RBX Blockchain Whitepaper

Executive Summary:

This document will detail the ReserveBlock RBX platform as an NFT Centric blockchain that will function as a blockchain and store the needed NFT data & scalable utility use within a decentralized structure. This blockchain will support fully scalable and programmable smart contracts and on chain media storage & transfer to create a truly autonomous NFT driven blockchain that is fully decentralized enabling a complete peer-to-peer ecosystem.

This document will contain links as well to click to learn more about potential technology stacks or tools aggregated, created and integrated with proprietary code. All information in this document is open-source and can also be referenced in the RBX Github. All information below has had a feasibility analysis done and has been determined to be programmatically viable for integration, coding and implementation.

Table of Contents

Introduction

With blockchain technology on the rise and being utilized to create a number of layer 1 and layer 2 applications, an inevitable discovery of obvious weaknesses have organically been revealed through innovation. Two of the largest issues facing blockchains today are their scaling challenges and their ability to maintain consensus. These immediate acknowledgments have triggered rapid research and technology developments in order to address. Another global issue is the high entry cost and support of proof-of-work. The energy inefficiency and hardware cost to actually mine PoW have increased exponentially over the years making it costly to enter and transact, which has caused deviations of underlying Nakamoto's principles of "one CPU one vote". Additionally, with the rise of NFTs popularity it has been seen that current blockchains are strained while trying to serve all facets of their technology, with none intended to be built to serve the massive current and anticipated future demand of NFTs with Smart Contracts as the center driving force of these respective blockchains. To date, there have been some alternative solutions to address some scaling and gas fees issues, however current layer 1 & 2 applications have failed to allow NFTs to evolve from their current static and centralized state, utilizing unsecured storage, limited utility, lack of true royalty enforcement, forced marketplace centralization and limited scalability.

RBX has created a resolution that is a consensus that can impose a masternode rule and as the blockchain scales have a secondary layer that is also present to prevent the Masternodes from running things in an untrustworthy manner. A masternode is defined as a governing hub in some cryptocurrency networks. It requires initial assurance of coins to operate where Masternodes play a role in certain blockchains at the time of the validation of the transaction. Masternodes are required to hold balances in native coins to validate and coins that are not locked for any period of time and can be moved freely at the users discretion. The chain also follows a practical Byzantine Fault Tolerance and implements a new model where a relationship is created between validators, adjudicators, and the chain to create a truly decentralized ecosystem where no one party can take advantage over another.

The RBX protocol is a blockchain designed and scaled to handle the heavy demand of NFTs, as well as evolving NFTs with limitless use cases. RBX will allow both the metadata and the actual digital & physical good locator beacon references, to be cryptographically stored in one single transaction. This means that each transaction will contain the data needed for the NFT and if required, a locator beacon to download the NFT asset peer-to-peer. In essence RBX is designed to help decentralize issuance, governance, media, marketplaces and the tokenization of physical assets all in one chain. This dramatically resolves many of the issues facing NFT platforms and marketplaces today as well as the minters and traders who utilize them. Additionally, the RBX blockchain accomplishes this by using a self-created language called Trillium that allows for programmable smart contracts that are solely focused on driving NFT technology further while being an open source for both blockchain capable programmers and non-programmers with infinite possible use cases.

Introduction to Current Technology and Blockchain Solutions

A Brief History

By now Satoshi Nakamoto's vision of Bitcoin has been realized and due to its popularity blockchain technologies have evolved far beyond even his original vision. Decentralized digital currencies have been around for some time even

before 2009. With the popularity of Bitcoin rising, it became apparent that new alternatives would be needed, alt coins like Litecoin and other forks for the Bitcoin source code had launched, but none of which really changed much other than things like block times and maybe consensus algorithms. Eventually it was theorized that it would be possible to execute pieces of decentralized code in what has become to be known as a 'Smart Contract'. The project that would bring this to market would be known as Ethereum. Ethereum is a Proof of Work based system, however it is in the process of the Eth2 merge which will make it a Proof of Stake system. Ethereum has opened the door to many new markets and products, one of which being the ERC-721, better known a Non-Fungible Token (NFT). NFTs saw an explosion in the market in 2021 and due to this rise scaling has become problematic within the Ethereum network causing slow transaction times and very high fees associated with creating and trading NFTs, as well lack of real utility use. This scope of this paper will not dive into all the things Bitcoin or Ethereum could have or could be doing as both can be justified and said to have a place and purpose, but rather to explain how the RBX blockchain will solve the issues that individuals, enterprise users and marketplaces are facing today, and with new solutions through consensus and on chain tools can help facilitate further adoption as well as use case utility.

Current Mining

To better understand what RBX solves, it is necessary to understand Proof-of-Work mining and how it is currently operating today. Proof of work is a requirement to define an expensive computer calculation, also called mining, which needs to be performed to create a new group of trustless transactions (referred to a block) on a distributed ledger called blockchain.

Mining serves as two purposes:

1. To verify the legitimacy of a transaction or avoiding the so-called double-spending.

2. To create new digital currencies by rewarding miners for performing the previous task.

When a transaction is created the following is what happens within a mining infrastructure:

- Transactions are bundled together into what is called a block.
- Miners verify that transactions within each block are legitimate.
- To do so, miners solve a mathematical puzzle known as proof-of-work problem.
- A reward is given to the first miner who solves each blocks problem.
- Verified transactions are stored in the public blockchain.

This "mathematical puzzle" has a key feature: asymmetry. The work, in fact must be moderately hard on the requester side but easy to check for the network. This idea is also known as a CPU cost function, client puzzle, computational puzzle or CPU pricing function.

All the network miners compete to be the first to find a solution for the mathematical problem that concerns the candidate block, a problem that cannot be solved in other ways than through brute force so that essentially requires a huge number of attempts.

When a miner finally finds the right solution, they announce it to the whole network at the same time receiving a cryptocurrency reward provided by the protocol.

From a technical point of view, the mining process is an operation of inverse hashing: it determines a number (nonce), where the cryptographic hash algorithm of block data results in less than a given threshold.

This threshold, called difficulty, is what determines the competitive nature of mining: more computing power is added to the network, the higher this parameter increases also increases the average number of calculations needed to create a new block. This method also increases the cost of the block creation, pushing miners to improve the efficiency of their mining systems to maintain a positive economic balance. This parameter update should occur approximately every 14 days and a new block is generated every 10 minutes.

Proof of work is not only used by the Bitcoin blockchain but also by Ethereum and many other blockchains.

Conversely, Ethereum developers are attempting to migrate away from PoW using a new consensus system called proof of stake. Proof of stake is a different way to validate transactions and achieve the distributed consensus. Though still an algorithm and while the purpose is the same as proof of work, the process to reach the end result is quite different.

The Proof of Stake concept was first suggested on the Bitcoin talk forum in 2011, but the first digital currency to use this method was Peercoin in 2012, together with ShadowCash, Nxt, BlackCoin, NuShares/NuBits, Qora and Nav Coin.

Unlike Proof-of-Work where the algorithm rewards miners who solve mathematical problems with the goal of validating transactions and creating new blocks, with proof of stake the creator of a new block is chosen in a deterministic way, depending on its wealth, also defined as stake. No block reward.

Additionally, all the digital currencies are previously created in the beginning and their number never changes. This means that in the PoS system there is no block reward, miners earn the transaction fee. This is why in this PoS system miners are called forgers.

Merkle Tree Explanation

Merkle trees are binary trees of hashes. Merkle trees in Bitcoin use a double SHA-256, the SHA-256 hash of the SHA-256 hash of something. If when forming a row in the tree (other than the root of the tree), it would have an odd number of elements, the final double-hash is duplicated to ensure that the row has an even number of hashes. First they form the bottom row of the tree with the ordered double-SHA-256 hashes of the byte streams of the transactions in the block. Then the row above it consists of half that number of hashes. Each entry is the double-SHA-256 of the 64-byte concatenation of the corresponding two hashes below it in the tree.

This procedure repeats recursively until it reaches a row consisting of just a single double-hash. This is the Merkle root of the tree. For example, imagine a block with three transactions a, b and c. The Merkle tree is:

```
d1 = dhash(a)
d2 = dhash(b)
d3 = dhash(c)
d4 = dhash(c)            # a, b, c are 3. that's an odd number, so we take
the c twice

d5 = dhash(d1 concat d2)
d6 = dhash(d3 concat d4)

d7 = dhash(d5 concat d6)
```

Where

```
dhash(a) = sha256(sha256(a))
```

d7 is the Merkle root of the 3 transactions in this block.

Note: Hashes in Merkle Tree displayed in the Block Explorer are of little-endian notation. For some implementations and calculations, the bits need to be reversed before they are hashed, and again after the hashing operation.

Timestamp Servers

The RBX solution starts with a timestamp server or device. A timestamp server works by taking a hash of a block of items to be timestamped and then published widely to the hash. What the timestamp does is prove that the data must have existed at the time to proceed into the hash. Each timestamp includes the previous timestamp in its hash forming a chain, with each additional timestamp reinforcing the ones before it. This is a core principle and contribution of Nakamoto's ideals. The timestamps follow a UNIX epoch standard so that everyone starts from the same point and creates a timestamp that follows strict timing standards. RBX uses its epoch in its ClientCallServices in the DoWorkV3 algorithm.

A timestamp is a unit of time that is measured from a specific point. In the case of RBX the network uses a unix epoch timestamp that time ticks from January 1st, 1970. To create this timestamp, RBX uses a simple DateTime ticks method that returns a timestamp. The code below is the RBX method that returns it.

```
0 references
public static long GetTime(int addTime = 0)
{
    long epochTicks = new DateTime(1970, 1, 1).Ticks;
    long nowTicks = DateTime.UtcNow.AddSeconds(addTime).Ticks;
    long timeStamp = ((nowTicks - epochTicks) / TimeSpan.TicksPerSecond);
    return timeStamp;//returns time in ticks from Epoch Time
}
```

This timestamp is validated on the network and the users timestamp must align with the networks order. What this means is the following:

- A user must have submitted a timestamp that is not less than the most recent timestamp
- A user must not submit a timestamp that is too far beyond the most current crafting block
- If a user breaks either of those two rules then the blocks they have crafted will be rejected per the network.

```
//ensures the timestamps being produced are correct
var prevTimestamp = Program.LastBlock.Timestamp;
var currentTimestamp = TimeUtil.GetTime(60);
if (prevTimestamp > block.Timestamp || block.Timestamp > currentTimestamp)
{
    return result;
}
```

Communication with Other Nodes

An essential part of a node is to share and sync the blockchain with other nodes. The following rules / parameters are used to keep the network in sync.

- When a node generates a new block, it broadcasts it to the network
- When a node connects to a new peer it queries for the latest block
- When a node encounters a block that has an index larger than the current known block, it either adds the block to its current chain or queries for the full blockchain.

Nodes at present use a P2P layer to communicate and share information. SignalR is the tool being used that primarily uses websockets to keep an open connection so messages can be shared easily over the web. A hubconnection is created that allows a user to connect to other peers. This includes validators, adjudicators, and regular peers.

HubConnection code (Basic Implementation):

```
hubConnection1 = new HubConnectionBuilder()
.WithUrl(url, options => {

})
.WithAutomaticReconnect()
.Build();

hubConnection1.On<string, string>("GetMessage", async (message, data) => {
    if (message == "tx" || message == "blk" || message == "val" || message == "IP")
    {
        if (message != "IP")
        {
            await NodeDataProcessor.ProcessData(message, data);
        }
        else
        {
            ipList.Add(data.ToString());
            ReportedIPs = ipList;
        }
    }

});


hubConnection1.StartAsync().Wait();
```

HubConnection Code (Current Implementation):

```
private static ConcurrentDictionary<string, bool> ConnectLock = new ConcurrentDictionary<string,
bool>();
        private static async Task Connect(Peers peer)
        {
            var url = "http://" + peer.PeerIP + ":" + Globals.Port + "/blockchain";
            try
            {
                if (!ConnectLock.TryAdd(url, true))
                    return;
                var hubConnection = new HubConnectionBuilder()
                       .WithUrl(url, options =>
                       {

                       })
                       .Build();

                var IPAddress = GetPathUtility.IPFromURL(url);
                hubConnection.On<string, string>("GetMessage", async (message, data) =>
                {
                    if (message == "blk" || message == "IP")
                    {
                        if (data?.Length > 1179648)
                            return;

                        if (message != "IP")
                        {
                            await NodeDataProcessor.ProcessData(message, data, IPAddress);
                        }
                        else
                        {
                            var IP = data.ToString();
                            if (Globals.ReportedIPs.TryGetValue(IP, out int Occurrences))
                                Globals.ReportedIPs[IP]++;
                            else
                                Globals.ReportedIPs[IP] = 1;
                        }
                    }
                });

                await hubConnection.StartAsync(new CancellationTokenSource(8000).Token);
                if (hubConnection.ConnectionId == null)
                    return;

                var node = new NodeInfo
                {
                    Connection = hubConnection,
                    NodeIP = IPAddress,
                    NodeHeight = 0,
                    NodeLastChecked = null,
                    NodeLatency = 0,
                    IsSendingBlock = 0,
                    SendingBlockTime = 0,
                    TotalDataSent = 0
                };
```

```
                (node.NodeHeight, node.NodeLastChecked, node.NodeLatency) = await
GetNodeHeight(hubConnection);

                node.IsValidator = await GetValidatorStatus(node.Connection);
                var walletVersion = await GetWalletVersion(node.Connection);

                if (walletVersion != null)
                {
                    peer.WalletVersion = walletVersion.Substring(0,3);
                    node.WalletVersion = walletVersion.Substring(0,3);

                    Globals.Nodes.TryAdd(IPAddress, node);

                    if (Globals.Nodes.TryGetValue(IPAddress, out var currentNode))
                    {
                        currentNode.Connection = hubConnection;
                        currentNode.NodeIP = IPAddress;
                        currentNode.NodeHeight = node.NodeHeight;
                        currentNode.NodeLastChecked = node.NodeLastChecked;
                        currentNode.NodeLatency = node.NodeLatency;
                    }

                    ConsoleWriterService.OutputSameLine($"Connected to
{Globals.Nodes.Count}/8");
                    peer.IsOutgoing = true;
                    peer.FailCount = 0; //peer responded. Reset fail count
                    Peers.GetAll()?.UpdateSafe(peer);
                }
                else
                {
                    peer.WalletVersion = "2.1";
                    Peers.GetAll()?.UpdateSafe(peer);
                    //not on latest version. Disconnecting
                    await node.Connection.DisposeAsync();
                }
            }
            catch { }
            finally
            {
                ConnectLock.TryRemove(url, out _);
            }
        }
```

A user will only allow up to 8 outgoing connections (may change in future) and will only accept as many incoming connections as their machine can handle. With a distributed network the more nodes the less burden on every machine. Also incoming connections is not turned on by default and a user can opt to not participate.

The Mining Process

The mining process, or as ReserveBlock RBX defines it 'validating', is the process in which users will assure an amount of native coin to join the network of validators for a chance to craft a block for the current block height reward. The RBX validating process uses a consensus algorithm called Proof of Assurance (PoA). This assurance acts in two main ways (among others):

The first way of assurance is the digital signing of a wallet that has the required amount of RBX coin to join the network of validators and so long as those coins do not leave the validators core wallet, it is permitted to continue to mine on the

ReserveBlock network and craft blocks for a chance at the block reward (seeing halving schedule for rewards). There are no penalties for removing coin and it is always liquid in that you can move it at anytime creating a free will system that does not punish people for their service to the network, hence why an assurance of coin is required, not a non-liquid stake of it.

The second part of this assurance is to act as way points for regular nodes to send and have their transactions broadcasted amongst the network.

To ensure mining is an equal and fair process the network has adopted a randomized approach through theory's such as Nash's Equilibrium, where no one node has any better of an advantage over any other node, as well as game theory to provide a truly randomized outcome every time. A Computer science paper from Princeton University by Matheus V. X. Ferreira and S. Matthew Weinberg proposed an idea with this concept that introduced game theory and Nash's equilibrium produce a unique environment where the winners of a block are not only random, but do not have any advantage over any other validators. More from this paper can be read here: https://arxiv.org/pdf/2107.04069.pdf

This concept was applied to RBX where validators are all given the same task. This task can be outlined below:

1. A validator joins the pool and if not too late for the current block, will be given the current task.
    a. A task is defined as a unit of measurement they must choose without knowing what the correct answer to the task is.
    b. All other validators get the same task, but are not aware of the other answers, or what the answer to the task is.
2. Once a task is processed it is sent back to the adjudicators it will then relay it to the lead for selection.
    a. All incoming tasks contain an answer and a block.
    b. The block is processed with the on-chain consensus rules and if it passed, it will be permitted to enter the list of received task.
    c. If a block is inadequate it is rejected and disallowed.
3. Once a victor is selected the adjudicators are all notified of the winning block.
    a. The adjudicators then push the winning blocks to the validators, who then also send to other peers connected to them.
    b. Once a block is received, a new task is also received and the process starts over again.

To put into perspective this can defined further with this example:

- User A – Has 1 node.
- User B – Has 5 nodes.
- User C – Has 10 nodes.
- User D – Has 15 nodes

User A is competing against 30 other nodes, however the 30 nodes do not have any advantage of solving the task just because they are the larger pool of nodes. This is because once one node solves a task, the other nodes are not aware of this solve and therefore could solve the task the exact same way which provides no added benefit.

User B is competing against all other nodes as well as his own. However, in this scenario should one of his nodes beat his other 4 nodes he will still receive a reward, but the selected node has not received this reward because his 5 nodes beat everyone else, but rather his 1 node out of 5 beat his 4 other nodes and Users A, C, and D.

In game theory you can compare this to a lottery to a degree. The more tickets you have the higher your odds are, but that increase becomes negligible amongst a larger pool, especially when halving and block rewards are taken into consideration.

The mining process can best be broken down to the following lines of code:

```
if (Globals.CurrentTaskNumberAnswerV3.Height != blockHeight)
            {
                var num = TaskQuestionUtility.GenerateRandomNumber(blockHeight);
                Globals.CurrentTaskNumberAnswerV3 = (blockHeight, num, TimeUtil.GetTime());
            }

            await P2PClient.SendTaskAnswerV3(Globals.CurrentTaskNumberAnswerV3.Answer + ":" +
Globals.CurrentTaskNumberAnswerV3.Height);
```

A randomized number is generated and combined with block height and the current time. This is whats known as a task answer to a task question.

Also the pool of adjudicators that are acting as another series of 'checks and balances' also add another layer of randomization in that they create a system in which favoritism cannot and is actively fought against. More on adjudicators below.

Beacon Servers and Clients

RBX includes in its network a solution to transport media/assets between users to continue to keep all aspects decentralized. This solution is known as the Beacon System. The beacon system consists of two elements: the server and the client.

Should a user elect to become a beacon the server is started at wallet startup and a user will dictate the control of this beacon. It can be either public or private, and the amount of time assets are cached for storage is also up to them. In order to submit a beacon, the submitter must be the rightful owner of an NFT and that NFT asset must exist within the users wallet. If the user meets those requirements then the submitter will make a call out the beacon locator which will be included in the users transfer TX and thus allowing the receiving party to have the location of the users NFT assets.

The second part is the client. Each CLI running will have the means to become a beacon client to download the assets as they need. Once assets are downloaded they are wiped from beacon giving sole ownership to the recipient.

Another option to secure assets is for a user to be its own beacon. A user can open the ports on its own firewall and enable self-hosting of assets while allowing other peers to download them directly. This means a self-hosted user can always have assets available for retrieval. This is desirable for creators who want to ensure the delivery of their work is guaranteed and in a 100% p2p and decentralized way.

The normal process of a beacon server to beacon client is as follows:

1. A TX with an NFT asset is being transferred
2. The Minter/Owner of the NFT will either be the users own beacon or they will call out to a beacon existing somewhere on the network, or one they have manually entered through the CLI, GUI, or API.
3. Once connected to a beacon the user will produce a beacon locator string which will contain the needed information for the receiver of the NFT to connect to the beacon, validate, and authenticate the media is their own.
4. That beacon locator is attached to the TX and sent to the receiver.
5. The receiver validates the TX like any other and when confirmed the receiver then uses the locator the call out to the beacon.
6. The beacon will validate the request and if valid, it will allow the client to download the assets. Once all downloaded the beacon will delete the assets on its end forever. This system involves no centralized authorities and uses only the peers of the network for the process to be completed.

Beacons are a measure used to transfer data and giving users the ownership of that data requires them to act responsibly and sync those transactions as soon as they can. Once a network reaches a high level of maturity, then its theorized assets can remain with beacons for extended periods, but until then it will always be recommend to receive a TX as soon as possible or at the time of a transaction.

A beacon has 3 states:

1. Off or not setup
   The beacon is not running and will not accept or relay files.

2. On (Public)
   This means your beacon is running and will accept incoming NFT assets from anyone attempting to relay.

3. On (Private)
   A private beacon is a beacon in which only you may relay NFTs from.

   Wallets you send your NFTs to will be able to call your beacon to receive the files, but cannot send files for the beacon to relay.

Beacon connections are established through TCP and use a network stream to download and upload the information. Data packets are created and sent over the stream to receive the information. They are also encoded with specific case numbers to know exactly what to do. An example of creating a Data packet can be seen below:

```
8 references
private byte[] CreateDataPacket(byte[] cmd, byte[] data)
{
    byte[] initialize = new byte[1];
    initialize[0] = 2;
    byte[] separator = new byte[1];
    separator[0] = 4;
    byte[] dataLength = Encoding.UTF8.GetBytes(Convert.ToString(data.Length));
    MemoryStream ms = new MemoryStream();
    ms.Write(initialize, 0, initialize.Length);
    ms.Write(cmd, 0, cmd.Length);
    ms.Write(dataLength, 0, dataLength.Length);
    ms.Write(separator, 0, separator.Length);
    ms.Write(data, 0, data.Length);

    return ms.ToArray();
}
```

If a beacon is set to private then this beacon is designed for the owner to have a one way p2p connection to the people they are sending NFTs too. This allows an NFT creators from mint to transfer to be in control of the assets and data from start to finish. Beacons are decentralized amongst the network an so this option is not required, but if dealing with more sensitive assets it would be highly recommended.

Adjudicators

Adjudicators are the underlying altruistic portion of the network that connect all validators together and serve as a "checks and balances" to ensure ALL validators are acting in the best interest of the network. Validators connect to the pool of adjudicators and submit their crafted blocks to them, where a lead randomly select a winner based on a task that was delivered to the validator.

Adjudicators, unlike validators, are not required to put up any collateral and act as a network enforcement layer of protection to ensure that validators are acting correctly, as well as remove and warn others about any validators that may attempt to deceive the network.

Adjudicators are an entirely altruistic service created to allow people who cannot be a validator, due to the RBX requirement, but still be involved in the network and are the pseudo police of the validators. A validator is someone who dedicates a machine to allow "X" amount of validators to connect to them and receive task and send those task to the other adjudicators in the pool. Validators connect to adjudicators through the same P2P layer using SignalR. Validators receive the following responses from an adjudicator:

1. A task
2. A task Result
3. Status
4. TX for Mempool
5. Bad Block Submission response

This can be seen in the code here:

```
hubAdjConnection1.On<string, string>("GetAdjMessage", async (message, data) => {
    if (message == "task" || message == "taskResult" || message == "fortisPool" || message == "status" || message
    {
        switch(message)
        {
            case "task":
                await ValidatorProcessor.ProcessData(message, data);
                break;
            case "taskResult":
                await ValidatorProcessor.ProcessData(message, data);
                break;
            case "fortisPool":
                await ValidatorProcessor.ProcessData(message, data);
                break;
            case "status":
                Console.WriteLine(data);
                ValidatorLogUtility.Log("Connected to Validator Pool", "P2PClient.ConnectAdjudicator()", true);
                LogUtility.Log("Success! Connected to Adjudicator", "ConnectAdjudicator()");
                break;
            case "tx":
                await ValidatorProcessor.ProcessData(message, data);
                break;
            case "badBlock":
                //do something
                break;
        }
    }
});
```

The pool of adjudicators process everything on a set interval to ensure blocks continue to move at an average pace of 25 seconds. Once a winning task is found the pool will re-validate the crafted block to ensure it is a valid block. The pool is not not permanent, and can be switched to ensure the network is fair, equal and insure no bad actors are in place. Should a bad actor be found, it is immediately exposed to others and recommend their nodes ban that IP. Adjudicators can also be voted out should one be determined to bad and the equivalent can be voted in.

Voting requires for an adjudicator are as follows:

```
public static bool ValidateAdjVoteIn(AdjVoteInReqs adjVoteReq)
        {
            var result = false;

            switch(adjVoteReq.MachineHDDSpecifier)
            {
                case HDSizeSpecifier.GB:
                    if(adjVoteReq.MachineHDDSize < 200)
                    {
                        return result;
                    }
                    break;
                case HDSizeSpecifier.TB:
                    if(adjVoteReq.MachineHDDSize < 1)
                    {
                        return result;
                    }
                    //
                    break;
                case HDSizeSpecifier.PB:
```

```
                if (adjVoteReq.MachineHDDSize < 1)
                {
                    return result;
                }
                break;
            }

            if(adjVoteReq.InternetSpeedDown >= 100 &&
                adjVoteReq.InternetSpeedUp >= 100 &&
                adjVoteReq.MachineCPUCores >= 6 &&
                adjVoteReq.MachineCPUThreads >= 10 &&
                adjVoteReq.MachineRam >= 16 &&
                (adjVoteReq.Bandwith > 16 || adjVoteReq.Bandwith == 0) &&
                adjVoteReq.TechnicalBackground.Length > 100 &&
                adjVoteReq.ReasonForAdjJoin.Length > 100) { result = true; }

            return result;
        }
```

```
The requirements for adjudicator consideration are as followed:
1. HDD space of 200GB
2. Internet Speed (Up/Down) 100 MB/s
3. CPU Core of 6
4. CPU Threads of 10
5. RAM of 16 GB
6. Bandwidth of 16 TB
7. A Technical background writeup
8. A Reason for the voting of the proposed ADJ
```

Being an adjudicator does not yield any rewards, but it does allow one to take part in the network and monitor and see what validators are producing and doing. As referenced above this will allow people invested in the network to warn others of bad actors and keep RBX decentralized and a continued autonomous network.

```
Start of Consensus at height 593559
EncryptedAnswer Consensus at height 593559
My submission count 2558
Submissions Consensus at height 593559
Number of valid submissions: 4526
DecryptedAnswer Consensus at height 593559
Chosen answer: 172688033
Potential winners total: 30. Requesting total: 18
Received winners total: 18
Submitted Winner Consensus at height 593559
Winner collected total: 30
Winner found. Hash: 4dba489163046598508f27910e25eee87b36947c45097aed211013d49c928971
WinnerHashSignature Consensus at height 593559
Task Completed and Block Found: 593559
01/24/2023 05:00:06
Sending Blocks Now - Height: 593559 at 1674536406011
Done sending - Height: 593559 at 1674536406017
Fortis Pool Processed
```

Adjudicator process for example above.

Private Keys

A blockchain allows anyone to send value anywhere in the world where the blockchain file can be accessed. A user must have a private cryptographically created key to access only the blocks that user owns. By giving a private key that a user owns to someone else, it effectively transfers the value of whatever is stored in that section of the blockchain to the new holder of the private key.

To use the Bitcoin example, keys are used to access addresses which contain units of currency that have financial value. This fills the role of recording the transfer, which has traditionally been carried out by banks. It also fills a secondary role, it establishes trust and identity, because no one can edit a blockchain without having the corresponding keys. Edits not verified by those keys are rejected. Of course, the keys — like a physical currency — could theoretically be stolen, but a few lines of computer code can generally be kept secure at a nominal expense. (Unlike, say, the expense of storing a cache of gold in a proverbial Fort Knox.)

RBX follows the ECDSA curve to produce private keys and with respect to Bitcoin, follows the pattern for then producing a human readable address. The algorithm that does this can be seen below:

```
public static string GetHumanAddress(string pubKeyHash)
{
    byte[] PubKey = HexToByte(pubKeyHash);
    byte[] PubKeySha = Sha256(PubKey);
    byte[] PubKeyShaRIPE = RipeMD160(PubKeySha);
    byte[] PreHashWNetwork = AppendReserveBlockNetwork(PubKeyShaRIPE, Program.AddressPrefix);
    byte[] PublicHash = Sha256(PreHashWNetwork);
    byte[] PublicHashHash = Sha256(PublicHash);
    byte[] Address = ConcatAddress(PreHashWNetwork, PublicHashHash);
    return Base58Encode(Address); //Returns human readable address starting with an 'R'
}
```

A private key can also be encrypted to protect it. When calling keys a method was made to ensure you always get a key back, whether its encrypted or not.

```
public string GetKey{ get { return GetPrivateKey(PrivateKey, Address); } }
```

HD Wallets

RBX also supports the user of deterministic wallets through the BIP-32 and BIP-39 standard. These 12 or 24 word mnemonic phrases can be created and used to have an easier and secure way to manage large sets of keys without having to backup each private key for each wallet address.

Once a mnemonic is created a user can then use a derivation path to create deterministic wallet addresses. This process can be used in both the CLI as well as third party applications like web wallets, and other solutions to continue to create a decentralized and trustless environment for ReserveBlock RBX users.

An example of this would be to produce a 12 word mnemonic like:

```
memory kidney tuition describe rhythm expose display dress unique course midnight notice
```

This Mnemonic produces the following Master Seed in hex.

38584fea368bc7f0d1074d914d41d60006a4b3525c12972d4620c13b9b057200dc0db5a926e5d05e2e0fff86a5db1731d2818466c35b9e82582e43a033c88197

This seed can be used to produce deterministic wallet address with a derivation path. RBX uses the following path:

"m/0'/0'"

The CLI and GUI both support this and since they follow the BIP-32 and BIP-39 standard many current practices for other cryptocurrencies can be applied to easily create other decentralized products or integrate to current ones.

The Wallet encryption process can best be seen here:

```
public static async Task GenerateKeystoreAddresses(bool convertCurrentAddr = true)
        {
            List<Keystore> keystoreList = new List<Keystore>();

            var accounts = AccountData.GetAccounts();
            var accountList = accounts.FindAll().ToList();
            var amount = 1000;
            var progress = 0.00M;
            //Create 1000 keystore addresses
            await AnsiConsole.Progress()
                .StartAsync(async ctx =>
                {
                    var task1 = ctx.AddTask("[green]Generating Encrypted Keystore[/]");
                    var task2 = ctx.AddTask("[green]Encrypting Current Keys[/]");

                    while (!ctx.IsFinished)
                    {
                        for (var i = 0; i < amount; i++)
                        {
                            //generating addresses, then encrypt private key and save to account
db
                            var account = AccountData.CreateNewAccount(true);
                            var keystore = await WalletEncryptionService.EncryptWallet(account);
                            if (keystore != null)
                            {
                                keystoreList.Add(keystore);
                                progress += 0.1M;
                                task1.Increment(0.1);
                            }
                        }
                        task1.Increment(100);

                        if (convertCurrentAddr)
                        {
                            var accountListCount = accountList.Count();
                            double incr = 100 / accountListCount;
                            //Update current accounts to have private keys encrypted.
                            foreach (var account in accountList)
                            {
                                var keystore = await
WalletEncryptionService.EncryptWallet(account, true);
```

```
                        if (keystore != null)
                        {
                            keystoreList.Add(keystore);
                            task2.Increment(incr);
                        }
                    }
                    task2.Increment(100);
                }
                else
                {
                    task2.Increment(100);
                }
            }
        });

    var keystores = GetKeystore();
    if(keystores != null)
    {
        keystores.InsertBulkSafe(keystoreList);
    }
}
```

RBX Blockchain & Decentralized Vision

The ReserveBlock Foundations launch of a Decentralized, NFT Centric Blockchain that functions as an on-chain solution to store needed NFT data & transfer of media for use by any users who are a validated entity to the underlying smart contract. The RBX blockchain supports smart contracts that have true utility, scalable use features, on-chain media / file storage and empower DEX features for users to monetize & / or utilize on a truly decentralized NFT Centric Blockchain with trustless tools. Upon mainnet launch by the foundation, all governance will be held by the validators of RBX and allow for users, minters / creators and NFT traders to utilize their own web-based platforms & social media channels without the need for a centralized authority to complete a transaction.

Vision

The main vision of RBX can be outlined below:

1. **Ease of Use**: Elimination of high entry level burdens for utilization. Provide simple, yet powerful tools and convenience of data storage within a stable and secure environment as the RBX chain scales. Both preprogrammed smart contracts and raw programming available by everyone are accessible from the non-programmer to the most advanced.
2. **Adaptation**: No hard or soft forks. An on-chain set of parameters and rules implemented for new features that can be added and adopted by the community over time to allow nodes to immediately receive new protocols without the need to update, similar to that of an operating system update.
3. **Speed:** The ability to exceed today's standard as a masternode-hybrid infrastructure while achieving faster confirmations and validations of transactions than current systems without the need to solve any complex mathematical problems and maintain a secure scalable decentralized chain.
4. **Unregulated and Non-Discriminatory:** The RBX chain and its protocols do not restrict or attempt to control any specific areas of usage as a centralized authority. All regulatory mechanisms are designed to self-regulate harm and not attempt to stop an application proposed so long as it receives the necessary voting & acceptance as

required by the protocol. Additionally, self-governing protocol mechanisms prevent bad actors from completing transactions that would violate platform parameters.

The RBX network and platform is an alternative to current solutions and protocols that have failed to allow for true decentralization of all aspects of Smart Contracts, NFTs and the currencies that govern. While solving scaling, storage and utility, the RBX protocol provides a convenient mechanism for joining the network as a node and securing block rewards for it through a system called Proof of Assurance (PoA).

Code for this process:

```
public static async Task<string> StartValidating(Account account, string uName = "")
        {
            string output = "";
            Validators validator = new Validators();

            if (Globals.StopAllTimers == true || Globals.BlocksDownloadSlim.CurrentCount == 0)
            {
                output = "Wallet is still starting. Please wait";
                return output;
            }


            if (account == null) { throw new ArgumentNullException(nameof(account)); }
            else
            {
                var sTreiAcct = StateData.GetSpecificAccountStateTrei(account.Address);

                if (sTreiAcct == null)
                {
                    output = "Account not found in the State Trei. Please send funds to desired
account and wait for at least 1 confirm.";
                    return output;
                }
                if (sTreiAcct != null && sTreiAcct.Balance < 1000.0M)
                {
                    output = "Account Found, but does not meet the minimum of 1000 RBX. Please
send funds to get account balance to 1000 RBX.";
                    return output;
                }
                if (!string.IsNullOrWhiteSpace(uName) && UniqueNameCheck(uName) == false)
                {
                    output = "Unique name has already been taken. Please choose another.";
                    return output;
                }
                if (sTreiAcct != null && sTreiAcct.Balance >= 1000.0M)
                {
                    //validate account with signature check
                    var signature = SignatureService.CreateSignature(account.Address,
AccountData.GetPrivateKey(account), account.PublicKey);

                    var verifySig = SignatureService.VerifySignature(account.Address,
account.Address, signature);

                    if (verifySig == false)
                    {
```

```
                        output = "Signature check has failed. Please provide correct private key
for public address: " + account.Address;
                        return output;
                    }

                    //need to request validator list from someone.

                    var accounts = AccountData.GetAccounts();
                    var IsThereValidator = accounts.FindOne(x => x.IsValidating == true);
                    if (IsThereValidator != null)
                    {
                        output = "This wallet already has a validator active on it. You can only
have 1 validator active per wallet: " + IsThereValidator.Address;
                        return output;
                    }

                    var validatorTable = Validators.Validator.GetAll();

                    var validatorCount = validatorTable.FindAll().Count();
                    if (validatorCount > 0)
                    {
                        output = "Account is already a validator";
                        return output;
                    }
                    else
                    {

                        //add total num of validators to block
                        validator.NodeIP = "SELF"; //this is as new as other users will fill this
in once connected
                        validator.Amount = account.Balance;
                        validator.Address = account.Address;
                        validator.EligibleBlockStart = -1;
                        validator.UniqueName = uName == "" ? Guid.NewGuid().ToString() : uName;
                        validator.IsActive = true;
                        validator.Signature = signature;
                        validator.FailCount = 0;
                        validator.Position = validatorTable.FindAll().Count() + 1;
                        validator.NodeReferenceId = BlockchainData.ChainRef;
                        validator.WalletVersion = Globals.CLIVersion;
                        validator.LastChecked = DateTime.UtcNow;

                        validatorTable.InsertSafe(validator);

                        account.IsValidating = true;
                        var accountTable = AccountData.GetAccounts();
                        accountTable.UpdateSafe(account);

                        Globals.ValidatorAddress = validator.Address;

                        output = "Account found and activated as a validator! Thank you for
service to the network!";

                        _ = StartupService.GetAdjudicatorPool();
                    }
                }
                else
```

```
            {
                output = "Insufficient balance to validate.";
            }
        }

        return output;
    }
```

The main items in the code above for an address to join the pool and be in consensus are the following:

1. Account must be found in the state trei.
2. The account must have a balance greater than 1000.00 RBX.
3. Account must produce a valid signature to prove they own this address.
4. Must not have a validator already active on wallet AND network.
5. Lastly if all checks pass then they will connect to an adjudicator.

RBX NFT Smart Contracts

The NFT centric smart contract system contains its own programming language that is computationally universal. These smart contracts are completely decentralized, and their code executed without the need of a centralized authority and without coding knowledge. This language allows users to easily create and customize their NFTs to their own specifications and allowing for new & improved tools to be developed as new use-cases are discovered. These improvements also solve current landscape platform issues such as EVMs and / or other tools that take data away from the blockchain.

The basic structure of an NFT with a royalty appears below:

```
Let RoyaltyType = "1"
let RoyaltyAmount = "10"
let RoyaltyPayToAddress = "Some RBX Address"
let Name = "Some SC NAME"
let Description = "Some Description"
let MinterAddress = "Some RBX Address"
let Address = Some RBX Address
let SmartContractUID = "65b6eec2081b402084f626ed923466f8:16646546846"
let Signature = "some signature"
let Features = "1"
let FileName = "evenmore.jpeg"
function NftMain(data : string) : string
{
    if data == "nftdata"
    {
        return GetNFTData(Name, Description, Address)
    }
    else if data == "getnftassetdata"
    {
        return GetNFTAssetData(FileName, Location, FileSize, Extension)
    }
    else if data == "getroyaltydata"
    {
        return GetRoyaltyData(RoyaltyType, RoyaltyAmount, RoyaltyPayToAddress)
    }

    return "No Method Named " + data + " was found."
}
function GetNFTData(name : string, desc : string, addr : string) : string
{
    return name + "|->" + desc + "|->" + addr
}
function GetNFTAssetData(fileName : string, loc : string, fileSize : string, ext : string) : string
{
    return (fileName + "|->" + loc + "|->" + fileSize + "|->" + ext)
}
function GetMinterAddress() : string
{
    return MinterAddress
}
function GetRoyaltyData(royaltyType  : string, royaltyAmount : string, royaltyPayToAddress : string) : string
{
    return (royaltyType + "|->" + royaltyAmount + "|->" + royaltyPayToAddress)
}
```

RBX State Accounts

The RBX Blockchain contains address accounts. The account contains the following:

- A counter **nonce** that will insure only each transaction is running once to prevent any duplicates from happening.
- The account address or **Key**
- The Accounts **RBX Balance**
- The accounts most recent **state root**
- **The account code hash**

RBX is the internal crypto of the Blockchain. It is used to feed the transactions on the network. It is also a reward to the users that help strengthen the network through the networks assurance system. The accounts as listed above are held in the **public key** and **private key** structure format. If a smart contract is issued against an account it will contain that code for others to then reference. These are the only accounts users can call into to activate their NFT smart contract contracts. This code will always be available on the blockchain once published.

A brand new account:

{

1. Id : 1
2. Key : REWa8G5kgREJdtcvxynweqjWGT63UX8WmD
3. Nonce : 0
4. Balance : 0.0M
5. StateRoot : will not exist until first transaction sent or received
6. CodeHash : “”

}

Transactions and NFT Deployment

A transaction in RBX is when an account is used to sign a package of data and by external accounts. A transaction will contain:

- The receiver of the data package
- A signature from the sender
- The amount of RBX being sent from the send to the receiver.
- Transaction Cost
- TX Data (this is for NFTs)

The first four items are expected in just about all cryptocurrency transactions and should not be anything that is found to be difficult. The transaction cost will however be at near zero value as the network is structured to not have periods of high transaction costs which is alleviated by rewarding the masternodes as opposed to rewarding them transaction costs. The second type of transaction that can happen is the deployment of an NFT. This will contain but may be modified later is the following:

- The account owner address
- Signature from them
- Amount of RBX required to deploy contract (this is rewarded to the data miners of network).
- Smart Contract Code
- Beacon Locator for Asset retrieval

These items will differ from current solutions as there is currently no ability for the storage of media on-chain, however with the RBX contract code being NFT centric allows for a vast array of tools and features to store and transfer media p2p. These on-chain features also allow users to facilitate NFT advancement outside the collectible sphere and into multiple use cases that can benefit greatly from their utility. RBX NFTs have standardized features that are more advanced than their marketplace counterparts and will continue to expand new features as user customization organically grows post beta. The network supports the following features in beta:

- Royalty Enforcement
- Multi Account Ownership on a Single NFT (crowd funding, fractionalization, Joint ownership)
- NFT Locking and Privatization
    - Users can restrict the movement of an NFT and select private data only.
- Media for NFTs stored on-chain no longer requires web based IPFS addresses or centralized storage
- Evolving NFTs or State-Change Event driven NFTs
- Fractionalized NFTs
- Music Based NFTs
- Self-Destructing NFT
- Ticketing NFTs
- Soulbound NFT
- Token NFT
- Wrapped NFT
- Pairing NFT
- Programming language that allow for NFT growth and development

NFTs parameters reside within the smart contract. A contract will be called anytime the NFT is needed and its functions. This is how the NFT tools will be utilized and how the NFT can be sent to a new recipient. A contract, should it provide for, will contain the royalty information to allow for decentralized royalty payment enforcement to the original creator of the NFT.

Example of a base transaction will look like the following:

{

1. Hash: c3d5e34b0bad7286862b75ffde2fa825372a67e838c86d50c9b517a37a9b641c
2. ToAddress: RJhKML8fogVcRCgocyy5G58AVQgVjSDeAP
3. FromAddress: Coinbase_BlkRwd
4. Amount: 32.00
5. Nonce: 0

6. Fee: 0.00
7. Timestamp: 1654872843
8. Data: *null*
9. Signature: *null*
10. Height: 5518

}

A transaction hash is created from the combination of the following in this order:

1. var data= Timestamp + FromAddress + ToAddress + Amount + Fee + Nonce + TransactionType + Data
2. Then the value from the above is taken and a SHA256 hash is ran twice.
    a. `HashingService. GenerateHash(HashingService. GenerateHash(data))`
3. This will produce the hash for the TX above.

The signature algorithm uses ECDSA key signing. You can see the example below:

```
11 references
public static string CreateSignature(string message, PrivateKey PrivKey, string pubKey)
{
    //1. Get signature with message and private key
    Signature signature = Ecdsa.sign(message, PrivKey);

    //2. Base64 the outputted signature
    var sigBase64 = signature.toBase64();

    //3. Base58 public key and remove '04' if it was appended.
    var pubKeyEncoded = Base58Utility.Base58Encode(HexByteUtility.HexToByte(pubKey.Remove(0, 2)));

    //4. Concat the base64 sig and the base58 public with a period '.'
    var sigScript = sigBase64 + "." + pubKeyEncoded;

    //5. validate new signature
    var sigScriptArray = sigScript.Split('.', 2);
    var pubKeyDecoded = HexByteUtility.ByteToHex(Base58Utility.Base58Decode(sigScriptArray[1]));
    var pubKeyByte = HexByteUtility.HexToByte(pubKeyDecoded);
    var publicKey = PublicKey.fromString(pubKeyByte);
    var verifyCheck = Ecdsa.verify(message, Signature.fromBase64(sigScriptArray[0]), publicKey);

    if (verifyCheck != true)
        return "ERROR";
    return sigScript;
}
```

The timestamp is also derived from an in-built function as seen below:

```
20 references
public static long GetTime()
{
    long epochTicks = new DateTime(1970, 1, 1).Ticks;
    long nowTicks = DateTime.UtcNow.Ticks;
    long timeStamp = ((nowTicks - epochTicks) / TimeSpan.TicksPerSecond);
    return timeStamp;

    //returns time in ticks from Epoch Time
}
```

The fee for a transaction is kept at a near zero evaluation of cost. The algorithm responsible for this can be seen here:

```
public static decimal CalculateTXFee(Transaction tx)

        {
            var txFee = new decimal();
            var baseFeeMultiplier = 0.00001M;
            var kb = 1024.0M;

            var txSize = JsonConvert.SerializeObject(tx).Length;

            txFee = (txSize / kb) * baseFeeMultiplier;

            return txFee;
        }
```

Non-Fungible Tokens

Non-Fungible Tokens (NFTs) are currently being viewed through a surface-oriented lens. This means that current blockchains are capable of producing the tokenized equivalent of an NFT, however are not equipped at present to handle storage, utility and scalability. Most in fact are truly limited in actual Smart Contract execution and enforcement, as a result leaving them with no option but to act as a centralized authority. Current blockchain NFT capabilities are limited to:

- Mint a token(s) to represent a physical or digital asset(s).
- Assign ownership to a specific address.
- Change ownership to another specific address.
- Store limited metadata about the item being tokenized.

With early adoption of NFTs, the above actions were sufficient enough for digital collectors and traders, however it has become evident with increased use cases, that a decentralized solution is needed in order to create real efficiencies and scalability that extend far beyond what is capable today. The following are some of the immediate needs identified by creators and minters:

- Royalty enforcement.
- Multi-Sig Single NFT Tokens.
- Assurance (insurance).
- Media Regulation.

- Media Stored within TX chain data (Achieved through the beacon locator).
- Ability for NFT(s) to evolve.
- Programmable NFTs.
- The ability for NFT holders to become their own NFT DEXs.

These features are devoid within the marketplaces today as the respective chains and layer 2 solutions developed were never initially constructed to be NFT Centric and are unable address these issues and scale beyond their capabilities without forking and / or creating bridges which seem to currently scale inefficiencies not actual utility.

In order to save space, smart contract and their metadata is stored in the data section of a TX:

An example of this can be seen below:

Tx Details


```
[
{
"Function": "Mint()",
"ContractUID": "04e97e875dc9464ea331d223c9407b82:1654872825",
"Data":
"H4sIAAAAAAAAE+1ZWW/bRhCeZ/8KQg+FlDC2rCjpAduAYucQEhtFnARI31QdthIdLUVfaf3f8+2QK3J3Z6mlnSYvBUFruTv3fLOXZzSmlCJ6S0u6oQHN8HVD7/D+hZGI9vE2aBfvFsYk2h7N8X1BCx7L6Nu0jfdxBdfva
Ck9S/CP8CSgWuHREt7Sa/pIXfx9Tn9g9CMNYcV7+kQ/0yXar6FvDq2n9Jn+pkP8PYTMK3pB1/hdYnTJ2i+hJ4HE52gtofsC2qc8rvpvwJ+ilZZ8/YcewPZbw3Yf9zF+rwUZHZH7Et9H4FLcc8gZ5vQpZF1grOA4YYpC3jG
4ItazZL1lGUccuSH6p8iZtk9zjiBpRRP8naCtWpHzu0JLj5Yl/7eZ0VqOYfGCo5f8AI12pOfr/IzpnCnGBu8pRlTuU+hYspQE30P8vqc+clFUQBccv8LCMf2Cv0/g1xDfXXrKIwNUx2N4MQJWOmhlY21Q/gn6Dv2Gsafg6
jJ3h/ueWJZM6YxtTeF7UvKhj94V2y7TlaW8YFv0SBHzXVjQtlB8zTFTsk2MfQLuziypU24r3V9KlnXhQ5s9LlO/gbQhW2HK3aEPedXN2bYdrrdhnr0les5B9wpR30H0M48VlZpfZvnXgKtO1cwOrJwxv8qskjtac2Tatys
8sXHi5+gxdlW7x/PFOddsKM4mPJMOS7E4QV/Kniu/m1zRar6JkCFVuylX/gJWtIS+LcxnW2hn9a4593MrFixb9zdy6oStV4hQ+l/y1wki8Q6Rziiba39i7wwUOzNIXFntLWi+xTtGJFccKZ/VZ2xRZvtgHe1wL4oMFf7Ye
Y4FXMYCsmOxMuKNOKjrbWKsniG+mutt4am80seV67o76lu/tV+mRQ34veTsj/NIjEqr3IgpInpo+P8w773iHEeITmbNCFXXyLW41SKjVVeaWx8x68zwK48Ocv/k0XmOZx2F8Jo0I1RYmPn9Lz2ig1JUtIXSaGGhNGpbGBI
5qUImzkwoxWPGNSOPTZxVQaIaczX5ciHVVJ2Z0Iy65JUUQ+2VNOZ6JVFpr+T8SV61gjLVR06bwf5u2r2EaLR3DOHabc4Qbe7upY63Nm+1Rt+MmYgzZuTBaOKdRUPo/eeicIxH/NhI93khIbLKh830m9YGX/wP2dqEV9LUO
DUVJ6yq7Gd+6xPfmPnts9nmc6AcP1uazxPZ6uoqcbWFnjdDrGgKkVB2ZKvCpmhmuxJXwt5aQjPY2tZaaiFfPzprC9BdeaKYZdC1RSFw15GoLG9ukHdwJy9aji7XHwlDDcxog3y1UHuelPdEZduynY0p49bqqXeToSu0nN/
qqLTWNX4rWGN6VEfqluXP3WIj4/0oR8X9EO/DukJJO6e5D04f5TgNQeYe62yJ65ocKzMGRbS+N17ugxDp5FDHVxcZh9hNDVj72KP3lNfBIWdF3cl9a/TscXTrzzIR/YRHxuN+CZHFIr+2jjo5dXnD8xha33LmepWravU+Q
N9pF/Vs3s/0MXJjUOgzmHzP2gd18ZT5it29fQc0ZR3qjkpFaRtSF/ldkmm7vjk27ffR9UD1LD+xfDao7R2hvJLae7uwk6kdlepzTvVpS/Z6E6Xpt3+vKWPp2b2w1KnE0gfe0Q5qoEndx6+MO/nNaLpkLRf/4+k748n8j46
08y50FutIvNZWjq27zmRfEgq1VBuNbYey0CDRZruRKpn7lZLkPYpPnnTrWaX94Afq3ruzbone1C7xtnMa9X4FX4v6VJgdAAA="
}
]
```


The actual Data in this section is the entire NFT contract code. The code is compressed and then base64 encoded to produce a much smaller string that will not increase the cost of sending maintaining a near-zero approach.

Since this was a mint there is no beacon information as it is not needed. If this was a transfer one would see a beacon locator in here as well so the recipient could retrieve the asset(s).

Smart Contract Code Execution

The code in an RBX smart contract is written in a basic level language and ran on RBX nodes. This is known as “RBX Assembled Compiler Code” or “RAC code”. The code is taken from the smart contract when called and pulled into the node or machine calling it and broken down into operations that the “caller” can then use to execute commands. The code when called does the following:

- Pulls the current code into memory and will begin processing from start to finish, or line 0 to *infinite*.
- Once code is pulled it is decompiled and then assembled into executable parameters the “caller” can use to manipulate the NFT as allowed from the smart contract.
- Once the “caller” no longer wishes to interact with the NFT the code is cleared from memory and is no longer used unless the “caller” wishes to store said data for easier use in the future.

This process is specifically designed to allow the “caller” to do as little work as possible. Once the code is decompiled for use the “caller” simply chooses the functions needed and once done can terminate the code and clear from memory. The user will also gain access to the smart contract data like Block, owner, cost, etc.

An example of taking the decompiled smart contract below:

```
var repl = new TrilliumRepl();
repl.Run("#reset");
repl.Run(textFromByte);

var scUID = repl.Run(@"GetNFTId()").Value.ToString();
var features = repl.Run(@"GetNFTFeatures()").Value.ToString();

var minterName = repl.Run(@"MinterName").Value.ToString();
var name = repl.Run(@"Name").Value.ToString();
var description = repl.Run(@"Description").Value.ToString();
var minterAddress = repl.Run(@"MinterAddress").Value.ToString();
var address = repl.Run(@"Address").Value.ToString();
var signature = repl.Run(@"Signature").Value.ToString();

var extension = repl.Run(@"Extension").Value.ToString();
var fileSize = Convert.ToInt32(repl.Run(@"FileSize").Value.ToString());
var location = repl.Run(@"Location").Value.ToString();
var fileName = repl.Run(@"FileName").Value.ToString();
var assetAuthorName = repl.Run(@"AssetAuthorName").Value.ToString();

var mainData = repl.Run(@"NftMain(""nftdata"")").Value.ToString();
var mainDataArray = mainData.Split(new string[] { "|->" }, StringSplitOptions.None);

var assetData = repl.Run(@"NftMain(""getnftassetdata"")").Value.ToString();
var assetDataArray = assetData.Split(new string[] { "|->" }, StringSplitOptions.None);

var smartContractMain = GetSmartContractMain(name, description, address, minterAddress, minterName, scUID, signature, features);
var smartContractAssset = SmartContractAsset.GetSmartContractAsset(assetAuthorName, fileName, location, extension, fileSize);
smartContractMain.SmartContractAsset = smartContractAssset;
```

Blockchain

The RBX Blockchain is inherently similar to the Bitcoin Blockchain albeit different in many ways, we share the same sentiment that network does. The main differences are the ability for storage of smart contract code and media for NFTs, how mining happens, transaction cost, and the over network topology is different, but the ethos can be felt in all these areas.

The block structure however is similar to Ethereum in that the network does the following:

1. Checking if the previous block referenced exist and that is passes validation.
2. Checking the timestamp to ensure this is a block in the future and not the past. There are limits to this to avoid block collisions.
3. Check the block number, diff, transaction root, uncle root, and the media.
4. Ensure masternode nonce is valid.
5. Other checks will then validate previous block into current block.

Masternodes

What do they do?

There are 3 different kinds of nodes on the RBX network, a peer node that shares blocks to download, an adjudicator node which as previously mentioned is altruistic to the network, and a Full node which is a device running an RBX client on the RBX network that allows peers to connect to them to receive updates about all events on the network. This node has the entire chain data synced to the nodes server or device.

These nodes are very important for securing the legitimacy of the network as well as its health. They provide all the new data, connecting the clients ability to synchronize with the network and validate the transactional messages going across the network.

Current Issues

Networks have seen many full nodes disappearing across networks like Bitcoin and Ethereum because the incentive is no longer feasible to become a full node and is considered a burden on the owner. Only if the owner is able to mine to their full node is the cost justifiable. Large data storage and bandwidth are needed to be a truly successful node, which is another current barrier to independent decentralized participants.

As the cost rises the number of full nodes will inevitably decrease and possibly centralize mining overall. This could and in some cases prove to be very dangerous as a network is only as strong as its diversity.

Masternodes as a solution

Masternodes are full nodes, however they do not provide the service of both node and miner to the network and pledges its RBX native coin as collateral (assured) in order to participate as a node on the network. This collateral is never taken and is secured to the node while it is operating. This allows the masternode to operate on the network and provide the service needed to secure a transaction. These nodes earn payment (rewards) for their services to the network like Bitcoin mining and drastically help reduce volatility & friction. Since the coin is not locked a client may choose to stop being a masternode at anytime with no penalty or punishment.

Becoming a Masternode

To turn a client into a masternode the owner of the client must secure an address belonging to them and have control over the address with (X) amount of RBX native coin. While the client is active, the masternode provides the service to the blockchain similar to that of a miner and as a reward for this service their client receives block rewards that are also similar to a miner, but they are not solving incredibly complex problems and therefore will not need specialized hardware. A masternode may run in the background on any laptop as an example.

The primary requirements are outlined as the following:

- Must be a unique address
- Must contain 1000 RBX
- Must have a validator name

Masternode Rewards

The obvious truth is that the number of masternodes will fluctuate over periods of time and because of this the expected masternode reward will also vary in accordance to the current number of **active** masternodes.

We can calculate the standard payment with the following equation:

*(m / tm) * r * b * p*

Where:

m is the number of masternode clients someone owns

tm is the total masternodes active on the network

r is the current block reward

b is the average blocks being submitted in one day

p is the average payment for a masternode.

Below is the current structure for how a block reward is determined. RBX uses a halving schedule that roughly halves every ~3-3.7 years or every 4,730,400 blocks.

```
2 references
public static decimal GetBlockReward()
{
    decimal blockReward = 32.00M;
    int currentBlockHeight = Program.BlockHeight != -1 ? (int)Program.BlockHeight : 1;
    int blockHalvingInterval = 4730400; // Roughly every 3 year halving

    //An int will always result in a rounded down whole number. 4730399 / 4730400   = 0 | 9,460,799 / 4730400  = 1
    int halving = currentBlockHeight / blockHalvingInterval;

    var n = 1;
    while (n <= halving)
    {
        blockReward /= 2;
        n++;
    }

    return blockReward;
}
```

Reward and Halving Schedule

```
================================================================
Year 0 - No Halving Block Reward: 32.00
Coins mined at beginning of chain start: 78,255,056 (Chain was reset during alpha a few times
hence the larger number at genesis block).
================================================================
Halving Number 1) New Block Reward: 16.00 at year 3.75
Coins mined at beginning of halving: 220,627,184 at a block reward of: 32.00
Total percent of coins mined: 59.31%
Block Halving Num: 1. Block halving interval: 4449129
================================================================
Halving Number 2) New Block Reward: 8.00 at year 7.5
Coins mined at beginning of halving: 296,313,584 at a block reward of: 16.00
Total percent of coins mined: 79.65%
Block Halving Num: 2. Block halving interval: 4730400
================================================================
Halving Number 3) New Block Reward: 4.00 at year 11.25
Coins mined at beginning of halving: 334,156,784 at a block reward of: 8.00
Total percent of coins mined: 89.83%
Block Halving Num: 3. Block halving interval: 4730400
================================================================
Halving Number 4) New Block Reward: 2.00 at year 15
Coins mined at beginning of halving: 353,078,384 at a block reward of: 4.00
Total percent of coins mined: 94.91%
```

```
Block Halving Num: 4. Block halving interval: 4730400
============================================================
Halving Number 5) New Block Reward: 1.00 at year 18.75
Coins mined at beginning of halving: 362,539,184 at a block reward of: 2.00
Total percent of coins mined: 97.46%
Block Halving Num: 5. Block halving interval: 4730400
============================================================
Halving Number 6) New Block Reward: 0.50 at year 22.5
Coins mined at beginning of halving: 367,269,584 at a block reward of: 1.00
Total percent of coins mined: 98.73%
Block Halving Num: 6. Block halving interval: 4730400
============================================================
```

Trustless Plenum

On network launch, masternodes are in place by requiring (x) amount of RBX native coin and assured to be a node. This system is built in which no one entity can control the entire network or detrimental critical mass of masternodes.

For example, if an entity wanted to control 51% of the network, the client would need to purchase over half of the RBX coin required to exceed 51% of the masternodes active in the pool at any given time. As the client purchased this vast amount, the price of RBX would rise inevitably making it implausible and non-feasible for the client to purchase the full amount. In the event someone does manage to get 51% of the current validator pool and submitted a bad block, this block would have to be accepted by the adjudicators and if by some way made it through the adjudicator pool, the other 49% of the validators would reject that block and request a new task to create a new block and would no longer accept blocks from the crafter or the adjudicator that approved it. This is the final layer of consensus that will forcibly put the bad 51% actors into their own chain while the rest of the chain will then operate without them.

With the masternode network and the minimum RBX requirements, the network can be utilized to do highly sensitive tasks in a trustless environment, where no single entity can control the outcome. By selecting X number of random masternodes from the total active pool to perform the same task, these nodes act as an oracle, without needing the whole network to perform that same task.

RBX Consensus Algorithm

The RBX Consensus Proof of Assurance (PoA) is a system in which validators are agreed upon and the pool of them is created with those validators then each agreeing on the submission of blocks and the transactions inside them. Proof of Assurance is based on all validators agreeing to put up a specific amount of RBX and then honestly and accurately crafting transactions into blocks for the network and acting as beacons for asset moving.

Proof of Assurance (PoA)

Proof of Assurance is the process in which a node will put up a (X) amount of RBX to become part of the consensus pool of other validators that will agree on blocks and transactions making it to finality. So long as a node has the required coins in place they can join the validator pool and have a chance at solving blocks for the reward and helping the network continue to move forward.

Consensus is reached by all adjudicators agreeing a node meets the requirements:

- 1000 RBX
- Unique Address
- Unique Environment
- Funds remain during validation

If these requirements are all met, a validator is allowed into the pool and others will agree.

Once in the pool agreement on transactions and blocks are met through a series of approvals, mainly:

- Balance/TX checking
  - `var txResult = await TransactionValidatorService.VerifyTX(transaction, blockDownloads);`
- Hash checking
  - `if (!newBlock.Hash.Equals(block.Hash))`
- Previous Hash checks
  - `if (!newBlock.Hash.Equals(block.Hash))`
- Signature checks
  - `var verifyBlockSig = SignatureService.VerifySignature(block.Validator, block.Hash, block.ValidatorSignature);`
- Timestamp checks
  - `if (prevTimestamp > block.Timestamp || block.Timestamp > currentTimestamp)`
- Merkleroot check
  - `if(!newBlock.MerkleRoot.Equals(block.MerkleRoot))`
- Blockchain Ref ID must match
  - `if(block.ChainRefId != BlockchainData.ChainRef)`
- Block Versions must match
  - `var blockVersion = BlockVersionUtility.GetBlockVersion(block.Height);`

The chain also has its own replay and double spend protection.

```
public static async Task<bool> DoubleSpendReplayCheck(Transaction tx)
public static async Task<bool> HasTxBeenCraftedIntoBlock(Transaction tx)
```

All the above must match perfectly with every other node. If there are any irregularities, then the other nodes will reject the blocks due to consensus failing. Should consensus fail a new block is requested by the validators and adjudicators will be required if enough of the pool request.

Anti-Inflationary Measure (TX Fee Burning)

RBX is taking measures to ensure inflation is mitigated in addition to a fixed lifetime supply. All transactions fees, while they are near zero, are burned on the tx.

Another measure put in place is a finite supply of the coin with block rewards halving roughly every ~3 years to reduce the amount of 'new' coins being created and put into circulation.

At the time of a craft all the transactions in a block fees would traditionally go to a miner. Since RBX fees are near zero, the network still requires users to send those fees, however at the time of crafting they are not concatenated into one TX for the crafter and instead are dropped over thus forever removing those coins from the ecosystem.

Trillium

Trillium is the smart contract language that powers RBX smart contracts. This is from scratch language that has roots from C#, C, and some JavaScript.

This language is Turing complete and has many features. Below are most of the current implemented features:

- Numbers (ints)
- Strings ("Any")
- Booleans (true, false)
- Addition ('+')
- Subtraction ('-')
- Multiplication ('*')
- Division ('/')
- Basic Operator Tokens(&, !, ^, |, =, (,), >, <, [,], :, ',', &&, !=, ^=, ==, >=, <=, and more)
- If Statements
- Else Statements
- Else If Statements
- Returns
- While Loops
- For Loops
- Do While Statements
- Block Statements
- Expressions
- Variable Declaration
- LiteralExpression,
- Name Expression
- Unary Expression
- Binary Expression
- Parenthesized Expression
- Assignment Expression
- Call Expression

The Languages purpose is to drive all self-executing NFT (SENs) functions.

Some examples are below:

```
function HelloPerson(name : string) : string
{
    var text = "Hello " + name
    return text
}
HelloPerson("Trillium")
```

outputs Hello Trillium

```
» function calculator(firstNum : int, secondNum : int, operator : string) : string
· {
·     var result = 0
·     if operator == "+"
·     {
·         result = firstNum + secondNum
·         return string(result)
·     }
·     else if operator == "-"
·     {
·         result = firstNum - secondNum
·         return string(result)
·     }
·     else if operator == "*"
·     {
·         result = firstNum * secondNum
·         return string(result)
·     }
·     else if operator == "/"
·     {
·         result = firstNum / secondNum
·         return string(result)
·     }
·     else
·     {
·         return "No known operator was given"
·     }
· }
```

```
» calculator(2,2,"+")
4
» calculator(4,2,"+")
6
» calculator(4,2,"*")
8
» calculator(4,2,"/")
2
```

Trillium was inspired from a multitude of sources and was created with the help of Microsoft's C# and Microsoft's Immo Landwerth teachings on programming languages.

You can learn more about Trillium and try the language currently at: https://trillium.rbx.network/

Node Voting

In order for the network to be truly decentralized and autonomous a voting mechanism will be activated on mainnet to give the validators helping secure the network the proper channel and voice to improve the protocol over time. Validators will each be given one vote per cpu and the right to propose network changes, updates, and anything else that they believe will be a benefit to the network. The process of voting will be the following:

1. A proposal will be submitted with adequate information and this proposal will have a unique ID. This proposal can start on any website, forum, p2p layer, etc. It does not matter where just somewhere other validators can publicly see it. This proposal will have a base signature that will only allow them to submit the proposal once it reaches approval.
2. Once a proposal has been seen, validators can take that unique id and sign it in their wallet. This is the time validators should research the proposal to ensure the developer(s) submitting can do what they are suggesting.
3. Once a proposal has reached 51% amount of signatures (as defined by the size of the pool) then the original author of the proposal can take the all the signatures and the original base signature with the proposals unique ID and submit this to the network for ALL validators to receive and then vote on-chain for it.
4. Voting timeline will be defaulted to 30 days. Should more time be desired the Author can increase to a max of 180 days.
5. All votes are collected and published to chain. Once deadline has passed everyone can view results.
    a. If it passes then changes will be submitted to the main repository and pulled by all forks and accepted. A pass is 51% or more votes in favor of.
    b. If it fails then nothing is done and a record of this fail is recorded on-chain.

Every change to RBX will be required to undergo this process so that no one, even the original creators, can do anything to the chain that the majority of the network does not agree with. Should changes be made without the approval of network, then those changes can be rejected and validators can refuse the update of those changes.

Applications

With an NFT centric blockchain, RBX applications are considered to be limitless. While the current areas of collectibles, art, gaming, and tokenization of physical assets are areas that immediately benefit, there are multiple use-cases that will continue to evolve and why the chain is completely adaptable without the need for forking.

Tokens

RBX allows for the creation of tokens to exist. The way tokens work is very similar to today's standards; however, each user will receive a copy of the smart contract to be stored locally on their wallet acting as the token. This will allow them to send, receive, and burn tokens as they see fit.

Tokens can be defined as:

1. The token supply
2. Token features (burn, assure, hold, send, balance)
3. Token Ticker
4. Token Name
5. Token mint address

Tokens are a feature that will enable users to create digital currencies that can be transferred amongst other users to provide a wider range of use on the RBX network.

Evolving NFTs

NFTs with a programming language can be programmed with set standard of events that allow for them to adapt with given parameters and evolve over time. NFTs issued that evolve with given parameters will not have a need to issue multiple NFTs as the use may now encompass code with specific customized parameters for what to show, do and how they should evolve.

Tokenized Physical Goods

NFTs can be linked to physical goods to act in their place. This allows for transparent and easy transfer of assets that provide a legitimate proof of ownership.

Gaming

Gaming stands to benefit heavily from NFTs as they can now tokenize in-game items such as weapons, armor, skins, devices which can be used in-game and used to create secondary markets. Gaming NFTs can provide abilities to avatars or determine the outcome of combining certain variables.

Music

Artist can now issue digital albums with publishing stored directly on chain. With RBX royalty enforcement artists can also receive royalties every time music is purchased / transferred without the need for centralized authorities.

Art & Collectibles

Art & Collectibles (digital and physical) are currently the highest use cases for NFTs at the moment. To date, the majority of these NFTs are minted and traded on centralized marketplaces and closed end systems. The RBX platform provides for on-chain storage and p2p transfer of the media and the ability to add utility & functionality to these NFTs that evolve and are completely programmable.

Name Service and Domains

With the RBX blockchain, a user can secure name services and domains to then serve decentralized web applications completely devoid of any centralized authority. The network additionally provides for minters to take complete autonomous control over their NFT assets and provides the infrastructure support for those assets to be deployed over the users own web-based platforms, social media accounts or any other means of remarketing or trading RBX NFTs through its decentralized open-source network.

Real Estate

Houses, buildings, land, development projects and other related assets (both virtual & physical) can now be minted into an NFT which can be tracked in a trustless manner with immutable proof of ownership. Transactions are able to occur between owner of the asset and the buyer(s) of the underlying asset without the need of an intermediary.

Additional Programmable Verticals

Additional programmable applications such as virtual worlds, sports, fashion, ticketing, alternative assets, logistics, crowd funding, branded assets, and private tokenization are just some of the known use cases that are deployable today on the network. The RBX network is agnostic to all known and unknown use cases as the platform is developed to

provide programmable smart contract & deployment in a decentralized environment that adapts, scales and provides NFT utility controlled by each user not a centralized authority.

RBX Coinomics

The RBX native coin is the in-platform currency for the ReserveBlock network. This coin creates an internal ecosystem and brings value to the platform through the coins utility by exchange of it for services, transactions, transfers, tracking, mining and governance in marketplaces and swaps. The RBX coin is also a reward currency to users for participation, development, and verification. Additionally, the coin held by validators has voting rights for the migration, direction and expansion of the RBX decentralized Blockchain. The coin will be swappable and users will have access to an organic market that will not be influenced or manipulated by the platform developers or foundation.

The coin will have standard functions will be defined as follows:

- Fixed supply of 372,000,000 RBX
- 67,500,000 to be minted into the platform and with set reserves and pre-mines for the platform issuance where needed. Additionally mined during the testnet by validators.
- Send and receive standard functions.
- Rewards for Masternodes, Nodes, and decentralized developer participation.
- Utility use as platform currency to purchase services, transfers and swaps.
- One vote, One CPU.

Miscellanea

Media Storage

One of the obvious issues is that media cannot be stored uncompressed. As the network scales media it must be stored in a way that creates as little friction as possible.

One of the solutions involve compressing the media, however that would then require multiple beacons.

Media Storage Abuse

Beacons will only accept files that are 150MB or smaller. This is more than enough for standard sized NFTs. If someone is trying to mint other large types of media they will need to use more reliable storage measure or other decentralized tools like IPFS, or open their machine to be a beacon itself to house the data to be downloaded.

Size Scaling

One issue all blockchains face is the size of their blockchain. For example, if a blockchain starts at 20 GB and is growing at 1 MB per hour or If a network were to process transactions in the thousands per second thus causing it to grow by 1 MB per three seconds (1 GB per hour, 8TB per year) the network can risk the centralization of the blockchain as the data will be limited to who can actually store it.

One ways RBX is handling the above is by not requiring the media stored on the blockchain to be synced but to still be a full node. The beacons will house the data only as long as it is needed. Once the media is claimed from the rightful

owner they will delete the data to make room for new data incoming. This way no node is incurring the cost of data storage and the responsibility of the asset lies with the receiver and the sender.

Smart Contract Loops

Something to note with a smart contract based blockchain that will support a Turing-complete is that infinite loops can happen.

If a user were to create a malicious smart contract that when sent to nodes for execution the contract would then cause an infinite loop until the node would eventually timeout. Since the RBX network does not use Gas like Ethereum, there will have to be a time check. Another solution would be to pre-test code to prevent this specific event. If the code were to enter a loop and sit for (X) amount of minutes then it could be determined that the code is malicious in that regard and it would be rejected. To ensure this works the network could take it a step further and have an oracle section of masternodes to test this and if they sit in an infinite loop for (X) amount of minutes then the smart contract could be rejected across the network and not published to the other nodes.

Trillium has now been coded to address this concern with recursion protection, as well, as a system in the syntax tree that will detect looping token words and throw a new item into the diagnostic bag to prevent the loop, or only allow it to run for a limited number of cycles that does not affect the rest of the code or functions of the wallet.

Conclusion and Final Thoughts

The RBX Blockchain allows for further expansion, utility and scalability of NFTs. The protocol provides for a decentralized open-source network agnostic to industry, category, or marketplace and allows for a “everyone, by everyone” ethos to leverage programmable tools enabling a globalized DEX without vertical specificity. Supported through a decentralized node infrastructure, a suite of on-chain tools and storage enables organic growth of an ecosystem that helps drive NFTs forward into a strong dominating global marketplace in lieu of centralized control. The RBX network empowers all users with programmable smart contract automation without limitation to asset class, experience or action.

The protocol additionally helps solves the ever growing demand the mining impact has on the environment by embracing a carbon neutral solution through a Proof of Assurance node infrastructure, while maintaining decentralization and reward system. With the RBX Blockchain being completely dedicated to NFTs and the tokenization process, the imbedded suite of tools will inevitably help scale NFT utility well beyond the current global market utilization today.

References and Citing's

1. The Bitcoin White Paper – By Satoshi Nakamoto
2. Bitcoin Community Wiki - https://en.bitcoin.it/wiki/Main_Page
3. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. By Vitalik Buterin (2014)
4. Proof-of-Stake Mining Games with Perfect Randomness By: Matheus V. X. Ferreira and S. Matthew Weinberg - https://arxiv.org/pdf/2107.04069.pdf
5. Programming The Blockchain – Community Edition - https://programmingblockchain.gitbook.io/programmingblockchain/
6. Overview of ASP.NET Core SignalR – Microsoft https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-6.0
7. RBX GitHub - https://github.com/reserveblockio