PROOF OF TOSS

a smart betting ecosystem

TECHNICAL PAPER

VERSION
0.2.6

Table of Contents

1. Glossary

Protocol: a set of smart contracts and its parameters which determine the work rules and logic of all participants.

White Label is our **Solution for Bookmakers**: a ready to use, open source platform for sportsbook suppliers, allowing them to create a unique application for any specific area of betting.

Peer-to-Peer Solution (P2P, a decentralised platform where users interact without a middle-man or third party): an online platform based on White Label, deployed on the PROOF OF TOSS website.

Event: a real world event happening in the future, one with more than one outcome where bets can be placed.

A **wager**'s created on the PROOF OF TOSS platform by either an originator or an operator.

A **bet** is placed by the player on the outcome of an event the wager was created for.

TOSS: a token serving as an internal unit of value for the PROOF OF TOSS ecosystem. It's used to bet, judge, create deposits and collateral, or receive prizes and rewards.

SBTC: the cryptocurrency used inside the RSK blockchain to pay for transactions, using Gas.

User: defined by a wallet in the PROOF OF TOSS ecosystem.

Originator: a user of the platform with enough expertise and knowledge in a particular sphere to create interesting wagers for others to bet on.

Player: a user of the PROOF OF TOSS ecosystem who tries to predict the outcome of an event by placing a bet on a wager.

Operator: a professional bookmaker who's able to utilize the PROOF OF TOSS ecosystem, just as in a traditional betting platform, by creating wagers, setting their own odds on various outcomes and delivering results for their wagers.

Judge: a user of the platform who's responsible for confirming the outcome of an event a wager was created for. Also, they resolve any challenges to events created by operators.

Challenge: a process that can be initiated by players if they disagree with the outcome of an event published by an operator.

Distributed Crowd Judging (DCJ): we're using the wisdom of the crowd in all our judging and challenge mechanisms. Employing the "prisoner's dilemma" with a clear risk and reward for any action taken by the crowd, and relying on our random number generator to select judges, we ensure every decision will be right and true. If such a decision is unreachable the wager's cancelled and players immediately receive a full refund.

Challenge Fund: a transit storage of TOSS tokens deposited to start a challenge.

Judge Fund: a transit storage of TOSS that holds deposits allocated for judging. By the end of the judging period, all tokens in the judge fund are distributed amongst the users, according to the results of the judgement.

Collateral: the amount of TOSS tokens an operator must allocate within the ecosystem to ensure all prizes, rewards and fines that may arise in a wager are covered.

Deposit: the amount of TOSS tokens an originator or judge chooses to place, in order to back their intentions of creating a wager, or to participate in the judgement of a wager.

Prize: tokens that players receive, if they win a bet.

Reward: tokens that originators, operators and judges receive as compensation for their active participation in the evolution of the ecosystem.

Fine: a penalty for unprofessional or unsound actions within the ecosystem.

Active Event: an event open for placing bets.

Completed Wager: a wager with a finished event and declared result.

Serialization: conversion of data, regarding an event, into byte sequence for transfer to a smart contract.

2. Project Summary

Gambling and betting markets are rapidly evolving, bringing the user experience to a whole new level. According to multiple sources, today's total estimated gambling market capitalisation is $400 billion, and has great potential for growth as it embraces new technologies and innovation.

For the past couple decades everything's been moving toward being "smart", from smart homes to our daily lives, with smart phones, smart watches, and smart cars. We've entered the wonderful age of optimization, where technology's aimed at tuning and adjusting existing systems to increase productivity, elevate user experience, and make them smarter and faster. Perhaps most importantly, we're increasingly able to eliminate unnecessary components. This trend's supported and embraced by manufacturers around the globe.

Blockchain's a fine example of tech currently changing the world. This technology's aimed at making transactions faster, safer and transparent. That's why we've chosen it to build the future PROOF OF TOSS ecosystem. We're inspired by the idea of making betting effortless, enjoyable and profitable for all participants.

PROOF OF TOSS is an ecosystem for placing bets, serving both as a peer-to-peer platform and an open source solution for professional bookmakers, sportsbook providers, and operators, powered by the blockchain. PROOF OF TOSS is a win-win solution for both.

The aim of PROOF OF TOSS is to improve the experience for all gambling market participants. Utilizing blockchain technology and smart contracts, the ecosystem will solve the following problems of bettors and bookmakers.

Problems for Bettors:

- Long verification processes
- Low levels of control of personal funds
- The necessity to trust providers and operators
- The necessity to wait for an operator's approval for prize payouts
- There's no proof of an operator's collateral for wagers
- Small event pools are fully controlled by a bookmaker
- Low odds

Problems for Bookmakers:

- The risk of chargebacks
- Dependency on centralized systems
- High transaction costs
- Limited market coverage
- Overpriced fees of online payment systems
- Transaction restrictions and delays
- The necessity to trust information suppliers
- Fraudulent activity
- The risk of human error

The business vision of PROOF OF TOSS is an ecosystem for betting, developed for both the end-user (as a peer-to-peer platform) and companies supplying events at any scale. Everything's embedded in the logics of the smart contracts, powered by RSK's blockchain and fueled by TOSS tokens. TOSS is the utility token of the ecosystem, used to create wagers, place bets and vote on results.

PROOF OF TOSS will be a smart betting ecosystem for the new era, based on blockchain tech and smart contracts rather than trust in a centralized party. All users of PROOF OF TOSS are protected by the ironclad logics of smart contracts, blockchain cryptography and the internal judgement mechanism used to reach high levels of accuracy in defining and challenging the outcome of an event.

3. General Project Architecture

The project consists of two main components:

- Protocol: a set of solidity smart contracts working on the RSK blockchain (https://www.rsk.co/)
- White Label: a user interface consisting of a website developed on JavaScript, React.js, HTML, CSS, and using cache in the form of Elasticsearch (https://www.elastic.co/).

4. Why are Blockchain and Smart Contracts Helpful?

In a traditional betting sector, responsibility for transaction management and all guarantees of trust and security lie on a bookmaker who provides an infrastructure for placing bets and making payments. Such systems are fully centralized. The end user isn't able to verify the system works transparently, nor can they rely on any kind of secure personal data protection.

Building the PROOF OF TOSS ecosystem, we value principles of transparency, fairness, equality and safety. These principles are fundamental in choosing a technical base and building an architecture for the future ecosystem. Blockchain and smart contracts enable us to capitalize on these principles while fully satisfying the main industry requirements. Moreover, the blockchain allows us to increase the transaction speed, provide independence and a high level of reliability.

The blockchain and smart contracts are ideal for the bookmaking industry because they allow a system to be decentralized, independent, transparent and automatic. The most important features blockchain provides for the betting arena are as follows:

- The blockchain is peer-to-peer and decentralized by design. Thanks to that there's no central control over the network, it's permissionless and uncensored, no one can "decide" which transactions are allowed.
- Users of blockchain-based apps can be sure network fees for transactions and conditions programmed in the smart contract will be included in the network, without any hidden logic on the app owner's side.
- The blockchain network is transparent, accessible and irreversible, therefore everyone can observe and verify any transaction at any moment and verify it's held in the exact condition it was sent. Thus users can easily verify there are no manipulations.
- The blockchain uses elliptic-curve cryptography. It means the owners of a private key can always prove the transaction has been sent by them.
- The blockchain has several important technical features. In traditional server-client architectures developers must implement these independently, which usually isn't an easy task. Here are such features:
  - –replication: the data is stored in every node;
  - –irreversibility: it's impossible to change transactions already recorded in the network;
  - –reliability and no single point of failure: if one node breaks down the network keeps working, unchanged.
- Smart contracts on the blockchain are a sort of software inside the network permitting us to program execution logic, keeping the same characteristics as common transactions, including irreversibility, reliability and transparency. Basically, smart contracts execute a code based on the input of certain transactions. Thus:
  - –users can be sure smart contract code can't be changed once deployed;
  - –users can always review the smart contract code to ensure it executes exactly what's been programmed and nothing else.

• In the blockchain, and smart contracts-based systems, all obligations of the parties, with all conditions, are programmed and automated inside smart contracts.

The PROOF OF TOSS mission's to create a smart and reliable ecosystem for fair betting, giving every user an equal chance to win.

5. Why the RSK Blockchain?

RSK (Rootstock, https://www.rsk.co/) is a blockchain with a virtual machine (RSK Virtual Machine) for executing smart contracts. It has opcodes (operation codes) compatible with the Ethereum Virtual Machine. RSK works as a Bitcoin sidechain. Bitcoin is the basis of the modern decentralized economy and RSK expands its functionality adding smart contracts, near instant transactions and higher scalability.

Main Advantages of RSK:

- it scales up to 300 transactions per second
- there's a possibility to increase it up to 2000 tps with the Lumino protocol
- support of smart contracts which are compatible with the Ethereum virtual machine
- it increases safety with the help of bitcoin merged-mining (done by bitcoin miners, it's fixated in RSK)
- there are technologies and protocols for:
  - –fast transactions and blocks propagation;
  - –distribution of rewards among miners of competing blocks (DECOR+);
  - –additional securing of the best chain by counting uncle blocks as normal blocks (GHOST)
- the average block generation time is 10 seconds

We'll launch our MVP in an RSK testnet.

6. Protocol Description

A protocol is a set of smart contracts with public attributes and functions one can address.
To interact with smart contracts it's necessary to setup an available client locally to work with RSK (currently only available as rskj), or use public and permanent servers. An RSK client provides a JSON-RPC server one can interact with via curl or web3.

We'll provide a working protocol in the first version in two networks: mainnet and testnet. Working with both is similar, with the exception of a port address to connect with the JSON-RPC server.

In the future, we plan to develop a JavaScript API to make it easier to interact with smart contracts, for companies who want to integrate a protocol but don't need a White Label solution.

In terms of components, the protocol can be divided into several smart contracts:

- the "Token" smart contract is an ERC20 token, implementing such functions as token blocking, sending tokens to transfer an additional data parameter and calling another smart contract code.
- the "Main" smart contract is the main system smart contract, responsible for creating events, jackpots, or events distribution for judging.
- the "Event" smart contract is the heart of the system, an event users can bet on.
- the "RandomNumberGenerator" smart contract is a random number generator used to distribute events to be judged.

7. Protocol API

Here's a table with a brief abstract description of the protocol API. More details can be found in the full protocol API manual.

Operation Type: event creation

The operation to create common and operator events.

Parameter	Type	Description
tags	string[]	Array of event tags
results	string[]	Array of event results
sourceUrl	string	Resource of event outcome (URL)
description	string	Event description
name	string	Event name
category	bytes32	Event category
bidType	bytes32	Bet type
locale	bytes2	Event locale
resultCoefficients	uint64[]	Array of event outcomes odds. Filled with zeros if an event is not being created by an operator.
tagsCount	uint8	Amount of event tags
resultsCount	uint8	Amount of possible event outcomes
endDate	uint64	Event end time (UTC)
startDate	uint64	Event start time (UTC)

Operation Type: resolving an operator's event.

This operation is for operators to publish the right event outcome, when the event's finished.

Parameter	Type	Description
result	uint8	Right outcome (outcome index, begins from 0).

Operation Type: permits a smart contract to give another smart contract the right to block tokens.

This operation is for operators.

When users bet on an operator's events they bet against the operator. That's why the operator must provide a certain number of tokens to cover participants' potential prizes, called the operator's collateral.

The operator's collateral is to be blocked until the event is over, while users are making bets. There are operations which can block an operator's tokens almost automatically. An operator has to execute the operation which gives a smart contract the right to block the operator's tokens.

- Once a certain number of tokens are blocked they can't be used for any other operations in the ERC-20 interface.
- The number of tokens blocked is the exact sum to cover the players potential winnings.
- Tokens are unblocked when an event is finished and one of the event participants executes a "withdrawal of funds".

Parameter	Type	Description
contract	address	Address of smart contract that is allowed to give another smart contracts the right to block tokens.
permission	bool	Array of event results

Operation Type: making a bet on an active event.

This operation is for any event participants.

Parameter	Type	Description
eventContract	address	Address of event smart contract
tokens	uint256	A number of TOSS tokens for making a bet
result	uint8	Outcome (outcome index, begins from 0).

Operation Type: challenging the outcome of the event.

This operation is for the bettors who took part in the event.

Parameter	Type	Description
tokens	uint256	A number of TOSS tokens needed as a deposit for challenging.
result	uint8	An outcome for challenging (outcome index, beginning form 0).

Operation Type: adding a **"seed"** number for starting random number generation.

This operation is for the judges to start a random number generation.

Parameter	Type	Description
seedHash	bytes32	Hashed **"seed"** number (sha3).

Operation Type: confirmation of a **"seed"** number for finishing a random number generation.

This operation is for the judges to finish a random number generation. The operation is available after 50 confirmations from other judges. As a result, a judge gets a random number and an automatically chosen event for judging.

Parameter	Type	Description
seed	bytes32	A **"seed"** number

Operation Type: voting for an event outcome.

This operation is for judges. It's available for common events when judging, or for an operator's events, after the successful initiation of a challenge.

Parameter	Type	Description
result	uint8	Outcome (outcome index, starting from 0)

Users can withdraw funds from an event via the "funds withdrawal" operation:

• Event.withdraw() - withdraw all funds in total (including the prize and any deposit made)

• Event.withdrawPrize(uint bet) - withdraw a bettor's prize from a certain wager

• Event.withdrawReward() - withdraw a reward (or return a deposit) by an event creator

A Typical Event Scenario:

• An event creator starts an operation, "event creation", and makes a deposit

  –If an event creator's an operator he must execute an operation beforehand to "allow the smart contract to give the right to another smart contract to block the tokens". Execution of this operation is only required once.

• After the event begins, bettors start to make bets via the operation "making bets"

• If it's an operator's event, after it's finished, the event creator resolves the event via the operation "resolving event"

  –Bettors have the right to challenge the event outcome by initiating the judgement process described below

• If a common event or challenge of the operator's event was initiated, judging begins

  –For a random number generation, and picking an event, for judging a judges work, with operations "adding a "seed" number for starting a random number generation" and "confirmation of a "seed" number for finishing a random number generation", see the algorithm below in the "Algorithms" section.

  –For voting, judges use a "voting for an event outcome" operation

• At the end, all event participants including an event creator, judges, and bettors may withdraw their funds such as deposit returns, winning sums, bet returns, etc, via the "funds withdrawal" operation.

8. Serialization, Compression and Effective Data Storage

Such operations as event creation use a specific format when transferring data. The data is being serialized and transferred in the form of a bytes sequence.

Key Features:

• We can transfer data in the most suitable form, evading the limitation of a number of function arguments.

• Data is being deserialized in a smart contract via the Solidity assembly code which makes this operation fast, and the best option in terms of resource usage.

• Only data vital for the business logic of the system is stored on the blockchain. For example, the start and end dates of an event, because this data's used for various verifications of smart contracts.

• Some data's important for system operability, but doesn't affect the business logic, for example, an event name and description. Such data, before transferring, is being compressed on the browser side via the BWTC algorithm, to consume less memory while executing the smart contract.

• Instead of being stored in smart contract storage, data is stored on the blockchain logs and passed to the Elasticsearch cluster via emitting an event (emit NewEvent) from a smart contract. This approach makes it possible to considerably reduce the gas expenditure.

• By transferring all the data, using only the serialized parameter, it becomes possible to use the transferToContract function ("Token" smart contract) for transferring tokens, while calling a code of another smart contract (similar to ERC223) within one transaction, without having to use the "approve" function.

9. Protocol Updating Process

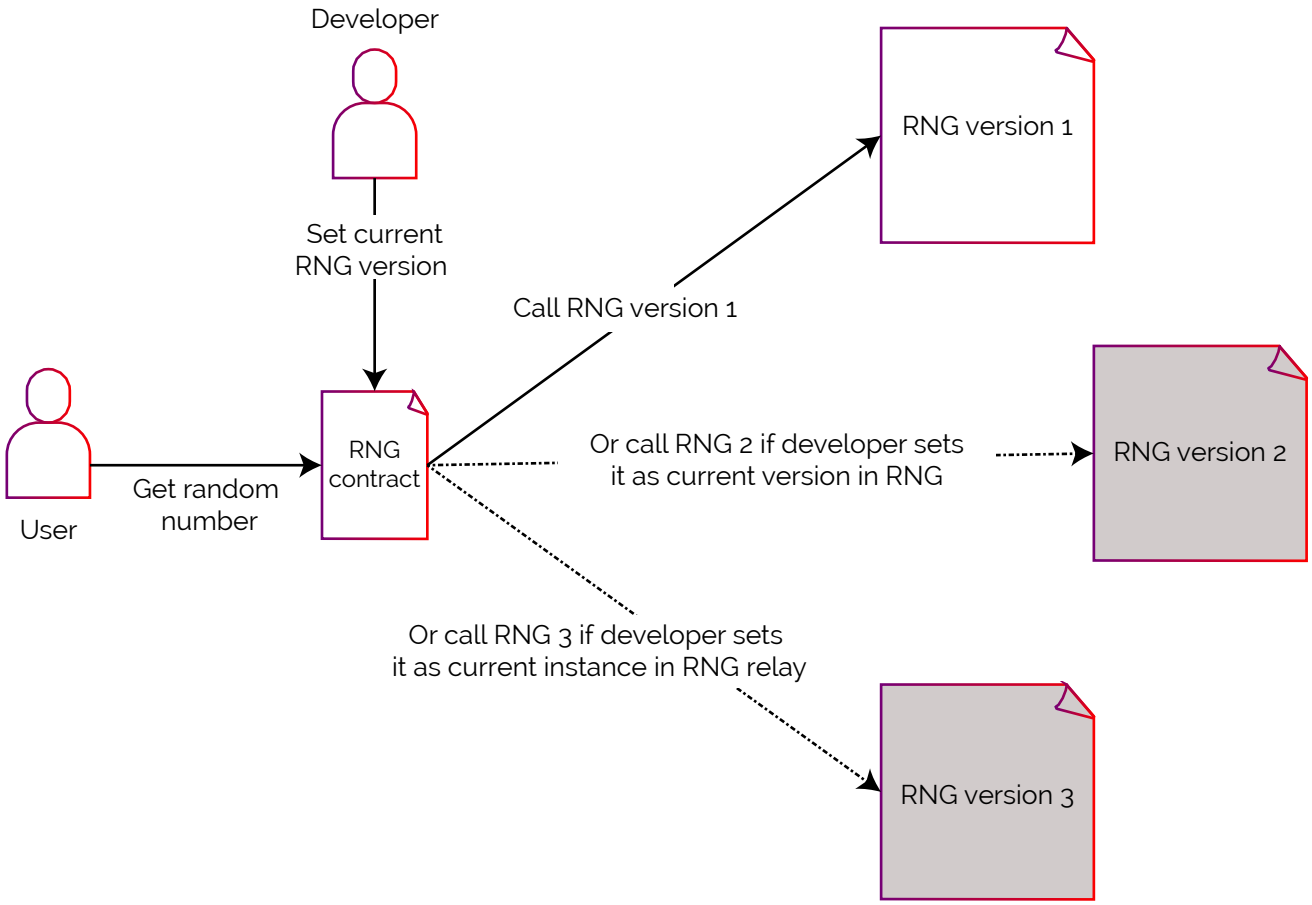
As the project evolves, we may want to change the code of some smart contracts. This option won't be available for sensitive and important smart contracts, such as token contracts or event contracts with bets. But it will be provided for service contracts and for business logic, which may evolve in time.

We're planning to integrate the ability to update smart contracts for the following cases:

- Random number generation to distribute events to be judged
- Event distribution for judging
- Creation of new events
- Updating an event prototype (used for the creation of new events)
- Jackpot drawings

In the following scheme, the process of calling a function from an updatable smart contract, and updating the smart contract, is an example of a random number generation:

- Client (a user or another smart contract) calls a function of the random number generator smart contract (RNG contract, in our case it's getRandomNumber())
- The RNG contract stores the address of the current smart contract implementation with an updatable function, so the RNG contract redirects the function call from the RNG contract to the current RNG version.
- The Smart contract owner has the right to update the address of an updatable smart contract, and to add or change a functionality.



Events are implemented as follows:

• There are two smart contracts: Event and EventBase.

  – Event is an instance of an event.

  – EventBase is a sort of prototype of an Event.

• When creating a new event, a new Event smart contract's created. The memory of an event prototype EventBase smart contract's allocated to it.

• To allocate memory for a new event, the smart contract stores the address of a prototype, EventBase.

• If any business logic changes it's possible to develop a new EventBase prototype.

• In previously created events nothing will change, code will be changed only for new events.

10. Description of a Token and its Functions

The TOSS token is a standard ERC-20 token. The token is used in all operations of the system:

• making a deposit when creating an event;

• blocking an operator's collateral;

• making a bet;

• making a deposit for judging;

• and the withdrawal of prizes and returns of funds.

The token's functionality is extended for the crowdsale (pause of exchange, freezing tokens etc.) and to provide the functionality of the system.

• Functions for blocking an operator's collateral in operator's events (grantToAllowBlocking, allowBlocking, blockTokens, unblockTokens).

• Function transferToContract is used to avoid any extra calls of the "approve" function, allowing users to execute some operations at once. This function duplicates a "transfer" function and takes an additional parameter of data in the form of bytes. This parameter is then transferring to another smart contract. For example, this way the process of placing a bet, and making a deposit when creating an event are both implemented.

11. Algorithms

11.1. The Period of Completing a Wager

A wager's ready to complete when the specified time of an event's end arrives. Formula:

now ≥ endTime, where now and endTime - unix timestamp stored in a smart contract (now is the current time and doesn't depend on smart contracts).

When this time has come:

• placing bets are prohibited;

• common events are ready for judgement;

• the operator's event is ready to publish the outcome, by an operator (after that, bettors will have the right to initiate a challenge of the outcome).

11.2. The Sum of an Operator's Collateral and What it Depends On

When a user places a bet, part of an operator's collateral must be blocked. Also 0.5% from the sum of the whole betting pool must be blocked additionally in case there's a challenge requiring a judgement process. This 0.5% will be transferred to judges as a reward. An operator's collateral is considered a bet, so it's included in the betting pool.

Formula for blocking an operator's collateral:

b*[(k-1)+(k*0.5/100)], where:

b - a bet

k - odds of an event's outcome a bet was made on.

11.3. The Sum of an Event Creator's Deposit and What it Depends On

To create a common event, making a deposit of any sum's required. The deposit's used in the calculations of an event creator's reward and in some cases an event creator can be fined the amount of the deposit, in favour of the judges/bettors.

You can read more details about when and how an event creator's deposit requires further calculations in the White Paper:

- PROOF OF TOSS / Toss - the smart way to bet
- PROOF OF TOSS / Use cases

11.4. The Sum of a Judge's Deposit and What it Depends On

All judges deposits are stored in a judgement fund. A judgement fund takes part in an event funds distribution following the event.

You can read more details about when and how a judge's deposit requires further calculations in the White Paper:

- PROOF OF TOSS / Toss - the smart way to bet
- PROOF OF TOSS / Use cases

11.5. Random Number Generation for Event Distribution for Judging

Random number generation on the blockchain is a nontrivial task. An algorithm of a random number generation with a seed number's described below. The seed's the value needed for a random number generation. In traditional client-server apps an RNG usually uses the current time of the machine executing the program as a seed. On the blockchain, a block's generation occurs at different intervals and similar blocks can be generated on different machines. That's why it's not possible to use the value based on the characteristics of a particular machine as a seed: different machines have different characteristics, hence a random number generation algorithm will not be deterministic. So we needed to create an algorithm based on a seed value originating from the blockchain.

Requirements:

- Algorithms must be deterministic and produce the same results for one seed.
- The result must not be predictable.
- Seeds cannot be reused.
- Picking an event, with the help of a random number, must proceed a fixed number of operations regardless of the number of events.

Chosen Algorithm:

The pseudorandom number RA is calculated as follows:

RA=SHA3(A+Sj+1+Sj+2+...+Sj+50), where

A– judge's address

S=...,x,...– set of seeds,

x– judge's seed

Set of seeds S must satisfy the following conditions:

$\forall T : |S|^T \leq |H|^T - 50$

$j \geq i + 50$

$S_j = x$

where

H=[...,SHA3(x+A),...]– set of hashes, H_k=SHA3(x+A)

$|S|^T$ and $|H|^T$– sizes of S and H sets at time T

T2>T1, T1– time of seed hash sending, T2– time of seed confirmation

$i=|S|^{T1}$; $j=|S|^{T2}$; $k=|H|^{T1}$– sizes of S and H sets at times T1 and T2

The algorithm steps:

1. A judge calls Token.approve() for a deposit sum.

2. A judge locally generates a unique number x– seed. It must be unique for one address. There are $2^{32} \approx 4{,}29$ billion possible seeds for one address.

3. A judge generates hash SHA3(x+A)and sends it using a smart contract. A deposit is withdrawn from the judge's account.

4. A judge waits for another 50 users to send and confirm their seeds. It's necessary to make fraud unbeneficial (for example sending seeds by one person from different addresses and defining a random number in advance). If a process is too slow the system can send seeds by itself.

5. A judge sends the seed x to a smart contract. The smart contract computes SHA3(x+$\hat{A}$), where $\hat{A}$ is the sender's address. If the hash matches with a hash previously saved in step 3, the seed goes into the set of seeds.

6. Another user must confirm the seed. After that SHA3(A+S_{j+1}+S_{j+2}+...+S_{j+50}) is calculated. This will be a random number R_A. Note: a seed sent by a judge does not influence the R_A.

7. All random numbers are added to the cumulative hash string and it will be used to determine a random selection for a jackpot.

11.6.Algorithm for Choosing an Event to Judge

A judge can judge any event if the following conditions are met:

• A judge made a deposit for judging and got a random number.

• If it's a common event, an end date must not be earlier than T=Tn-Tj, where Tn - unix timestamp (current time), Tj - time for judging, 7 days

• If it's an operator's event

  –a date of resolving an event by an operator must not be earlier than T=Tn-Tj' where Tn - unix timestamp (current time), Tj' - time for judging a challenged event, 8 days.

  –a necessary number of challenges must occur within a particular time period.

• Voting didn't lead to consensus.

• The bet sum for an event depends on a judge's deposit and must be within the range described in the following table:

Deposit size for judging	**Range of a size of an event betting pool**
0 - 50	0 - 250 000
51 - 100	251 000 - 500 000
101 - 200	501 000 - 1 000 000
201 - 400	1 000 001 - 2 000 000
401 - 800	2 000 001 - 4 000 000
801 - 1600	4 000 001 - 8 000 000
1601 - 3200	8 000 001 - 16 000 000
3201 - N	16 000 001 - N

The last range has a fixed starting length, an ending length may be unlimited. But the number of TOSS tokens can't be more than 1 billion. This number's the limit.

Judging's possible if an event ended more than a week ago. The end date of the event rounds up to 30 minutes. Thus, there are **7 * 24 * 2 = 336** time periods for events to be judged.

The bet's sum is distributed by ranges on a logarithmic scale (see the table above). Thus, there's eight price ranges where it's possible to have events for judgements. Events will be distributed in a 8336 matrix by 8 price ranges and 336 time ranges:

E=...,[...,[...,event,...],...],...– event distribution matrix, $E_{i,j}$– sets of events, $E_{i,j,k}$– event addresses, i– index of the price range that matches deposit j– index of the time range that matches event's finish time.

A set of events addresses will be stored for each time period and price range. If the corresponding event of the new bet matches the next range, it's added to the set of this range. Then, the address of the last element's copied to the index of the event in the old set. The old element is then deleted.

After a new vote, if a consensus has been reached, the address of the last element in an set's copied to an event index in an old set, then the last element is deleted.

Events with 50+ judges, but without consensus, will go to a separate store. 10 minutes after the last vote such events will be assigned to new judges.

The event is determining by pseudorandom number R∈[0,...,65536] as:

$event_R=E_{D,x,(p-\sum_{i=0}^{x}M_{D,i})}$, where

E– event distribution matrix (see above)

M=([...,[...,|Ei,j|,...],...])– matrix containing sizes of sets of events

D– index of the price range that matches judge's deposit

p– a random number scaled by the number of suitable events, $p\in[0,...,\sum M_D]$; $p=[R/65536*\sum M_D]$

x– index of the time range that contains p-th event. x: $\sum_{i=0}^{x}M_{D,i}>p \leq \sum_{i=0}^{x+1}M_{D,i}$

$event_R$ determination needs M_D matrix to be traversed two times. First time for calculation of number of suitable events $\sum M_D$, second – to find time range x. Thus the algorithm is executed in not more than 2336=672 steps.

12. White Label Description

The PROOF OF TOSS White Label is a ready to use, open source platform for sportsbook suppliers. Bookmakers will be able to integrate it into their business, rebrand it and launch it, adjusting it as needed. PROOF OF TOSS will serve bookmakers as a gateway to the world of blockchain and cryptocurrencies with all it's benefits.

In the first version, White Label will include the following functionalities:

• Authentication and authorization

• Creating common and operator's events

• Browsing an events list with filters

- Making bets
- A system for events distribution for judgement
- A judgement system
- A personal user account allowing a user to:
    - browse a list of user's events, bets and events for judging
    - receive funds from events, such as rewards, deposits etc.
    - utilize their TOSS wallet for token transfers

12.1. White Label Architecture Details

- It's a frontend application based on React and Redux.
- It utilizes ElasticSearch as a search system and a secondary system for storing data.
- The website has no server side, except from a service for listing events on the blockchain (creation and updating events, making bets), for updating ElasticSearch indexes on time.
- Interactions with the blockchain network (RSK) uses the following scheme:
    - JS code of the White Label
        - JS library Web3
            - MetaMask which addresses the blockchain nodes

13. Platform Fees

In the PROOF OF TOSS ecosystem a fee's set only for paying a reward to event creators and judges. It totals 1% of the sum of all bets made. The team and founders don't receive any monetary rewards from the system's operation and don't depend on its size and income. Only under this condition can the system be decentralized and protected from developers' manipulations. To achieve the goals we've outlined, part of the money raised during the token sale and a portion of the tokens will be used to establish a fund. The fund will be responsible for evolving the project, interacting with the community and making decisions regarding project development, by voting.

14. Blockchain Fees

Approximate calculations of gas consumption and RSK mainnet fees are presented in the table below:

- The amount of gas for an operation multiplied by 1.5 to cover an operation's performance 100%.
- The price for gas is arbitrary, in reality it changes in real time and depends on the network miners.

Operation	**Gas**	**Gas price**	**Total SBTC**
Event creation	681000	183 MWei	0,000124623
Making a bet	260000	183 MWei	0,00004758
Events distribution for judging	No data	183 MWei	No data
Voting for an outcome (judging)	No data	183 MWei	No data
Publication of an outcome (by an operator)	50000	183 MWei	0,00000915
Challenging the resolved event	No data	183 MWei	No data
Deposit return of a created event	172000	183 MWei	0,000031476
Withdrawal the reward for an event creator	172000	183 MWei	0,000031476
Withdrawal the reward for a bettor	106000	183 MWei	0,000019398
Withdrawal the judge reward	No data	183 MWei	No data

15. Project Governance

This project is a non-profit organisation. That's why 12% of the tokes reserved and used set up a fund. The aim of the fund is project development, including code development, functional extension and ecosystem maintenance. Also the fund will work to expand the community, communicate with its members and attract new members.

16. Road Map of the Protocol and the White Label

16.1. Stages of the PROOF OF TOSS Protocol Development

- MVP: a decentralized online betting system, a set of smart contracts (June 2018)
- 1st version: smart contracts for wisdom of the crowd (August 2018)
- JavaScript API: an API making it easier to interact with smart contracts, for companies who want to integrate a protocol but don't need the White Label (autumn 2018)
- Live-betting support (December 2018)

16.2. Stages of the PROOF OF TOSS White Label

- MVP: an online wallet working via MetaMask, placing bets on centrally created events (June 2018)
- MVP design: MVP layout markup and system setup (July 2018)
- 1st version: peer-to-peer betting, judgement of the crowd (August 2018)
- Online wallet without MetaMask (autumn 2018)
- Open source mobile application (1st part of 2019)

PROOF OF TOSS

a smart betting ecosystem

THANK YOU FOR YOUR ATTENTION

FOR MORE INFORMATION

Visit our website - https://toss.pro/

Join our Telegram chat - https://t.me/proofoftoss

Subscribe to our Facebook page - fb.me/ProofOfToss

Fllow us on Twitter - https://twitter.com/proof_of_toss

VERSION
0.2.6