Welcome to RubiX - ProofChain!

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.

Distributed Transaction and Data Network

Blockchain protocols in general achieve a globally accessible singleton state by organising all global transactions sequentially as blocks (each block having a finite number of transactions) and organising 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. Further, such protocols require every node to store the entire global state, which results in 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 is made of 16.9 million ProofChains. Each ProofChain is bound by one unique Token. A ProofChain P_T is made of all transactions that use token T_n 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. These tokens are various entities in Rubix platform based upon its design and application. The two main categories are Asset tokens and Utility tokens.

Asset tokens are derived from the digital form of any real-world asset 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 is restricted in creation. When compared to its Ethereum model ERC 721, Rubix tokens are more dynamic. Unlike ERC721 that uses a parameter "**value**" while token creation itself, Rubix asset token's value is decided with respect to the utility tokens bounded to it during a transaction.

A land asset token A_i bounded 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 bounded to a transaction with utility tokens worth *y* units. The current value of the asset token A_i is *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 utility tokens represent the overall cap of the network. Much like every real world and virtual currencies in the world, Rubix utility tokens also has a finite monetary supply. With Rubix's breakthrough ProofChain architecture, the network does not require expensive miners or centralized stakes men with proof of stake protocols to maintain the networks genuineness, the tokens can be pre-mined and published.

Rubix generates 16.9 million tokens and each of these tokens will be distributed among Rubix token issuance nodes. The following conditions must be satisfied while pushing value-based tokens into any distributed trust less network. Firstly, the tokens should be finite in supply, no one even Rubix developers should be able to create more tokens. Secondly, each of these tokens should be uniquely identifiable and publicly verifiable.

Rubix generates its utility tokens from a combinatorics design called Latin square. By design, the number of Latin squares that can be generated from a set of inputs of a size is limited. This design also complements with unique values for each of the Latin squares which is important to satisfy the second condition of tokens in a blockchain network.

Rubix platform uses a reduced Latin square of size 7×7 that can create 16.9 million unique Latin squares. Each of these Latin squares are hashed (SHA3-256) to obtain Rubix utility tokens. Such representation of unique finite tokens supply security over infiltration of more pseudo tokens into Rubix network. There is a total of 16.9 million utility tokens, numbered T_1, T_2, T_3, …. Token T_n is essential to confirm all the transactions in the ProofChain P_T. While each

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 16.9 million Rubix utility tokens. Rubix achieves this by representing its tokens in the form of Latin square hashes and along with its custom version of the Interplanetary File System (IPFS) protocol. Each of the 16.9million hashes in Rubix chain is pushed into IPFS and pinned by any one of the token issuance nodes.

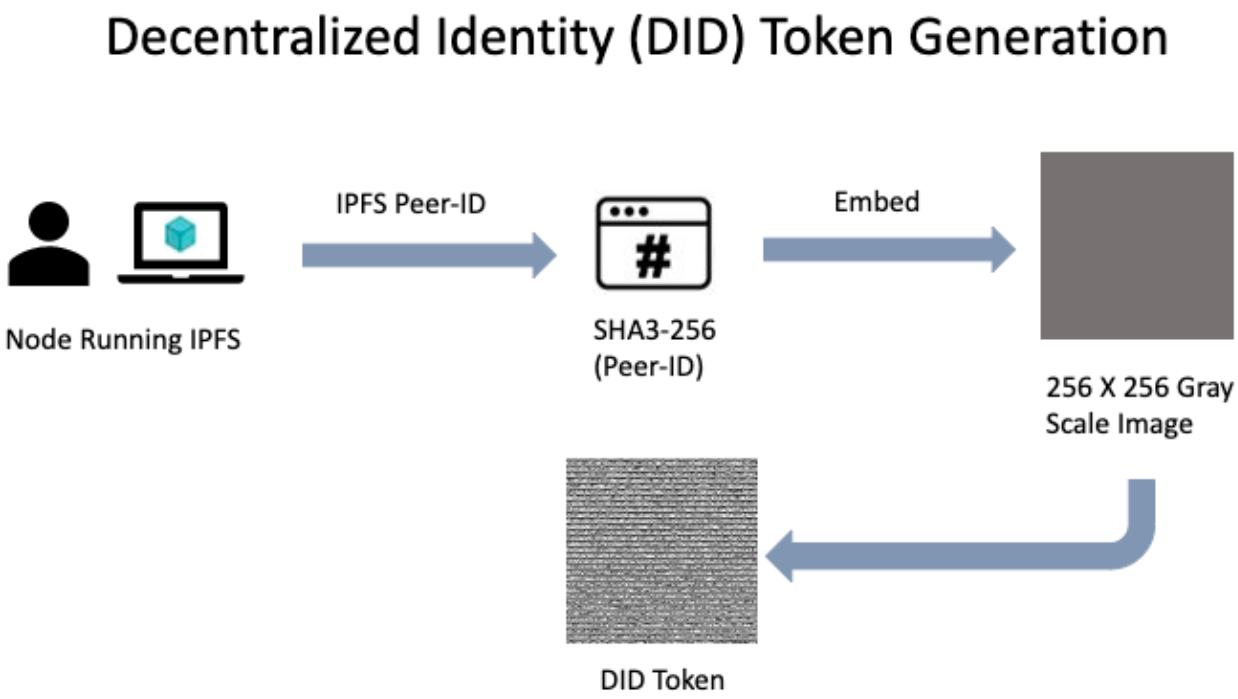
Since IPFS follows content-based addressing, any alteration even on a single bit will result in an entirely different hash value that will not be verified by any of the token chains. Once all the hash values are generated, they are passed into a bloom filter. Bloom filter is a memory efficient probabilistic 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 ProofChain code is lightweight and can even be run from most laptops and 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.

Each Rubix node is assigned a DID token. The DID token is a 256×256 greyscale image and made up of Peer ID and an identity unique to the node such as mobile phone number.



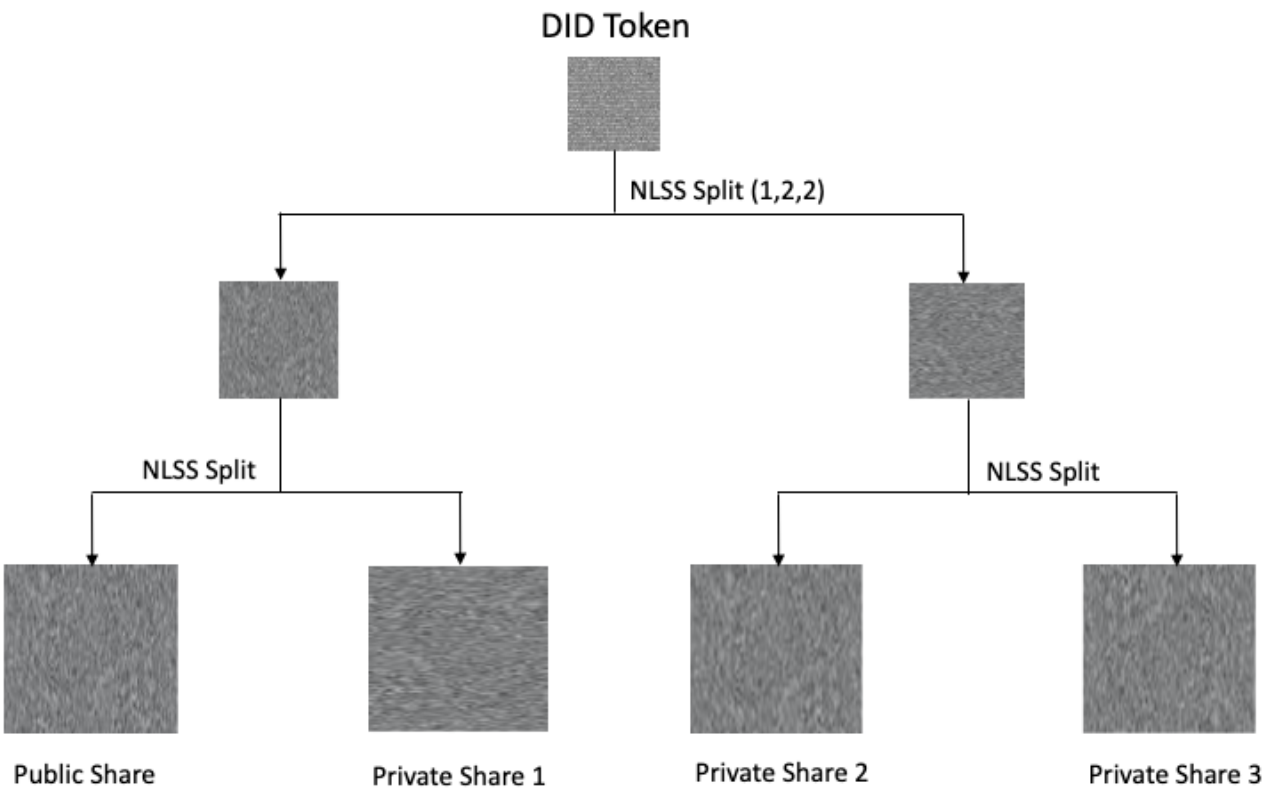
Non-Linear Secret Sharing (NLSS)

Rubix DID tokens uniquely represent every node in the Rubix network. DID tokens are represented using two-dimensional $m \times n$ images. DID token images are split using **(1,2,2)** image based Non-Linear Secret Sharing Scheme.

The DID is split into 2 shares, each share will be 8 times $8m \times 8n$ the original DID token. Each of these shares are further split into two shares each, such that each of the four shares are 64 times $64m \times 64n$ the original DID token.

One of the four shares is a wallet share which is publicly available and the remaining three are private shares which can be securely stored in cloud, secured devices, etc.,

All the three private shares are mandatory for peer-to-peer authentication



The DID Token for node i, K_i is split into two secret shares using 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 64 times upon splitting. K and K_{in} are stored on the IPFS and are globally accessible and verifiable. K and K_{in} are inseparable from the Peer ID of the node and hence cannot be faked. The private share K_{ip} is secretly and securely stored by the node. 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 K_{in} or K_{ip} does not help in reconstructing K. To reconstruct K, knowledge of both K_{in} and K_{ip} and the Non-Linear function F_n used to split into secret shares is needed.

Since K_{ip} is the secret knowledge possessed by the node i, only the 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 node i. Let us say that 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 Ki. Node j would randomly select the 256 position values that make up the challenge. Note that there is a total of $2,097,152$ pixels 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_i to compare with the values of the corresponding 32 pixels of K_i. A complete match would successfully authenticate Node i to Node j.

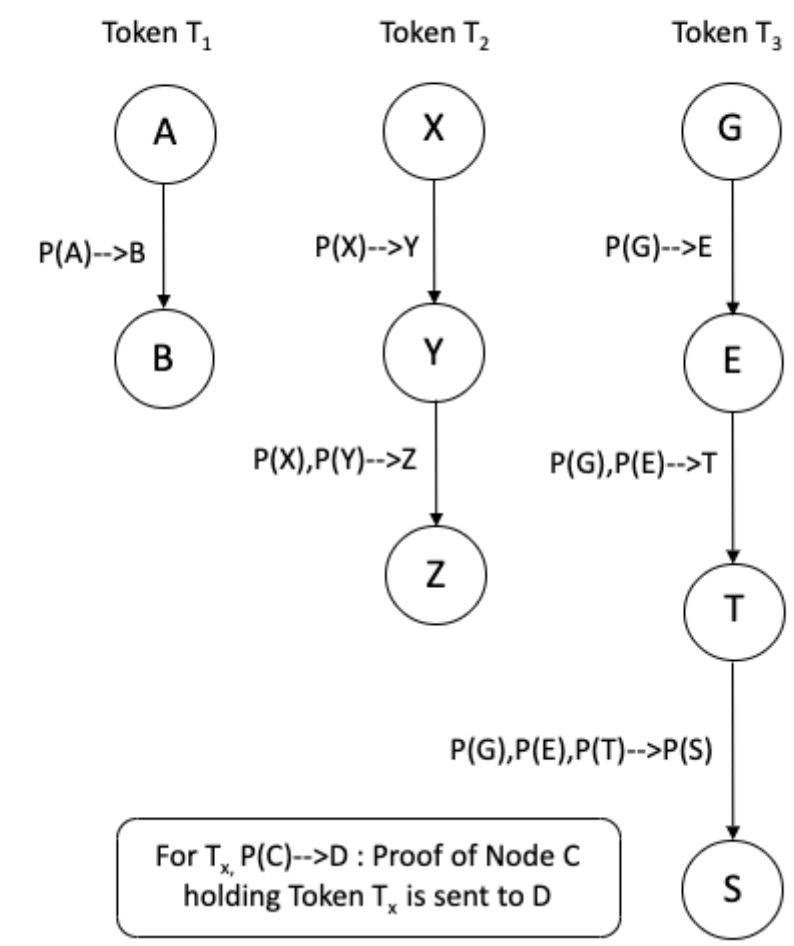
Transaction

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, there are 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 a peer node. To initiate the transaction, the peer node must use at least one native token. Depending on which native token (ti) 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

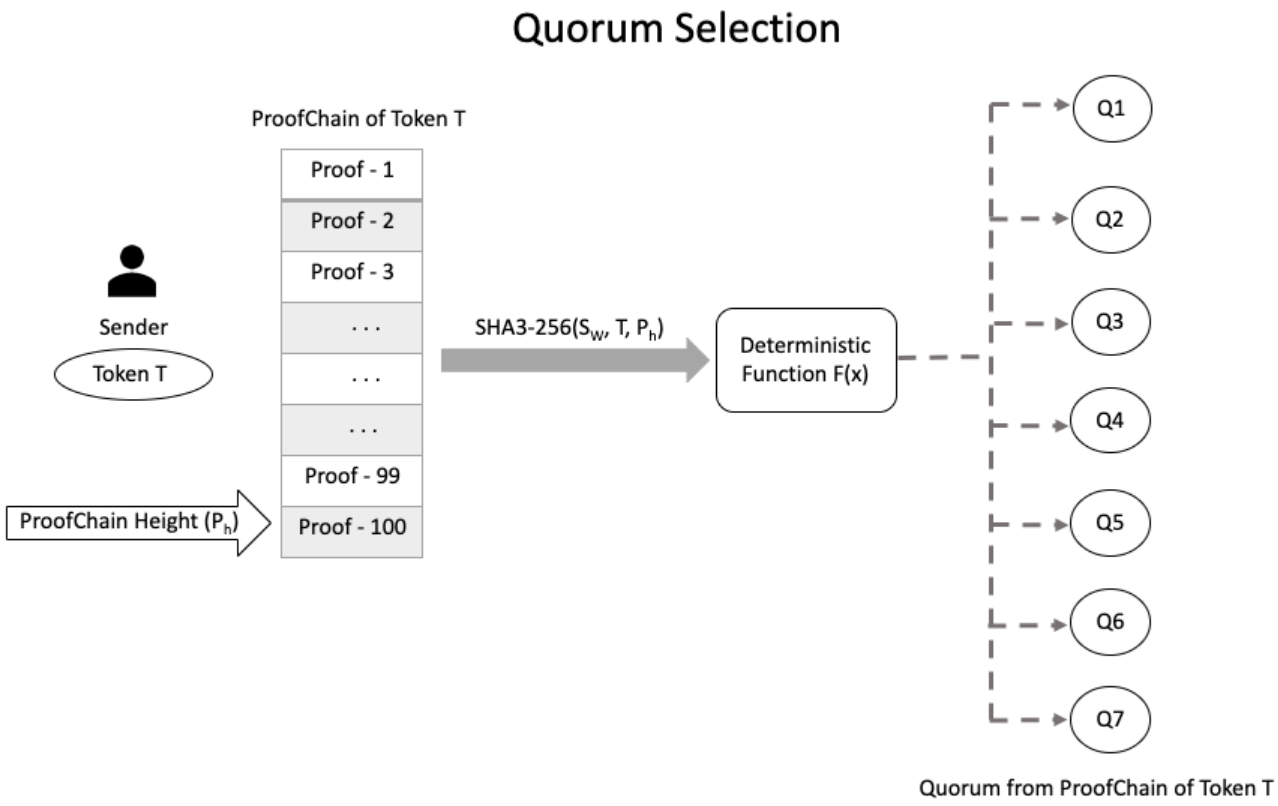
A ProofChain P_{Ti} is a chain of all transactions bound by the token T_i. All ProofChains start from the Genesis node G. The Rubix node is the Genesis node G which has all the 16.9 million tokens to begin with. All the tokens are stored and committed on the IPFS by the genesis node G. Since all tokens are committed by the genesis node G, the node's ownership is globally verifiable.



Consensus Protocol

Rubix consensus is a lightweight Practical Byzantine Fault Tolerance (PBFT) algorithm that require $2n + 1$ out of $3n + 1$ members to reach an agreement. Rubix PBFT algorithm is designed to work in practical, asynchronous environments.

For each transaction, a set of 7 (the minimum number of nodes required for a PBFT consensus) nodes are selected by the initiator. The 7 nodes are deterministically chosen from the ProofChain(s) of the token(s) to be transferred. These nodes, called quorum members perform a Multi-Party Computation (MPC) based consensus. If 5 out of the 7 nodes in the quorum agree to the transaction, the consensus is achieved.

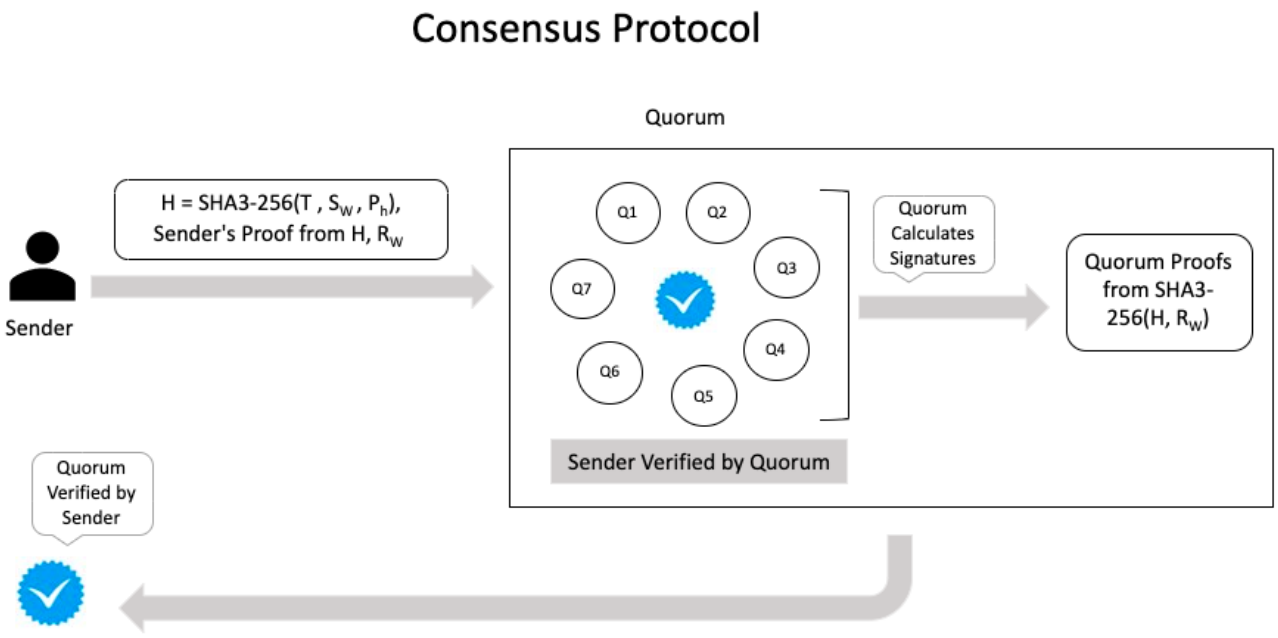


NLSS based Consensus process

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 the all nodes to synchronise with other peers in the network makes these models inoperable. In Rubix platform consensus protocol is run for each transaction independently. This way each transaction can be independently verified reducing forks.

Rubix Consensus involves an Initiator (I), Seven Nodes (Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7) – Quorum Initiator creates the Transaction Data in the following way:

> Transaction Data = SHA3-256(Initiator's WalletID + Token + ProofChain Height)



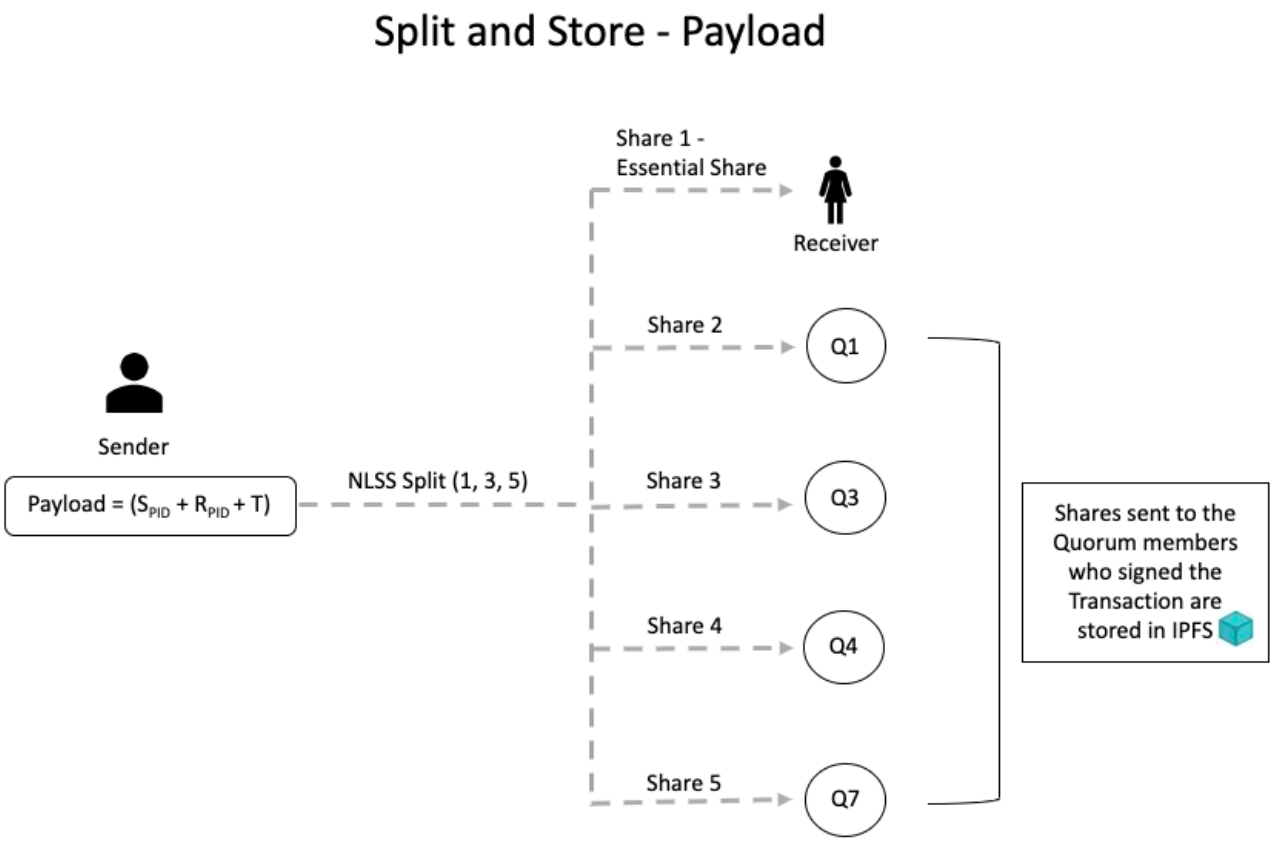
Post-quantum Cheating Immune Algorithm that facilitates Multi-Party Computation (MPC) based consensus process. To recreate the transaction data image, each quorum member should have minimum of two shares of which one should be the essential share. The value 7 is chosen based on Practical Byzantine Fault Tolerance (PBFT) algorithm. Hence, 5 out of 7 votes from the Quorum nodes is required for a successful consensus.

Steps:

1. Sender picks quorum members Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7 from ProofChain using a deterministic well dispersed function.
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 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 this particular transaction of sender and token. (This can be Light-weight PoW quorum performs)
4. When 5 or more signatures are received by sender, he performs Peer-Peer authenticated token transfer and IPFS Commiting & Uncommiting.
5. Once the BFT count is reached, the Rubix consensus is successful.

Payload Split and Store

Visual Secret Sharing scheme is used during this storage module at the Sender after successful Consensus. The payload is split into 5 shares using **(1,3,5)** scheme. The essential share is stored/retained with Receiver of the token. The shares related to each transaction are stored in different nodes (Recipient, Notaries, Backup Notaries) to prove provenance of the transaction. In case the notaries (who were part of the consensus and hold shares) fail, the two backup notaries have rest of shares stored.



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.

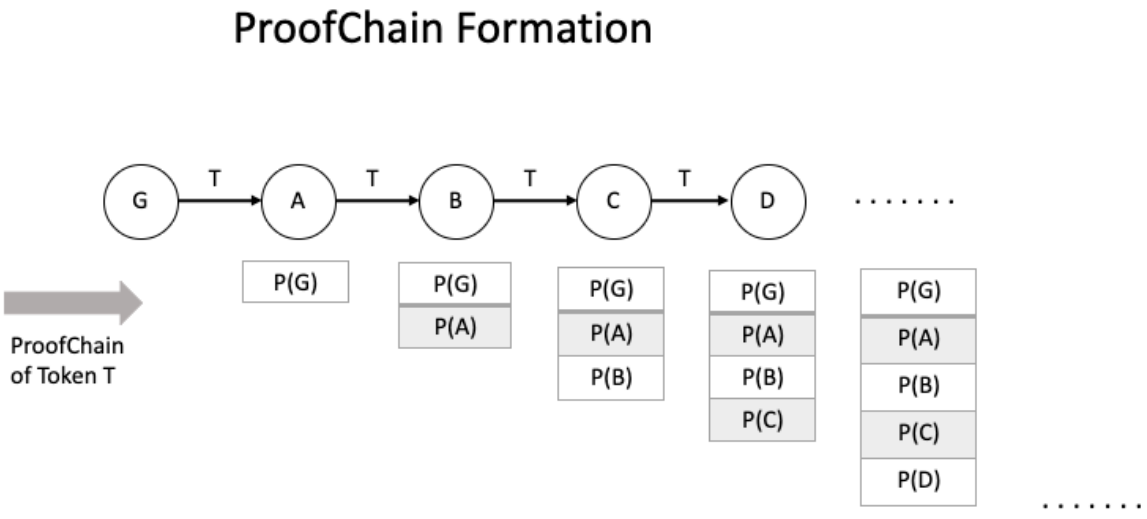
If no fork is found and G is the sole committer of T_x, then G proceeds to complete the transaction with G.

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.



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)

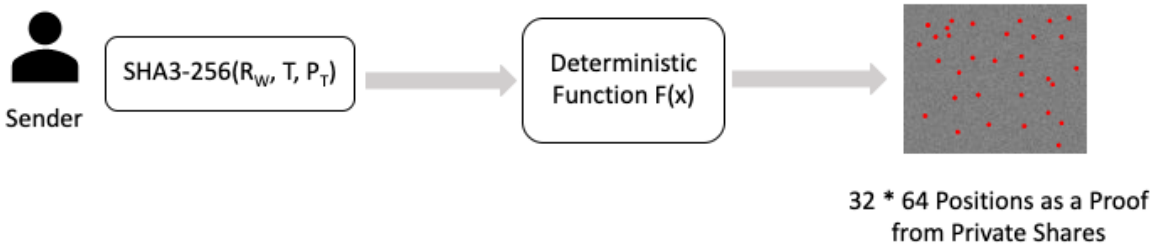
7. The sender calculates SHA-256 hash of receiver's wallet share (R_W), token (T) and the ProofChain (P_T).

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

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].

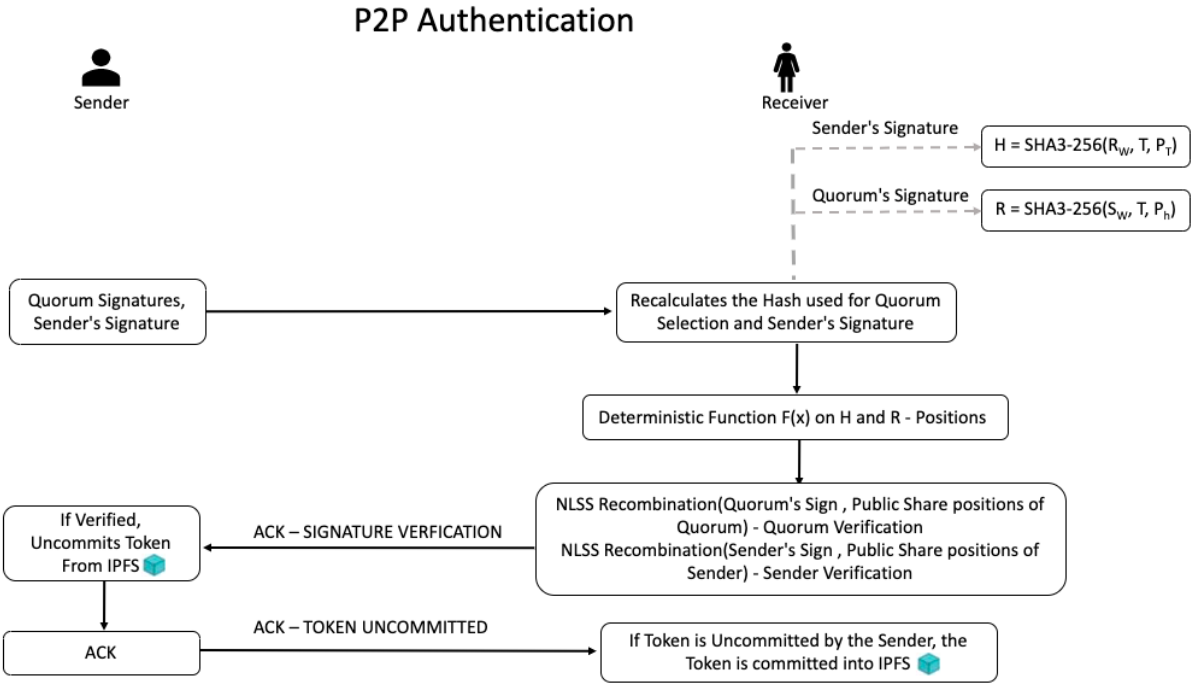
14. ProofChain is used because, apart from the sender no other user in the network possesses the latest token chain of the token. Therefore, no party, not even the receiver cannot calculate the hash and pre-compute the 32 challenge - response positions for the transaction of token T.

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

20. The token to be transferred is unpinned by the sender and pinned by the receiver thereby claiming the ownership of the token

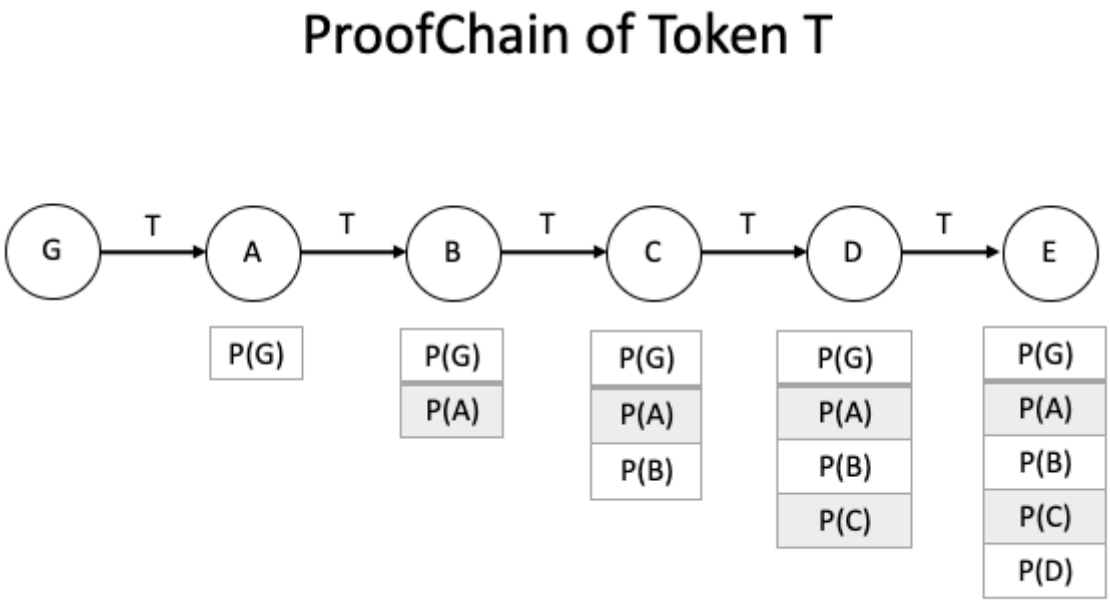
The transaction proof generated in the process detailed from Step 1 to Step 20 is deterministic and highly secure. The proof confirms that A has received T_x from G. A can only claim that it received T_x from G and not any other token T_y. This is because A will not be able to produce the knowledge of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

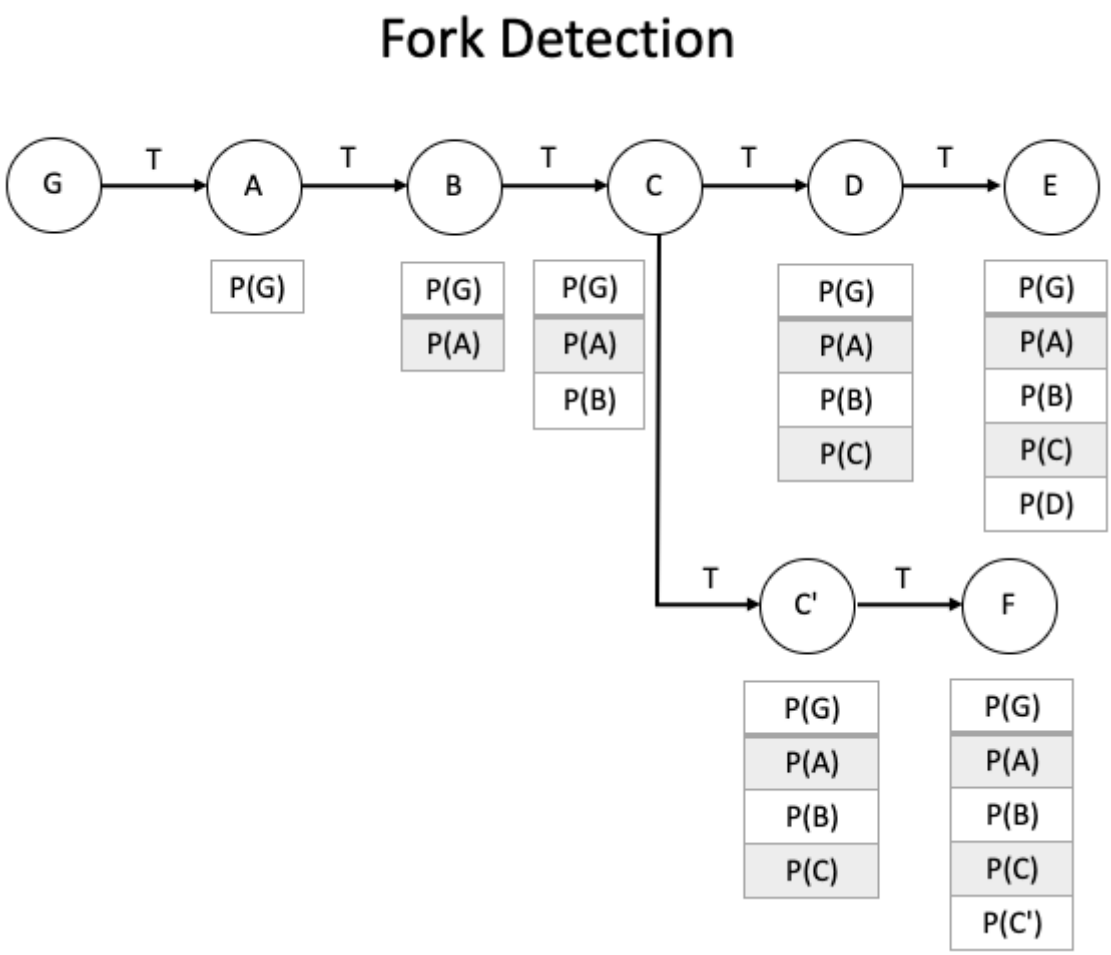
Double spending

Double spending is not possible since a node wishing to enter a transaction to acquire a token 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 IPFS). 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. E will not proceed with the transaction unless C and D provide further proof. Hence double spending is not possible. There is not 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, forking can be used for denial of service 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©



By traversing back, the token chain, it is easy to determine where the fork(s) occurred. The nodes causing fork can easily be determined. While forks can easily be determined, how does the Rubix protocol resolve the forks?

To resolve a fork, each transaction goes through a consensus protocol and the quorum members sign on the transaction hash. In case of a fork, the signature of the consensus quorum members is verified to decide the correct chain

Asset tokens

Asset-backed tokens or security tokens carry an actual value as they are associated with the value of an existing real-world asset. Asset-backed and security tokens provide secure, fast and minimal cost trading of standard assets via blockchain technology, and boost liquidity for conventional securities.

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}.

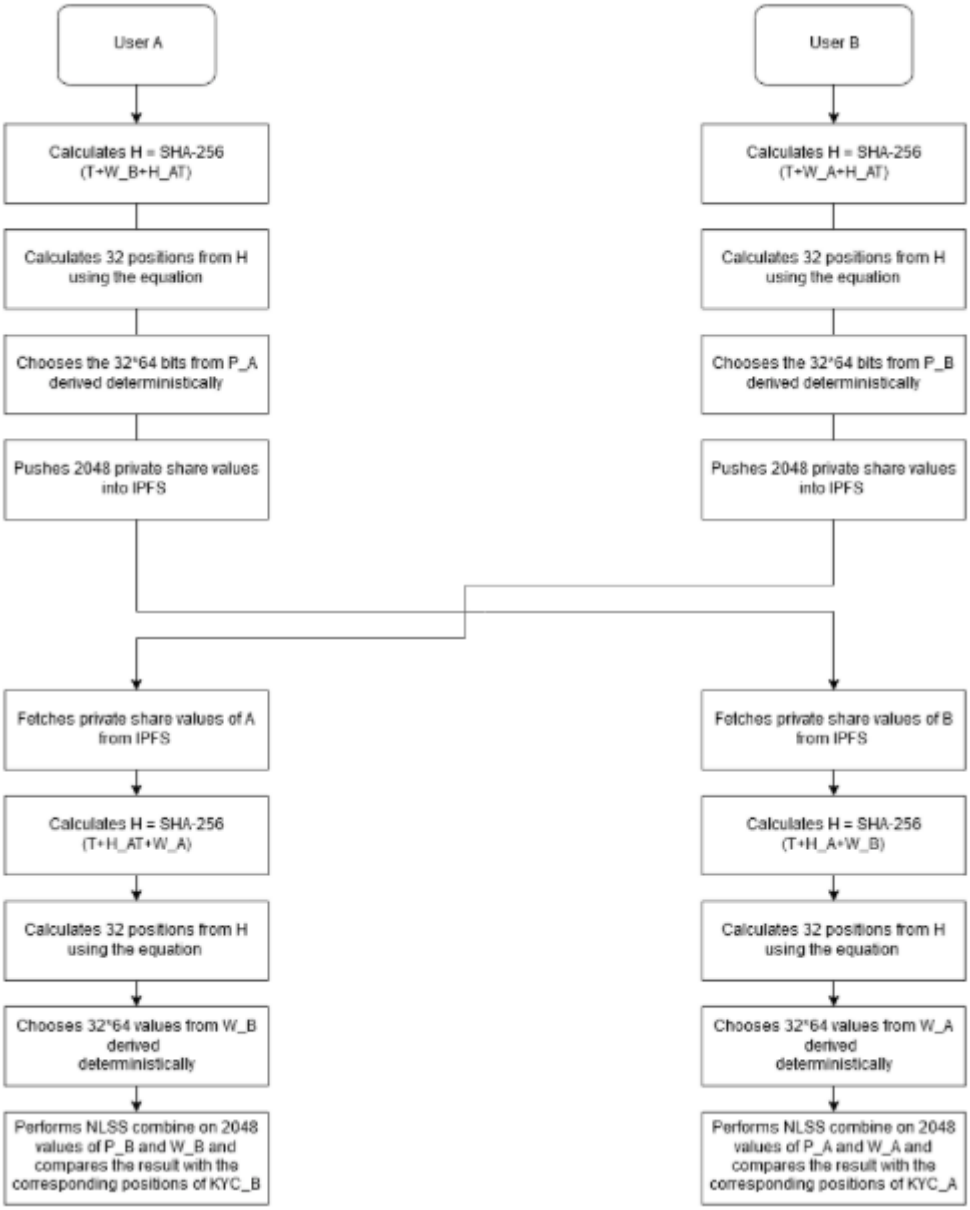
From the calculated hash H, User A deterministically find out 32 positions using a deterministic well-dispersed function.

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 $20, 97, 152$. 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.



A Proofchain 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 commodities. 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.

LIBP2P

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 peerID 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.

IPFS in 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. 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. Commiting 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.