Rublix

Whitepaper

Abstract

Until now, the primitive nature of blockchain technology and the lack of on-chain resources have hindered complex blockchain innovation and has posed restrictions on the development of exciting and elaborate DApps. Many blockchain architectures suffer from a number of issues: extensibility, scalability and on-chain data availability. We believe these problems stem from the relationship between two very important parts of the consensus architecture - canonicality and validity - being too closely associated. Furthermore, the use of Oracles adds a centralized party to manage off-chain data which defeats the true purpose of blockchains' core functionality.

This Rublix whitepaper introduces an architecture that integrates a strategic consensus method for reliable market data accessible on-chain.

Rublix is proposing a trustless and decentralized blockchain that enables real-time market pricing data to be accessed on-chain while maintaining integrity by utilizing several data sources and a unique consensus agreement to determine accurate values. This gives traders, fund managers, investors and users of any kind the power to write smart contracts with the added ability of executing commands based on an underlying price of a stock, cryptocurrency, foreign exchange market, et al.

Table of Contents

References

Blockchain Mission

In order to develop an efficient and high quality blockchain for worldwide use, the Rublix Blockchain will need the ability and flexibility to address the following criteria:

Purpose & Functionality

The Rublix Blockchain will source financial data from multiple data feeds and post the authenticated information on-chain. This can be used to build decentralized applications that require trusted market data. Moreover, due to the complex nature of the blockchain and to encourage widespread adoption, Rublix users must be welcomed with an interface that is easy to navigate, understand and use. The Rublix blockchain will be built as an Ethereum Side-Chain with a modified consensus protocol to handle common scalability issues present on the Ethereum mainnet.

Quality of Information

In order to ensure the highest quality of information, the appropriate incentive models and consensus mechanisms must be established. Rublix is implementing a novel verification process combined with the most advanced architectures on the market to ensure utmost integrity for data validation and access.

Scalability

Disruptive centralized projects like Facebook, Uber and Youtube all handle millions of active connections simultaneously. In order to achieve similar worldwide scalability, decentralized applications built on the Rublix Blockchain will utilize a unique consensus model (structured as a hybrid between Proof-of-Authority and Proof-of-Stake) that can support massive user load.

Hybrid Consensus Model

Proof-of-Authority / Proof-of-Stake Model

The Proof-of-Authority model provides one of the highest levels of security, as an attacker with unwanted connection or hacked authority cannot overwhelm the entire network by potentially reverting or disrupting all transactions. By initially allowing a trusted set of individuals to validate blocks, we dramatically lower the risk of malicious nodes trying to alter the chain with false information.

To start the chain, we will assign 16 internal validators to secure the network. This number of initial validators will be sufficient to operate the chain diligently.

Benefits of the Rublix Consensus Model:

1. Lower transaction fees

2. Micro transaction friendly

3. More energy efficient

4. More secure than traditional Proof-of-Work protocols

5. Substantially higher scalability

Network Specs:

Initial Validators: 16

Blocktime: 10 Seconds

Max Supply: 100 million

Algorithm: Aura

As tested on the Kovan network[1] and Parity recommendation:

Simple and fast consensus algorithm, each validator gets an assigned time slot in which they can release a block. The time slots are determined by the system clock of each validator.[2]

Issues with a faster block time in traditional protocols:

Traditional Proof-of-Work protocols operate poorly when blocktimes are set too low. These issues include:

Orphan Blocks and wasted disk space

Increased bandwidth

More/longer forks, and even longer re-org time

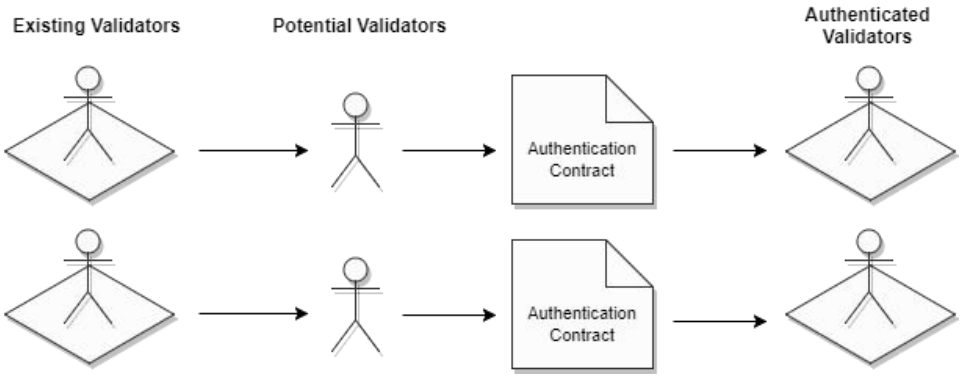
A greater portion of wasteful hashing power

Benefits of faster block times in the Rublix Blockchain:

Ability to handle a higher volume of transactions due to expeditious block generation.

Reduction of risk as it relates to double spending attacks.

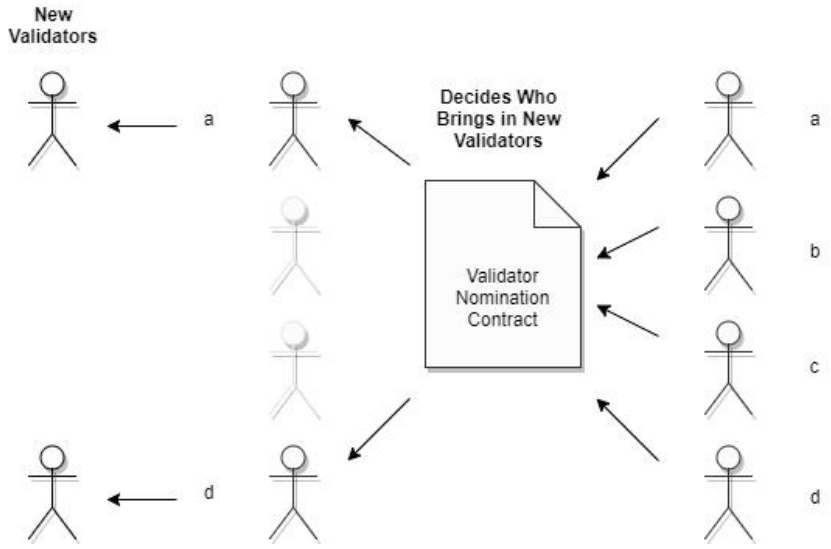
We anticipate new validators to be by invitation only by existing block creators.



As the network continues to grow and more resources are required to scale, validators will be added by invitation only. A smart contract will be used to facilitate and confirm the admission of newly appointed validators. The contract will be part of the blockchain state. Existing validators will be held responsible for whom they bring onto the network. By being a block producer and having the ability to earn transaction fees, there is an expectation of assisting with the chain advancement when required.

Scaling Only When Needed

When the chain requires scaling, we use the 'Validator Nomination' smart contract to decide which existing validators will be responsible for bringing on new block creators. A smart contract will decide which validators will participate in the selection process.



Economic Sustainability

Validators will start to create blocks and generate a fixed reward for securing the network. For every block generated, the validator who created it will receive all transaction fees. Every validator has the same opportunity to create a block.

Block Rewards

There are no plans on utilizing a block reward mechanism like Ethereum and Bitcoin. We believe the scarcity of the RBLX native cryptocurrency will contribute to reduced selling pressure.

Upgrading & Maintaining the Protocol

Due to the nature of blockchains, hard forks, and bugs, the initial release of the Rublix Blockchain will retain a controlled group of validators for a period of 6 months. This period will allow the software to perform as normal but allow the development team to retain control of upgrading software quickly. After a sufficient amount of time with no software issues, the validators will then be able to extend invites to the public. This method will allow Rublix to test the network and its validators sufficiently and prevent hard forks from occurring at early stages.

Validators acknowledge a change is being requested to the software.

Validators propose a change to the constitution and obtain 66% approval.

Validators maintain 66% approval for 14 consecutive days (2 weeks).

By default configuration of the Rublix Blockchain software, the process of updating the blockchain to add new features takes 2 to 3 months, while updates to

fix non-critical bugs that do not require changes to the constitution can take 1 to 2 months.

All validators that do not upgrade to the new code shut down automatically.

In case of emergency situations, validators may accelerate the process if a bug fix or software update is required to fix an extremely critical, time-sensitive issue.

Other Distinctions

Economical Network

The hybrid consensus model provides an inexpensive way to secure the network. Users can run existing DApps on the Rublix Blockchain and spend less money on transaction fees. Overall cost of the network's security will also be lower due to considerably lower market cap.

No Miners

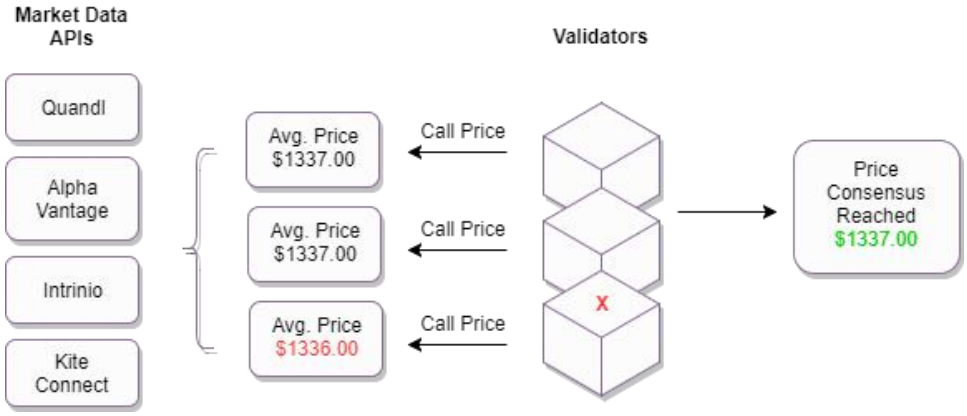
Due to consensus lying in the validators (nodes) with no mining required, scalability concerns subside as transactions are not reliant on resource intensive confirmations.

Governance Model

The Rublix Blockchain implements a governance process that combines a hybrid Proof-of-Authority and Proof-of-Stake protocol. Initial validators on the Rublix platform must be chosen by the power of a select few in order to secure the network. In order for a user to become a validator, they must be invited by an authoritative figure and stake the same number of tokens as the existing validator who elected them. The agents' stake can be gifted or purchased on an exchange. This procedure ensures the newcomer is committed to becoming a trusted validator on the Rublix network. The Rublix Blockchain acknowledges that power lies within the token holders and initial validators. The validators are monitored and given limited authority to freeze accounts, perform updates, push bug fixes and propose changes that require a hard fork to the underlying protocol. Before any changes can be made to the blockchain, validators must approve the proposed modifications. If validators refuse to undertake the suggested changes made by the token holders, the proposals can be rejected. If validators make changes without permission of the token holders then all other non-producing full-node validators will decline the change.

Market Data Validation Model

Market Data Validation Diagram



Validator Weighting

We anticipate a reputation algorithm to reward truthful validators that provide consistently accurate information. Validators with a higher reputation will be prioritized to provide data over their weaker brothers.

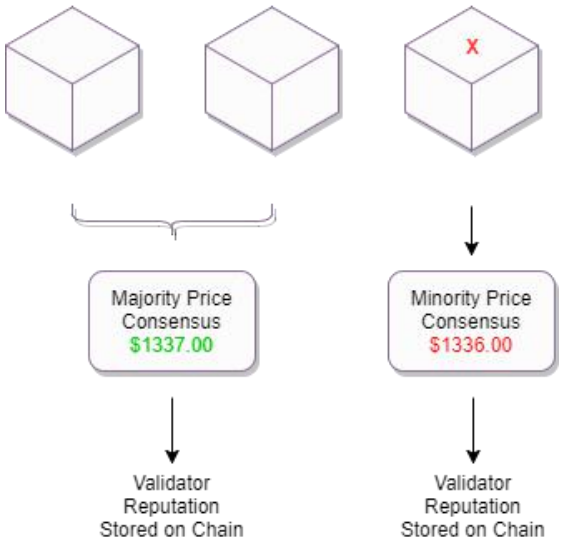
Rublix will attempt to provide reputable data on-chain by:

Providing agent incentives across the network

Incorporating stake disincentives to put a cost on malicious behavior

Developing a strong, statistics-powered reputation engine which incorporates a degree of machine learning in future iterations

Developing a scalable, decentralized reputation scoring system which allows the network to store historically informed reputation scores across all validators, or so that validators are enabled to inquire about reputation information via others which have transacted with the agent in question



Remaining Trustless

In case of a faulty validator on the network delivering data, we want to be more accurate than taking a simplified average.

We introduce a validator/node layer with real-time market data integration using multiple data sources to reach price action consensus. The values are based off validator reputation and averages from multiple real-time Websockets.

Prototype Smart Contracts

We have created Ethereum smart contracts using Oraclize which demonstrate the functionality of bringing external market data from a single API onto the chain for smart contract resolution. Unfortunately the real-time aspect is not possible due to excessive fees from high frequency API queries. Using Oracles on the Ethereum chain we run into four major problems:

1. Inability to obtain real-time data

2. Centralized API source removes trust factor

3. Expensive GAS fees on the Ethereum network

4. Consensus cannot be automated

Resolving a Contract on the Ethereum Blockchain

Determination

By integrating the data supply system into the validation level at genesis we are able to overcome these limitations on the Rublix Blockchain.

Example Code

Example of a basic Blueprint contract:

This contract compares the end of day data of BTCUSD using Quandl API.

```
pragma solidity 0.4.18;

import "github.com/oraclize/ethereum-api/oraclizeAPI.sol";

import "github.com/Arachnid/solidity-stringutils/strings.sol";

import "./SafeMath.sol";

contract EscrowVault is Ownable

{

using SafeMath for uint256;

address public wallet;

mapping(address => uint256) balances;

RBLXToken token;

mapping (address => bool) public isRefunded;

event Logrefund(address _useraddress, uint256 _time, uint256 amount);

event Logvalutclose(address _wallet, uint256 amount, uint256 time);

/**

* @param _wallet Wallet Address

* @param _token Token Contract Address

*/

function EscrowVault(address _wallet, RBLXToken _token) public
```

```
{
require(_wallet != address(0));
require(_token != address(0));
wallet = _wallet;
token = _token;
}
// Investors Cn Claim Refunds
function Refund(address[] _recipient,uint256 _numberOfBetters) onlyOwner
public
{
require(token.balanceOf(this) >0);
uint256 balance = token.balanceOf(this);
if(_recipient.length==0)
{
require(token.transfer(wallet, balance));
}
else
{
balance = balance.div(_numberOfBetters);
for(uint256 i = 0; i< _recipient.length; i++)
{
if(!isRefunded[_recipient[i]])
{
isRefunded[_recipient[i]] = true;
```

```
require(token.transfer(_recipient[i], balance));
Logrefund(_recipient[i],now,balance);
}
}
}
}
function close() onlyOwner public
{
require(token.balanceOf(this) >0);
uint256 balance = token.balanceOf(this);
require(token.transfer(wallet, balance));
Logvalutclose(wallet,balance,now);
}
}
contract Blueprint is Ownable, usingOraclize
{
using SafeMath for uint256;
// start and end timestamps where investments are allowed (both inclusive)
uint256 public startTime;
uint256 public endTime;
//ERC20 Token declarations
RBLXToken public token;
// decimalFactor
uint256 private constant decimalFactor = 10**uint256(18);
```

```
// vault used to hold tokens while Blueprint is running

EscrowVault public vault;

//owner amount of token to bet

uint256 public price;

//stores users list

address[] recipient;

//users count

uint256 numberOfBetters = 0;

struct better{

RBLXToken token;

address sender;

uint256 amount;

}

//maintain users details

mapping(uint => better) betters;

mapping(address => bool) public userExist;

mapping(bytes32=>bool) validIds;

using strings for *;

uint256 public highValue;

uint256 public predictedValue;

uint256 public lowValue;

event LogPriceUpdated(string price, uint256 time);

event LogNewOraclizeQuery(string description);

event LogUserinfo(address userAddress, uint256 aomunt, uint256 time);
```

```
        /**

        * @param _tokenaddress is the address of the token

        * @param _wallet for when contract owner Reached goal sending tokens to
wallet

        * @param _price for how many token to owner to bet

        * @param _predictedValue is predicted value

        * @param _endTime is in how many seconds will the Blueprint expire from now
(EPOCH)

        */

        function Blueprint(address _tokenaddress, address _wallet, uint256 _price,
uint256 _predictedValue, uint256 _endTime) public

        payable

        {

        require(_tokenaddress != address(0));

        require(_wallet != address(0));

        require(_price! = 0);

        require(_predictedValue! = 0);

        require(_endTime >= now);

        oraclize_setCustomGasPrice(5000000000 wei);

        oraclize_setProof(proofType_TLSNotary | proofStorage_IPFS);

        token = RBLXToken(_tokenaddress);

        assert(token.balanceOf(msg.sender)>=_price);

        vault = new EscrowVault(_wallet,token);

        price = _price;

        predictedValue = _predictedValue;
```

```
startTime = now;

endTime = _endTime;

token.sendToken(vault,_price*decimalFactor,msg.sender);

getPrices();

}

/**

* @param amount Number of Tokens for Buy-in

*/

function sendCoin(uint amount) public returns(bool)

{

require(price == amount);

assert(owner != msg.sender);

assert(endTime >= now);

assert(userExist[msg.sender] == false);

userExist[msg.sender]=true;

better bet = betters[numberOfBetters]; // Creates a reference bet

bet.token = RBLXToken(token);

recipient.push(msg.sender);

bet.sender = msg.sender;

bet.amount = amount.mul(decimalFactor);

bet.token.sendToken(vault, bet.amount, bet.sender);

LogUserinfo(msg.sender,bet.amount,now);

numberOfBetters++;

return true;
```

```
}

// @return true

function hasEnded() public view returns (bool) {

return now > endTime;

}

// @return true

function ownergoalReached() public view returns (bool) {

return highValue >= predictedValue && lowValue <= predictedValue;

}

function __callback(bytes32 myid, string result, bytes proof)

{

if(msg.sender != oraclize_cbAddress()) throw;

LogPriceUpdated(result,now);

setPrices(result);

if(ownergoalReached()){

vault.close();

} else if (hasEnded()){

vault.Refund(recipient,numberOfBetters);

} else {

if(endTime.sub(now)>86400){

getPrices();

} else {

getupdatedPrices(endTime.sub(now));

}
```

```
        }

        }

        function setPrices(string _result)

        {

        var s = _result.toSlice();

        var delim = ",".toSlice();

        var parts = new string[](s.count(delim) + 1);

        for(uint i = 0; i < parts.length; i++) {

        parts[i] = s.split(delim).toString();

        }

        // highvalue conver to uint256

        highValue = parseInt(parts[1],2);

        //lowvalue conver to uint256

        lowValue = parseInt(parts[2],2);

        highValue = highValue.div(100);

        lowValue = lowValue.div(100);

        }

        function getPrices() payable

        {

        if(oraclize_getPrice("URL") > this.balance) {

        LogNewOraclizeQuery("Oraclize query was NOT sent, please add some ETH to
cover for the query fee");

        } else {

        bytes32 queryId =oraclize_query(86400,"URL",
"BACwdQwK0DRrTxTZj83+5gsF/R4RQIlHwRd84MkH3Dh3uxPnNsBBWegEnCq8nw/+7
```

05Fr91prJ65tsgmQttVCmniw16KxIKYp7U0xS4ZCRuOAO+dce//7n6jJblzK10o3WwAhNz8/incFhFxwgVkC37GQfJ2aJlm/jtgNj9oHiMPtoxC43S7RTJXE57GnvqFU7pLbdCRjJRt0QPrLHYn6Ak57Scbh/ACIw8bB5QpqntRPSys5jkkK+RM0mLRFM8=",500000);

validIds[queryId] = true;

}

}

function getupdatedPrices(uint256 _delay) payable

{

if(oraclize_getPrice("URL") > this.balance) {

LogNewOraclizeQuery("Oraclize query was NOT sent, please add some ETH to cover for the query fee");

} else {

bytes32 queryId = oraclize_query(_delay,"URL", "BACwdQwK0DRrTxTZj83+5gsF/R4RQIlHwRd84MkH3Dh3uxPnNsBBWegEnCq8nw/+705Fr91prJ65tsgmQttVCmniw16KxIKYp7U0xS4ZCRuOAO+dce//7n6jJblzK10o3WwAhNz8/incFhFxwgVkC37GQfJ2aJlm/jtgNj9oHiMPtoxC43S7RTJXE57GnvqFU7pLbdCRjJRt0QPrLHYn6Ak57Scbh/ACIw8bB5QpqntRPSys5jkkK+RM0mLRFM8=",500000);

validIds[queryId] = true;

}

}

}

Decentralized Applications

Once the Rublix Blockchain is established, decentralized applications can be built on the platform that require trusted, on-chain financial data. We are building, testing and launching our initial finance related DApps on other chains before migrating onto the Rublix Blockchain.

Hedge Platform

Our flagship DApp, Hedge, is in development and is currently being built on the Ethereum network but will eventually be migrated to the Rublix Blockchain. Hedge is

a networking hub for financial and cryptocurrency trading experts and newcomers seeking trading predictions for cryptocurrencies, stocks, options, commodities or any other tradable product.

The Hedge platform incorporates blockchain technology directly into the functionality whereby traders submit predictions into a smart contract-driven "Blueprint" that will execute true or false results based on real market information. Hedge rewards traders with RBLX tokens for successful predictions as paid for by the Blueprint purchasers. Traders with successful predictions will also be rewarded positive ranking points on the Hedge platform based on the smart contract authority and Hedge's Proof-of-Ranking algorithm. Blueprints will thus carry an intrinsic value based on the trader's track record and ranking.

Users who identify a trader with a high ranking and reputation can purchase their Blueprints using RBLX tokens to gain access to that trader's predictions. If the trader makes a correct prediction via the Blueprint, they will receive the tokens from the purchasers. Otherwise, the tokens are returned to the purchaser as coded in the smart contract. The higher the trader's ranking on Hedge, the more expensive the Blueprint. This ranking, reward and verification system greatly enhances a trader's credibility, motivation to succeed and earning potential, in addition to filtering out poor performers with unproven track records.

Hedge v1: Smart Contract Integration (Blueprints)

The first and official public release of Hedge (v1) will be in the form of an Ethereum based dApp. This will provide the groundwork necessary to release a fully functioning product utilizing our planned smart contract architecture on the Rublix Blockchain, which will handle all of the anticipated features.

By incorporating smart contract capabilities, Hedge v1 allows for the facilitation of outcomes for analyst Blueprints. This system will be able to compare market predictions coded into each smart contract using Oracles, and then execute positive or negative rankings for the trader based on the end result of what took place in the market at a future point in time. Traders who correctly anticipate market movements will automatically be rewarded with positive ranking points on the Hedge platform based on the smart contract authority.

In our context, an Oracle is an agent that finds, validates and submits market information to the blockchain to be used by our smart contracts. When a particular value (i.e., date/time, price, etc.) is reached an event automatically triggers. The primary task of an Oracle is to provide these values to the smart contract in a secure and accurate manner.

This heightened level of validation will greatly enhance traders' credibility and potential to attract subscribers to their private channel, while at the same time filtering out poor performers with unproven track records.

Hedge v2: Proprietary Blockchain

Hedge v2, which is planned for a late-2018 release, will run off Rublix's core blockchain. By upgrading to this unique, industry specific technology and framework, the Rublix platform will incorporate the following upgraded features:

Full empowerment, customization and control for future updates and integrations to the Hedge platform, as well as the ability to build out more complex features.

Real-time market data will allow Blueprints to be solved on-chain automatically without the need for an Oracle.

A reduction in dependency on the Ethereum network and its inherent limitations.

Ability to work with community members who want to build on top of our finance platform using our Developer Toolkit.

Blueprint Smart Contract Details

Each smart contract is written based on parameters set by the trader using the Hedge web application. Any audience member can then 'buy' this analysis, or 'Blueprint.' Depending on the result of the Blueprint (a correct or incorrect trade prediction), the contract will then execute an outcome when specific parameters are carried out. The executed contract will affect the trader's reputation, ranking and RBLX earnings; RBLX is only awarded from the audience to the trader upon making a correct Blueprint, otherwise the RBLX tokens are returned to the audience. This means that audience members will only 'pay' for trader recommendations with RBLX if their Blueprint is 'true,' otherwise the smart contract will execute a 'false' outcome and return the RBLX to the audience member.

The purpose of the smart contract integration into the Blueprints is to create a higher level of validation, verification and transparency of analyst performance, which in turn affects their ranking and reward. Plenty of traders and analysts who make market predictions on social media, during interviews or to clients directly have uncertain ramifications for being incorrect and may even delete or refute previous predictions. The Hedge platform intends to filter out poor performers and allows traders' true wisdom to speak for itself by utilizing verified smart contracts posted on the blockchain.

Step by Step creation of a Blueprint contract:

John accesses the Hedge application, undertakes technical analysis on the BTCUSD chart and decides to post the dynamics of a trade he is going to make. He clicks “Create Blueprint” then enters the following data:

Expiration of the Contract - When does the trade end?

Category - What kind of product is it (cryptocurrency, token, equity, ETF, option, etc.)?

Entry and Exit points - At what price does this trade start and finish?

Address - Where will the RBLX tokens be distributed if this trade is successful?

The remaining data will be automatically populated based on the fundamentals of the Blueprint:

Potential gain from the trade as a percentage.

Blueprint purchase price based on John's ranking on the Hedge platform.

Ranking impact for a correct or incorrect Blueprint.

Proof-of-Ranking Algorithm

Overview

The Proof of Ranking algorithm is a proprietary multi variable set of formulas which assigns each trader on the platform an overall rank.

This scheme allows data to be indexed relative to importance and value. Our implementation of multiple variables within the algorithms will consist of numerous factors, a sampling of which are listed below as examples:

1. Birth of account

2. Number of successful Blueprints

3. Viewership

4. Unique contributors

5. Amount of RBLX earned

6. Account age of contributors

7. Streak

All these factors, among a proprietary list of others, are taken into consideration to output an assessment of users/data and distinguish a level of credibility for the platform and each user.

Implementation

The premise of the formula is to filter each individual and provide an accurate risk analysis on the user as they grow their track record.

Let R(u,t) be the Proof-Of-Rank function describing a user u at a time t. R(u,t) is a real-valued function where u=(u_1,u_2,...,u_N) is an N-dimensional vector of real values.

R(u,t) = f(u,t, w(t))

Where w(t) is a real-valued scalar weighting function, independent of the user u, and only depending on the time.

Ranking and Valuating Blueprint Contracts

A general Proof-of-Performance model is used to determine the amplitude of change in a blueprint's dynamic pricing value. The below factors will be used to determine the impact on valuation:

m = multiplier

r = current ranking

k = target ranking

a = contract demand

g = potential gain (in %)

t = time until Blueprint expires

Note: The formulas we have developed are continually being optimized and will be updated within this document when the final variables and structures are decided upon.

Spam and Manipulation

A key advantage of the Hedge platform compared to other trading platforms is the degree of difficulty it will present to spammers and manipulators, making the

execution of such fraudulent activities perverse and impractical. Almost all traditional social media investing and sharing websites are plagued with fake upvotes, comments and followers.

Many of these heavily manipulated sites are frequently used to carry out pump-and-dump schemes and therefore can pose a massive risk to new investors in the space. Rublix's core architecture is dependent on a complex algorithm that eliminates gaming and abuse.

An integral element of spam prevention is the ability to detect anomalies, such as:

Account Creation Spam: Multiple accounts created by one person or a robot.

Shilling: Accounts used to artificially manipulate the perception of a person or product.

Usage of Accounts: Are the accounts only used to contribute to one person?

Amongst the tools used to help us identify account legitimacy is Artificial Intelligence (AI):

We build a base model from the data that asks the following question:

"What is the probability that the selected data is not an anomaly?"

Employing this model that we built utilizing our data structure, we can differentiate if other examples are anomalous or not.

Having built this model, we can assume:

if	$p(x_{test}) < \epsilon$	Possible Anomaly
if	$p(x_{test}) \geq \epsilon$	OK

ε = is a probability value which we define depending on our needs.

We model each of the features by assuming each one is distributed according to a normal distribution:

Assuming that there are n features assumed for each training data, p(x)p(x) is:

We now have a group of data which fits valid interactions and can continue to filter the data through more validity tests.

Other Projects in Development

Rublix, as well as its partners, are also developing other innovative applications which are currently under internal R&D. Unlike Hedge, these DApps may not be publicly disclosed and/or launched on other blockchains before launching on the Rublix Blockchain.

Milestone Releases

Due to the nature and complexity of blockchain technology, the Rublix project will be spread into multiple stages.

Rise

The Hedge v1 platform operates at a preliminary stage on the Ethereum network using a centralized Oracle to bring market data on-chain for basic validation. Consensus is reached by manually executing the smart contract by a 3rd party in order to collect data from the Oracle. The user is rewarded transaction fees as bounty.

Genesis

We will migrate from Rise to Genesis once our testnet is live and functioning appropriately, market data is correctly implemented on-chain and has passed rigorous testing from our senior developers and the Rublix audit committee.

The first release of the Rublix Blockchain will be established with 16 internal validators. The chain debut will feature several simple DApps and extensive public testing will occur.

Token Swap

A token swap or bridge will occur after the Rublix network is deployed, tested and deemed to be in a stable state by the Rublix audit committee and 3rd party consultants.

Epilogue

The Hedge platform will be completely migrated onto the Rublix Blockchain. Subsequent DApps may be built and deployed on the newly established chain.

RBLX Token

Purpose

The primary purpose for the RBLX token is to be used with the Hedge v1 platform while on the Ethereum network. It will allow for an easy transition to the token swap or network bridge which will take place at a later date.

The Rublix tokens issued during the token distribution event will be identified as 'RBLX'. Rublix will not create any additional supply of tokens for the rewards pool. The fixed token supply created at genesis will reduce deflationary concerns and ultimately the market will decide the fair value of the token.

Reward Mechanism

The RBLX token is used as a reward token within its eco-system and is required for the Rublix Blockchain to properly operate.

These tokens will be the proprietary utility token for the network, Hedge contribution and reward system on the platform. RBLX will be required for Blueprint buy-ins and will pay out in the same currency. RBLX will also be used for other Rublix applications as they are developed, in various ways relevant to each tool, such as platform fees and use credits.

Disclaimer

This Rublix whitepaper is for information purposes only and is subject to change. Rublix does not guarantee the accuracy of or the conclusions reached in this white paper, and this whitepaper is provided as is. Rublix does not make and expressly disclaims all representations and warranties, express, implied, statutory or otherwise, whatsoever, including, but not limited to:

Warranties of merchantability, fitness for a particular purpose, suitability, usage, title or non-infringement.

The contents of this whitepaper are free from error or misinformation.

That such contents will not infringe third-party rights.

Rublix and its affiliates shall carry no liability for damages of any kind arising out of the use, reference to, or reliance on this white paper or any of the content contained herein, even if advised of the possibility of such damages. In no event will Rublix or its affiliates be liable to any person or entity for any damages, losses, liabilities, costs or expenses of any kind, whether direct or indirect, consequential, compensatory, incidental, actual, exemplary, punitive or special for the use of, reference to, or reliance on this whitepaper or any of the content contained herein, including, without limitation, any loss of business, revenues, profits, data, use, goodwill or other intangible losses.

References

Kovan - A Stable Ethereum Public Testnet - Parity - https://github.com/kovan-testnet/proposal

Aura (Authority Round) is a pluggable Blockchain consensus algorithm - Parity - https://paritytech.github.io/wiki/Aura