DOI: https://doi.org/10.15276/aait.03.2021.6
UDC 004.75

Smart contract sharding with proof of execution

Igor E. Mazurok[1)]
ORCID: https://orcid.org/0000-0002-6658-5262; igor@mazurok.com. Scopus ID: 57192064365
Yevhen Y. Leonchyk[1)]
ORCID: https://orcid.org/0000-0003-1494-0741; leonchik@ukr.net. Scopus ID: 57192064365
Oleksandr S. Antonenko[1)]
ORCID: https://orcid.org/0000-0001-9680-3446; asantonenko@gmail.com. Scopus ID: 17433258300
Kyrylo S. Volkov[1)]
ORCID: https://orcid.org/0000-0002-7705-8994; cyrillicw@gmail.com
[1)] Odessa I. I. Mechnikov National University. 2, Dvoryanskaya Str. Odessa, 65082, Ukraine

ABSTRACT

Nowadays, Decentralized Networks based on Blockchain technology are actively researched. A special place in these researches is occupied by Smart Contracts that are widely used in many areas, such as Decentralized Finance (DeFi), real estate, gambling, electoral process, etc. Nevertheless, the possibility of their widespread adoption is still not a solved problem. This is caused by the fact of their limited flexibility and scalability. In other words, Smart Contracts cannot process a large number of contract calls per second, lack of direct Internet access, inability to operate with a large amount of data, etc. This article is devoted to the development of the Sharding Concept for Decentralized Applications (DApps) that are expressed in form of Smart Contracts written in WebAssembly. The aim of the research is to offer a new Concept of Smart Contract that will increase the scaling due to applying the idea of Sharding that allows avoiding doing the same work by all nodes on the Network and flexibility due to the possibility of interaction with the Internet without special Oracles. During the research, decentralized 0ata storages with the possibility of collective decision-making were developed. The scheme of forming Drives that assumes that each Contract is executed by a set of randomly selected nodes that allows avoiding cahoots and prevents Sybil Attack is offered. Such an approach allowed using Drives as a base layer for Smart Contracts. Moreover, Drives can be used as a standalone solution for decentralized data storing. The features of coordination of results of Contracts execution that greatly expands the possibilities of the Contracts compared to Ethereum Smart Contracts, and, in particular, allow the Contracts to interact with the Internet are described. The Rewards Concept that incentivizes all nodes that honestly execute the Contracts, unlike other systems where only the block producer is rewarded, is developed. It is based on the specially developed Proof of Execution – a special algorithm that allows detecting all the nodes that honestly execute the Contracts. In order to make the Proof of Execution more compact, an extension for the existing discrete logarithm zero-knowledge proofs that makes it possible to consistently prove knowledge of dynamically expanding set of values with minimal computational and memory complexity so-called Cumulative Discrete Logarithm Zero-Knowledge Proof is developed. Thus, in this article, the new concept of Smart Contracts Sharding empowered by economic leverages is researched. The main advantages of the proposed approach are the possibility of interaction with the Internet and big data processing. Moreover, the mechanism of incentivizing nodes to honestly execute the Smart Contracts is developed. In addition, the Cumulative Proof that is necessary for the cryptographic strength of the specified mechanism is offered and its correctness is proven. The obtained results can be used to implement Smart Contracts in decentralized systems, in particular, working on the basis of Blockchain technology, especially in the case of demanding high bandwidth and performance.

Keywords: Proof of execution; cumulative proof; sharding; smart contracts; zero-knowledge proof

For citation: Mazurok I. E., Leonchyk Y. Y., Antonenko O. S., Volkov K. S. "Smart contract sharding with proof of execution". ***Applied Aspects of Information Technology.*** **2021; Vol. 4 No. 3: 271–281. DOI: https://doi.org/10.15276/aait.03.2021.6**

1. INTRODUCTION, FORMULATION OF THE PROBLEM

The recent years the decentralized and distributed networks are actively studied and problems that arise in such networks are researched [1, 2], [3]. Special attention is paid to Smart Contracts on the base of blockchain.

For the first time the idea of Smart Contracts was offered by Nick Szabo in 1994 [4]. They became widespread after the emergence of the Ethereum blockchain network that, in fact, provides an environment and infrastructure for execution Smart Contracts written in Solidity. Since then, contracts have been widely used in many areas: finance (lending [5], exchanges [6]), guaranteeing property rights [7], etc.

Despite its innovation and flexibility, Ethereum Smart Contracts have a number of disadvantages, including those related to the network architecture. Among the main disadvantages can be noted low bandwidth – the Ethereum is not able to process a large number of contract calls per second, lack of direct Internet access, inability to operate with a large amount of
data, etc. They are, in particular, called by the fact that the Smart Contracts are executed by the all blockchain nodes.

The Ethereum and similar systems offer the algorithm of encouragement of Smart Contract Execution that is developed at the Blockchain Consensus Level based on Proof of Work and assumes that only the producer of the block is rewarded. Such an approach does not incentivize the nodes of the Network to validate correctness of the execution.

Thus, the following problem arises: to develop an environment for Smart Contracts Execution that:

– supports Sharding – dividing the Work among the nodes of the Network that assumes that each Contract is executed by a large enough set of nodes, whose size is significantly less than the whole number of nodes in the Network

– allows the Contracts to interact with Internet and other Contracts

– incentivizes the nodes to honestly execute the Contracts

The last point assumes that the nodes must be rewarded for their work. Therefore it is necessary to have the possibility to determine the nodes that have indeed executed the contract. Since the results of the execution must be public, then it is impossible to perform such determination via checking possession the results of the execution. That's why the algorithm that allows determining such nodes must be developed.

2. LITERATURE OVERVIEW

All the mentioned problems significantly decrease the possibility of using the existing solution in wider spectrum of cases. That's why they are actively researched.

In particular, Ethereum is actively working on Ethereum v2.0 [8] that aims to introduce Sharding that will allow to significantly increase the number of contract calls executed per second and the main idea of which is dividing the network into subnets called shards and parallelization of work among them.

In [9] the overview of Ethereum Layer 2 Scaling: Plasma, ZK-Rollups and Optimistic Rollups – is given. The main idea of the solutions is to take the most operations off-chain, in particular, via building child chains and registration of some events in the parent chain. Nevertheless all of the proposed ideas require either centralization, or significant decrease in functionality, for example, the possibility of using Contracts only for tokens transfers.

Hyperledger offers its solution for Private Networks called ChainCode [10]. In this solution similar ideas can be traced: only some nodes of the Network execute the code. Moreover, nodes are not the Harvesters – the nodes that produce the blocks.

Zero-Knowledge Proofs [11, 12] is a method used in cryptography for proving possession secret information to a Verifier without providing it any additional information about the secret except for the fact that the Prover knows it.

The Zero-Knowledge Proofs are actively used and researched in blockchain area. A special attention is paid to non-interactive protocols – those ones that do not suggest sequential messaging between the Prover and the Verifier but assume that the Verifier is able to check possession of the secret information after receiving a single message from the Prover.

The Proofs have become especially popular due to the development of blockchain technologies.

The most wide used algorithms are zk-SNARK [13]. This is a class of Zero-Knowledge Proofs that possesses a number of additional properties including succinct proof property that requires the proof size to be a constant value.

Most often these algorithms are used to prove the validity of token transfer transactions. In particular, such a technology is used by ZCash [14].

3. THE AIM AND OBJECTIVES OF THE RESEARCH

The aim of the research is to develop a new architecture of smart contracts that will make execution more flexible, scalable and increase incentives for nodes to work honestly.

Research objectives:

1. Offer a new concept of Smart Contracts Sharding with the possibility of interaction with the Internet.

2. Develop the mechanism of incentivizing nodes to honestly execute the Smart Contracts

3. Offer a new cryptographic protocol that allows proving the possession of expanding sequence of discrete logarithms. The protocol should allow proving without knowledge which part of the sequence has already been proved by the moment of new proof generation.

4. DRIVE

In order to introduce the offered concept of Smart Contracts we should preliminary introduce Drives. We use Drive concept to store both Smart Contract Code and Smart Contract Data. Each Drive has its Owner – user of Smart Contact System, which should upload Smart Contract Code and initial data for Smart Contract on stage of Deployment of Smart Contact.

The Drive is a decentralized storage served by a set of Network nodes called Executors. The simplest idea is to store all Drives information on the same set of Executors. But then we will not obtain sharding concept and we obtain system which is in many ways similar to Etherium 1.0 system. So we offer to store a Drive on a proper subset of all Executors. Only this subset Executors will run Smart Contract located on this Drive. Since Drive is not stored by all Network participants, its size can be quite large. This allows storing arbitrary information and process large files including databases.

All Executors of some Drive store the same information about Drive content which is negotiated and approved on the blockchain using the Multisignature mechanism. The essence of this approach is that changing the contents of the Drive is made via releasing the transaction that is signed with the supermajority of the Executors. This allows to work in standard assumptions for Byzantine Fault Tolerance [15] system that assume that there is less than one third of fault Replicators on the Drive.

The possibility of collective decision-making via Multisignature mechanism favorably distinguishes our solution from other decentralized storages [16, 17] which either does not provide the possibility of collective decision-making at all, or they are carried out in a centralized way.

Since the total size of information on Drive can be pretty large, it is logical to store in blockchain transactions not all the content of Drive, but some aggregate information of it, for example, hash of all Drive content.

In fact such Drives are analogs of Shards in other Networks but with the clarification that each Executor can be a member of several Drives.

The common problem for Sharding systems is the procedure of assigning nodes to Shards: if each node can select the wished Shard by itself, then the performing of Sybil Attack becomes much easier: it is sufficient to seize a single Shard in order to put under threat the functionality of the entire Network.

In order to prevent the possibility of the attack and avoid cahoots, Executors on the Drives are assigned randomly.

To summarize our thoughts, the Drives can be used on one's own, like decentralized analogue of cloud storage service, like Google Drive or Dropbox. But using them as a base layer for Smart Contracts is the most interesting.

5. SMART CONTRACTS

In this section we introduce the concept of Smart Contracts – the more advanced analogous of Ethereum Smart Contracts.

Smart Contract is a special kind of Decentralized Application in a form of software product. The Contract code is stored on a separate Drive and is public that means that everybody is able to download the code and make sure there are no vulnerabilities. The code is immutable that guarantees that all the terms of an electronic contract will not be changed over the time.

The Smart Contracts are written in WebAssembly [18] and executed by Executors.

Usage of WebAssembly allows:

– avoiding development of own programming language and virtual machine like it is made in Ethereum

– lowering the entry threshold for developers

5.1. Contract deployment

At first, the creator of a Contract need to upload source code of the Contract and all the data needed by the Contract to some drive. We call this process Contract deployment. On this stage Contract creator can modify the source code and the data on Drive as he wants, but after finishing of deployment process, he loses direct control on both source code and data, since all data on Contract Drive can be modified only through Contract code execution. This is used to protect users of Contract against abuse by the creator of the Contract (for example, in case of online auction, the initiator of the auction cannot cancel it after start, or change the auction conditions or terms).

The results of successful or unsuccessful deployment of the Contract, as well as initial state of the Drive after deployment are fixed by a corresponding transaction in the blockchain.

Thus, the Contracts code is immutable after finish of deployment process, so the developers must carefully analyze it before uploading, in particular, using the method proposed in [19].

5.2. Contract runs

The Contract run is initiated via posting a corresponding transaction into the blockchain. We will call a concrete request of the Contract run as a Call.

The Call starts to be executed as soon as the block with the corresponding transaction is finalized. It allows guaranteeing that all the Executors run the Call in the same order.

The transaction contains all necessary information for the Call Execution including the arbitrary parameters. It allows locking tokens on behalf of the Contract.

5.3. Interaction with data

The Smart Contracts can interact with all the files on their Drives. The interaction includes reading, modifying, and removing any files.

Interaction with the blockchain is limited. The limitation is caused by the fact that Contracts are executed in parallel to the blockchain so different Executors may have different states of the blockchain by the moment of execution. The similar problem is considered by Hyperldger [10].

They offer introducing two sets:

– Read Set - a list of unique blockchain keys and their committed version numbers that are read during the execution.

– Write Set - a list of unique blockchain keys modified during the execution.

The execution is then considered as a successful one if the values mentioned in the Read Set have not been modified during the execution.

Nevertheless, we refuse such an approach and prefer limiting the information accessible during the execution because of the following disadvantages of Read Set:

– modification of some information in the blockchain that was used during the execution does not always mean incorrect contract execution: for example, if the balance of an account has increased, then it should not be a block for transferring money from the account initiated with the contract;

– a malefactor can maliciously modify some values in order to fail the execution.

The limitation consists in the fact that the Contract can interact only with immutable part of the blockchain: blocks and transactions and not the state.

Due to the approach of approving the Execution Results the Contracts are able to interact with the Internet. Despite reading from Internet and Writing to Internet are almost indistinguishable from the network point of view, it should be noted that since the Contract is executed by a couple of Executors, writing to the Internet can have unexpected consequences so in fact the interaction with the Internet is limited with the possibility of downloading data.

5.4. Interaction of several Contracts

Typical Ethereum Smart Contract calls other Smart Contracts during its execution. That's why it is necessary to provide Contracts with the similar possibility of interacting with each other. The difficulty of the problem is explained by the fact that Contracts are run in parallel independently of each other so different Executors of both Contracts may receive different results.

We offer the next solution of the problem:

– Running of a Smart Contract by another Smart Contract can be performed only via posting the corresponding transaction to the blockchain

– Obtaining information from a Contract can be performed only as downloading a file with known hash.

Such an approach allows guaranteeing that all the Executors always execute the same actions and receive the same information.

5.5. Approving the Execution Results

After the Call is executed, the Executors have to agree on the results of its execution and approve them in the blockchain via posting a special transaction.

Since during its work, the Contract can modify data stored on its Drive and issue transaction to the blockchain, then the process of agreement on the execution results comes down to the process of agreement on the final contents stored on the Drive and the contents of the issued transaction. In order to decrease network load a typical solution is agreement on hashes of the corresponding values.

Since the Contracts have access to the Internet, the situation when different executors receive different results can arise. Despite the fact that it obviously indicates that the contract does not work properly, such a situation must be processed in normal mode.

Thus, the procedure of agreement should support the next scenarios:

– The supermajority of Executors agrees on the same results of Contract execution.

– There does not exist a supermajority of Executors that agrees on the same results of Contract execution.

In the first case the Contract is executed successfully and the results on which an agreement has been reached can be considered the results of the Contract execution. Such a task can be solved by any Byzantine Fault Tolerance consensus or via Multisignature mechanism.

The second case is more difficult since it is not possible to find out whether the desired supermajority exists until all the Executors express their opinion. The consensuses that are able to solve such problem usually use the concept of time [20]. It allows avoiding endless waiting for the responses from all the nodes, considering that the nodes that have not sent a response in the allotted time will never send it.

A feature of the problem being solved is that the Executors that the process of agreement is not regular like accepting blocks in the blockchain but takes place only when the Executors complete their work. Due to different computing power of Executors, network lags, etc. the moments when the executions are finished can differ a lot.

Taking into all the above we offer the next solution of the problem based on the Multisignature mechanism:

1. As soon as the Executor completes the execution, it sends signed opinion about the results of the execution to all other Executors at time T

2. If at some moment of time the Executor collects necessary number of signatures for the same execution results, it means that the agreement is reached and all the corresponding signatures are sent to the blockchain.

3. If by time $T + t$, where t is the maximum waiting time the results of the execution are not approved in the blockchain, the Executor sends signed opinion about the unsuccessful execution to all other Executors.

4. As soon as the Executor collects necessary number of signatures for unsuccessful execution, they are sent to the blockchain.

Since we assume that the supermajority of Executors is honest, in the worst case they all sooner or later will agree on the unsuccessful results of the execution. Since each Executor uses its local time, there is no need to synchronize time among the Executors and the algorithm works properly even if Executors complete execution with a very large time difference.

That's why we offer using a so-called two-phasic approving:

1. Executors try to approve the successful results of the execution using Multisignature mechanism described above.

2. If the results are not approved after expiration of a time limit, the Executors try to approve the unsuccessful results of the execution

In the worst case, when the Executors receive different results of the Execution, they are always able to cope with the situation and continue working in a regular mode.

The approach described above allows solving problem of interacting with the Internet: the information downloaded from the internet will be applied only if the Executors will be able to agree on the final results of the execution. At the same time, problems with interaction on the one Drive do not influence other Drives and Smart Contracts.

5.6. Rewarding the Executors

The Executors perform their work in order to receive rewards. The amount of the reward is proportional to the amount of work executed. As is other systems, this amount is measured as the number of opcodes executed by the WebAssembly Virtual Machine.

The algorithm of rewards accrual should satisfy the two given properties:

– If the Executor has honestly executed the Smart Contract, it must receive the reward.

– If the Executor has not executed the Contract, it must not receive the reward.

Malicious Executors may try to lie whether they have executed the contract in order to receive improper gain.

6. PROOF OF EXECUTION

The main idea of Proof of Execution is that those Executors who honestly execute the Contract Calls possess some information that is not available for other Executors. This allows them to prove the fact they have executed the Contract via proving the knowledge of this information. Such proof must be made in a Zero-knowledge proof manner in order to prevent disclosure of secret information.

6.1. Secret Information Generation

In this section we consider the ways of secret information generation. Since possession of the information proves the execution of the Contract Call, this information should be a digest of execution log. Thus, the process of the information generation can be divided into two parts – Execution Log Generation and Execution Log Information.

6.1.1. Execution Log Generation

The purpose of Execution Log Generation is to create such an array that reflects the process of the Call Execution. We offer two ways of the array creation: Standard and Custom. They are not mutually exclusive and can be used together.

Since Smart Contracts are executed in a Virtual Machine, the Contract Call is in fact an ordered list of opcodes, each of which has its own id and parameters o_i. This allows us to introduce the next representation of the array $a = o_1, \dots, o_n$. We call this approach a Standard one. The Standard Generation can be enabled or disabled at any moment.

The main drawback of the Standard approach is that it requires that all the Executors execute exactly the same list of opcodes. It does not suit very well for the Contracts that work with the Internet. That's why we offer an additional way for the formation of the array.

The Contract Creator can foresee calls of a special function that forcibly adds to the array the specified value. It allows the Creator to require checking the correctness of some critical values. We call the approach the custom one. The values are stored in the same array with the Standard

Generation so these two approaches can be used together.

6.1.2. Execution Log Aggregation

The purpose of the aggregation is to create a short digest of the Execution Log the possession of which more convenient to prove. We offer using hashing for these purposes. It allows reacting on small changes in the input array and produces a non-trivial-predictable random variable. At the same time, it doesn't matter that the algorithm is cryptographically strong: hash speed is much more important. In our experiments we use xxHash [21] that satisfies all mentioned requirements.

6.2. Providing Proofs

When the Executors approve the result of execution, they additionally approve the public information, necessary to verify the possession of secret information.

The verification is made by the blockchain nodes. Since information in the blockchain is public, proof of the Secret Information possession should be made in a Zero-Knowledge manner. In such cases a standard tool is proving knowledge of the discrete logarithm of a group element. Usually, such a group is either group of elliptic curve points [22] or multiplicative group of integers modulo n.

In order to prove the possession of the Secret information, the Executor should post a special transaction with the proof. In order to decrease the number of transactions, the Executor should have the possibility to provide proofs for several executions at once. Moreover, in order to be able to execute the next Calls before the transaction with Proof of Execution for the previous one is posted in the blockchain, the proof generated by the Executor should allow to verify that he has honestly executed all the Contract Calls starting with arbitrary Call in the past.

This is the reason for introduction Cumulative Discrete Logarithm Zero Knowledge Proof.

6.3. Proof of Execution Properties

The purpose of the Proof of Execution is to determine those nodes who honestly execute Smart Contract in order to reward them. More formally the algorithm should satisfy the following conditions:

– The Executors that honestly execute the Contract are able to generate correct Proof of Execution.

– The Executors that do not execute the Contract are not able to generate correct Proof of Execution except for negligible probability.

Both of these conditions are satisfied due to using Cumulative Zero-Knowledge Proof that satisfies Completeness, Soundness and Zero-Knowledge properties:

7. CUMULATIVE DISCRETE LOGARITHM ZERO-KNOWLEDGE PROOF

Let G to be a cyclic group with generator g such that the finding the discrete logarithm in this group is computationally difficult.

The standard problem of discrete logarithm zero-knowledge proof is formulated as following for given value $y \in G$ prove knowledge of such value x that $y = g^x$.

In the [23] the following non-interactive proof is proposed:

1. Peggy generates random value v and computes $\mathrm{t} = \mathrm{g}^{\mathrm{v}}$.

2. Peggy computes $\mathrm{c} = \mathrm{hash}(\mathrm{g}, \mathrm{y}, \mathrm{t}),\ \mathrm{i} = \overline{1, \mathrm{n}}$.

3. Peggy computes $\mathrm{r} = \mathrm{v} - \mathrm{cx}$.

4. The proof is the pair(t, r).

The primitive algorithm for proving knowledge of several values $x_1, \dots, x_n$ such that $g^{x_i} = y_i,\ i = \overline{1, n}$ is generating n independent single proofs for each value y_i. But such an approach is not efficient from the memory point of view: it would be more convenient if the proof's size was constant regardless of value of n.

The approach described in [24] allows us to extend the standard algorithm of discrete logarithm knowledge proof for proving knowledge of several logarithms with a constant proof size:

Peggy wants to prove Victor that she knows values $x_1, \dots, x_n$ such that $g^{x_i} = y_i,\ i = \overline{1, n}$.

1. Peggy generates random value v and computes $\mathrm{t} = \mathrm{g}^{\mathrm{v}}$.

2. Peggy computes $\mathrm{c_i} = \mathrm{hash}(\mathrm{g}, \mathrm{y_i}, \mathrm{t}, \mathrm{P}),\ \mathrm{i} = \overline{1, \mathrm{n}}$

3. Peggy computes $\mathrm{r} = \mathrm{v} - \mathrm{c_1 x_1} - \cdots - \mathrm{c_n x_n}$

4. The proof is the pair (t, r).

5. In order to verify the proof Victor computes values $\mathrm{c_i}$ and verifies the equality $\mathrm{t} = \mathrm{g}^{\mathrm{r}} \cdot \mathrm{g}^{\mathrm{c_1}} \cdot \cdots \cdot \mathrm{g}^{\mathrm{c_n}}$.

Nevertheless, such an approach requires that both, the Prover and the Verifier agree on the last proved value. As it is shown above, sometimes it is not possible to guarantee. Therefore, it is necessary to develop a mechanism that will not rely on the last proved value. A straightforward solution is each time generate a proof for all the values $y_1, \dots y_n$. But despite the proof's size is still constant, it is very non-efficient from the computational efficiency point of view: in order to generate and verify each proof it is necessary to process all preceding sequence values that makes the computational difficult of verifying n proofs $O(n^2)$.

Thus, an efficient algorithm must satisfy the next condition:

– Each single Proof allows to verify knowledge of all the sequence values from the beginning to some element y_m.

– Each Proof generation and verification is performed in time linear to the number of values for which the proof has not yet been generated and verified.

– The size of the Proof does not depend on the number of elements in the sequence.

We offer the Cumulative Discrete Logarithm Zero Knowledge Proof as the solution that satisfies all the conditions.

The main idea behind the Cumulative Zero Knowledge Proof is borrowed from prefix sums. Prefix sum $p_1, \dots, p_n$ of the array $a_1, \dots, a_n$ being the sum of the first i elements of the array allows finding the sum of any subarray $a_{m+1}, \dots, a_n$ in a constant time via finding the difference $p_n - p_m$.

In the same way Cumulative Zero Knowledge Proof allows verification of the knowledge of new values of a sequence via finding the "difference" of Proofs.

The Cumulative Proof consists of two parts: **Main Proof** and **Safety Proof**. Below we describe the algorithm of the proof generation and verification.

7.1. Cumulative Proof Generation

Main Proof

Let Peggy has already generated the Proof for values $x_1, \dots, x_m$ and now she would like to prove Victor that she possesses values $x_1, \dots, x_n$, $n > m$ that are discrete logarithms of values $y_1, \dots y_n$ base g. In order to generate the Main Proof Peggy

1) computes $c_i = \text{hash}(g, y_i, P)$, $i = \overline{m+1, n}$ where P is Peggy's public key

2) randomly picks $v \in \mathbb{Z}_l$, where l is the size of the group G.

3) computes $t_n = g^v$.

4) computes $r_n = v - c_1 x_1 - \dots - c_n x_n \mod l$

The tuple $b = (t_n, r_n)$ is called Main Proof.

Safety Proof

Unlike the original scheme for proving knowledge of a single discrete logarithm, the values of c_i in Main Proof do not depend on value t_n. This makes it very vulnerable. Safety Proof is an auxiliary part of Cumulative Proof and servers for The Safety Proof serves for the avoidance of fraudulent generation of the Main Proof. It allows proving that Peggy truly knows the value of the discrete logarithm of t_n using standard Fiat–Shamir heuristic.

In order to generate Safety Proof Peggy:

1) randomly picks $w \in \mathbb{Z}_l$;

2) computes $f = g^w$;

3) computes $d = \text{hash}(f, t_n, P)$;

4)computes $k = w - dv \mod l$.

The tuple $q = (f, k)$ is called Safety Proof.

7.2. Cumulative Proof Verification

We assume that Victor has already verified the Cumulative Proof for values $x_1, \dots, x_m$ with Main Proof (t_m, r_m). If $m = 0$ we assume that $r_m = 0$ and t_m is equal to the neutral element of the group G.

The Verification of the Cumulative Proof for values $x_1, \dots x_n$ consists of two parts:

Safety Proof Verification and **Main Proof Verification.**

Safety Proof Verification

In order to verify whether the Safety Proof is valid, Victor:

1) computes $d = \text{hash}(f, t_n, P)$;

2) verifies the equality: $f = g^k t_n^d$.

If the equality is true, then Victor goes to Main Proof Verification else the entire Proof is invalid.

Main Proof Verification

In order to verify whether the Main Proof is valid, Victor:

1) computes $c_i = \text{hash}(g, y_i, P)$, $i = \overline{m+1, n}$;

2) verifies the equality $t_n t_m^{-1} = g^{r_n - r_m} y_{m+1}^{c_{m+1}} \cdot \dots \cdot y_n^{c_n}$.

If the equality is true then the entire Cumulative Proof is considered as a valid one.

7.3. Cumulative Proof Correctness

In this section we attempt to prove three properties of the proposed Cumulative Zero-Knowledge Proof: completeness, soundness and zero-knowledge [12].

Completeness

Completeness property provides that any honest Prover that is that one who possesses all values $x_1, \dots, x_n$ will pass the verification.

The completeness of the proposed Cumulative Proof follows from the next equality:

$$t_n t_m^{-1} = g^{v_n} g^{-v_m}$$
$$= g^{r_n + c_1 x_1 + \dots + c_n x_n} g^{-(r_m + c_1 x_1 + \dots + c_m x_m)}$$
$$= g^{r_n - r_m} g^{c_{m+1} x_{m+1} + \dots + c_n x_n}$$
$$= g^{r_n - r_m} y_{m+1}^{c_{m+1}} \cdot \dots \cdot y_n^{c_n}$$

Soundness

The Soundness means that the Prover that does not really know values $x_1, \dots, x_n$ can pass the verification with negligible probability.

The original scheme requires computing the value c based on the value of T . It allows guaranteeing that the Prover who does not know the Secret Information has to solve a difficult Problem of Discrete Logarithm Computing. Nevertheless such an approach does not suit our algorithm because otherwise, there will be no mutual annihilation of the corresponding terms during verification and completeness property will not be satisfied.

Now we will prove that the introduction of Safety Proof in addition to Main Proof provides the same protection as in standard Fiat-Shamir Heuristic.

Suppose that the Prover has already successfully passed Verification for values $x_1, \dots, x_m$ with Main Proof (t_m, r_m) hense he has proved that he knows all of them. Now he attempts to prove knowledge of $x_1, \dots, x_n$, $n > m$ without possession the knowledge about at least one of these values.

The Prover should find such pair (t_n, r_n) that $t_n t_m^{-1} = g^{r_n - r_m} y_{m+1}^{c_{m+1}} \cdot \dots \cdot y_n^{c_n}$. Then for such a Prover, there are two action strategies.

First Strategy. The Prover does not fix the value of v at the corresponding algorithm step. This allows him to select the value r_n in an arbitrary way that uniquely identifies the value of $t_n = g^{r_n - r_m} t_m y_{m+1}^{c_{m+1}} \cdot \dots \cdot y_n^{c_n}$. But now to pass Safety Proof the Prover should find the value of the discrete logarithm of T that is computationally difficult.

Second Strategy. The Prover fixes the value of v at the corresponding algorithm step that uniquely identifies the value of $t_n = g^v$ (thus, the Prover generates correct Safety Proof according to the algorithm) and now the Prover should find the value of r_n such that $g^{r_n} = t_n t_m^{-1} g^{r_m} y_{m+1}^{-c_{m+1}} \cdot \dots \cdot y_n^{-c_n}$ that again makes him solve the discrete logarithm problem (the Prover cannot use the formula $r_n = v - c_1 x_1 - \dots - c_n x_n \bmod l$ since he does not know at least one of x_i).

Zero-Knowledge

Single Proof does not disclose more information than the original scheme so the Zero Knowledge of the single Proof follows from Zero Knowledge of the scheme.

Two given proofs for values $x_1, \dots, x_m$ and for $x_1, \dots, x_m$, $m < n$ do not provide the Verifier with any additional information since in the difference $r_n - r_m = (v_n - v_m) + c_{m+1} x_{m+1} + \dots + c_n x_n$ the value of $v_n - v_m$ is unknown for the Verifier.

7.4. Cumulative Proof Properties

One of the important properties of the Cumulative Proof is that it allows proving knowledge of the sequence elements not from the very beginning but starting from the arbitrary element. It allows the Executors to start Smart Contract execution at any moment without the need of executing Calls that were initiated long time ago in order to be able to generate a valid Proof of Execution.

8. COMPARISION WITH THE EXISTING SOLUTIONS

The existing schemes of increasing scalability of Smart Contracts propose Sharding blockchain nodes [8] or executing Smart Contracts offchain [9, 10]. The Concept proposed in the article in fact merges these ideas in such a way that Contracts are executed off chain but with formation of concrete Shards.

Such scheme has a row of advantages comparing to the existing solutions:

– *The possibility of allocating an arbitrary amount of information for the Contract work.* The property is achieved due to allocating the data on the Drives that allow information overwriting and are stored only on a subset of Network nodes.

– *The possibility of interaction with the Internet.* The property is achieved due to a specially developed algorithm of agreement on the Smart Contract Execution result.

– *Incentivization of all nodes to honestly execute the Smart Contract.* The property is achieved due to first proposed algorithm of Proof of Execution.

9. CONCLUSIONS

The work is devoted to the development of Sharding Concept for Decentralized Applications (DApps) that are expressed in form of Smart Contracts.

The concept assumes that Smart Contract data is stored on specially designed decentralized file storages called Drive. The Drives unlike other solutions, such as [15, 16] allow decentralized collective-decision making. It allows considering them as separate Shards.

The special algorithm of agreement on the results of Smart Contract execution is designed. Due to its ability to determine impossibility to find consensus on successful Contract Execution the interaction with internet becomes possible that favorably distinguishes the proposed system from the existing ones, in which obtaining information from the Internet requires special oracles.

The main advantages of the proposed scheme comparing to the existing ones are lack of any centralization, the possibility of storing arbitrary large amount of information and interaction with the Internet.

One of the key features of the proposed system is the rewards distribution scheme. Unlike other systems it assumes all nodes who honestly execute the Smart Contract remuneration.

In order to make it possible a special scheme called Proof of Execution is developed. It allows determining which nodes honestly receive their rewards based on the assumption that such nodes possess some information that is unavailable to the malicious nodes. The way of constructing the information as execution fingerprint is offered.

A special cryptographic protocol necessary for making computational and memory complexity of Proof of Execution called

Cumulative Zero-Knowledge Proof is developed. It allows proving the possession of expanding set of discrete logarithms without knowledge which part of the sequence has already been proved by the moment of new proof generation. This algorithm is superstructure over existing discrete logarithm ZK proof and so is applicable to any group in which the discrete logarithm is computationally difficult. The correctness of the protocol that consists of Completeness, Soundness and Zero-Knowledge is proven.

Thus, the new concept of Smart Contracts Sharding with the possibility of interaction with the Internet is offered, the mechanism of incentivizing nodes to honestly execute the Smart Contracts is developed, the Cumulative Proof that is necessary for cryptographic strength of the specified mechanism is offered and its correctness is proven. The goals set in the article have been achieved.

REFERENCES

1. Mazurok, I. E., Leonchyk, Y. Y. & Kornylova, T. Y. “Proof-of-Greed Approach in the NXT Consensus”. *Applied Aspects of Information Technology. Publ. Science i Technical*. Odessa: Ukraine. 2019; Vol.2 No.2: 153–160. DOI: https://doi.org/10.15276/aait.02.2019.6.

2. Kovalev, I. S., Drozd, O. V., Rucinski, A., Drozd, M. O., Antoniuk, V. V. & Sulima, Yu.Yu. “Development of Computer System Components in Critical Applications: Problems, their Origins and Solutions”. *Herald of Advanced Information Technology. Publ. Nauka i Tekhnica*. 2020; Vol.3 No.4: 252–262. Odessa. Ukraine. DOI: https://doi.org/10.15276/hait.04.2020.4.

3. Kalnauz, D. V. & Speranskiy, V. A. “Productivity Estimation of Serverless Computing”. *Applied Aspects of Information Technology. Publ. Nauka i Tekhnica*. Odessa: Ukraine. 2019; Vol. 2 No.1: 20–28. DOI: https://doi.org/10.15276/aait.01.2019.2.

4. Szabo, N. “Smart Contracts”. – Available from: https://www.fon.hum. uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html. – [Accessed 15 Feb 2020].

5. Hugo Hoffmann, C. “Blockchain Use Cases Revisited: Micro-Lending Solutions for Retail Banking and Financial Inclusion”. *Journal of Systems Science and Information*, 2021; Vol. 9 No. 1: 1–15. DOI: https://doi.org/10.21078/JSSI-2021-001-15.

6. Malamud, Semyon & Marzena Rostek. “Decentralized Exchange”. *American Economic Review*, 2017; 107 (11): 3320-3362. DOI: https://doi.org/10.1257/aer.20140759.

7. Ante, L. “The Non-Fungible Token (NFT) Market and its Relationship with Bitcoin and Ethereum”. *BRL Working Paper Series*. 2021; Vol. 21: 1-15. DOI: https://dx.doi.org/10.2139/ssrn.3861106.

8. CRYPTO.COM. “Ethereum 2.0. An Introduction”. – Available from: https://assets.ctfassets.net/hfgyig42jimx/7j3AVp5aCnx2Ct2P6T6kY/e8991d5da1972d5d760233868f237609/Crypto.com_Macro_Report_-_Ethereum_2.0.pdf. – [Accessed 15 Feb 2020].

9. Marukhnenko, O. & Khalimov, G. “The Overview of Decentralized Systems Scaling Methods”. *Fifth International Scientific and Technical Conference “Computer and Information Systems and Technology”*. 2021: 37-38. DOI: https://doi.org/10.30837/csitic52021232174.

10. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C et al. “Hyperledger fabric: a distributed operating system for permissioned blockchains”. *In Proceedings of the Thirteenth EuroSys Conference (EuroSys '18). Association for Computing Machinery*, New York: NY, USA. 2018; Article 30, 1–15. DOI: https://doi.org/10.1145/3190508.3190538.

11. Feige, U., Fiat, A. & Shamir, A. “Zero-Knowledge Proofs of Identity”. J. Cryptology 1. 1988. p. 77–94. DOI: https://doi.org/10.1007/BF02351717.

12. Goldreich, O. & Oren, Y. “Definitions and Properties of Zero-Knowledge Proof Systems”. *Journal of Cryptology”*. 1994; Vol. 7 No. 1: 1–32. DOI: https://doi.org/10.1007/BF00195207.

13. Ben-Sasson, E., Chiesa, A., Tromer, E. & Virza, M. “Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture”. *SEC'14: Proceedings of the 23rd USENIX Conference on Security Symposium.* 2014. p. 781–796.

14. Ben Sasson, E. et al. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. *IEEE Symposium on Security and Privacy*. 2014. p. 459–474. DOI: https://doi.org /10.1109/SP.2014.36.

15. Castro, M. & Barbara, L. “Practical Byzantine Fault Tolerance”. *In Proceedings of the third symposium on Operating systems design and implementation (OSDI '99). USENIX Association. USA.* 1999. p. 173–186.

16. Y. Psaras and D. Dias, "The InterPlanetary File System and the Filecoin Network," 2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S). 2020. p. 80–80. DOI: 10.1109/DSN-S50200.2020.00043.

17. Figueiredo, S., Madhusudan, A., Reniers, V., Nikova, S. & Preneel, B. “Exploring the Story Network: a Security Analysis”. *In Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC '21). Association for Computing Machinery*. New York: NY, USA. 2021. p. 257–264. DOI: https://doi.org/10.1145/3412841.3441908.

18. Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A. & Bastien, J. F. “Bringing the Web up to Speed with WebAssembly”. *In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017). Association for Computing Machinery*. New York: NY, USA. 2017. p. 185–200. DOI: https://doi.org/10.1145/ 3062341.3062363.

19. Paulin, O. N., Komleva, N. O., Marulin, S. U. & Nikolenko, A. A. “Method for Constructing the Model of Computing Process Based on Petri Net”. *Applied Aspects of Information Technology. Publ. Science i Technical*. Odessa: Ukraine. 2019; Vol. 2 No. 4: 260–270. DOI: https://doi.org/10.15276/ aait.04.2019.1.

20. Gilad, Y., Hemo, R., Micali, S., Vlachos, G. & Zeldovich, N. “Algorand: Scaling Byzantine Agreements for Cryptocurrencies”. *SOSP '17: Proceedings of the 26th Symposium on Operating Systems Principles*. 2017. p. 51–68. DOI: https://doi.org/10.1145/3132747.3132757.

21. GITHUB. “xxHash – Extremely fast hash algorithm”. – Available from: https://github.com/Cyan4973/xxHash. – [Accessed 15 Feb 2020].

22. Washington, L.C. Elliptic Curves: Number Theory and Cryptography, Second Edition (2nd ed.). Chapman and Hall/CRC. 2008. DOI: https://doi.org/10.1201/9781420071474

23. Bernhard, D., Pereira, O. & Warinschi, B. “How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios”. *Advances in Cryptology – ASIACRYPT*. 2012. p. 626–643. DOI: https://doi.org/10.1007/978-3-642-34961-4_38.

24. J. Camenisch and M. Stadler, “Proof systems for general statements about discrete logarithms,” ETH Zurich, Technical Report No. 260. Tech. Rep. 1997. DOI: https://doi.org/10.3929/ETHZ-A-006651937.

Conflicts of Interest: the authors declare no conflict of interest

Received 15.12.2020
Received after revision 25.02.2021
Accepted 16.03.2021

DOI: https://doi.org/10.15276/aait.03.2021.6
УДК 004.75

Шардування смарт контрактів з доказом виконання

Ігор Євгенійович Мазурок[1)]
ORCID: https://orcid.org/0000-0002-6658-5262; igor@mazurok.com. Scopus ID: 57192064365
Євген Юрійович Леончик[1)]
ORCID: https://orcid.org/0000-0003-1494-0741; leonchik@ukr.net. Scopus ID: 57192064365
Олександр Сергійович Антоненко[1)]
ORCID: https://orcid.org/0000-0001-9680-3446; asantonenko@gmail.com. Scopus ID: 17433258300
Кирило Сергійович Волков[1)]
ORCID: https://orcid.org/0000-0002-7705-8994; cyrillicw@gmail.com
[1)] Одеський національний університет імені І. І. Мечникова, вул. Дворянська, 2. Одеса, 65082, Україна

АНОТАЦІЯ

Останнім часом активно досліджуються децентралізовані мережі на основі технології блокчейн. Особливе місце в цих дослідженнях займають Смарт Контракти, що широко використовуються в багатьох галузях, таких як децентралізовані фінанси (DeFi), нерухомість, азартні ігри, виборчі процеси тощо. Тим не менше, можливість їх широкого застосування є досі не вирішеною проблемою. Це викликано тим, що вони мають обмежену гнучкість та масштабованість. Іншими словами, Смарт Контракти не можуть обробляти велику кількість викликів у секунду, відсутність прямого доступу до мережі Інтернет, неможливість роботи з великою кількістю даних тощо. Дана робота присвячена розробці концепції шардування для децентралізованих програм (DApps) у формі контрактів, написаних на WebAssembly. Пропонується концепція, яка передбачає, що кожен Контракт виконується набором випадково обраних вузлів, що дозволяє уникнути змови та запобігти атаці Сивілли. Під час дослідження були розроблені децентралізовані Сховища даних з можливістю колективного прийняття рішень. Запропонована схема формування Сховищ, яка передбачає, що кожен Контракт виконується набором випадково вибраних вузлів, що дозволяє уникнути змови та запобігти атаці Сивілли. Такий підхід дозволив використовувати Сховища як базовий рівень для Смарт Контрактів. Крім того, Сховища можна використовувати як автономне рішення для децентралізованого зберігання даних. Описано особливості узгодження результатів виконання Контрактів, що значно розширює можливості Контрактів порівняно з Ethereum Smart Contracts і, зокрема, дозволяє взаємодіяти Контрактам з Інтернетом. Розроблено концепцію винагороди, яка стимулює всі вузли, які чесно виконують Контракти, на відміну від інших систем, де винагороду отримує лише блок продюсер. Вона базується на спеціально розробленому Доказі Виконання (Proof of Execution) – спеціальному алгоритмі, який дозволяє виявляти всі вузли, які чесно виконують Контракти. Для того, щоб зробити Доказ Виконання більш компактним, розроблено кумулятивне розширення існуючого алгоритму доведення знання дискретного логарифму з нульовим розголошенням, що дає можливість послідовно доводити знання динамічно розширюваного набору значень з мінімальною обчислювальною та пам'ятною складністю. Таким чином, у цій статті досліджується нова концепція шардування Смарт Контрактів, що наділена економічними важелями. Основними перевагами запропонованого підходу є можливість взаємодії з мережею Інтернет та обробка великих об'ємів даних. Крім того, розроблено механізм стимулювання вузлів до чесного виконання Смарт Контрактів. А також пропонується Доказ Виконання, що необхідно для криптографічної міцності зазначеного механізму, та доведена його коректність. Отримані результати можуть бути використані для реалізації Смарт Контрактів у децентралізованих системах, зокрема, що працюють на основі технології Blockchain, особливо у випадку вимог до високої пропускної здатності та продуктивності.

Ключові слова: доведення виконання; кумулятивне доведення; шардування; смарт контракти; доведення з нульовим розголошенням

ABOUT THE AUTHORS

Igor E. Mazurok, PhD in Computer Science, Associate Prof. of the Department of Optimal Control and Economic Cybernetics. Odessa I. I. Mechnikov National University. 2, Dvoryanskaya Str. Odessa, 65082, Ukraine
ORCID: https://orcid.org/0000-0002-6658-5262; igor@mazurok.com; Scopus ID: 57192064365
Research field: Distributed computing; decentralized system design and modeling; artificial intelligence

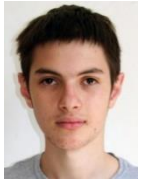
Ігор Євгенійович Мазурок, кандидат технічних наук, доцент кафедри Оптимального керування та економічної кібернетики. Одеський національний університет ім. І. І. Мечникова, вул. Дворянська, 2. Одеса, 65082, Україна

Yevhen Y. Leonchyk, Ph.D. in Physics and Mathematics, Associate Prof. of the Department of Mathematical Analysis. Odessa I. I. Mechnikov National University. 2, Dvoryanskaya Str. Odessa, 65082, Ukraine
ORCID: https://orcid.org/0000-0003-1494-0741; leonchik@ukr.net; Scopus ID: 57192064365
Research field: Mathematical modelling of computer; environmental and economic complex systems; blockchain technology

Євген Юрійович Леончик, кандидат фізико-математичних наук, доцент кафедри Математичного аналізу. Одеський національний університет ім. І. І. Мечникова, вул. Дворянська, 2. Одеса, 65082, Україна

Oleksandr S. Antonenko, PhD, Associate Prof. of the Department of Mathematical Support of Computer Systems. Odessa I. I. Mechnikov National University. 2, Dvoryanskaya Str. Odessa, 65082, Ukraine
ORCID: https://orcid.org/0000-0001-9680-3446; asantonenko@gmail.com; antonenko@onu.edu.ua; Scopus ID: 17433258300
Research field: Automata theory; automata groups and semigroups; computability theory; algorithmic information theory; blockchain

Олександр Сергійович Антоненко, кандидат фізико-математичних наук, доцент кафедри Математичного забезпечення комп'ютерних систем. Одеський національний університет імені І. І. Мечникова, вул. Дворянська, 2. Одеса, 65082, Україна



Kyrylo S. Volkov, Bachelor of Applied Mathematics, Master Student. Odessa I. I. Mechnikov National University, 2, Dvoryanskaya Str. Odessa, 65082, Ukraine
ORCID: https://orcid.org/0000-0002-7705-8994; cyrillicw@gmail.com
Research field: Blockchain; DeFi; machine learning

Кирило Сергійович Волков, магістрант. Одеський національний університет імені І. І. Мечникова, вул. Дворянська, 2. Одеса, 65082, Україна