The Proofchain

Technical Whitepaper
Version 1.8
May 3, 2022

https://github.com/rubixchain/rubixnetwork

Abstract

The Rubix Proofchain 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 of all global transactions. The protocol divides the global state-machine into a large, but finite number of state-machines called Proofchains. While each Proofchain maintains one state, together all Proofchains 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 inefficiencies.

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

ProofChain

The Rubix global state will be made of about 51.4 million Proofchains. Each Proofchain is bound by one unique utility token. A ProofChain P_T is made of all transactions that use token Tn to confirm. All transactions within a Proofchain P_T are validated individually and sequentially. However, transactions of different Proofchains are validated asynchronously and parallelly.

Tokens

Every transaction in Rubix network is bound to one or a set of tokens. There are two types of tokens: Asset 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.. These 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 use 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 Tokens are bound to the proofchains 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.

Rubix native utility tokens (RBT) are capped about 51.4 million in total. With Rubix's breakthrough Proofchain 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.

Rubix introduces Proof of Pledge (PoP) protocol where all Rubix nodes consent to be validators (miners) & any node can be chosen as a validator based on the quality of network, system requirements & balance of proof credits left (more on this explained later in this paper). Rubix mining protocol is eco-friendly & carries low carbon footprint. Mining also means that nodes accumulated 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 prevent forking or denial of service. The validators can enhance the robustness & reliability of the network by storing proofs & transaction data (referred to as mining). Hence the validators also can earn utility tokens based on their proofs of pledge.

Rubix 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 the Rubix developers should be able to create additional tokens (other than miners). 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 Proofchain P_T (number of transactions validated on the chain). For example, Proofchain P_{Tx} could be longer than P_{Ty}, but corresponding tokens T_x and T_y are equal in their utility for validating a new transaction.

To keep integrity of the network, it is important to verify the genuine ownership of each of the pre-mined million Rubix utility 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 Rubix chain is pushed into IPFS and committed by any one of the token issuance nodes.

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 token chains. Once all the hash values are generated, they are passed into a bloom filter. Bloom filter is a memory efficient probabilistic data structure that tells whether an element is present in a set or not. The set generated by passing all the hash values to bloom filter can be used by any node in the network to perform the first level of verification of the token in an efficient manner.

Nodes

A Node looking to transact must register on the Rubix network. Rubix full nodes can be set up on most laptops & desktops. A Rubix node will automatically be registered as a peer-node on the Inter Planetary File System (IPFS). An IPFS peer node is automatically assigned a Peer ID. Peer ID is a cryptographic hash of the public key of the peer node.

IPFS uses a public-key (or asymmetric) cryptographic system to generate a pair of keys: a public key which can be shared, and a private key which needs to be kept secret. With this set of keys, an IPFS peer node can perform authentication, where the public key verifies that the peer with the paired private key actually sent a given message, and encryption, where only the peer with the paired private key can decrypt the message encrypted with the corresponding public key.

Decentralized Identity (DID)

Every node joining Rubix platform creates a unique network Decentralized Identity (DID). DID is a 256x256 PNG image that is self-created but verified decentralized by peers in the network.The DID is uniquely paired with the Peer ID created on the IPFS.

DID creation process

DID creation take 2 input parameters, first a 256x256 PNG image of users' choice. Second seed is a SHA3-256 hash H, that is generated from user and machine characteristics. Decentralized Identifier needs to satisfy two properties: Uniqueness and Randomness. Uniqueness is required to distinctly refer each node and randomness property is essential for share generation phase. The PNG image provides randomness since its chosen by user whereas the hash H contributes towards achieving uniqueness.

DID Embedding Process

1. The 256*256 PNG image is converted to bits.

2. First 256 bits performs a XOR operation with the SHA3-256 hash H.
3. A hash of H is created to embed with the next bits 256-511 of the PNG image bits.
4. Subsequently, a hash chain of hashes is created, and each hash is embedded to corresponding PNG image bits
5. The embedded bits are finally converted back to a 256*256 image

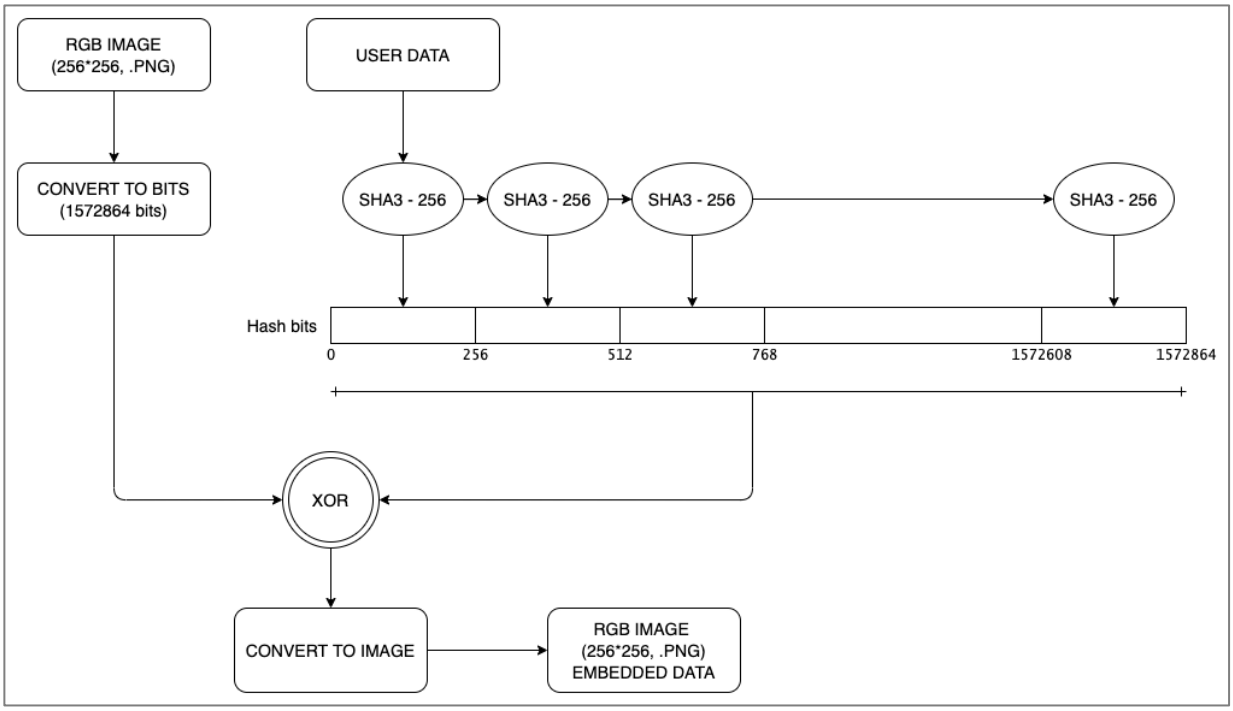

Figure 1: DID embedding process

DID Share Generation Process

The DID for node i, K_i, is split into two secret shares using the Non-Linear Secret Sharing (NLSS)[6] cryptographic techniques – (a) a public network share K_{in} and (b) a private share K_{ip}. Upon splitting, the two shares are 8 times the size of the original DID. K_i and K_{in} are stored by the node in the IPFS. The IPFS hashes of K_i and K_{in}, H(Ki) and H(K_{in}) respectively, are shared across other peer nodes in the network using gossip protocol. Any peer node, to perform a transaction with node i, fetches the K_i and K_{in} from the IPFS using H(K_{in}) and H(K_{in}).

K_{ip} is the secret knowledge known only to the node i.To authenticate the node i to the Rubix Network, knowledge of the private share knowledge of the private share K_{ip} is necessary. Since only the node i has the knowledge of the private share K_{ip}, only it can authenticate successfully to the rest of the network.

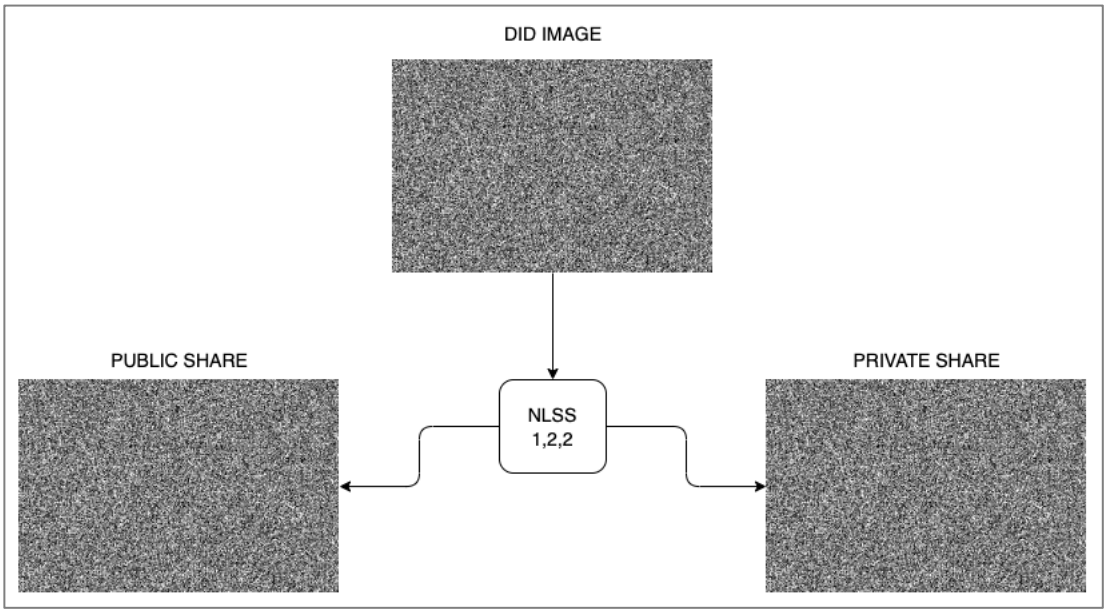

Figure 2: DID Shares creation using NLSS 1,2,2 scheme

Authentication using DID

The DID Token for node i, K_i is split into two secret shares using the Non-Linear Secret Sharing (NLSS) cryptographic techniques – (a) a public network share K_{in} and (b) a private share K_{ip}. Each of the shares expand into 8 times upon splitting. K_i and K_{in} are stored on the IPFS & are globally accessible as well as verifiable. K_i and K_{in} are inseparable from the Peer ID of the node i and hence cannot be faked. The private share K_{ip} is secretly and securely stored by the node i. Using the NLSS cryptographic techniques, each pixel of the DID token K is split into two secret shares. K_{in} and K_{ip} are reconstructed from the split shares of all pixels of K. Knowledge of either Kin or Kip does not help in the reconstruction of K_i. To reconstruct K_i, knowledge of both K_{in} and K_{ip} as well as the Non-Linear function F_n used for splitting into the secret shares is needed.

Since K_{ip} is the secret knowledge possessed by the node i, only node i can authenticate itself to another peer node. In other words, to authenticate to another peer node, knowledge of the private share K_{ip} is necessary. The authentication process involves cryptographic Proof of Work from the node i. Let us say that the node i would like to authenticate itself to node j. Node j would issue a cryptographic challenge by asking for the values of the 256 pixels from the private share K_i. Node j would randomly select the 256 position values that make up the challenge. Note that node j would be honest in picking the 256 position values since it is seeking the identity of another node. There is a total of 1,572,864 pixels (each RGB pixel can take 24 values & there are a total of 256x256 or 65,536 pixels; 65,536 x 24 = 1,572,864) in a DID token. Upon receiving the challenge, Node i would return the values of the 256 pixels of the private share K_{ip}. Node j combines the values of the 256 pixels of K_{ip} with the values of the corresponding 256 pixels of K_{in} to compare the result with the values of the corresponding 32 pixels of Ki. (note that each pixel of K_i is expanded to 8 pixels while splitting into K_{ip} & Kip. A complete match would successfully authenticate Node i to Node j.

Non-Linear Secret Sharing (NLSS)

In a secret-sharing scheme, a set number of parties share a common secret. The information about the secret a single party owns is called a secret share. In secret-sharing schemes for ***n*** parties the shares t_i for i = 1,2, . . . , n are often computed according to some arithmetic functions $t_i = f_i(s_1, s_2,...,s_u)$ where s_1 is the secret and $s_2,...,s_u$ are randomly chosen elements of some fields. We call such a scheme linear if all f_i are linear with respect to the variable $s=(s_1, . . . , s_u)$, and nonlinear otherwise. Linear secret sharing schemes are not cheating-immune[4]. Nonlinear secret sharing shame is a cheating-immune secret sharing scheme that prevents a cheater, who submits a corrupted share, from gaining an advantage in knowing the secret over the honest participants. A perfect secret sharing scheme realizing the access structure Γ is a method of sharing a secret among a set of participants P, in such a way that the following two properties are satisfied:

i. Every authorized subset B ⊆ P can determine the secret.

ii. Every unauthorized subset B ⊆ P obtains no information about the secret.

An (n, n)-secret sharing scheme can be represented by a defining function $f : F_q^n \to F_q$ which maps to each vector of shares a secret value in F_q. The defining function of an *(n, n)*-secret sharing scheme over F_2 is a Boolean function.

Theorem 1: *An (n, n)-secret sharing scheme with defining function f is* 1*-cheating-immune if and only if f is* 1*-resilient and satisfies the SAC.*

Theorem 2 : *Let C be a binary* $[n-m, m, d]$ *linear code with dual distance* $d\perp$*. Assume that* $d\perp \leq d$*. Let G be a generator matrix of C and let* α *be a vector in* F_2^{n-m} *which is not in C. Consider the function* $f(x, y) = xGy^T \oplus \alpha y^T$*, where* $x \in F_2^m$ $y \in F_2^{n-m}$ *and* $\oplus$ *is the addition over* F_2*. Then we have the following:*

i. f is t-resilient with $t = \min_{c \in C}\{wt(c \oplus \alpha)\} - 1$

ii. f satisfies the strengthened propagation of degree l with $l = d^{\perp} - 1$.

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 parallelly 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 Proofchain P_{Ti}. If multiple tokens are used in a transaction, the transaction is added to multiple Proofchains.

Proofchain

A Proofchain P_{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 proofchains 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 Proofchain of that token starts with the validators node that has mined the token.

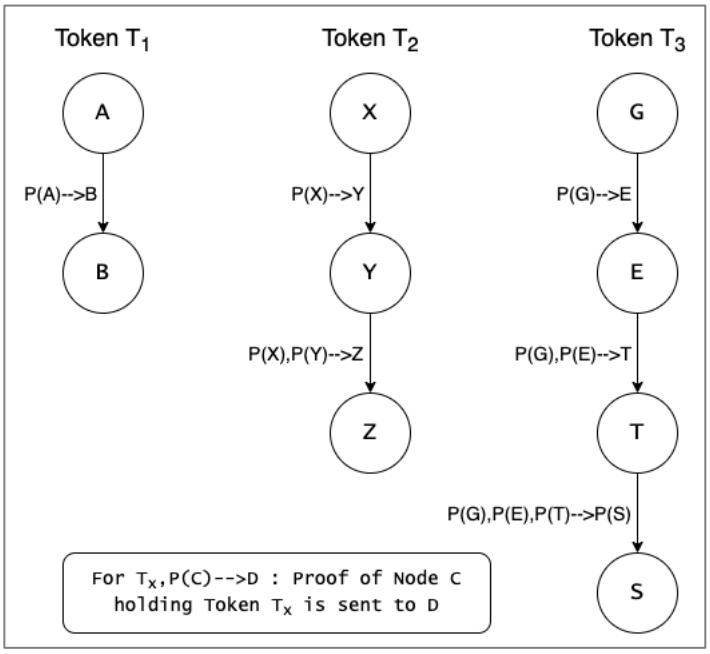

Figure 3: Independent ProofChains of Tokens

Rubix Consensus Protocol

Rubix introduces the lightweight, low carbon footprint consensus protocol, a combination of (a) Proof of Work (PoW) & (b) Proof of Pledge (PoP).Validators perform work to validate the transaction & verify cryptographic signatures. Validators earn proof credits in return for the work performed. The credits earnt as a result of work are pledged by the validators to secure the network using the PoP algorithm. Proof of Pledge (PoP) is the most important part of the consensus in the Rubix 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. In the case of PoP, only outstanding proof credits are pledged, not the native RBT tokens. When the outstanding proof credits are converted into RBT tokens, validators need to earn new proof credits before being eligible for **α**

validators (more in this later in this section). Since nodes are in continuous competition with each other to convert the outstanding proof credits into RBT tokens (before the credits deprecate in their value as the network moves into next level), accumulation of proof credits in order to control the network is not possible. Hence, every node in the network gets equal chance to become a **α** validator & there is no scope for concentration in validator powers. Eliminating the need for staking also obviates the need for rent seeking, a crucial bone of contention with PoS algorithms.

For each transaction (a transaction may involve multiple nodes & proofs could be built on multiple proofchains using multiple RBT tokens), the transacting nodes choose a set of minimum 21 validators. Out of these, 7 validators (named **α** validators) are chosen by the transacting nodes based on the combined proof credits outstanding while the remaining 14 (including 7 **β** validators & 7 **γ** validators) are chosen by each of the participating nodes respectively. The number 7 is chosen in order to achieve PBFT consensus ($3n+1$) where $n = 2$. The key design principle is that any honest buying node in the transaction would choose the α validators such that, the combined outstanding proof credits of the **α** validators **>** the combined value of the tokens involved in the transaction. Cheating **α** validators lose all their valuable outstanding proof credits, while not gaining anything directly from the forking attempt. This prevents any remote possibility that validators will collude with the transacting nodes to cheat & cause forks.

<u>Proof credits</u>

Each of the 15 successful validators (5 out of 7 **α** validators following PBFT consensus, 5 out of the 7 **β**, 5 out of the **γ**) in any transaction receive one proof credit (**p**). For a transaction involving an asset token, **α** validators get an additional proof credit each (more on this in the asset transfer section later in this paper). Validators will accumulate proof credits from various transactions across different Proofchains. When a node's Σ (**p**) reaches the threshold level **τ**, the node can convert the proof credits into one new RBT token. Nodes continue to earn proof credits & thereby mine new RBT tokens by participating in consensus of various transactions & perpetually storing the consensus proofs. When a node transfers its outstanding proof credits or the mined RBT tokens along with the supporting proof credits to a new node, the receiving node will continue to store the transaction proofs (otherwise the corresponding RBT token will be invalid).

Each validator node keeps the latest count of outstanding proof credits as well as the number of RBT tokens they have validated in various transactions by pledging the proof credits. New transacting nodes use net outstanding proof credits data to select the α validators. The authenticity of the proof credits & pledge count is confirmed by PBFT consensus among the **α** validators as described later in this paper.

Conversion of proof credits into RBT tokens

Each validator in the Rubix transaction stores the proofs to secure the Proofchain. By burning energy on their computing systems & using the memory to store the proofs, 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 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 the **Annexure 1** (the threshold levels, τ, are tabulated).

α validators

α validators play a vital role in preventing & resolving forks by achieving consensus. Any node in the network can become an α validator. The higher the number of outstanding proof credits (that are yet to be converted into RBT tokens), the higher the chance of being selected as an α validator. Since the combined outstanding proof credits of the 5 successful **α** validators is higher than the token value in the transaction, the **α** validators have no economic incentive to collude with any malicious transacting node since the economic loss from such collusion is greater than what could be gained even in the case of unlikely collusion. This is because nodes initiating forks & the validators behind forks can easily be determined by any other node in the Rubix Network. The proof credits earnt by validators behind forks will be discredited, resulting in forfeiture. Given that there are a total 5 reward earning **α** validators, the maximum number of tokens that can be transferred in any single transaction without fork risk is 5. If more than 5 tokens need to be transferred, transacting nodes need to choose a higher number of nodes as alpha quorum to ensure security. For example, if 500 tokens need to be transferred, sufficient set of **α** validators must be chosen such that the outstanding credit count of these validators is higher than the credit value of 500 tokens at the given mining level. This can be achieved by either choosing **α** validators with largest outstanding credits or a large number of **α** validators. Rubix is designed to facilitate large value transactions equally faster as small value transactions, all the while requiring no third party transaction fees.

β & γ validators

In addition to the 7 **α** validators, an additional set of 7 **β** validators & 7 **γ** validators are chosen for every transaction. The **β** & **γ** validators are selected each of the transacting nodes, based on a deterministic function of the length of chain & the identities of the transacting nodes. The idea behind **β** & **γ** validators is to democratize the network, reduce the concentration & speed up the consensus.**β** & **γ** validators are chosen by the transacting nodes to meet two specific rules: (a) these validators should have low number of outstanding credits (less than $1/5^{th}$ of credits needed to meet the RBT token threshold level) and/or (b) these validators must have the ending character in their DID from one of the last five characters in the TID

(Transaction ID). α validators secure the Rubix Network based on economic incentives, while β & γvalidators increase the reliability & decentralization of the network. A node can be α or β or γat any stage of time in the network depending on the number of credits outstanding.

In the Rubix Network, an honest node will never enter into a double spending transaction as the node can easily find out if the token is double spent or not by checking the ownership of the token using IPFS hashes & the Proofchain proofs.

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 Proofchain 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 Proofchain. The Proofchain 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.

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 proofchain 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 α validators as discussed earlier in this paper. No α validator will approve the consensus

of a forking transaction due to the risk of losing the valuable proof credits. The malicious node setting up other Sybil nodes to become **α** 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 **α** 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.

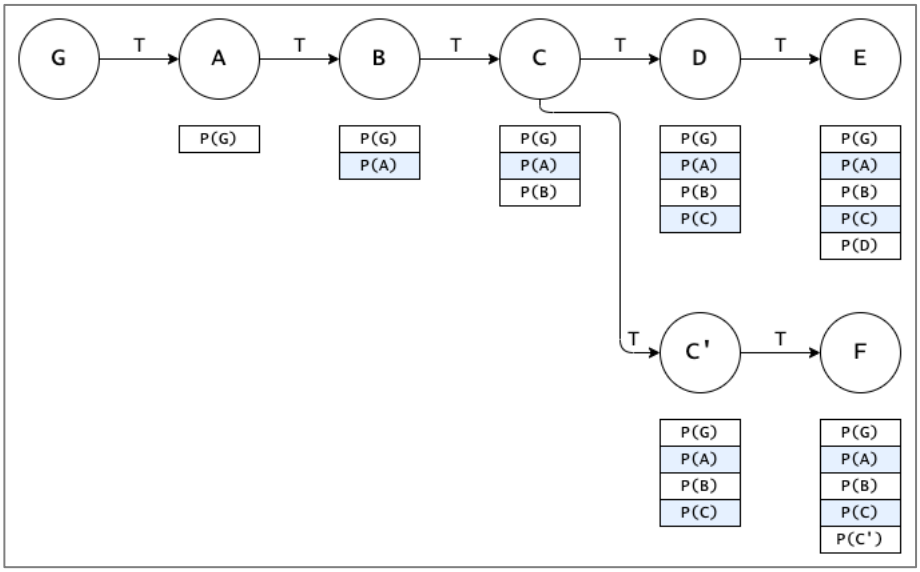

Figure 4: Fork Detection

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 **α** validator 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 creditchain with the previous transaction details.

A malicious node cannot form sybil nodes & sybil validators to generate proof credits without work. Generation of proof credits without a genuine transaction require the same amount of proof verification & cryptographic signature work as a genuine transaction. Since same amount of work & time is spent even in a potential sybil credit attack, there is no loss to the network. Further, sybil attacks on credits may not involve longer term costs that

greatly outweigh any benefits.

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 Three (3) RBT tokens for a minimum number of transactions of the newly minted RBT token. Technically, the three RBT tokens are pledged temporarily, not staked as such.

Once the minting node submits new proof credits to mine, 3 of the alpha validator nodes signing the minting transaction will need to pledge (or stake) 1 RBT token each. The pledged RBT tokens by alpha quorum are 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 alpha quorum nodes with at least one token to stake? The quorum can be populated by borrowing the required three alpha nodes that have at least one token to stake token for $4*h$ transactions.

The number of tokens to be staked for new RBT minting will reduce to 2 from 3 from the level 14 onwards (please refer to the Annexure I for the difficulty level information) and further reduce to 1 from level 24 onwards.

Solving Hashing Problem for Mining

All mining transactions need to solve an iterative hashing problem to complete the mining process. The iterative hashing requires the mining node to repeatedly hash (using SHA3 – 256) the Transaction Hash ID (TX_i) until the last 3 characters of the $f_n(TX_i)$ matches the last three characters of the DID of each of the three nodes that have staked tokens for mining, where n is the number of hashes.

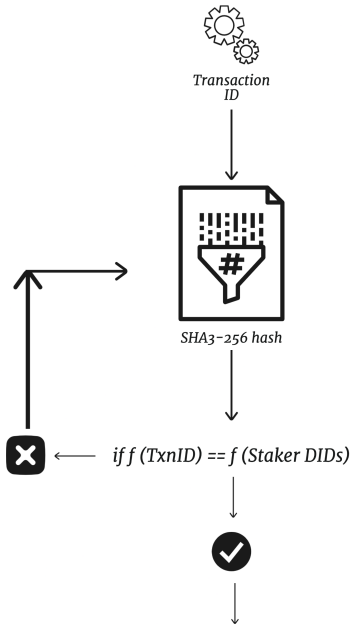

Figure 5: Hashing Problem

Since TX_i is random & unique (can't be created by the mining node apriori), the mining node needs to repeat hashing for *n* number of times to achieve the required result. Given TX_i & n, any verifying node can conclusively prove that the mining node has solved the required problem.

NLSS based PBFT

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 an Sender (S), Receiver (R), seven α validator nodes (α_1, α_2, α_3, α_4, α_5, α_6, α_7), seven β validator nodes (β_1, β_2, β_3, β_4, β_5, β_6, β_7) & seven γ validators (γ_1, γ_2, γ_3, γ_4, γ_5, γ_6, γ_7). Quorum Initiator creates the Transaction Data in the following way:

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

Each transaction requires three separate, parallel consensuses that occur among α, β &γ validator quorums. Post-quantum, cheating immune NLSS algorithm based Multi-Party Computation (MPC) determines the consensus. To recreate the transaction data image, each quorum member should have minimum of two shares of which one should be the essential share. The value 7 is chosen based on Practical Byzantine Fault Tolerance (PBFT) algorithm. Hence, 5 ($2n+1$) out of 7 ($3n+1$) votes from the Quorum nodes are required for a successful consensus.

Steps:

1. Sender picks quorum members α_1, α_2, α_3, α_4, α_5, α_6 & α_7 from the global validator set based on the outstanding proof credits of the nodes.
2. Sender calculates the SHA3-256 hash H of T, S_w and the Proofchain height P_h and sends H, Sender's Signature from H and Receiver ID R_w to all the quorum members.
3. The quorum members provide 32 × 64 bits of their private share from positions derived from H and R_w. 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. Steps 1 to 5 are repeated among β &γ validator quorums simultaneously.
7. Steps 1 to 5 are repeated among multiple sets of α validators where necessary.
8. Subsequent to the successful α, β &γ 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 G.

1. For a user in the Rubix network, after successfully completing the DID registration and share generation, a gossip protocol broadcasts the Public share and DID token 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 (ProofChain) is known only to the sender and therefore, after the acknowledgement the latest ProofChain is sent to the receiver for picking challenge-response positions.
4. The ProofChain represents the universal proven state of each token, which is publicly verifiable.

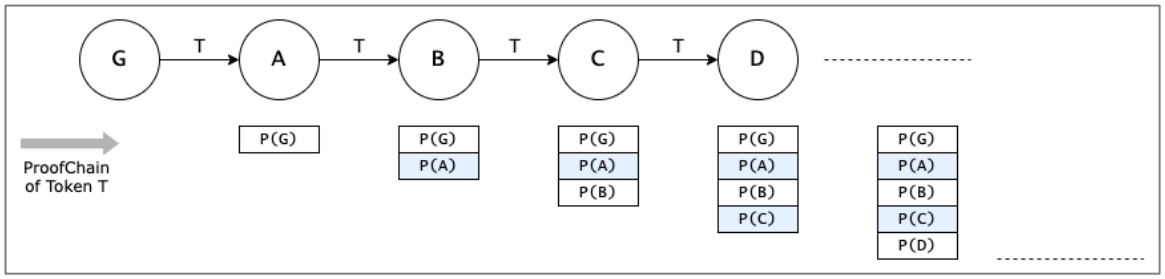

Figure 6: ProofChain Formation

5. If token T is purchased from Rubix genesis node G by user A. Node A gets empty token 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 ProofChain. Hence the current ProofChain contains two proofs, P(G) > P(A)
7. The sender calculates SHA-256 hash of receiver's wallet share (R_w), token (T) and the ProofChain (P_T).
8. A Deterministic Function F(x) is used on the hash to select 32 challenge - response positions `F(x) = (2402 + hashCharacters[k]) * 2709 + k + 2709 + hashCharacters[k])%2048`
9. The challenge response positions are picked from a hash which is calculated using R_w, T and P_T

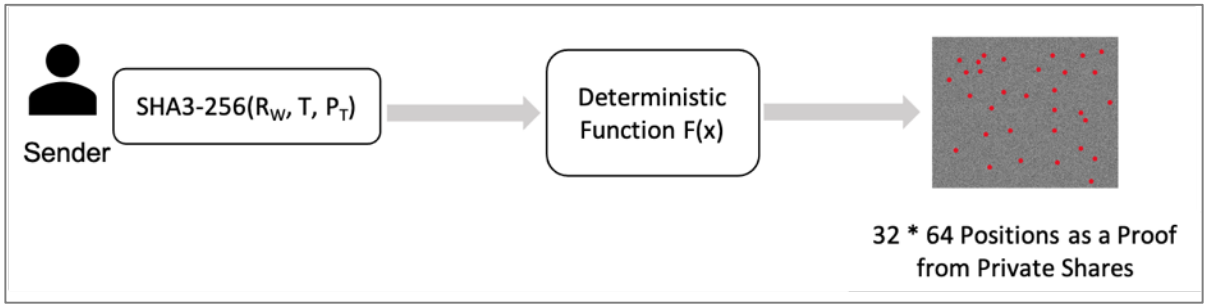

Figure 7: P2P Challenge Response Creation

10. R_w and T are used in hash calculation to ensure that the agreed token is being transferred to the recipient
11. For instance, A is transferring a token, T to B [A (T)—> B]. The inputs for the hash calculation are B_w, T and P_T.
12. Any other user out of this transaction, C cannot claim token T has been transferred to C by A [A (T)—> C].
13. B cannot claim A has transferred some other token by A [A (~T)—>C].
14. ProofChain is used because, apart from the sender no other user in the network possesses the latest token chain of the token. Therefore, no party, not even the receiver cannot calculate the hash and pre-compute the 32 challenge - response positions for the transaction of token T.
15. The sender picks the corresponding 2048 (32 positions pre-calculated along with 63 trailing bits of each 32 bits is chosen from the total 20, 97, 152 of the shares. Making it a total of 32 * 64 = 2048 bits) positions values from each of the 3 private shares and is shared with the receiver.
16. On the other side, the receiver also calculates the SHA3-256 hash of W_R, T and P_T and

subsequently determines the 2048 positions by applying the same deterministic function on the hash.

17. Since the wallet shares and DID of all users are publicly available, corresponding 2048 bits from W_S are chosen and is recombined using NLSS recombination with private positions shared by the sender. The recombined result yields 32 bits. The receiver picks the corresponding 32 positions from the DID sender and performs a comparison check.

18. If the 32 bits obtained from the recombination and the 32 bits picked from the DID sender matches, the sender is authenticated.

19. Receiver checks whether the token is held by any other node in the network and then acknowledges the sender

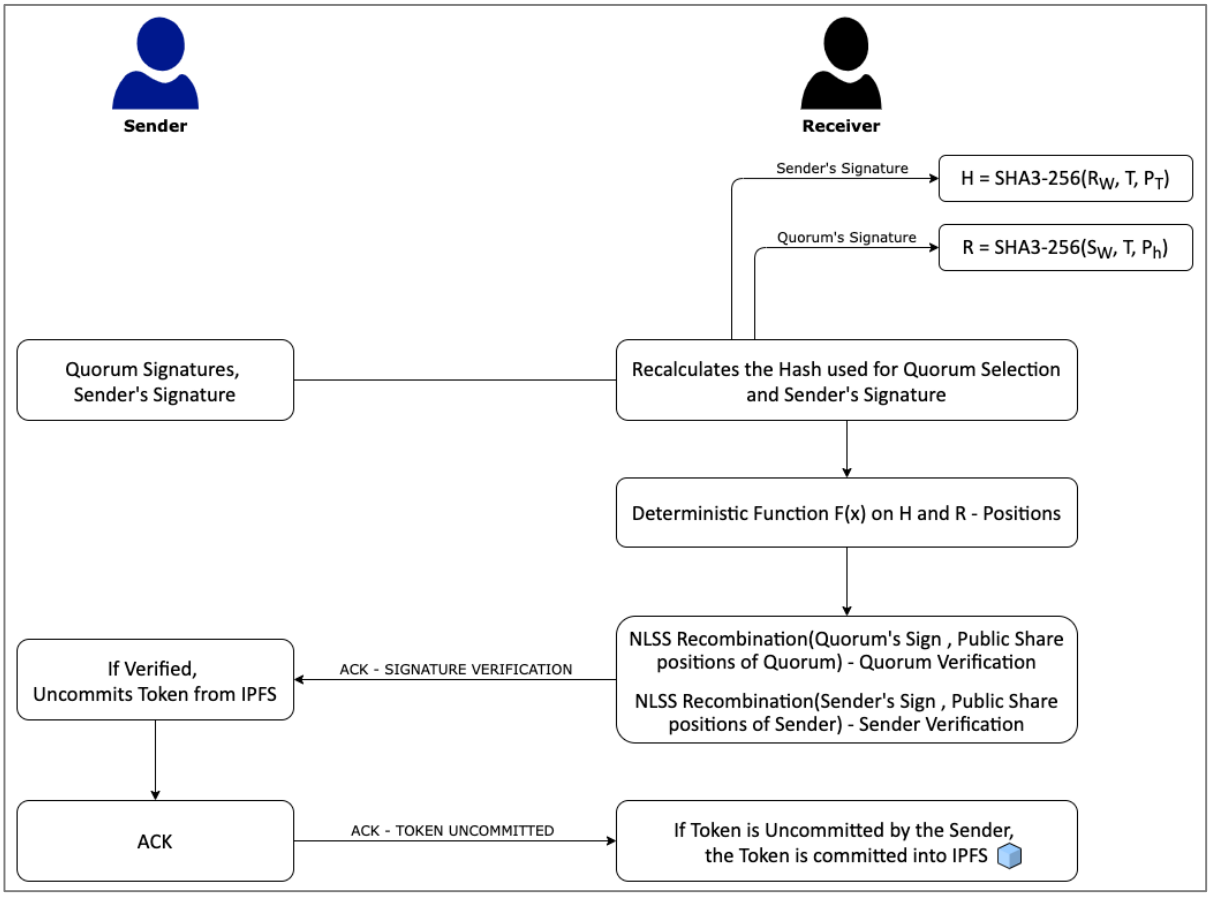

Figure 8: P2P Authentication

20. 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.

Furthermore, a third node B cannot claim the receipt of T_x from G since node B will not be able to produce the proof that G has signed the transfer of token T_x. Note that G has signed the transfer to node A by hiding the knowledge of G's private share with A's network share.

For node B to prove the transfer, it would have needed G to sign the transaction with B's network share.

The probability that same 32 bits will be selected to sign the transaction for two different tokens and/or for the same token to a different receiver is $(1/262144)^{32}$ or $6.44e^{-94}$. User authentication as described in steps 1 to 21 has a brute force resistance of 2^{196} .

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

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

Objects (Asset tokens/NFTs/Appcoins/Smart Contracts)

Every object on the Rubixchain has its own Objectchain. Objects can be RBT tokens (utility), NFTs, asset tokens or smart contract tokens themselves. Unlike the RBT utility tokens, object tokens carry an actual value as they are associated with the value of an underlying real-world or digital asset/service. Asset-backed and NFTs provide secure, fast and minimal cost trading of real or digital assets via blockchain technology.

Each Objectchain tracks the history & provenance of the associated Object. Unlike how other blockchains create complex code to create & track the movements of NFTs & other assets, Rubix enables handling data & feature rich Objects much more efficiently. It is easy to create rich Rubix Objects, modify or expand the metadata associated with them. Objects have a life of them on the Rubixchain independent of other Objects or contracts & therefore are easier to program around. Each Object can contain rich data to become an exchange or wallet itself.

Objects on the Rubixchain are defined by the RAC (Rubix Asset Contract) standard. RAC standard lets issuers easily mint rich Objects & commit them on the chain. The structure of an RAC standard Object is shown below:

Predefined type*	Creator's DID*	Total Supply*	Token Number*	Comment / Creator Input	Hash*	URL	Private Key Signature*

*denotes required fields

Predefined type*

The type of object in the first RAC field defines the nature of the underlying Object. The first three defined types are mentioned below:

RAC 1: Asset Tokens
RAC 2: Secondary/App Tokens
RAC 3: Smart Contracts

Creator's DID*

Decentralized identity (DID) is the unique public identity of any user in Rubix. Each RAC object is signed by the creator. The DID will help other nodes in the network to verify the authenticity of the token.

Total Supply*

The Total Supply field denotes the total number of asset tokens minted for any unique Object or asset.

Token Number*

Token number will be unique for each token in a range of RAC tokens created for a particular Object. To prevent double spending, validators will use this field to check if the token is being transferred or owned by any other node in the network.

The combination of all the fields is hashed to represent each token uniquely on the chain.

Peer-to-Peer Authentication of Asset Transfer

Consider a scenario when User A (K_A, W_A, P_A) want to transfer an asset AT with hash H_{AT} to User B (K_B, W_B, P_B).

User A calculates the SHA3-256 hash of token T, wallet ID of B (W_B), and Asset hash H_{AT}.

From the calculated hash H, User A deterministically find out 32 positions using a deterministic well-dispersed function.

Values from the private share corresponding to 32 positions pre-calculated along with 63 trailing bits of each 32 bits is chosen from the total 1,572, 864, making it a total of ($32 * 64 = 2048$ values). The 2048 positions and token T values are shared to the receiver B. B also calculates the SHA-3 256 hash token T, Asset hash H_{AT}, wallet ID of B, W_B and subsequently

determines the 2048 values of A's private share.

B performs Non-Linear Secret Sharing recombine of the 2048-private shares with corresponding wallet share of A and compare the result with 32 positions of DID_A. Only a perfect match results in authentication of user A. Once user B successfully authenticates user A, A tries to authenticate B. A calculates the SHA3-256 hash of token T, wallet ID of B, W_B, and Asset hash H_{AT}.

By following the same steps mentioned earlier B sends his 2048 private share positions to A. User A also calculates the SHA-3 256 hash of token T, Asset hash H_{AT}, wallet ID of A, W_A and subsequently determines the 2048 positions of B's private share.

A performs NLSS recombine of the 2048-private share with corresponding wallet share of B and compare the result with 32 positions of DID_B. Only a perfect match result in authentication of user B. Once the authentication of both A and B is completed the transfer of asset will occur.

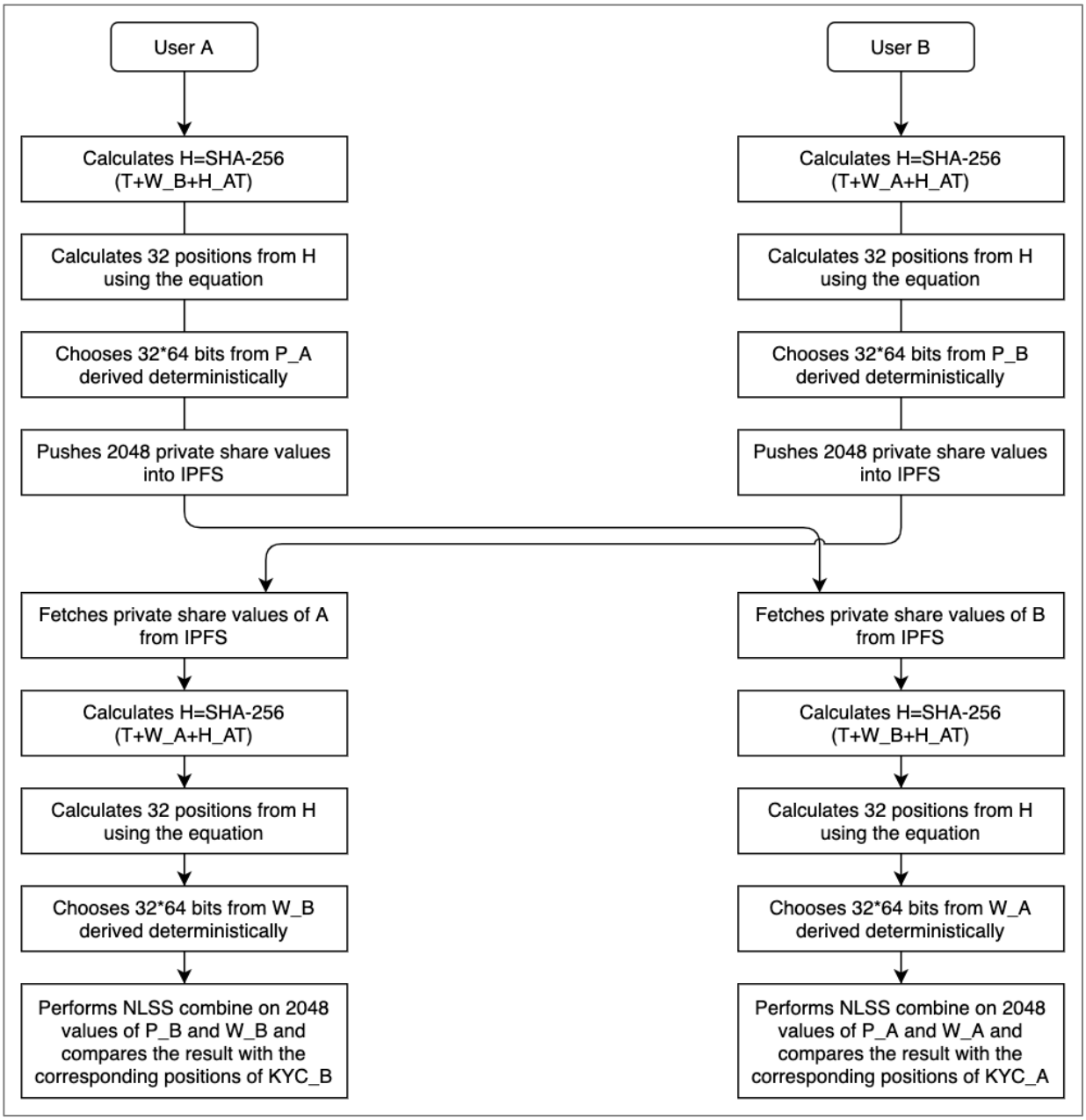

Figure 9: Asset Token / NFT / transaction flow

A Proofchain (Objectchain) can be used to process, validate and store peer-to-peer digital contracts. A digital contract is a contract denoting exchange of goods or services in return for other goods or services or in exchange of other means of value like fiat currencies or commodifies. Instead of sequentially pooling all global transactions into a block as a traditional blockchain would do, each transaction is processed and validated on its own with Proofchains. To include a transaction into Proofchain, the transacting parties need a token for provenance.

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

Rubix Network and Private IPFS

Every node who joins the Rubix network will be part of 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
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

Level	Cumulative tokens ('000)	Tokens awarded ('000)	PCs/token
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
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

Level	Cumulative tokens ('000)	Tokens awarded ('000)	PCs/token
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
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

Level	Cumulative tokens ('000)	Tokens awarded ('000)	PCs/token
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
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%			