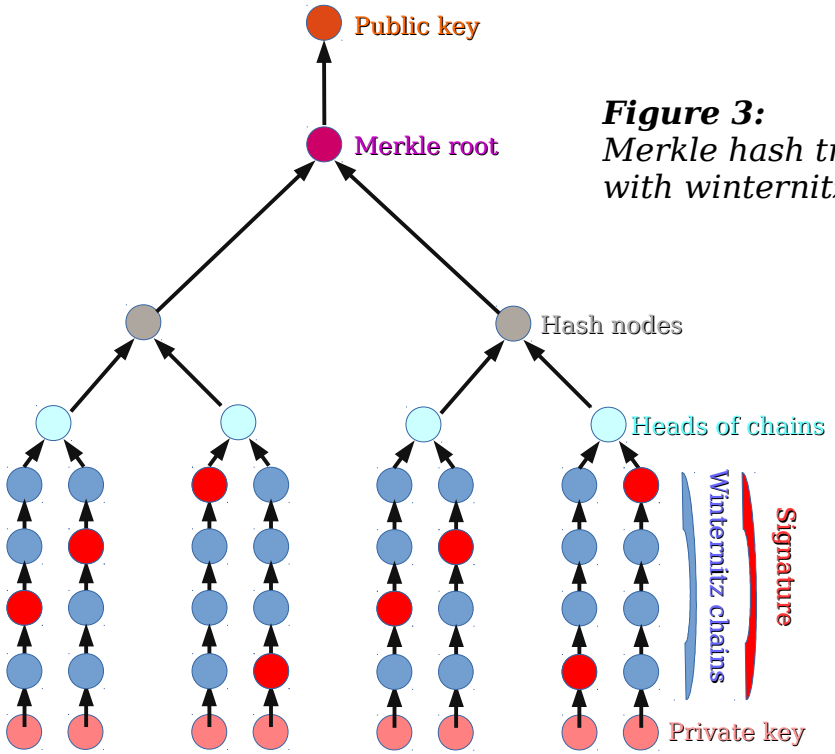


Figure 3:
Merkle hash tree signature with winternitz chains

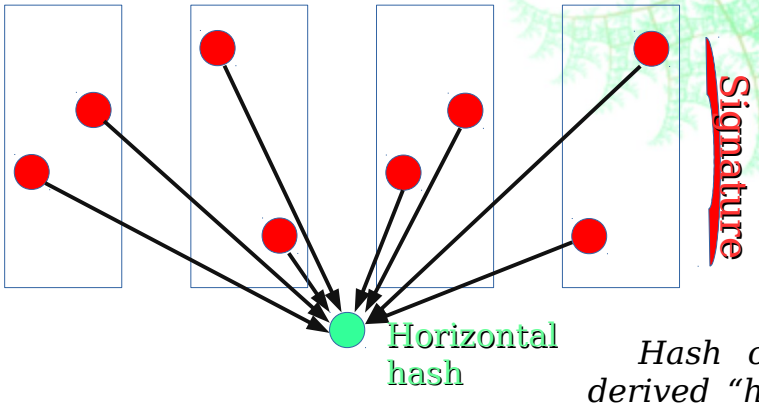


Figure 4:
Hash of the signature, derived "horizontally" from the middle hashes used in the signature

In order to shield Leaf from quantum computers we should set the following parameters: keccak 256bit hashes, 32byte public keys, 2kbyte private keys (which can be shrinked using pseudorandom number generators to reproduce them from smaller keys), and 2kbyte signatures. A little bit heavy, isn't it? If we put this on a blockchain with thousands of transactions per block, we need enormous amounts of memory and CPU power just for validating all of them. Well, we can do better.
Our trick is to move the validation responsibility to block producers.
Block producers are responsible to include only the correct signed amounts in the blockchain, providing a 256bit hash for every 2kb signature they receive. Moreover the public key, is not the root hash of the Merkle tree, but the hash of the root hash (one more hash), and block producers must include the hidden root hash also, which can be publicly validated by anyone. Hence, a block producer can try to make a counterfeit signature, if and only if he has seen a valid signature of the same amount.

Later, the signer of a transaction publishes a second signature in the next block to validate the transaction. Consequently a block producer cannot validate a transaction since the signer won't publish the second signature if there is no first signature already published in the blockchain.
If a block producer cheats and provides a fake hash of the signature, the supposed signer will reveal him by publishing the real signature, showing to the world that the supposed hash of his signature is counterfeit. As a result, the block producer will be punished, by losing all the funds associated with him.
Now we have two valid signatures. The first one, which was initially seen by the miner, and the second one, which was used by the user to prove that the miner was lying. So we need an additional mechanism for the validation of the real signature.

Second branch
We need to include two private keys in a transaction signature, in order to build two series of chains and corresponding trees, which merge to the same root hash. We 'll then add a second main branch to the tree. The extra private keys are also random and irrelevant with the first branch keys. Signer keeps them, and he doesn't expose anything about them, except if he needs to oppose the above cheater - block producer. In this case we reveal the second branch signature, and the network will validate the signature with the most branches. In this extreme case, we can accept the 4kbytes that these signatures need. This shrinking trick cannot be repeated for a second time on the same signature, because there is no other safety mechanism in case a block producer tries to cheat again.

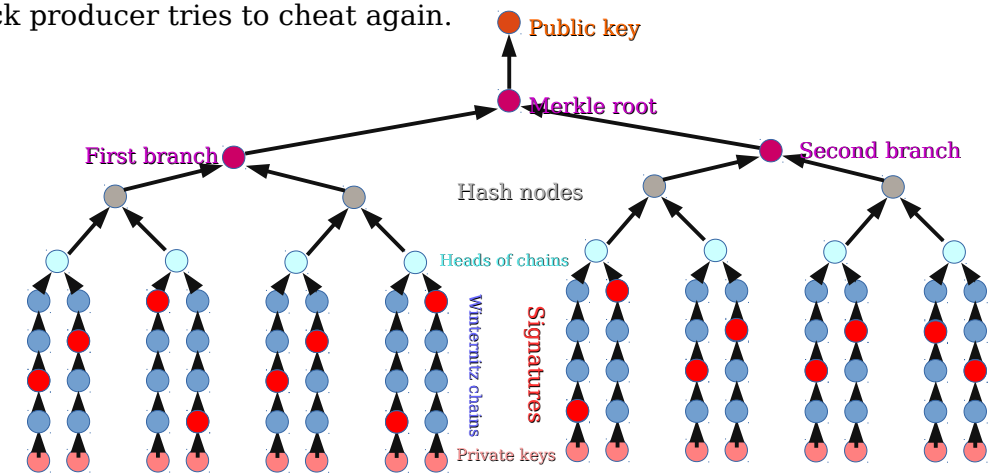


Figure 5:
Final form of our signature scheme

Bitcoin-like transactions with hash signatures
In our scheme transactions are only the output amounts (which are just public keys), signed with the input amounts. Instead of publishing full signatures by default on the blockchain, a shrinked version of them is provided in order to conserve block size small enough. Consequently the transaction size is practical. But in order to achieve anonymity we will need some modifications. You will find more information about them in the following chapters.

2. CONSENSUS

Proof of Computational Power
We propose a periodically interrupted Proof of Work model, in which miners are working during 20 seconds every 2 hours, resulting in a big short of the environmental impact. This idea also has a great decentralization potential, since a mining computer won't stall, but it remains a fully working normal computer, consuming no extra power. Hence there is no cost for someone to mine, which leads a lot of people to mine for free. Consequently there is no incentive for anyone to buy specialized hardware or to keep his computer activated more hours a day.

How do we achieve this? We can imagine an adversary who starts mining prior to the desired time point in order to multiply his portion of the hashing power of the network, resulting in taking over 51% of the network. The network has to verify that the miners started mining after a specific time point.

Timehash
Miners need a method to convince all nodes in the network that they started mining **after** a specific time point.
This problem is solved by the timehash protocol:

```
Every node produces a secret random value and hashes it.
Every node sends this hash to the network.
All nodes mix their hashes.
All miners will mine on a block which includes the final
hash of all their random values.
```

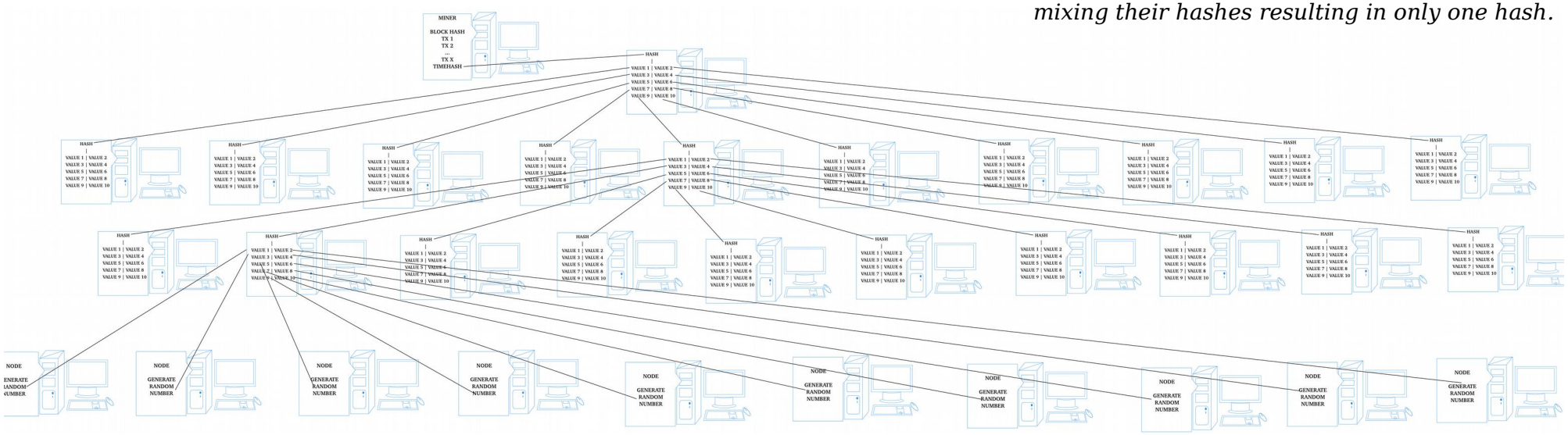


Figure 6: *Random values affect the final hash of a miner, by mixing their hashes in multiple layers. We can imagine thousants of nodes mixing their hashes resulting in only one hash.*

The above drawing seems a little bit centralized. We can imagine the same procedure applied horizontally to the whole network resulting in multiple different hashes for multiple miners:

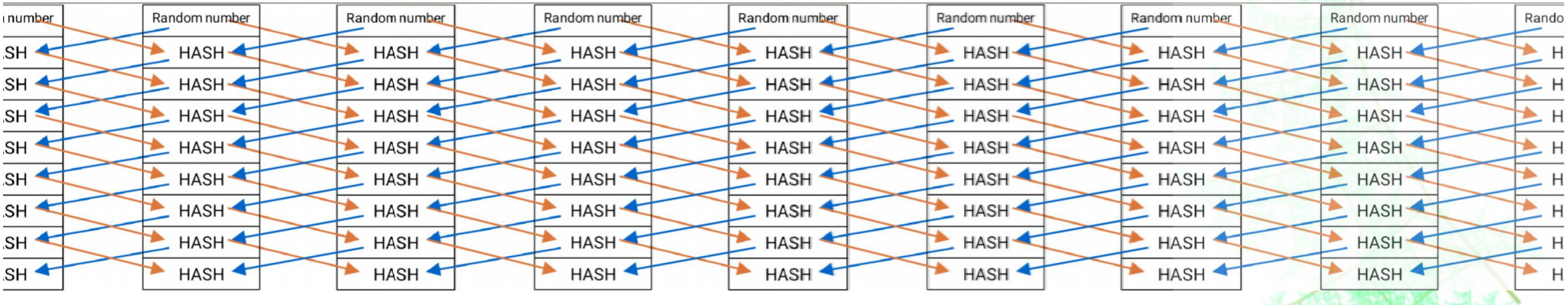


Figure 7: *A simplified version of this procedure (it includes mixing of only two hashes per node). The last raw of hashes are those of the miners, and they are affected by the hashes of the whole network.*

This algorithm is simple enough. It just requires a pre-image resistant Hash algorithm with 256 bit length. Our choice is 256 bit SHA3. *(Consequently, even a quantum computer needs enormous amounts of time to compute its input when given only the output).*

During timehash all nodes simultaneously execute every step of the following algorithm:

```
Select a secret random value, store it and hash it.
Send the hash to all nodes in direct connection.
Store the received hashes.
Hash all stored hashes with the secret random value to one single hash.
Repeat the 3 previous steps several times, constructing a tree of all
received hashes. (this tree will help later on verification)
```

Now we can imagine a network of 1.000.000.000 (10^9) nodes. If a node has an average of 10 neighbors, and the above procedure is synchronously repeated in the whole network 10 times, in the end, **every node will have a different root hash influenced by all secret values of all nodes in the network.**

Thus, if the miners include one of the billions of different (result) root hashes in the block they mine, the proof of work hash is created **AFTER** the timehash procedure. Now they have to prove it to the network, by the reverse procedure:

During the block propagation procedure, miners provide the mined blocks to their neighboring nodes. All mined blocks include the root hash of all the hashes that these nodes gave to the miners (10 in the above example).

Each neighboring node validates the block only if one of the hashes (produced previously by itself), is included in the hash list provided by the blocks, (and if they reproduce hashing the provided hashes together, and result in the same root hash).

Each node constructs a list of all the hashes received during the corresponding step of the previous timehashing process, and binds it to the hash list provided with the valid block.

Every node propagates the received block and hash lists.

Every node repeats the above procedure until everyone in the network gets a valid block with a virtual hash tree including its initial secret random value.

In the end **every node knows that the miner started mining after the agreed time**, so nobody is able to multiply his hashing power and control the bigger portion of the network.

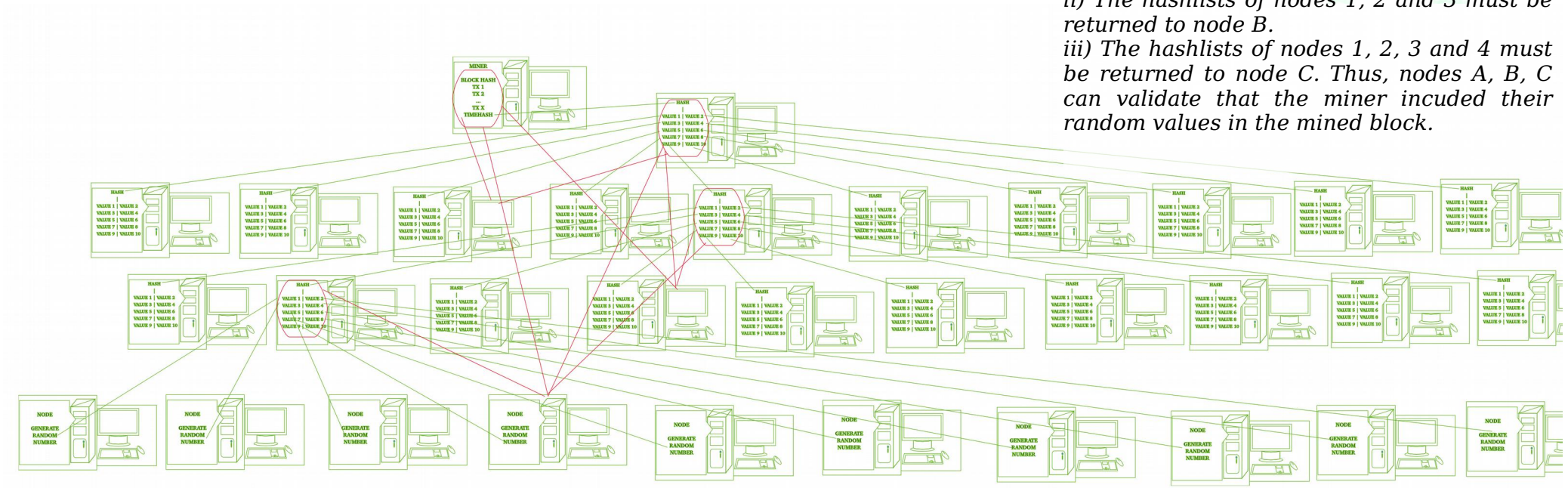
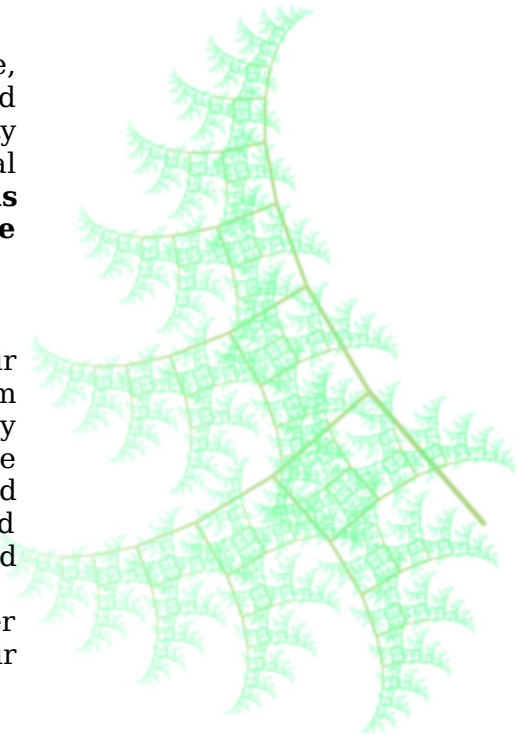


Figure 8:
i) The hashlists of nodes 1 and 2 must be returned to node A.
ii) The hashlists of nodes 1, 2 and 3 must be returned to node B.
iii) The hashlists of nodes 1, 2, 3 and 4 must be returned to node C. Thus, nodes A, B, C can validate that the miner incuded their random values in the mined block.



Optimizations
We propose to use an asic resistant (but not memory bound) algorithm, in order to keep computers usable, without any noticeable performance impact. We think that the appropriate algorithm is the well tested **cryptonight** algorithm, and we specifically choose the light version, which gives smart-phones the ability to participate in mining. In combination with our interrupted mining model, it could be really practical since it won't destroy smart-phone's battery and performance. **Decentralization potential of this scheme seems huge: every kind of computing device can participate in the network and take reward, without significant cost.**

Reinventing the block propagation mechanism
If the whole block is propagated, then transmission and validation takes too much time, resulting in unfair competition among miners and possibly to blockchain fragmentation. Thus we propose a mechanism which separates block propagation in 2 phases: In the beginning nodes start propagating “tickets” (only the necessary part of the block), including the transactions root hash, the hashlists, and of course the Proof of Work hash. Consequently, each node will validate that a proof of work came inside the desired time interval, but it won't know if the included transactions under the root hash are valid. In the second phase, as soon as nodes have finished propagating tickets, blocks are being propagated and validated (ticket propagation priority is always bigger than block propagation).
If the block which contains counterfeit transactions is revealed, it is considered invalid, and the miner loses his opportunity to earn some reward. So there is small probability that miners will drop their computing power just to flood the network with invalid block tickets during ticket propagation time.

Synchronous mining
Mining starts with timehashing procedure every 2 hours and it ends approximately after 20 seconds (block creation target time set to 20 seconds). Instead of mining on the longest chain, every miner, mines on the only chain which he considers valid, because the number of blocks is fixed according to time.
This modification provides some benefit. Since the possible valid blocks are only the blocks approved by timehash and their number is fixed, there is no possibility of a completely new block coming later. Thus, if a transaction or a contract is included in all possible chains, it is instantly validated. There is no need to wait for the other blocks. Hence, the consensus about a transaction or a contract in the blockchain is achieved before the whole blockchain consensus.

Timing problems
But there is also a drawback in this scheme. Miners are able to cheat by starting mining before timehashing is finished, including just a part of the network in their timehash tree. If they have more time to mine (~1 second), they consequently have a bigger possibility to win the PoW competition, by convincing a small portion of the network (nodes which are closer to them), that their blocks are valid. Despite that this situation will rarely occur, it could affect the consensus process, since one part of the network mines on blocks considered invalid for the rest of the network.

An extra safety mechanism for keeping the network robust when some miners decide to cheat, is the following:

- Double timehash steps, and
- the concept of semi-validity, if a block is considered IN_TIME for a portion of the network, and SEMI_OUT_OF_TIME for the rest.

Consider 20 steps instead of 10. Hence

- a node considers a block ticket IN_TIME if it has less than 10 steps distance in the timehash chain between the root hash and a hash produced by this node,
AND if it arrived before every other ticket.

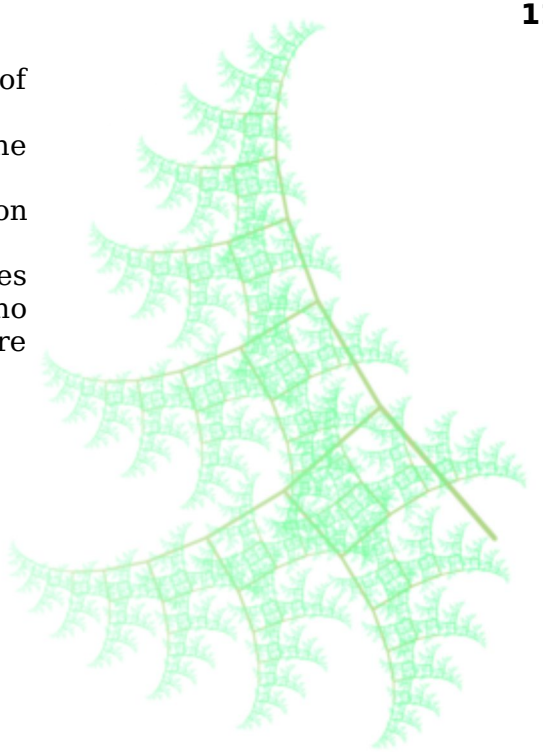
- A node considers a block ticket SEMI-OUT_OF_TIME if it has less than 20 steps distance in the timehash chain between the root hash and a hash produced by this node,
OR if it arrived T seconds after the first ticket.
(T=10*NET_LATENCY, where NET_LATENCY is the average time needed for the tickets to propagate to their neighbors)

The rule:

- a node **never** chooses a SEMI-OUT_OF_TIME node as head, but it could tolerate it if the majority of the network mines on it, if and only if this block came in intermediate time (in terms of the position in the timehash list or time arrived).

This means that if a block is completely OUT_OF_TIME, the nodes which chose it as head are not honest, since in the best case scenario the ticket has arrived to them SEMI-OUT_OF_TIME. Hence, honest nodes, don't accept it.

Even with these optimizations we 'll have blockchain fragmentation which demands many blocks to reach consensus like every PoW blockchain.



Proof of Burn

What if someone exploits the idle time of mining to mine an alternative chain, concentrating easily 51% of the network's PoW? How a new node could choose over two versions of the past chains?
Since every node won't be in the network from the beginning, what are the validation requirements in the absence of timehasing?
We need an additional mechanism to supplement interrupted Proof of Work. We propose a model based on Ian Stewart's Proof of Burn.
This model is the emulation of mining inside the chain, by a process with similar economical properties with mining. In this process, instead of miners who buy expensive mining rigs, there are burners who burn their coins to buy expensive virtual rigs inside the blockchain. Furthermore, instead of mining, there is a process like lottery, with which the next burner is chosen to sign a block. *In specific:*

Virtual rig construction:

Burner constructs 1024 one-time hash based signatures as defined above.
Burner combines all the above 1024 public keys in a Merkle tree.
Burner stores the above information locally. (~16mb)

Buying virtual rig (burning):

Burner publishes a special transaction: He spends a specific amount of coins to sign the root hash of his virtual mining rig.

Burner selection:

Real mining procedure generates consensus on previous blocks. Every node picks the nonce which produced the mined block 12 PoW blocks earlier (since consensus is already achieved on it), and seeds a Pseudorandom number generator with it.

The produced numbers (common for all the network) are used for the selection of the burners which will sign the next 720 PoB blocks.

Every 10 seconds a competition starts. Every burner searches in his public keys included in his virtual rigs to find the one with the value which is the closest to the above pseudorandomly generated number.

Burners publish their selected public keys, combined with proofs that they are valid (paths inside the Merkle tree of the virtual rig)

The burner who published the public key with the minimal distance from the generated pseudorandom number wins the competition.

PoB-block signing:

The winner collects all transactions and contracts of the previous 10 seconds, earns the fees and produces the PoB block.

The winner signs the block using the private key of the public key he announced.

Therefore, the PoB part of the blockchain will be deterministic, and if there are not any forks, transactions and contracts will be validated almost instantly (latency 0-10 seconds)

If any burners try to cheat, by including invalid elements or voting on more than one chains, they are identified by the next burners. Identification of cheaters is easy, since all used virtual rigs are listed by every node. Hence, cheaters are punished by losing all of their unspent funds associated with the virtual rig they used and their fees go to the next burner. If they sign the correct chain, they earn the block reward plus the transaction fees. So every burner is benefited if he stays inside the network running a node and waiting to be chosen for signing.
If there are blockchain forks, the blockchain with the less sum of distances between the signers' keys and the generated pseudo-random numbers, is considered valid.

In case an adversary concentrates 51% of virtual rigs, and violates the rules by announcing later a fake blockchain with the smallest sum of distances, it won't be accepted from the real miners who follow the blockchain in real time.
This conflict will cause all virtual rigs used in the fake version of the blockchain to be invalid in the real chain, and their rewards will be distributed among the next honest signers. Hence both the reward and the probability of the honest burners to sign a block becomes bigger, resulting in more burners involved and the network will soon recover.
If a new node doesn't know the real history of the blockchain in front of a conflict, it waits for the network recovery to accept a blockchain as valid.
Hence, **if an adversary concentrates 51% of the (real) PoW computational power of the network, he won't convince the new nodes unless he ALSO owns at least 51% of the virtual rigs.**
Consequently, **this dual mechanism of mining and burning supplementing each other, results in a more secure blockchain.**

There is also another advantage in our scheme: **If all mining pools collude, they won' t have the ability to change anything on the blockchain, since the last word belongs to the burners.** But burners could also construct the mining pool equivalents. What if these collude? Well, since virtual rigs do not really exist we could substitute mining pool's functionality with a shrinked smart contract (which is described in the following chapters, and it won't allow any cheating.

Nowadays, there are mining farms which concentrate a large portion of the PoW power, and they are able to “occupy” any small altcoin’s hashpower for a few hours and a double spend attack is thus possible. Our model protects us from this case. Moreover, consensus of pure PoS coins is exclusively derived from inside the blockchain. This could lead to a situation of the blockchain from where it cannot recover. In our model we have an external source of randomness which is PoW. We thus combine the advantages of both strategies while we avoid their vulnerabilities.

3. ANONYMITY

We are inspired from cryptonote's core idea: mixing the input transactions, making indistinguishable which input actually signed the transaction linked to the output. Cryptonote implements this with ring signatures (with one private key and a few public keys, you can make a valid signature while which of them is the private is hidden). Moreover in order to prevent double spending, there are "key images", derived from private keys through an one-way function. It is not possible to recover the private key from the key image, but it identifies that a private key is used, (hence it is spent), because it is unique.

Leaf transaction mixing

We will try to reproduce the mixing properties of cryptonote to our hash signature scheme, but with a completely different internal mechanism. There will be 64 kinds of outputs representing different amounts (powers of 2). So if someone wants to send 5 Leafcoins (101) in binary, he will send a Leafcoin of amount 1, and a Leafcoin of amount 4 (100 in binary).

In our model, the input of a transaction, is the XOR of 8 previous outputs. This is produced simply by the following procedure: (Alice pays Bob)

Bob finds 7 random outputs of the same kind of amount on the blockchain or from other random wallets (through an encrypted anonymity mixnet) and stores them secretly.

Bob constructs a hash-based key-pair as described above.

Bob XORs the 7 random outputs and the public key of his key-pair (the next transactions input). This is his output.

Bob sends the output to Alice.

(Alice has done the above steps on the past)

Alice recovers her 7 past random outputs.

Alice publishes them together with her past output linking to the input of the new transaction.

Alice publishes the signature of the new output, using the private key of the input of this transaction constructed in the previous transaction.

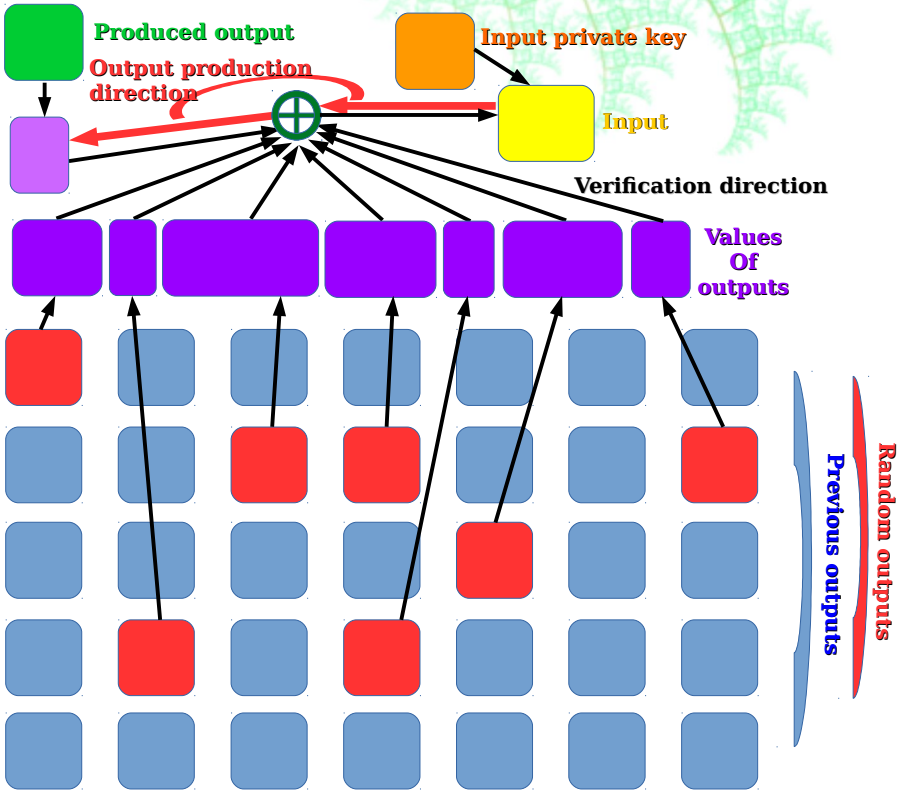
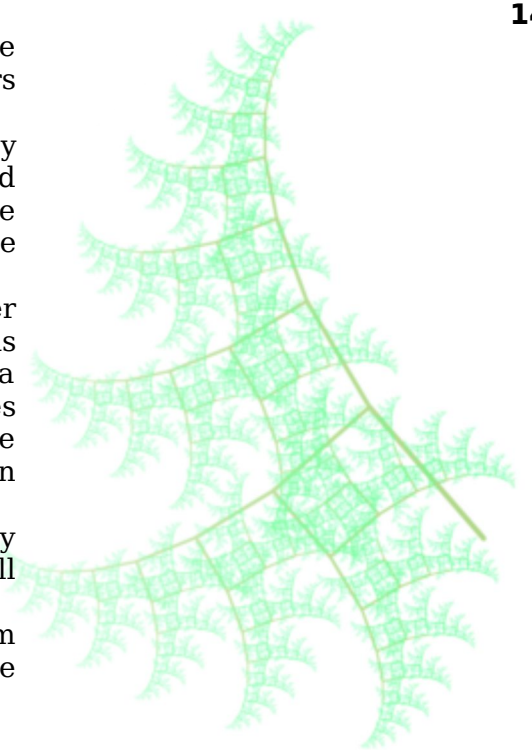


Figure 9: *representation of the relations between inputs and outputs.*



The above idea is not complete, due to the order in which outputs appear for the first time in the blockchain. Consequently, we need a mechanism to exchange unspent outputs between the payers through an anonymity mixnet, to obfuscate the order of first appearance in the blockchain.
This requires a warranty that the random output will actually be published. In case it won’t, the public key which was made with it, can't be spent either. We propose that whoever wants to hide better, could send his whole transaction to another user through the mixer, so that the user can use the outputs, while having the ability to keep them spendable, since he can release the transaction whenever he wants. The cost of hiding, is the delay of the transaction.
The receiver of the off-chain transaction also needs a warranty that the sender won't publish another transaction with his input prior to the agreed announcement of the transaction, which render his funds useless. Thus, sender has to publish a hash of his transaction associated with his input (combined with a secret obfuscating nonce shared with the receiver), before sending his transaction. If the sender violates the agreement, the receiver can reveal the original transaction associated with the hash and takes the output for himself. This transaction hash announcement will be obligatory for every transaction, and in case the transaction is not sent to an intermediate user, it could be just a random number.
This is a way of obfuscating the order in which the amounts created, aiming to make completely indistinguishable, which of the 8 inputs of a transaction is made by the signer. Hence, nobody can tell which of the inputs is actually related with the outputs.
In our model, Cryptonote's key image, is substituted by the root hash which leads to the public key (sum of the inputs). So the network constructs a list of the public keys of the hash-based signatures, so that the same inputs combination won't be able to be spent twice, since they are identified on the list.

Wallet

With all the above ideas, it is clear that the wallet must be sophisticated enough to manage all those sets of keys and signatures, watching the blockchain, inform other wallets about payments and taking care of its users anonymity by exchanging keys and outputs, or setting up connections in the mixnet. A wallet has to deliver the sense of simple homogeneous money and amounts to the user, while in the background will store thousands of keys, and exchange them with other clients to provide anonymity.
Actually we move the complex computations from the blockchain to wallets, in order to decentralize and reduce the overall overhead. We also want it to be integrated with the private cloud platform, so anyone anywhere can restore it. Leaf wallets will thus provide the ease of use of all cryptocurrency wallets.

Mixnet

The mixnet is a core component of our anonymity structure. Without it there is no real anonymity. Nowadays currencies use external networks such as tor and i2p, which suffer from the same problems.
First of all they are not yet adapted to the rise of quantum computers. Secondly, they have a fundamental problem. When a client of tor passes through 3 proxies to anonymize his internet traffic, his ability to hide leans on the assumption that the 3 proxies will never collude. (Actually, NSA runs tor nodes in order to intercept traffic and deanonymize users (and it might use more advanced techniques which require less than 3 nodes per circuit to deanonymize users). Last but not least, because of the asynchronous basis of these networks, they are vulnerable to traffic analysis and timing attacks.
Our cryptocurrency is based on synchronous mining and burning, hence it can be based on synchronous transactions also. This gives us an advance when we want to anonymize transactions and wallet interactions. We can synchronize all wallet activity in fixed time points in the middle between two PoB blocks signings, and thus exclude any traffic analysis from our model, since all transacting wallets will send at the same time similar encrypted packets.

In specific: Our model is based on Dining Cryptographers networks. This concept can be described as follows:

> Many users interact with each other exchanging seeds for PRNGs
>
> Every user mixes all the generated random numbers they agreed, with the other users with a XOR gate.
>
> Every user except the anonymous sender sends their mix results to common channel.
>
> If the anonymous sender wants to publish digit 0, he sends his mix result, else if he wants to publish digit 1, he sends the opposite of his mix result (through a NOT gate)
>
> Every user reads all sent data from the channel and XORs them together to define the result.

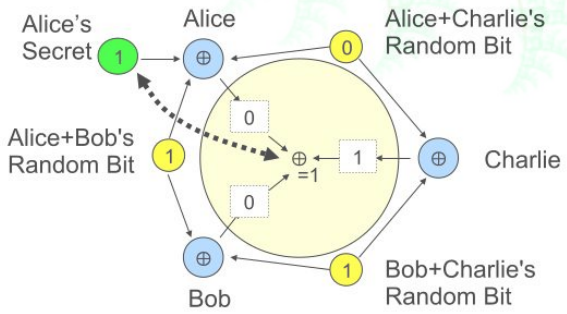


Figure 10: *Anonymous communication in DC-nets. Everyone can see that someone sent the digit 1, but noone can find who sent it.*

Hence they can send anonymously, since the other users don't collude (at least one honest user prevents identification leak).
If every user plays by the rules, the result is what exactly the anonymous sender wanted to be.

The most serious problem of this scheme is that a malicious user can send random data to destroy the communication channel without being identified, since the network is anonymous.

In our model, some nodes will act as mixers, without knowing anything about the mixed data. We also divide the communication procedure in two phases: establishment, and transmission.

Before the establishment phase, every user sends fees to the mixer.
During the establishment phase, every wallet publishes a hash based signature public key.
Users exchange their random PRNG seeds (different for every phase or slot they are going to use) signed with the above public keys, and publish one hash for every seed.
Every user sends anonymously through a DC-net, an anonymous public key of a hash-based signature and an identification hash of his data to be transmitted. (During this phase, they can follow an Ethernet-like protocol to define the time in which everyone sends.)
The mixer announces publicly some slots to emit in the next phase. Every identification hash gets a valid position for sending the whole packet of data later.
If the establishment phase results in corrupted data, all users open their PRNG seeds and they reveal their hashes without revealing the data to be sent. They collude by the protocol to find who did the corruption, and he thus loses his fees without transacting. No important info are linked, only the hashes which can easily changed during the next round (the data include some randomness except of the transaction).

If the establishment phase succeeds, then we go to the transmission phase.
Sensitive encrypted data are being sent to the mixer.
If all the hashes can be derived from the transmitted data, it means the transmission is correct.
If one hash is corrupted, the mixer exposes all of the inputs of the corrupted part of the data for this slot.
The user who owns the corrupted slot, requests the missing signed seed which match with its announced hash (through a backup anonymous channel).
He publishes (through the backup anonymous channel), the signed seeds which do not match, in order to deanonymize the user who corrupted his data, and signs this packet with the signature of his anonymous public key.
The user who corrupted the data gets punished by losing his fees.

DC model has a second drawback. Every user must send random data all the time, consequently it multiplies the data to be sent per user with the number of users. So it is not scalable enough.

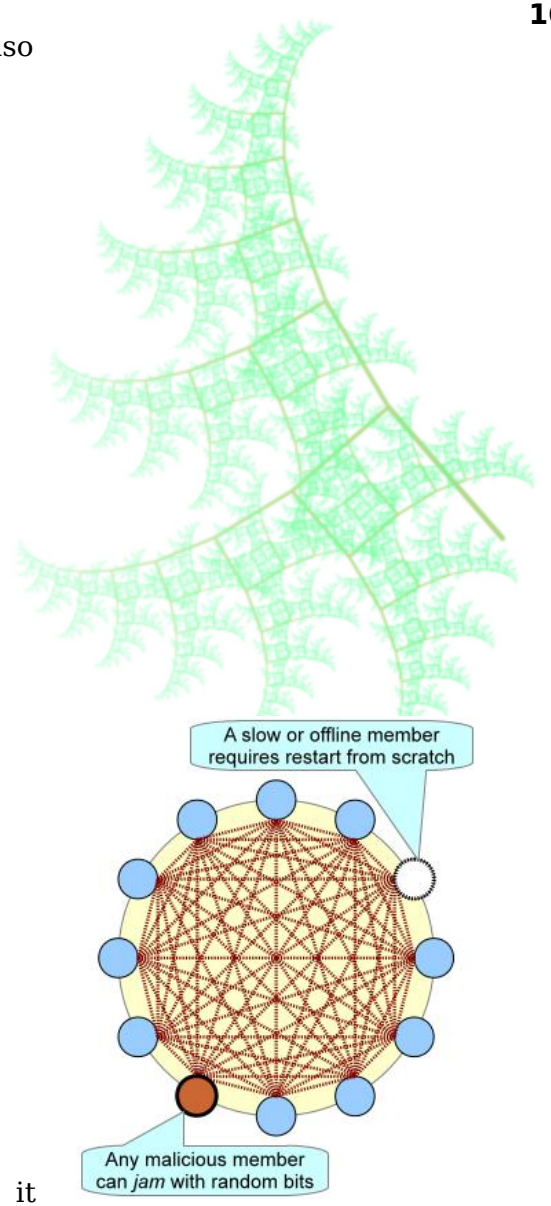


Figure 11: *DC-nets are not scalable and they are vulnerable to jamming attacks.*

In order to scale the anonymity network, we use this scheme as an enhancement to an onion network consisted of proxies which mix their inputs according to the DC-net scheme. The anonymity network is organized in 6 layers of encrypted tunnels, like tor. In each layer there is a proxy which also acts as a mixer. Each proxy-mixer is recognized by a public ID (the root hash of a hash-based signature).
Every wallet interacts with 12 different wallets in mixing and 6 mixers organized at levels. Hence, there are needed 18 different compromised nodes in one circuit, in order to deanonymize a user.
The onion tunnels between wallets and mixers at all layers, will have 256 bit AES with keys derived by Supersingular Isogeny Diffie-Hellman key exchange, which is already practical and quantum proof.

```
   A wallet communicates with two other wallets.
   The wallets agree  on their PNRG seeds, and on which mixer-
proxy to use.
   Mixer-proxy, signs his new keys (because they are one-time).
These are written in the blockchain, with the fees for the
mixing.
   All wallets communicate with the mixer to establish a DC-net
with the protocol defined above.
   Every wallet runs SIDH key exchange with the mixer-proxy in
order to start an AES encrypted tunnel.
   Through this encrypted tunnel, use the mixer as proxy and
repeat the above procedure, until you have 6 layers of proxies,
like onion routing.
   Send the sensitive data through the tunnel (inside the tunnel)
```

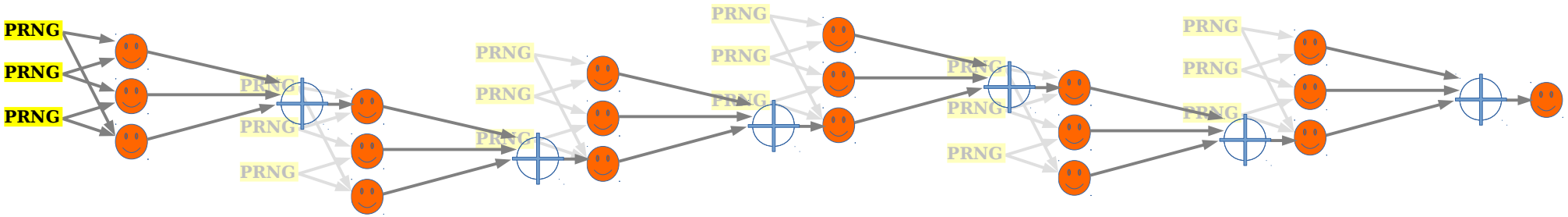
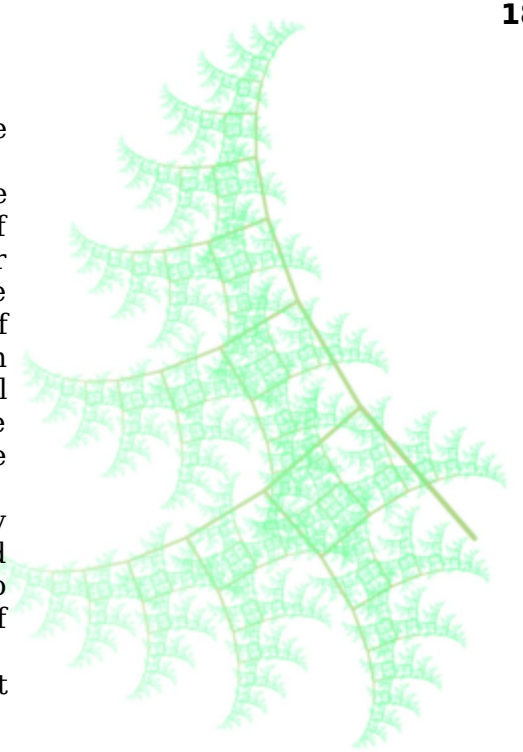


Figure 12:
Anonymous communication in several DC-nets, organized in layers, implementing onion routing.

Compared to existing anonymity networks like tor and p2p, this protocol seems paranoid and heavy, but the overhead is small since transactions are very lightweight in contrast with normal internet traffic (few KB for transactions vs several MB for browsing). With our scheme, every mixed transaction will hardly consume 1mb on client's connection. Furthermore, we aim to extend this network for sharing sensitive information in general.

4. BLOCKCHAIN SCALABILITY



In order to keep the blockchain scalable, it is necessary to use mixing transactions (cryptonote-like scheme) only in the head of the blockchain.
We need to brake the proposed transaction scheme in two phases. In the first stage we will have the mixing as defined above, and in the second stage we will stabilize the output to an “one to one” scheme. If we make this scheme mandatory, it won’t have any impact on the anonymity set, since the exact number of mixed transactions remains the same. In the second stage we end up with a transaction output suitable for long term storage on the blockchain. Thus, transactions available for mixing will remain on the head of the blockchain and the list of spent pub-keys (the equivalent of “key images” in cryptonote) will be much smaller and sustainable. These spent pub-keys lists, combined with mixing transactions they refer to, will live in a frame of blocks in which they are valid, before they expire. The size of frames in blocks will be different for every kind of outputs, since bigger amounts might appear more rarely in the blockchain. The actual size will be defined by an exact number of outputs to mix (~80.000 at every frame).
If we throw away the spent pub-keys list, how we recognize spent transactions? We can have a binary table of all one to one transactions of the blockchain, in which every spent transaction will be represented with 1, and every unspent with 0. The binary table will be changed in every block according to transactions spent on it. (More technically, burners will distribute the PoB blocks containing the list of changes and the hash of the new table).
In order to achieve real blockchain scalability we will use Bruce’s mini blockchain scheme with slight modifications.
We begin the description of this idea with a major modification in the blockchain validation process. Instead of validating the whole blockchain from the beginning, every node will collect all the alternative chains on which valid miners mine. Since nodes can validate current transactions and blocks and they are capable to distinguish valid miners due to timehashing, there will be a few blockchains to chose from. Consensus exists when all miners agree about the current blocks and revalidate them. But what about past blocks of this long blockchain with gigabytes of transactions? Well, if there are forks of the blockchain, nodes will have to run the validation process only on the part in which there is a disagreement, and they will choose the valid blockchain (with the smaller sum of distances of PoB scheme as defined above), only if it matches with miners’ choice. In case miners who mine on valid chains disagree, the new validating nodes will wait for the network to agree again before accepting a blockchain version. So the validation process will exclusively follow disagreements and there will be no waste of computational power and time. The final touch on our verification scheme, is to avoid a DoS attack of a malicious miner that floods the network with fake chains in order to slow it down. This is achieved by defining a small percentage of the mining power of the network (1%) as the limit under which a different version of the blockchain won’t be checked by any nodes. This limit is enough, since nobody wants to waste his mining power just to slow down the network (a little bit).

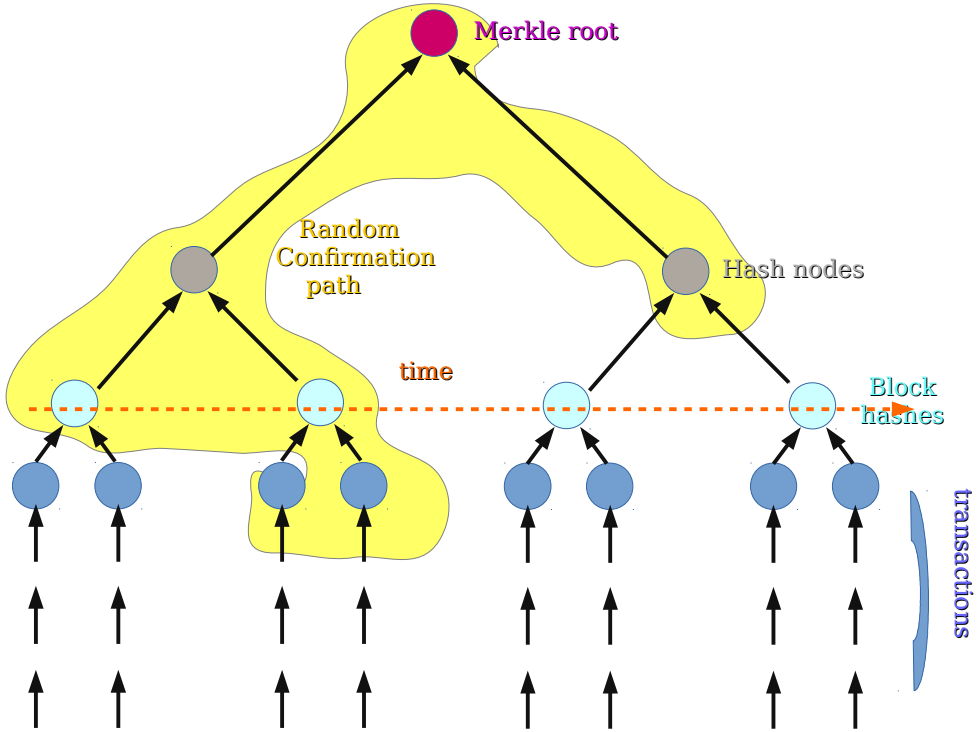
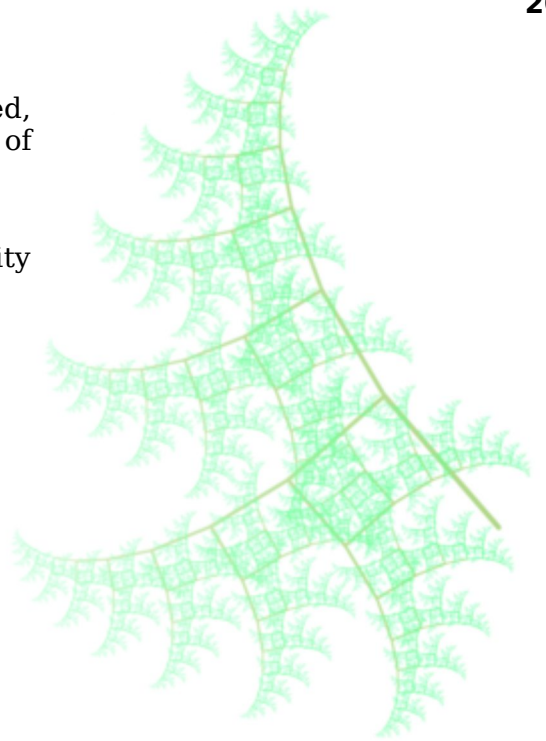


Figure 13:
Graphical representation of the compact blockchain scheme.

We have already reduced the verification time and cost. How about blockchain space? The old blocks of the chain will fold to a Merkle hash tree. More specifically, when new blocks produced, we actually keep a current blockchain of the recent PoW blocks and PoB blocks, and throw all past transactions, blocks and tickets to large Merkle trees divided to branches according to time. Every wallet is responsible for keeping the paths from the root hash to their transaction. So blockchain gets rid of transaction storage, and keeps only the proof that transactions happened: the root hash. When a wallet makes a transaction, it will have to provide the path of hashes from the previously spent output (new input) to the root hash.
In our model, nodes must share only the root hashes of the chains on which the miners mine and the binary table. Therefore, the overhead of a long term transaction in the blockchain, will be $n<64$ bits (where n is the number of digits with value 1 in the binary representation of the output amount). We can further reduce the size of the table during transmission by compressing it, since the most of the past transactions are spent, hence they can easily be represented in a zip file.
This model radically reduces verification time and hard disk space for the nodes, resulting in great scalability potential. Verification and storage of the blockchain is distributed, and coordinated through the Merkle tree.

5. SCRIPTS AND CONTRACTS



We want to keep this important functionality of modern blockchains in the above manner. Shrinked, private, and quantum secure. We insist on using simple and well tested hash algorithms instead of unexplored complex mathematics.
Our proposal is the following:

- We choose web-assembly as our main language for all scripts and contracts, due to its flexibility (every programming language ported to web assembly can be used).
- We agree on a protocol for the accepted inputs and outputs of our scripts and contracts.

Users who participate on a contract, agree on its script privately, and then they publish a signed hash of their contracts code on the blockchain, together with the signatures of the transactions that binded as inputs to it.

Contract's inputs also sign contract's outputs according to the rules.

In case the contract requires new input transactions, while already running, input transactions sign its hash with the associated protocol outputs (in a new block).

All the involved users must run their code locally, and follow its rules on the blockchain.

If someone violates the rules, he can be exposed by the other users, through publishing the full contract.

In case a violation is announced, all nodes run the contract to validate the violation.

Then the violator gets punished, by losing all the funds associated with him, and burners collect the fees (for the expansion of the contract in blockchain memory size and in CPU cycles required), from the violators of the contract.

For example, if there are off-chain private transactions inside the contract and someone attempts to sign an invalid new transaction as output with a contract's input, the rest of the users can reveal him and distribute his funds among them.

This model keeps **hidden all contracts by default**. It also saves space from the blockchain, since only a small number of contracts will be published, the violated ones (if they exist, they will probably be the result of bad programming, since nobody can be benefited of it) and the rest can be stored in 256bit hashes.

Furthermore, users who participate in a contract can improve the performance of their contract application, since they are able to run in their computers a native implementation of the contract, which reproduces the exact same results as the protocol defines.

Burning pools

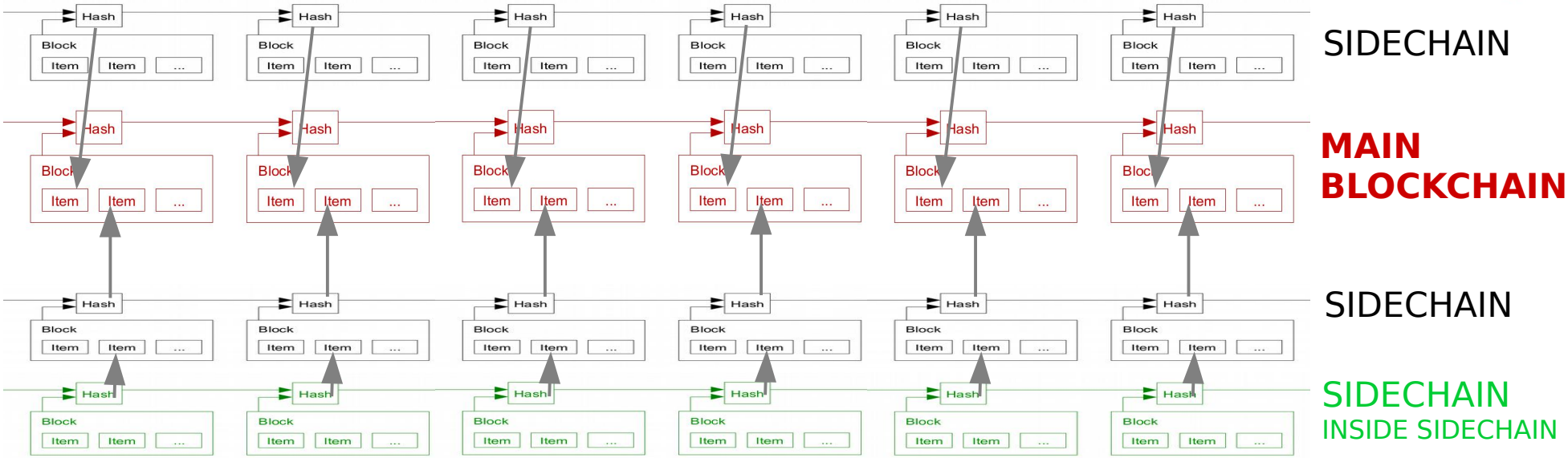
Burning pools, will be for burners, what mining pools are for miners, providing the equivalent functionality. They will be smart private contracts signed with a hash-based key-pair associated with every virtual rig, which says that every profit of these virtual rigs must be shared among them, according to the rules. If someone violates it, he loses his funds included as warranty input in the burning contract.

Infinite transaction capacity in a finite blockchain

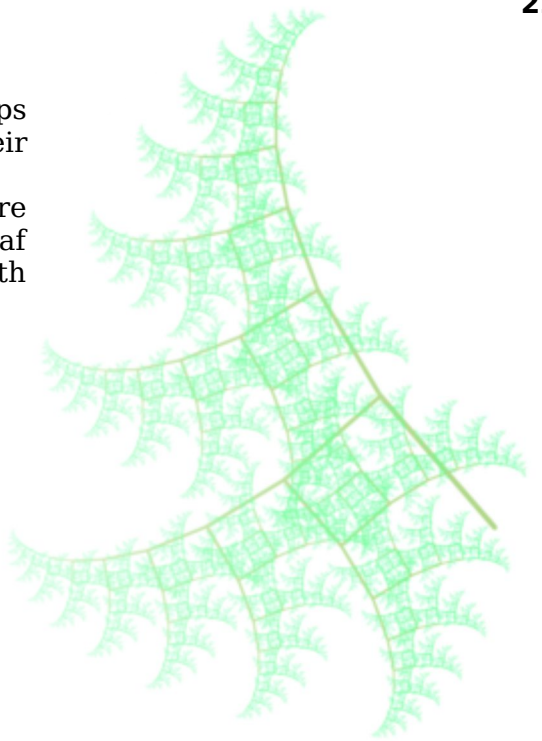
The above smart contract technology allow us to fit a whole parallel side-chain inside the main chain, using only one 256 bit hash per block. We could construct a private blockchain based on a smart contract which republishes its hash at every block. This hash, is the side-chain's new block hash including all recent transactions. Last but not least, side-chain could achieve consensus either with revealing possible double-spent attacks as violations in the main chain, either by running the same PoB protocol, using separate burners and virtual rigs, seeded with randomness from the main chain.

Transactions between the main chain and the side chain can be done either directly by binding new inputs and signing outputs, either indirectly with cross-chain atomic swaps.

And there can be multiple chains, even side-chains inside other side-chains, allowing infinite capacity and heavier protocols to be included (for example a a zk-STARK based blockchain, which moves its weight only to those interested in them).



6. PRIVATE FOG STORAGE AND PRICE STABILIZATION

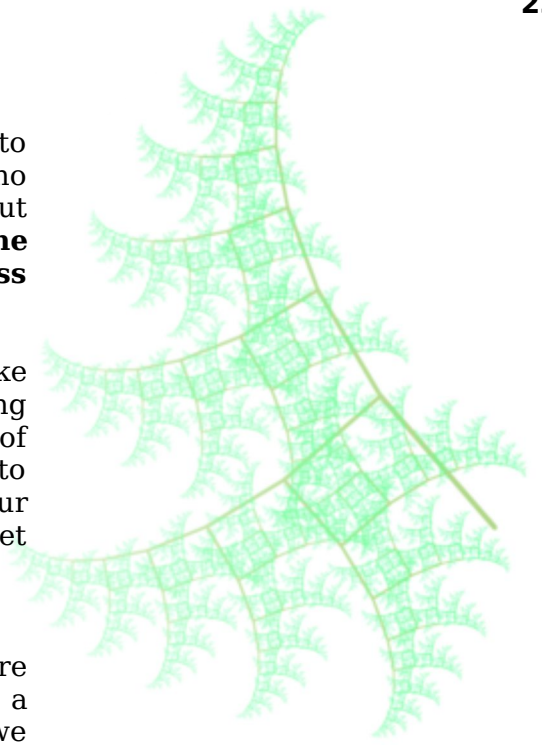


Why we have to integrate a private decentralized cloud service with a coin? Because this integration helps both of them. Private fog could use the coin as a reward system for the computers which offer their computing power and hard-disk space to the network.
The coin needs this to store the blockchain in a decentralized and secure way. It is also a required feature in order to provide on-line wallets, since Leaf wallets have to store a lot of info to be functional. But Leaf gets more benefits from this connection, as a mean to provide some real service. This could also help with price stabilization.
We propose a mechanism of controlling the price as follows:

> Depending on the average price of the provided cloud services since both of them are written on the blockchain, we build a global estimation of the current price at a given moment. Hence, anybody can use the on_chain data to see if Leaf's value goes up and down.
>
> We can't change miner reward, since this could affect security stability of the network. But we can control the supply of the newly mined Leafcoins by setting the appropriate time in which miner rewards are released for spending. So there is an extra level of intermediation between the market supply and the miners who are (and have to be) disconnected with the real market.
>
> This control mechanism, is integrated in the Leaf protocol, and it gets enforced by all nodes in a decentralized way. When the price seems to increase, we increase the supply by lowering the number of blocks which miner rewards have to wait before becoming spendable. When the price decreases, we do the opposite, in order to decrease the supply and increase the value again. We will only approve a slow and stable value increase.

This is a first step in the direction of reducing the price instability problem of all coins, to prevent price manipulation techniques, since there are no decentralized exchanges or price control mechanisms.
As for the underlying structure of this system, we refer to what Maidsafe does in terms of decentralized disk space usage: “farmers” (which store pieces of encrypted information) are getting paid (either with new coins, like miners, either) by the users who store it through the cryptocurrency. There are multiple copies of the data at several farmers and if a farmer fails to keep the data, automatically another farmer copies a clone of this data to a new farmer. The users select the number of backups they need for their data according to the importance of them and they pay for them. Farmers have to periodically provide proofs that they store the data, in the form of publishing the hash of a random piece of data. The randomness in our proposal comes from the current state of blockchain. Also, farmers reliability could be enhanced if they stake an amount of money in order to guarantee that they won't exit unexpectedly before they copy the data to another farmer with equal or bigger staking amount. In addition, “farmers” define the value of their services, and their average value defines the total supply.

7. EPILOGUE - ROADMAP



Ecology prioritized
We believe that there are already good algorithms to substitute PoW out there but they lack a way to convince miners to switch to it. In all major coins, miners decide the blockchain direction, and there is no way to convince them to throw away their expensive equipment and switch to an irrelevant algorithm. But with our solution things are really different. **If any project makes the switch to our algorithm, the miners can keep their mining machines and use it, they will just achieve to mine with less electricity.** So they have no reason to not support the switch if this new code is implemented correctly.

Consequently, we aim to see interrupted mining in more coins in the future. We found a way to make cryptocurrency networks more respectful for the environment, which is fully compatible with the existing form of the most of the cryptocurrencies. That 's why, we will initially distribute the pure source code of cryptonote including only the interrupted mining model. This aims to show the world it is possible, and to provide a source code paradigm of our model. We don't want to use this feature to promote exclusively our project. Consequently we decided to distribute it as soon as it is ready and functional. Our common planet is at risk, and reducing the cryptocurrency energy footprint is crucial for all.

Leaf implementation
There are many reasons to base our project on the cryptonote code-base. It has all the infrastructure needed for a coin. It has a lot of optimizations on the bitcoin protocol already coded, (for example a smooth decrease of the miners reward). Its code-base fits with our transaction mixing model, because we only need to substitute ring signatures with our hash based scheme and “key image” list with our list of spent pub-keys hashes. It also has the ideal PoW algorithm.

Our next step will be to implement blockchain foldability in order to make Leaf scalable from the beginning. This will reduce the usage of resources from the first stage.

When the new blockchain is up and running we will insert hash-based private contracts into the network, picking an external open source implementation of web-assembly, and defining the input output protocol for them.

Off_chain form of smart contracts, does not store itself on the blockchain but it uses it only as a source of enforcement. We think that this model will eventually substitute Etherum's serialized universal Turing machine, with a universal high parallelized computer.

After finishing our core implementation, our next priority will be the mixnet. It is necessary if we want cryptocurrencies to be really anonymous. And we really want this to be the initial spread of the next generation of anonymity networks.

The final step will be the integration of private fog storage services with our cryptocurrency. We aim to use existing code from open-source projects of the kind, or even cooperate with existing projects. We will interconnect the projects and stabilize Leaf's value on the market.

We hope that distributed storage will outperform centralized cloud services dominated by companies which steal users' data and we want to contribute in this direction.

GOVERNANCE OF THE PROJECT

We are not going to say something really new here. The problem of governance in the first and second generation of cryptocurrencies is well understood. It is about who decides the general direction of the project. While mining, verification and transactions are decentralized, there is actually a small team of people who decide what direction the code-base takes. In order to overthrow this problem, there have been many proposals. We consider Cardano's governing principles the best of them.

Our proposal will be based on smart contracts between investors and programmers who work on extensions or improvements of the code. If programmers fail, investors get a percentage of their money back. If programmers manage to do what they promise, they share the profits. But the final word belongs to the users of the network in a decentralized way. Burners of the network will vote on the hash of the code-base they want to switch to, in every mined block. By default, the hash to vote will be that of the current code-base. Every week there will be a decision of the network to switch or not to a new code-base. Hence we manage to keep the network up to date in a decentralized way. Consequently we will not stuck in an old and conservative code-base and protocol, but we can keep the project revolutionary and up to date, addressing security issues in time.

Epilogue

This is what we want to do, and now we want your help.

First of all, we wait for some feedback, Our ideas are extremely untested, so we really need criticism. Some code experiments are already on github. Test them and contribute. We need developers, so if you believe in our ideas, join us. Wait for the rest, and don't believe whoever pretends to be us without our pgp signature in the bottom of his message.