The Objectchain

Technical Whitepaper
Version 1.10
Jan 09, 2024

https://github.com/rubixchain/rubixgoplatform

Abstract

The Rubix Objectchain protocol is a deterministic state-machine that is designed to address the scale, cost, and privacy shortcomings of blockchain protocols that rely on one sequentially organized chain (monolithic) of all global transactions. The protocol divides the global state-machine into a large, but finite number of state-machines called Objectchains. While each Objectchain maintains one state, together all Objectchains represent a globally accessible singleton state that is immutable. This paper explains various components that make up the protocol.

Blockchain protocols such as Bitcoin[1] & Ethereum[8] in general achieve a globally accessible singleton state by organizing all global transactions sequentially as blocks (each block having a finite number of transactions) and organizing the blocks as a hashchain. Such blockchain protocols require exhaustive mining-based Proof-of-Work (PoW) consensus algorithms to secure the state, which results in high latency, low throughput, and high transaction costs. While the Proof-of-Stake (PoS)[7] consensus protocols may alleviate the energy & throughput issues associated with the PoW consensus, PoS protocols suffer from concentration (nodes with higher existing stakes may continue to gain larger voting power), security ("nothing-at-stake" & impersonation risks). Further, both PoW & PoS protocols require every node to store the entire global state, which results in significant storage & computational inefficiencies.

In contrast to the sequential transaction architecture of blockchains, Rubix Objectchain processes transactions in an asynchronous parallel manner. Each transaction achieves finality on its own without waiting to be pooled with unrelated transactions.

Objectchain

The Rubix global state will be made of at least 51.4 million Objectchains. Each Objectchain is bound by one unique utility token (alternatively described as a digital utility tool). An Objectchain O_T is made of all transactions that use token Tn to confirm. All transactions within an Objectchain O_T are validated individually and sequentially. However, transactions of different Objectchains are validated asynchronously and parallelly.

Objects

Object is the basic unit in the Rubix network. An Object can be a native utility token, asset token, data token, smart contract or identity (DID). These Objects are described in detail in the subsequent sections. Every Object, once created on the network, has genesis & its own Objectchain. The Objectchains can scale in parallel asynchronously, but always together form one global state at any point of time.

Tokens

Every transaction in the Rubix network is bound to one or a set of native utility tokens. There are three types of tokens: asset tokens, data tokens and utility tokens. There will be about 51.4

million native utility tokens (named Rubix or RBT tokens). Asset tokens represent both (a) unique digital assets like tickets, coupons, credits, vouchers or collectibles and (b) the digital form of any real-world assets like land, shares, vehicle etc.. Data tokens represent the underlying unique data objects that are stored on the chain. Asset and Data tokens are Non-Fungible Tokens (NFT) much like Ethereum's ERC721 tokens. They are unique and cannot be interchanged. Such tokens do not add any value on its own to the network and hence are not limited in supply nor are restricted in creation. When compared to its Ethereum model ERC 721, Rubix asset tokens are more dynamic. Unlike ERC721 which uses a parameter "value" during the creation itself, Rubix asset token's value can vary during the life, depending on the underlying utility token(s) value. Rubix asset and data tokens are bound to the Objectchains with the help of the utility token(s) (RBT).
An asset token A_i bound by utility tokens worth x units holds its price value at x. In the next transfer of token A_i, let us assume it is bound to a transaction with utility tokens worth y units; the value of the Asset token A_i then changes to y units. Thus, in Rubix platform, the current value of any Asset token depends upon the earlier transaction it was involved in. If an Asset token has not been transacted even once in the network, it effectively holds no value in the network. The properties of Asset tokens described in this para apply to Data tokens as well.
Rubix native utility tokens (RBT) are capped at about 51.4 million in total. With Rubix's breakthrough Objectchain architecture, the network does not require expensive miners or overpowering stakeholders with proof of stake protocols to maintain the network integrity. While a small portion of the tokens (56 in total) are pre-created to facilitate faster bootstrapping, all except the 56 tokens are mined by the nodes in the network. Since a full Rubix node can be set up even on a laptop with basic specifications, most computing nodes in the world would be eligible for being a miner.

RBTs are digital utility tools

The purpose of the Rubix network is to facilitate fundamental transformations in the way business processes are conducted in several industries including e-commerce, social, data, healthcare, advertising, media, financial services, payments, supply chain and ERP. The Rubix public blockchain is designed to fundamentally change the way identity, data & business processes are initiated, settled & recorded. RBTs are the digital utility tools that applications & businesses need to gain productivity. Businesses can either purchase the RBT tools needed for applications (capex), borrow (opex) or create by being validators (shared utility through security). The fundamental difference between RBT tools & the existing software tools is that RBT software tools are perpetual, reusable, saleable & have residual value. Fundamentally put forth, RBT software tools can be created, owned or borrowed by businesses much in the same way they deal with computer hardware & other factors of production like real estate and commodities. Software tools until now have no transferable rights & therefore no resaleable value, hurting business productivity. Rubix network changes the software and digital paradigm in a fundamental manner. RBTs are neither securities nor crypto currencies. They are simply a business utility like land and commodities that can either be transferred peer to peer or in any form of trading platform.

Proof of Pledge (PoP)

Rubix introduces the Proof of Pledge (PoP) protocol where all Rubix nodes are eligible to be validators (miners) & any node can be chosen as a validator (more on this explained later in this paper). Rubix validating protocol is eco-friendly & carries a low carbon footprint. Mining also means that nodes accumulate tokens in a decentralized & egalitarian method rather than by aggressively concentrating hash power to mine tokens. The PoP consensus is detailed in the subsequent sections.

Rubix is built to facilitate decentralized applications that power real world commerce at a significant scale on a decentralized network. Early app developers can earn pre-created tokens based on the velocity of transactions on their applications. Rubix consensus protocol deploys validators who engage in a consensus process to achieve PBFT[3], but the validators are not essential to prevent double spending. Rubix validators are not needed to prevent double spending (double spending is not possible in the Rubix Network) but are key to resolve forking or to prevent denial of service. The validators enhance the robustness & reliability of the network storing proofs & transaction data (referred to as mining). Hence the validators also can earn utility tokens based on their proofs of pledge.

Rubix Network will generate a total of approximately 51.4 million tokens. *Tokens are mined in perpetuity, but with a long tail*. The following conditions must be satisfied while pushing value-based tokens into any distributed trustless network. Firstly, the tokens should be near finite in supply; No node including from the Rubix core developers should be able to create additional tokens, beyond the specified total supply. Secondly, each of the RBT tokens should be uniquely identifiable and publicly verifiable at any point of time by all nodes.

While each RBT token is unique from another, they are all equal in their utility to validate transactions. In other words, all tokens are fungible with each other. The utility of a token T_x is also independent of the work done on the Objectchain O_T (number of transactions validated on the chain). For example, Objectchain O_{Tx} could be longer than O_{Ty}, but corresponding tokens T_x and T_y are equal in their utility for validating a new transaction.

To keep the integrity of the network, it is important to verify the genuine ownership of each of the RBT tokens. Rubix achieves this by representing its tokens in multi-hash format in the custom version of the Interplanetary File System (IPFS)[2] protocol. Each of the hashes in the Rubix chain is pushed into IPFS and committed.

Since IPFS follows content-based addressing, any alteration even on a single bit will result in an entirely different hash value that will not be verified by any of the object chains.

Nodes

A Node looking to perform token transfer , validator or be a smart contract participant must join the Rubix network. The ID of the node is published to the network via the Rubix network layer. The transactions involving the node are automatically stored in the chain distributed database. Since Rubix is a peer to peer protocol, nodes interact directly with each other through its end to end encrypted network channel assuring data privacy at the network layer. Nodes running the

Rubix application distribute storage , computation and pledging responsibilities in the network making Rubix more decentralized than monolithic chains.

Decentralized IDentity (DID)

Rubix DID is the unique identifier of nodes in the network. DID is self created and publicly verified. Rubix DID is generated by standard ECDSA P-256 curve. The public key , which in the Rubix ecosystem is termed as Decentralised Identity, is shared among peer nodes in the network. Nodes can create digital copies of real world assets / data points / identifiers as Non Fungible Tokens and link them to their DID.

Rubix Transaction

All Rubix nodes join the network to conduct transactions with each other. A transaction can be (a) a digital contract that involves exchange of services or goods in return for exchange of other services or goods (b) a digital contract that involves exchange of services or goods for exchange of any medium of value such as fiat currency or simple transfer of native token.

At any given point of time, several peer-to-peer transactions are submitted to the Rubix network. Transactions are processed in parallel independent of each other unless transactions involve common peers. A transaction is initiated by one peer node. To initiate the transaction, the peer node must use at least one RBT token. Depending on which RBT token (T_i) is used by the peer node initiating the transaction, the transaction is added to the corresponding Objectchain O_{Ti}. If multiple tokens are used in a transaction, the transaction is added to multiple Objectchains.

Objectchain

An Objectchain O_{Ti} is a chain of all transactions bound by the utility token T_i (note that there will be a total of about 51.4 million RBT utility tokens in the Rubix Network). A small set of RBT tokens are committed by the genesis node G. All Objectchains with the RBT tokens committed in the genesis node G originate with the genesis node. All such RBT tokens are stored and committed on the IPFS by the genesis node G. The node G’s ownership of such tokens is globally verifiable. All those RBT tokens that are not pre-committed to the node G are mined by validators. In such cases, once the RBT token is mined, the corresponding Objectchain of that token starts with the validators node that has mined the token.

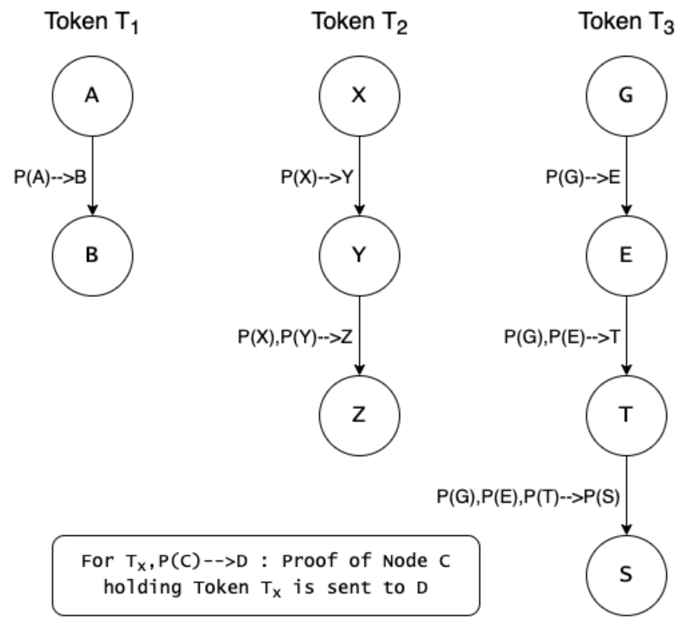


Figure 3: Independent Objectchains of Tokens

Rubix Consensus Protocol

Rubix introduces the lightweight, low carbon footprint consensus protocol, a democratic and decentralized Proof of Stake (PoS) variant. Validators perform work to validate the transaction by verifying (a) the ownership of Objects or tokens in the transaction (b) resolve forks if any by verifying the Objectchains. Validators also need to pledge RBT tokens to secure the chain. The total amount of RBTs pledged by the validators should be higher than the value of the transaction to be legit. Validators earn credits for pledging the RBTs to secure the network. Proof of Pledge (PoP) is not the same as PoS. PoS requires validating nodes to stake native tokens continuously, in order to validate & also to earn rewards. Further the power of the nodes with higher coin stakes continue to increase, leading to more concentration.

PoP is different from PoS. In PoS, the entire network is secured by a set of validators, where the number of votes is equal to the amount of native token staked. The stakers would return based on the amount of native tokens staked. The issue with staking in monolithic chains is that validators with the largest amount staked would continue to get more rewards and get bigger over time. Concentration of larger validators would increase over time, eventually making the chain more centralized. Since one monolithic chain is secured eventually by large validators or pools, it is easier for malicious actors to hack their wallets (or keys). This also means higher security risk.

On the other hand, with PoP, pledging is local to each Objectchain. Validators pledge their RBTs to secure each Objectchain. Since different validators are securing different Objectchains, the network is inherently more decentralized. The Rubix chain is therefore more decentralized and more secure. Further, validators pledging each Objectchain can always revoke their pledge as long as it is replaced by pledge from another set of validators.

Quorum Selection

In Rubixchain, quorum selection can be done in 2 days. In type 1 where nodes are in open Rubix network a Pseudo Random Function (PRF) is used to determine the set of validators. The validators are finalized if they have enough tokens for pledging. A new round of PRF is performed until a group of 7 validators with required pledgeable tokens is found. In type 2 where nodes are interacting within a subnet , the subnet can overwrite the rules of PRF and choose validator sets deterministically.

Periodic Pledging

In Proof of Pledge , the validators' nodes pledge their tokens against the state of objectchain for only a specific time interval. When a token is transferred (token state updated) , the validators that verify this state change attest their pledge for this state only for a period 7 days. Note that the validators need to pledge for state update of a token and not the entire object chain. By pledging the validators are notifying the entire network that the previous state of token is exhausted and thereby preventing double spending. After the 7 day period , a new self transfer state update is initiated by the token owner which finds a new set of quorum to pledge state for the next interval. Tokens in the rubix network are advised to perform a new state sync function call every 7 days in order to have a set of validators pledge against the new state.

Proof credit Accumulation

As explained in the above section, credits are earned via periodic proof of pledge consensus algorithm. The calculation of credits depends on how long the state was pledged by the validators. In scenarios where a token is not transferred within the pledge interval (7 days) , the validators earn one whole credit. Whereas in scenarios where a token is transferred within 7 days , only a fraction of credit will be accumulated. The net credit accumulation per token per day is 1 credit per periodic interval.

Since type 1 (open network) quorum operations involve more operations and makes the network truly democratic and decentralized , the transactions performed in type 1 network earn maximum credits per transaction (up to 1 credit per token for 7 days) . The incentive for quorum nodes to take part in consensus is solely to earn credit and contribute for network growth. Whereas in type 2, a subnet of systems (especially enterprise applications built on rubix) can overrule the PRF and select the validator from a predetermined set. This is advisable for networks that require high tps, low Round Trip Time (RTT) in communication. Note that the type 2 validator nodes can also participate in open network quorum selection and be part of maximum credit yielding transactions. In addition to this, subnet validators get a centi credit which is 1/100th of a credit point. At the time of mining verification, mining validators can verify the type of quorum selection and count credits accordingly.

Mining : Conversion of proof credits into RBT tokens

Each validator in the Rubix transaction stores the proofs to secure the Objectchain. By burning energy on their computing systems, using the memory to store the proofs & pledging RBTs to secure the transactions, validating nodes participate in securing the Rubix network. In reward for the proofs of pledge, each validating node mines a new RBT token after earning a certain threshold of proof credits. The conversion of proof credits into a RBT token is a new transaction in the Rubix Network undergoing a consensus of its own. The proof credits needed to mine a new token are set initially lower, but increase subsequently with the difficulty level needed to mine is set gradually higher in perpetuity. The number of proof credits required to mine a new RBT token is tabulated in Annexure **1** (the threshold levels, **τ**, are tabulated).

Mining is a special native NFT of Rubix network which can track and publish the list of mined tokens so far. The mine NFT objectchain consists of an iterative proof of each time a new RBT is mined and the credits used to mine the RBT.

Steps of Mining Procedure Explained

1. Rubix native mining NFT consists of a list of all tokens mined so far and credits so far.
2. The mined tokens list helps in identifying the token to be minted next. The tokens follow structure: hash (token level + token number).
3. When a node accumulates enough credits to mine new tokens, it calls mine operation in the Mining contract that performs 3 operations (a) initiates a consensus mechanism where the quorum/validators verify credits and calculate credit points. (b) calculates the next token level and number to be minted (c) Validators pledge for token minting transfer
4. Once the token is minted successfully, the contract is automatically updated with credits exhausted in mining procedure and also increment the token number

Double spending

Double spending is not possible since any node wishing to enter a transaction to acquire token(s) can verify if the token is committed by only one node or by multiple nodes at any point of time. If the token is also committed by another node which is not part of the transfer, acquiring node can verify easily and get alerted of a possible attempt to double spend.

Say C is the owner of a token (sole committer on Rubix chain). A transfer's a token to D (C uncommits and D commits). This makes D new owner of the token. If C were to make a digital copy of the same token and commit again, there will be two committers of the same token C and D. Now when a third node E wishes to acquire the token from the legitimate owner D, E finds out that two nodes C and D are committed to the token. Node E will not proceed with the transaction unless C and D provide further proof. Hence double spending is not possible.

There is no economic benefit for C to fraudulently commit the token again as C is not able to transfer the same token twice.

While double spending is easily prevented in the Rubix Objectchain protocol, there is a possibility that a malicious node can initiate forks to carry out denial of service or spam attacks. Denial of service attacks are possible if user create a fake node and transfer same token to fake node. Say node C create fake nodes C’ just to prevent F from transferring token further. In such cases, node F will challenge node C’ to provide it’s Objectchain. The Objectchain of node C’ will be P(G) > P(A) > P(B) > P(C). However, any node in the network can determine where the fork(s) occurred. The nodes initiating fork & the corresponding validators can easily be determined.

Double Spending Prevention by Validators Locking Token State

Validators for a transfer from A to B for a token T marks the current token state of Token (state where A was receiver) as exhausted in the network. This allows other genuine nodes to identify and prevent a double spend attempt by A.

Preventing Sybil/DoS Attacks

Any honest node will find it easy to enter a double spending transaction in the Rubix Network. However, malicious nodes that participated in a previous transaction in the chain could create sybil nodes & attempt to fork the chain (a node which has not been part of the Objectchain can’t fork).Such forking will not result in a direct benefit for the malicious forking node – preventing an honest token holder to transact further could be attempted though. However, Rubix Network protocol makes forking economically impossible using the concept of validators as discussed earlier in this paper. No validator will approve the consensus of a forking transaction due to the risk of losing the RBTs pledged to secure the transaction. The malicious node setting up other Sybil nodes to become validators would also find it economically impossible to get the forking approved. Since, it is easy for all the nodes in the Network to know the forking nodes & the validators that assisted in the transaction, the risk of being blacklisted for further transactions is an additional deterrent. Also, given the decentralized nature of the Rubix Network with little concentration, a global DoS attack to halt the network is almost impossible to succeed. As Rubix Network can expand to billions of full validating nodes, a coordinated global attack is highly impossible due to (a) lack of net economic gains & (b) the need to set up an extremely large number of independent computing nodes.

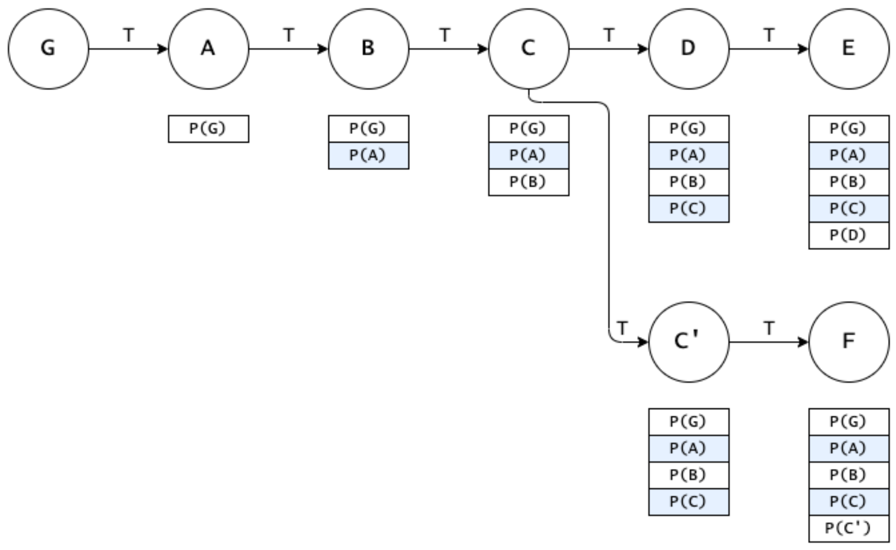


Figure 4: Fork Detection

Creditchain: Securing the proof credits

The Proof of Pledge (PoP) algorithm is a key component of security in the Rubix protocol. Ensuring the proof credits are legit & secure is a key consideration in the protocol. In order to ensure that the proof credits α of are valid, any node in the network (including the transacting parties in the new transaction & new quorum) can verify if the creditchains presented by the previous quorum members are not tampered. Since the new receiver & the quorum members are honest, they will verify if the creditchain provided by each of the validators is accurate by fetching the quorum list from the previous transaction's sender. Once acquiring the quorum list of the previous transaction, their signatures in the previous transaction can be verified. If the signatures match, then the legitimacy of the proof credit count of the concerned validator is determined to be accurate. It is evident any node in the network can ascertain if the credits declared by each validator are valid by simplifying cross checking the credit chain with the previous transaction details.

Staking for mining

Conversion of proof credits to RBT tokens is similar to a regular transaction in the Rubix network as explained earlier in this whitepaper. Additionally though, nodes converting proof credits into RBT tokens need to pledge RBT token for a minimum number of transactions of the newly minted RBT token. Technically, the RBT tokens are pledged temporarily, not staked as such.

Once the minting node submits new proof credits to mine, the validator nodes signing the minting transaction will need to cumulatively pledge (or stake) 1 RBT token. The pledged RBT token by quorum is not transferable for at least 4*h transactions where h is the height of the difficulty level at which the minting transaction is initiated. For example, at level 4, h = 64, hence pledged tokens will be locked for the first 256 transactions of the newly minted token. Without proof of the staked tokens, the newly minted token can't be transferred.

The staked tokens are automatically released for transactions after 4*h transactions or the completion of the difficulty level, whichever occurs earlier. The staking process significantly increases the security of mining in the Rubix chain. What happens if the node looking to mine credits does not have quorum nodes with at least one token to stake? The quorum can be populated by borrowing the nodes that have the required tokens to stake for 4*h transactions.

PBFT Consensus

Consensus involves agreement between multiple parties in a distributed network. This can be achieved by running a single chain with blocks created one after the other as an agreement to the previous blocks on the chain. However, such chains never reach a state of finality. Plus, the scalability factors for all the nodes to synchronize with other peers in the network makes these models inoperable. In Rubix Network, consensus protocol is run for each transaction independently. This way each transaction can be independently verified, thereby reducing the scope for forks.

Rubix Consensus involves a Sender (S), Receiver (R), seven validator nodes (**v**$_1$, **v**$_2$, **v**$_3$, **v**$_4$, **v**$_5$, **v**$_6$, **v**$_7$ (7 as per PBFT rule). Quorum Initiator creates the Transaction Data in the following way:

Transaction Data = SHA3-256(Sender's WalletID + Token(s) + Objectchain Height)

Steps:

1. Sender picks quorum members **v**$_1$, **v**$_2$, **v**$_3$, **v**$_4$, **v**$_5$, **v**$_6$ & **v**$_7$. The set of validators can be picked from any public set of validators available on the Mainnet of the Rubixchain or from any pre-determined subnets.
2. Sender calculates the SHA3-256 hash H of T, S_w and the Objectchain height O_h and sends H, Sender's Signature from H and Receiver ID, R_w,to all the quorum members.
3. The quorum members verify for unique ownership and state of token and signs using its private key. This way the quorum agrees to the transaction of the sender and the token.
4. When 5 or more signatures are received by the sender, the node performs Peer-Peer authenticated token transfer and IPFS Committing & Uncommitting.
5. Once the PBFT count is reached, the Rubix Network consensus is successful.
6. Subsequent to the successful consensus, the successful validators store the proof & transaction data, along with the latest proof credit count.

Simple token transfer transaction

Let us say node A would like to enter a transaction with node B. To validate the transaction, A needs at least one token (more than one token may be needed in certain transactions). For the current illustration, let us say one token is needed. A will enter a transaction to procure the required token T_x from the genesis node G. Nodes G and A enter a Peer-Peer transaction. Before entering the transaction, A requests the IPFS hash of the token T_x from G. A verifies if node G owns the token by checking if G and only G has committed T_x on the IPFS. If true, A proceeds to complete the transaction with G. If A finds that node(s) other than G found committing T_x, A will request for additional proofs of ownership from G and other nodes who are found committing T_x.

If no fork is found and G is the sole committer of T_x, then G proceeds to complete the transaction with A.

1. For a user in the Rubix network, after successfully completing the DID generation , a gossip protocol broadcasts the Public information across the network.
2. Initially the sender sends the token's IPFS hash to the receiver and the receiver acknowledges the availability of the token
3. The token's latest proof (ObjectChain) is known only to the sender and therefore, after the acknowledgement the latest ObjectChain is sent to the receiver for picking challenge-response positions.

4. The Objectchain represents the universal proven state of each token, which is publicly verifiable.

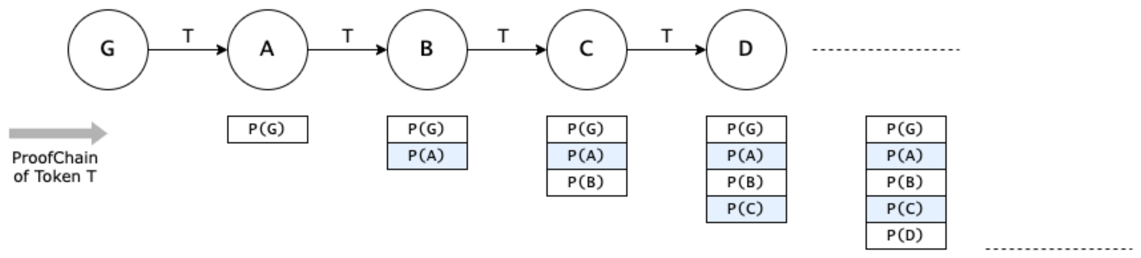


Figure 6: Objectchain Formation

5. If token T is purchased from Rubix genesis node G by user A. Node A gets empty object chain from G creates a new chain with proof P(G)
6. Now, when user A transfer token to user B, the proof for this transaction P(A) is appended to the existing Objectchain. Hence the current Objectchain contains two proofs, P(G) > P(A)
7. The sender calculates hash the transaction details which includes information about token(s) , sender , receiver and validators.
8. Receiver information (R) and Token information (T) are used in hash calculation to ensure that the agreed token is being transferred to the recipient.
9. For instance, A is transferring a token, T to B [A (T)—> B]. The inputs for the hash calculation are B, T and P.
10. Any other user out of this transaction, C cannot claim token T has been transferred to C by A [A (T)—> C].
11. B cannot claim A has transferred some other token by A [A (~T)—>C].
12. Sender A picks a set of 7 nodes as validators by Rubix Pseudo Random Quorum Selection Algorithm and shares the transaction information with them. The Validators check for unique ownership and verify the state exhaust flag before approving the transaction. Approving a transaction involves nodes pledging tokens that are cumulatively equal to transaction value.
13. Once Sender receives 5 approval votes from the validator group , the same data is shared to Receiver who verify the sender and quorum signatures and pledges.
14. The token to be transferred is uncommitted by the sender and committed by the receiver thereby claiming the ownership of the token.

The transaction proof generated in the process detailed from Step 1 to Step 20 is deterministic and highly secure. The proof confirms that A has received T_x from G. A can only claim that it received T_x from G and not any other token T_y. This is because A will not be able to produce the knowledge of the 256 bits of G that correspond to token T_y..

After the successful completion of the transaction $T_{øA}$, A creates Objectchain P(G) and both G and A store the Objectchain. When A further transfers token T_x to node B, the Objectchain expands to P(G) + P(A) and nodes A and B store the updated Objectchain. When B further transfers to C, Objectchain expands to P(G) + P(A) + P(B) and nodes B and C store the updated Objectchain. The Objectchain continues to expand perpetually. Note that the updated Objectchain is not propagated back to all the preceding nodes in the Objectchain.

The Objectchain for token T_y is constructed like that of T_x explained so far. The Objectchain represents the universal proven state of each token, which is publicly verifiable.

IPFS in the Rubix Network

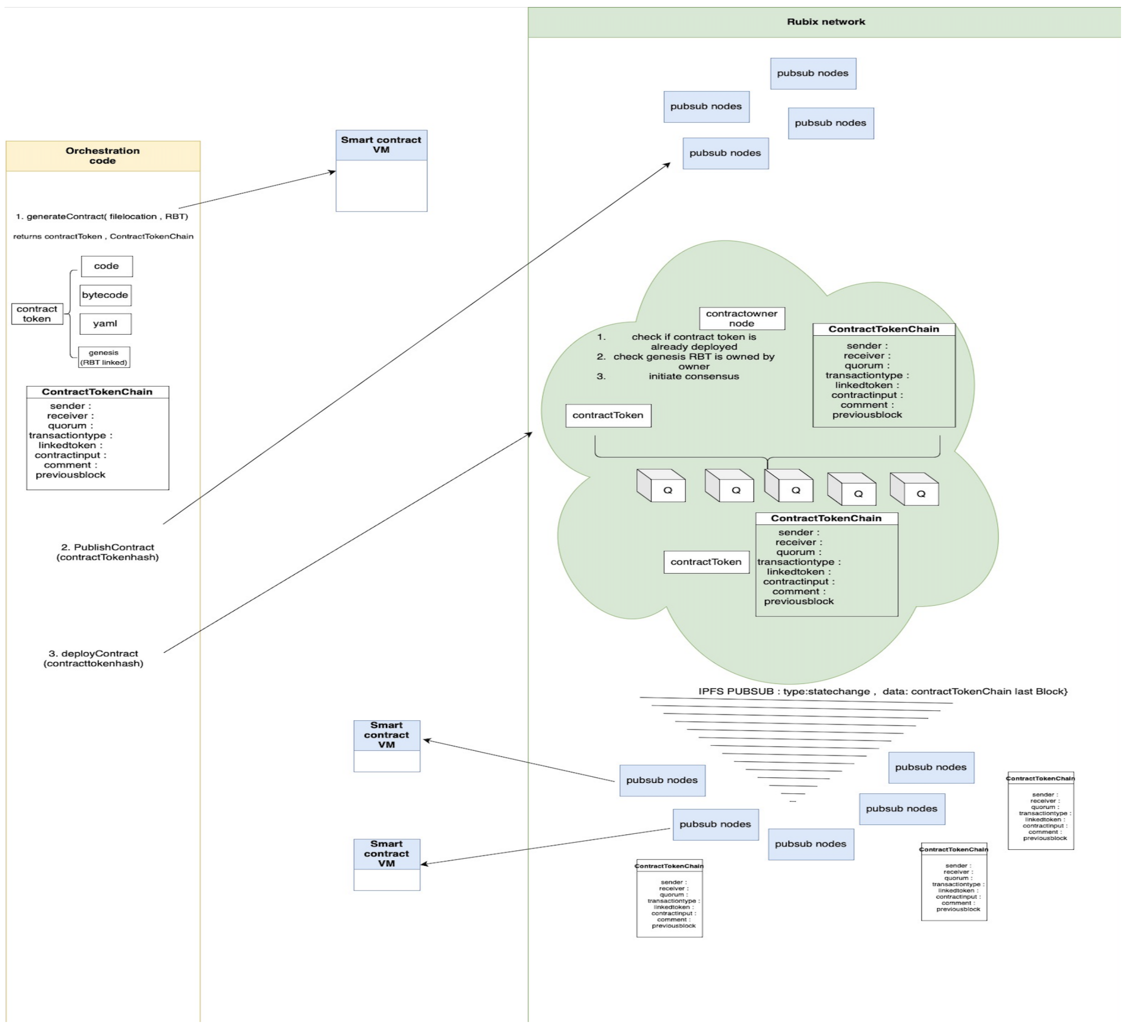
IPFS plays a key role in Rubix network due to below properties of IPFS:

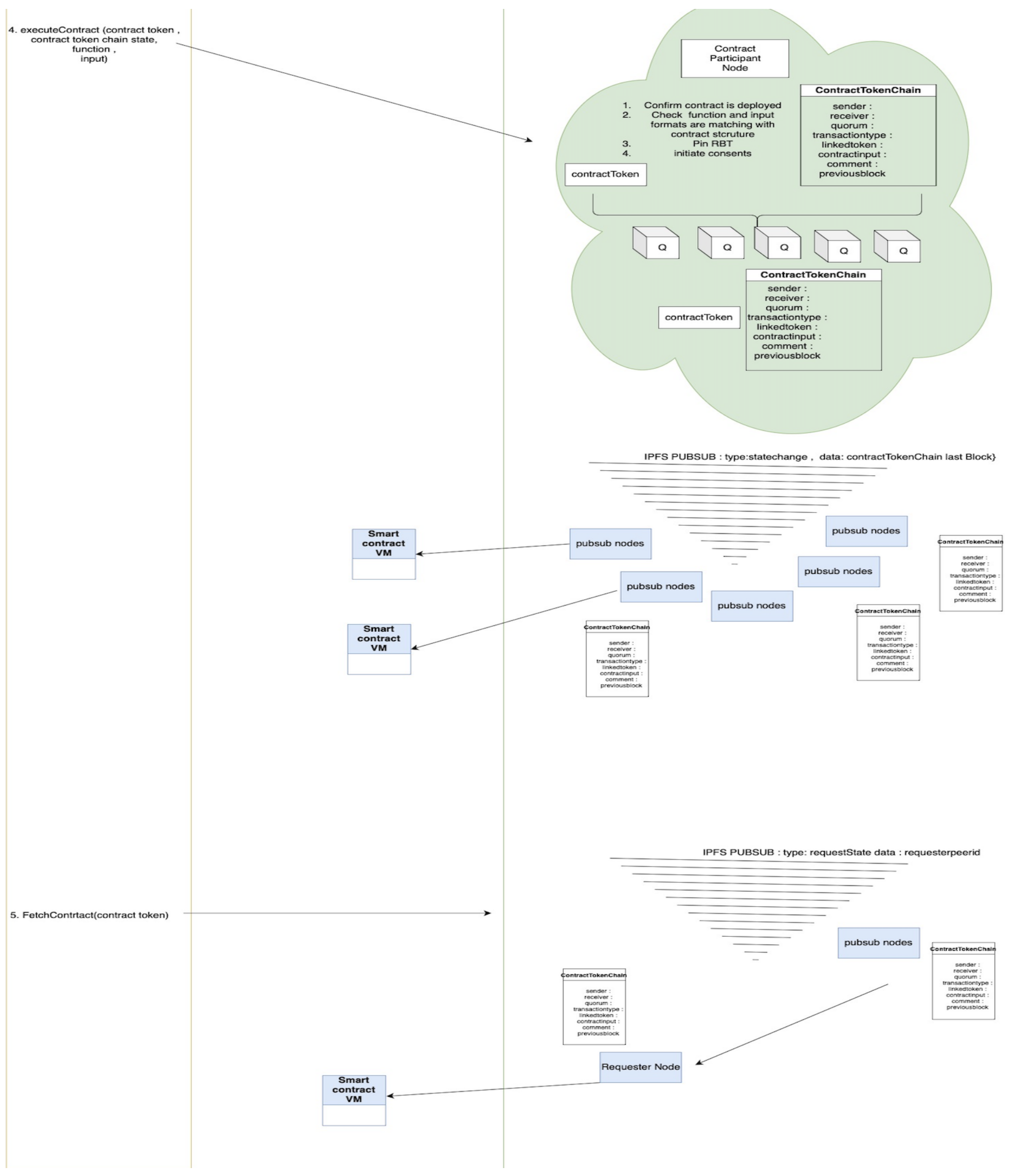
1. Immutable objects represent files
2. Objects are content addressed by cryptographic hash
3. Distributed Hash Table (DHT) to help locate data requested by any node and to announce added data to the network
4. IPFS removes redundant files in the network and does version control
5. Content addressing of the data stored in IPFS, every file name is unique if there is even a single character change in the content of the file
6. Private IPFS that can only be accessed by certain entities
7. Committing of the data - avoids double spending

Smart Contracts

Smart contracts represent sophisticated business logic encapsulated in machine-readable formats, typically articulated through programming languages. These contracts are executed within a network's nodes, operating in a deterministic, sandboxed environment.

Rubix smart contracts are treated as a specialized form of Non-Fungible Tokens (NFTs) possessing a dynamic state. Every invocation of a contract function leads to an update in this state, which is meticulously recorded and preserved on the Contract Object Chain. This dedicated chain furnishes an immutable ledger, ensuring transparent and tamper-proof documentation of each contract execution. To ensure versatility and adaptability, Rubix smart contracts are crafted in prevalent web2 languages, including Rust, JavaScript, and GoLang. These contracts are subsequently executed within a WebAssembly (WASM) environment.


Orchestration code
1. generateContract(filelocation , RBT)
returns contractToken , ContractTokenChain
contract token
code
bytecode
yaml
genesis (RBT linked)
ContractTokenChain
sender :
receiver :
quorum :
transactiontype :
linkedtoken :
contractinput :
comment :
previousblock
2. PublishContract (contractTokenhash)
3. deployContract (contracttokenhash)
Smart contract VM
Rubix network
pubsub nodes
contractowner node
1. check if contract token is already deployed
2. check genesis RBT is owned by owner
3. initiate consensus
contractToken
Q
IPFS PUBSUB : type:statechange , data: contractTokenChain last Block)
Smart contract VM


4. executeContract (contract token , contract token chain state, function , input)
Contract Participant Node
1. Confirm contract is deployed
2. Check function and input formats are matching with contract stcruture
3. Pin RBT
4. initiate consents
contractToken
ContractTokenChain
sender :
receiver :
quorum :
transactiontype :
linkedtoken :
contractinput :
comment :
previousblock
Q
IPFS PUBSUB : type:statechange , data: contractTokenChain last Block}
pubsub nodes
Smart contract VM
5. FetchContrtact(contract token)
IPFS PUBSUB : type: requestState data : requesterpeerid
Requester Node

Smart contract Life Cycle

Rubix protocol is committed to improve adaptability of blockchain technology. Along with its revolutionary proof of pledge protocol aided by zero gas fee transactions , Rubix focuses on making dApp deployment and execution easier for our ecosystem. With WebAssembly(WASM) based smart contracts, existing web2 codebases and developers can migrate their codebase and knowledge into Rubix with ease. WebAssembly (WASM) is a binary instruction format that allows code to be executed at near-native speed in a safe , sandboxed and deterministic manner across different platforms. Smart contracts can be written in languages that compile to WebAssembly, such as Rust and C/C++, and then executed on a blockchain platform that supports WASM.

Here are the steps in executing WASM in Rubix

1. **Writing Smart Contracts:** Smart contracts are written in high-level programming languages like Rust, GoLang or C/C++. These languages offer the flexibility and expressive power of high-level languages while compiling down to WebAssembly bytecode.

2. **Compiling to WebAssembly**: Once the smart contract code is written, it is compiled to WebAssembly bytecode. Compilers like rustc for Rust or Emscripten for C/C++ can be used to generate WebAssembly binaries from the source code.

3. **Deployment on Blockchain:** The compiled WebAssembly code is then deployed onto a blockchain platform. The contract code, along with any necessary metadata, is stored on the blockchain.

4. **Execution:** When a user or another contract interacts with the deployed smart contract, the contract's functions are called via transactions. These transactions contain input data that specifies which function of the contract to execute and with what parameters.

5. **Validation:** The transaction is validated by the blockchain nodes to ensure it follows the rules of the blockchain protocol. Once validated, the transaction and the associated smart contract function call are processed by the nodes

6. **State Change and Output**: Smart contracts can read data from the blockchain's state and modify it as per the logic defined in their functions. Smart contracts can also produce output data, which is typically returned to the caller after the contract function execution is complete. The smart contract is executed on the DApp side. The DApp should have an api endpoint which must be passed as a parameter to register-callback-url api. This api registers the api endpoint in the node. Once the endpoint is registered, each time an execution happens on the object chain, as per the logic in the smart contract deployed, the states in each of the subscribed nodes get updated.

Once deployed, the logic of a smart contract, represented by its WebAssembly bytecode, is immutable. This means it cannot be changed. If you need to update the contract's logic, a new version of the contract needs to be deployed.

To learn more about smart contracts and APIs to interact with Rubixchain , please visit https://learn.rubix.net/smartcontract/

Objectchain for Smartcontracts and Data Tokens

Rubix smart contracts are programmable NFTs that hold a sequential and deterministic proof of state execution. The sequential proofs provide irrefutable logs of a list of related events, be it as raw data or calls to the linked smart contract function. In case of smart contract NFTs (dynamic NFTs) , these proofs can be rerun by any node in the network to arrive at the present state of smart contract token. This means a node that syncs to the network can fetch the proofs , execute them one after another in its sandboxed environment and calculate the current state of contract. For static NFTs or data tokens in Rubix, the state chain represents a change of ownership or value of the NFT.

Pledging for Smartcontracts and Data Tokens

The value of a NFT is determined by the owner or creator of the NFT. For data tokens the value of NFT changes every time the ownership of the token is changed. Since every state update is performed with a linked RBT, pledging security for data token NFTs are secured by underlying RBT transfer.

In case of smart contract NFTs, the token ownership is never transferred. The state changes to the NFT object chain include the sequence of smart contract function calls. The smart contract provider needs to lock an amount of RBT to the contract (making RBT non transferable until NFT is burnt) that quantifies the total value of data points linked to the smart contract. These are customizable figures and depend on terms between smart contract apps and consumers.

Example : If a smart contract app charges 0.001 RBT for 2000 write operations to its consumers , the app needs to lock 20 RBTs to cover the value of data.

Rubix Network and Private IPFS

Every node who joins the Rubix network will be part of a private IPFS and therefore they are not connected to the external IPFS network and communicate only to those nodes connected to this private IPFS Swarm. All data in the private network will only be accessible to the known peers on the private network.

LIBP2P Protocol

LIBP2P is a modular network stack of protocols, libraries and specifications that enable development of peer-to-peer network applications. A peer-to-peer network in which the participants of the network communicate with each other directly without involvement of a privileged set of servers as in the case of a client-server model. LibP2P uses public key cryptography as the basis of peer identity, serving two complementary purposes. Each peer is given a globally unique name (PeerID) and the PeerID allows anyone to retrieve the public key of the identified peer enabling secure communication between the peers where in no third party can read or alter the conversation in-flight. To route the data to the correct peer, we need their

PeerID and the way to locate them in the network which is done in LibP2P using Peer Routing (discover peer addresses by leveraging the knowledge of other peers). Peer routing in LibP2P is done using Distributed Hash Table (DHT) (iteratively route requests to the desired peer ID using Kademlia routing algorithm).

LibP2P module is used in the Rubix network for communication of data from one end to another. The Rubix network traffic is tunneled through LIBP2P stream. We add all nodes who are part of the Rubix network, to a single private swarm network of IPFS. The communication over the internet from initiator to the notaries and participants during the consensus and from initiator to the receiver during the token transfer are performed using the IPFS listen and forward which is part of the libp2p library.

LIBP2P

1. **Listen for Incoming Streams**

   ```
   > ipfs p2p listen /x/<applicationName>/1.0 /ip4/127.0.0.1/tcp/<port>
   ```

2. **Forward**

   ```
   > ipfs p2p forward /x/<applicationName>/1.0 /ip4/127.0.0.1/tcp/<port>/p2p/<peerid>
   ```

SwarmConnect – Connection to Peer node either directly or through Realy service

1. **Relay**

   ```
   > ipfs swarm connect /p2p/<BootstrapNode>/p2p-circuit/p2p/<peerid>
   ```

   BootstrapNode – multiaddress of the bootstrap for relaying

   Peerid - ID of the node to connect

2. **Direct Connection**

   ```
   > ipfs swarm connect <Node>
   ```

   Node - multiaddress of the node to connect

3. **Swarm Peers**

   ```
   > ipfs swarm peers
   ```

   List of the peers a node can connect (fetched from DHT)

Token Access Operations and Commands

Add

```
> ipfs add <tokenname>
```

Returns Multihash of the file for reference

Get

```
> ipfs get <tokenhash>
```

Fetches the file from IPFS referenced using Multihash

Pin

```
> ipfs pin add <tokenname>
```

Pins the token– will not be removed during garbage collection

UnPin

```
> ipfs pin rm <tokenname>
```

Unpins the token– removed during garbage collection

References

1) Nakamoto, S. (2008) Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

2) Benet, Juan. "Ipfs-content addressed, versioned, p2p file system." arXiv preprint arXiv:1407.3561 (2014).

3) Castro, Miguel, and Barbara Liskov. "Practical Byzantine fault tolerance." OSDI. Vol. 99. No. 1999. 1999.

4) M. Tompa and H. Woll, How to share a secret with cheaters, Journal of Cryptology, Vol 1, Issue 2, Aug. 1988 https://dl.acm.org/citation.cfm?id=56181

5) Goldreich, Oded, and Yair Oren. "Definitions and properties of zero-knowledge proof systems." Journal of Cryptology 7.1 (1994): 1-32.

6) US Patent 009800408: Method of generating secure tokens and transmission based on (TRNG) generated Tokens and split into shares and the system thereof.

7) https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ

8) Keedwell, A. Donald, and József Dénes. Latin squares and their applications. Elsevier, 2015.

9) Wood, Gavin. "Ethereum: A secure decentralized generalized transaction ledger." Ethereum project yellow paper 151.2014 (2014): 1-32.

10) Renvall, Ari, and Cunsheng Ding. "A nonlinear secret sharing scheme." Australasian Conference on Information Security and Privacy. Springer, Berlin, Heidelberg, 1996.

11) Moni Naor and Adi Shamir. Visual cryptography; Workshop on the Theory and Application of Cryptographic Techniques. Springer. 1994, pages 1–12; and Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized Computation Platform with Guaranteed Privacy; preprint arXiv: 1506.03471, 2015.

Annexure 1: Difficulty Threshold Table (Proof Credits)

Level	Cumulative tokens ('000)	Tokens awarded ('000)	PCs/token
0	0.056	0.056	
1	4,300,000	4,300,000	0.125
2	6,725,000	2,425,000	16
3	9,028,750	2,303,750	32
4	11,217,313	2,188,563	64
5	13,296,447	2,079,134	128
6	15,271,625	1,975,178	256
7	17,148,043	1,876,419	512
8	18,930,641	1,782,598	1,024
9	20,624,109	1,693,468	2,048
10	22,232,904	1,608,795	4,096
11	23,761,259	1,528,355	8,192
12	25,213,196	1,451,937	12,288
13	26,592,536	1,379,340	18,432
14	27,902,909	1,310,373	27,648
15	29,147,764	1,244,855	41,472
16	30,330,375	1,182,612	62,208
17	31,453,857	1,123,481	93,312

Level	Cumulative tokens ('000)	Tokens awarded ('000)	PCs/token
18	32,521,164	1,067,307	139,968
19	33,535,106	1,013,942	209,952
20	34,498,350	963,245	314,928
21	35,413,433	915,082	472,392
22	36,282,761	869,328	590,490
23	37,108,623	825,862	738,113
24	37,893,192	784,569	922,641
25	38,638,532	745,340	1,153,301
26	39,346,606	708,073	1,441,626
27	40,019,275	672,670	1,802,032
28	40,658,312	639,036	2,252,541
29	41,265,396	607,084	2,815,676
30	41,842,126	576,730	3,519,595
31	42,390,020	547,894	4,399,493
32	42,910,519	520,499	4,949,430
33	43,404,993	494,474	5,568,109
34	43,874,743	469,750	6,264,122
35	44,321,006	446,263	7,047,138

Level	Cumulative tokens ('000)	Tokens awarded ('000)	PCs/token
36	44,744,956	423,950	7,928,030
37	45,147,708	402,752	8,919,034
38	45,530,323	382,615	10,033,913
39	45,893,807	363,484	11,288,152
40	46,239,116	345,310	12,699,171
41	46,567,160	328,044	14,286,567
42	46,878,802	311,642	15,179,478
43	47,174,862	296,060	16,128,195
44	47,456,119	281,257	17,136,207
45	47,723,313	267,194	18,207,220
46	47,977,148	253,834	19,345,171
47	48,218,290	241,143	20,554,245
48	48,447,376	229,085	21,838,885
49	48,665,007	217,631	23,203,815
50	48,871,757	206,750	24,654,054
51	49,068,169	196,412	26,194,932
52	49,254,760	186,592	26,587,856
53	49,432,022	177,262	26,986,674

Level	Cumulative tokens ('000)	Tokens awarded ('000)	PCs/token
54	49,600,421	168,399	27,391,474
55	49,760,400	159,979	27,802,346
56	49,912,380	151,980	28,219,381
57	50,056,761	144,381	28,642,672
58	50,193,923	137,162	29,072,312
59	50,324,227	130,304	29,508,397
60	50,441,500	117,273	29,951,023
61	50,547,047	105,546	30,400,288
62	50,642,038	94,992	30,856,292
63	50,727,530	85,492	31,319,137
64	50,804,473	76,943	31,788,924
65	50,873,722	69,249	32,265,758
66	50,936,046	62,324	32,749,744
67	50,992,138	56,092	33,240,990
68	51,042,620	50,482	33,739,605
69	51,088,054	45,434	34,245,699
70	51,128,945	40,891	34,759,385
71	51,165,747	36,802	35,280,775

Level	Cumulative tokens ('000)	Tokens awarded ('000)	PCs/token
72	51,198,868	33,121	35,809,987
73	51,228,677	29,809	36,347,137
74	51,255,506	26,828	36,892,344
75	51,279,651	24,146	37,445,729
76	51,301,382	21,731	38,007,415
77	51,320,940	19,558	38,577,526
78	51,338,543	17,602	39,156,189
Every subsequent level, token supply declines by 10% and threshold level increases by 1.5%			