

Received July 5, 2020, accepted July 24, 2020, date of publication July 30, 2020, date of current version August 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3013199

A Blockchain Consensus Protocol Based on Dedicated Time-Memory-Data Trade-Off

MIODRAG J. MIHALJEVIĆ 

Mathematical Institute, Serbian Academy of Sciences and Arts, 11000 Belgrade, Serbia

e-mail: miodragm@turing.mi.sanu.ac.rs

This work was supported by the Ministry of Education, Science and Technological Development of Republic Serbia.

ABSTRACT A problem of developing the consensus protocols in public blockchain systems which spend a combination of energy and space resources is addressed. A technique is proposed that provides a flexibility for selection of the energy and space resources which should be employed by a player participating in the consensus procedure. The technique originates from the cryptographic time-memory-data trade-off approaches for cryptanalysis. The proposed technique avoids the limitations of Proof-of-Work (PoW) and Proof-of-Memory (PoM) which require spending of only energy and space, respectively. Also, it provides a flexibility for adjusting the resources spending to the system budget. The proposed consensus technique is based on a puzzle where the problem of inverting one-way function is solved employing a dedicated Time-Memory-Data Trade-Off (TMD-TO) paradigm. The algorithms of the consensus protocol are proposed which employ certain unconstrained and constrained TMD-TO based inversions. Security of the proposed technique is considered based on the probability that the honest pool of nodes generate a longer extension of the blockchain before its update, and a condition on the employed parameters in order to achieve desired security has been derived. Implementation of the proposed technique in Go language and its inclusion as an alternative consensus option in Ethereum platform are shown. Implementation complexity and performance of the proposed consensus protocol are discussed and compared with the ones when PoW and PoM are employed.


INDEX TERMS Blockchain, consensus, proof of work, proof of memory, security.

I. INTRODUCTION

In a large number of scenarios we face particular instantiations of the following general problem: All updates of a huge database should be verified before becoming effective. A generic approach for performing the verification is the centralized one where a trusted party check and verifies all the updates. The main problems with this approach are requirement of a trusted party, and the problem of the single point of failure. Recently, as an alternative approach, the blockchain paradigm has been proposed within the bitcoin system [10], where the verification is performed in a distributed manner without requirement for the trusted party as the verification arbiter. The removal of the trusted party and the distributed verification approach requires an appropriate technique for achieving the verification decision: For this purpose, the blockchain employs the so called consensus protocol. This consensus protocol appears as a system

overhead. We could say that the escape from the centralized verification paradigm should be paid by the overhead related to the required blockchain consensus protocol. The overheads implied by the employed consensus protocol could be very large and it is an open research issue to construct dedicated consensus algorithms in order to minimize the overheads in a system based on blockchain technology.

An illustrative simplified framework of a system based on the blockchain paradigm is given in Fig. 1. Two main components of any blockchain are the ledger, a distributed database, and the consensus protocol. The consensus protocol is an algorithm which specifies the procedure for generation a candidate block and its inclusion into the blockchain ledger, i.e. update/extension of the blockchain database with a verified new block. Fig. 1 emphasizes components of the blockchain relevant for this article, i.e. the consensus protocol, its puzzle and employed cryptographic components, as well as their simplified relation with other components of the system. A key component of the puzzle is one or more cryptographic techniques employed for building the consensus protocol.

The associate editor coordinating the review of this manuscript and approving it for publication was Junggab Son .

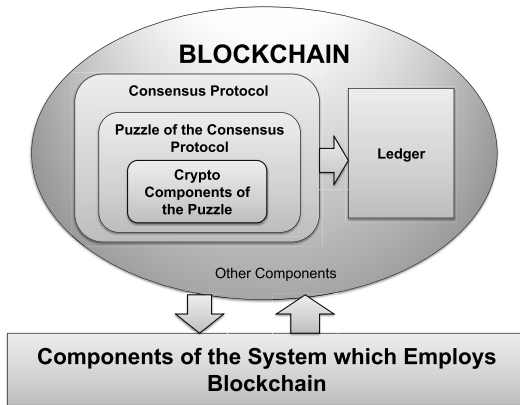


FIGURE 1. A simplified architecture of a class of blockchain based systems.

A lot of consensus protocols for blockchain based system have been proposed and the surveys [13]–[15], and [7], for example, provide summary of the main approaches. For the purposes of this article, we point out to the following two paradigms for achieving the consensus: Proof-of-Work (PoW) and Proof-of-Space (PoS) known also as Proof-of-Memory (PoM), and in order to avoid confusion with abbreviation of Proof-of-Space, the abbreviation PoM is used.

PoW approach, initially employed for protection against e-mail spam and denial of service, has been introduced in [4]: It is a proof approach where a prover, the party which spends some resources, convinces a verifier that some computation with respect to some statement x has been spent. An illustration of PoW technique is the following: Let $H(\cdot)$ be a hash-function which hashes the concatenation of given binary sequence x and a randomly selected seed s so that $H(s||x)$ begins with certain number t of zeros. Assuming that $H(\cdot)$ acts as a random function, the prover must evaluate $H(s||x)$ on 2^t different values of s (in expectation) before ending with a required s . Accordingly, the resource to be spent for PoW approaches is energy. Nowadays, a family of PoW approaches appears as the most widespread paradigm for securing the blockchains. The main, well known, drawback of PoW is requirement for heavy spending of energy at each participating node, and when this spending is considered cumulatively, it appears as a huge overhead. In particular, it is important to note that PoW design does not allow a trade-off between required energy and some other resource. Consequently, it appears as an interesting challenge to develop a blockchain consensus protocol which provides reduction of the required energy spending.

PoM is another paradigm for the consensus protocols - It is based on a proof that certain space/memory has been booked for participation in the consensus protocol, i.e. the resource which should be spent is “memory”. A simple illustration of the approach is the following: the verifier specify a random function f which maps a space of dimension N into the same dimensional space. During the initialization phase, the prover should compute the function table of f and sort it by the output

values. During the verification phase, to convince the verifier that prover posses the table, the prover must invert f on a random challenge. The previous is just an illustration of PoM underlying paradigm, but this simple PoM appears as the insecure one. A secure PoM has been proposed in [5] employing certain graphs in order to avoid time-memory trade-off approach for inversion reported in [6]. The approach [5] is such that a cheating prover needs $\Theta(N)$ space or time after the challenge is known to make the response which the verifier accepts. An application of this PoM has been employed for developing a crypto-currency reported in [11]. Another PoM has been proposed in [1] based on inverting random functions from a class of functions that are hard to evaluate in the forward direction, and even harder to invert. Also, [1] provides construction of functions satisfying this condition and proves lower bounds on time-memory trade-offs beyond the upper bounds given in [6]. Note that PoM is designed in such a way that each node should employ the claimed big memory in order to obtain a desired role into the consensus protocol. Although, certain approaches have been reported that the employed memory could have double functionality, as an archive storage and space with data required for PoM, the main drawback of PoM is requirement that each node should allocate a big memory for participation in the consensus protocol. In particular, it is important to note that PoM design does not allow a trade-off between required memory and some other resource. Accordingly, it appears as an interesting issue developing a blockchain consensus protocol which provides reduction of the memory required.

A. MOTIVATION FOR THE WORK

PoW based consensus is based on a puzzle which should be solved employing energy only: The underlying problem has such nature that employment of a memory does not help. On the other hand PoM based consensus is built over a problem which could be efficiently solved (and solved within given time slots) only if a memory large enough is employed and in this approach particular attention is payed in order to avoid possibility of using time-memory trade-off (i.e. energy-space trade-off) to obtain any benefit. So, a player who participate in the blockchain system running (operating), usually called a miner, has no option. The miner must employ a single type of the resources: energy or space - a combination is not allowed. We believe that in a number of scenarios a miner should have opportunity to design the mining budget based on a combination of different resources, and in particular as a combination of energy and space in order to deal with solving the consensus puzzle. The flexibility in selection of the resources to be spent is also important in a context of the incentive required for a miner when the mining should be performed for a small fee only. On the other hand, we reemphasize that the existing PoW and PoM underlying problems are intentionally selected so that efficient time-memory or time-memory-data trade-offs (see [6] and [2], for example) can not be employed. Note, PoW and PoM are built on the same paradigms as the cryptanalysis of encryption employing the

exhaustive search or the code-book table, respectively. Recall that the exhaustive search cryptanalysis assumes search for a solution based on testing all possible candidates for the keys, and the code-book based cryptanalysis assumes construction of certain table with the pairs (key, ciphertext) which provides inversion in the look-up manner. On the other hand, time-memory trade-off (TM-TO) and time-memory-data trade-off (TMD-TO) have been introduced to provide trade-offs of the resources required for the cryptanalysis. Consequently, our motivation is consideration of the consensus protocols design based on the TM-TO (TMD-TO) approaches for trade-off between the resources a miner should employ.

B. SUMMARY OF THE RESULTS

This article proposes a technique for the consensus in public blockchain systems. The proposed technique provides a flexibility for selection of the resources which should be employed by a player participating in the consensus procedure. It provides a possibility for different trade-offs between the required energy and space which should be spent during execution of the consensus protocol. The technique originates from the cryptographic time-memory-data trade-off approaches for cryptanalysis. The proposed technique avoids the limitations of PoW and PoM which require spending of only energy and space, respectively. Also, it provides a flexibility for adjusting the resources spending to the system budget. The differences of the developed consensus protocol in comparison with the previously reported ones are: (i) the puzzle which has to be solved, (ii) construction of the challenge for the puzzle, and (iii) the technique for solving the puzzle problem. The proposed consensus technique is based on a puzzle where the problem of inverting one way function is solved employing dedicated TMD-TO. Algorithms for the consensus protocol are proposed including dedicated ones for specific inversion problems of one way function where the challenge C is an ℓ -bits binary prefix (or suffix) of a ciphertext and the inversion yields one of the keys which encrypt n -bits all ones binary vector into an n -bits binary ciphertext with the given prefix C , where $\ell \leq n$. Two options for the inversion problem are considered: the unconstrained one and the constrained one. Security of the proposed technique is considered based on the probability that the honest nodes generate a longer extension of the blockchain before its update, and a condition on the employed parameters, in order to achieve desired security, has been derived. Implementation of the proposed protocol in Go language is shown and it has been incorporated into Ethereum platform. The complexity and the performance of the proposed consensus protocol are discussed and compared with the implementation complexity and performance when PoW and PoM are employed.

C. ORGANIZATION OF THE PAPER

A framework for the puzzle used in the consensus protocol is given in Section II. The algorithms of the consensus protocol employing the unconstrained and constrained TMD-TO

based inversions are proposed in Sections III and IV, respectively. Section V yields a security evaluation of the the proposed consensus algorithm. A discussion on implementation issues and performance of the proposed approach and comparison with the ones based on PoW and PoM are given in Section VI. Some concluding notes are given in Section VII. Finally, implementation of the consensus algorithms in Go language is shown in the appendix.

II. A CONSENSUS PROTOCOLS BASED ON THE PUZZLE REPLACEMENT

This section proposes a modification of the traditional PoW consensus protocol where hash-function based puzzle is replaced with a novel one.

A. THE CONSENSUS PROTOCOL

The consensus protocol proposed in this section follows the same framework as the traditional consensus protocols for public blockchains based on PoW employed in Bitcoin and Ethereum. Each execution of these protocols consists of the following three main phases: (i) Construction a block of transactions; (ii) Solving a puzzle; (iii) Inclusion the considered block into the blockchain (assuming that no one transaction from the block has been already included in the blockchain). The main differences of the novel blockchain consensus protocol in comparison with the previously reported ones are the puzzle which has to be solved, construction of the challenge for the puzzle, and technique for solving the puzzle.

B. THE EMPLOYED PUZZLE AND THE SOLVING METHOD

The puzzle for the consensus protocol proposed in this article requires, as the first step, construction of a challenge for the puzzle which should be solved. The challenge is specified as follows:

- find a nonce, which after added to the binary representation of the block, provides that hash value of the block with the nonce begins with certain number of zeros;
- take certain number of bits from the hash vector and form the binary challenge vector.

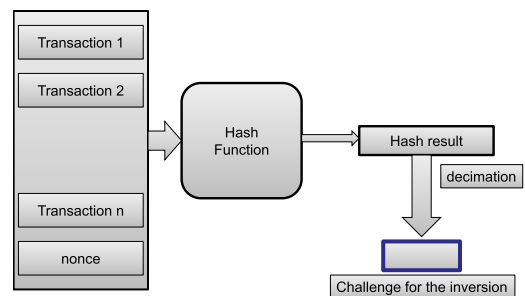


FIGURE 2. Paradigm of the challenge construction for the puzzle problem.

The step (a) could employ the same hash-function as employed in Bitcoin and Ethereum consensus protocols and could be considered as a mini PoW. The bits selected in the step (b) could be from arbitrary positions of the hash vector, and as a particular instantiation they could be a suffix (or prefix) of the hash value. Fig. 2 illustrates the approach

employed for construction of the challenge for the puzzle which should be solved later on: A number of the transactions and a nonce are organized as a binary vector; This vector is input to the hash function; The obtained hash-value is subject of decimation which yields the challenge for the inversion.

The puzzle itself is an inversion problem for the given challenge: The challenge is considered as the prefix/suffix of a ciphertext generated by certain encryption algorithm for the all-ones message (or any other given message), and the puzzle problem is to find a key which provides mapping of the message into a ciphertext with the considered prefix/suffix. The puzzle paradigm is illustrated in Fig. 3. Note that this inversion problem could have a lot of solutions because the message is longer than the challenge.

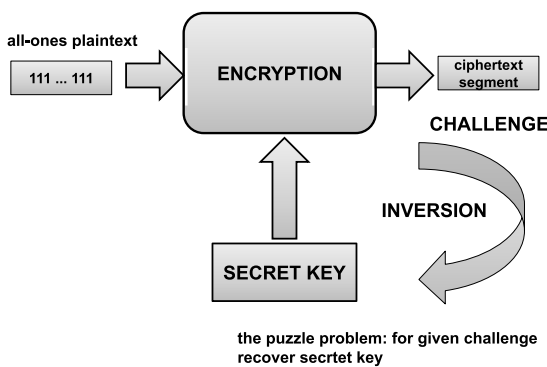


FIGURE 3. Paradigm of the puzzle.

As the method for solving the puzzle, this article proposes employment of a dedicated TMD-TO (time-memory-data trade-off) approach. This approach belongs to a family of cryptanalytic approaches which are the generic ones for cryptanalysis of certain encryption schemes. The proposed approach originates from the TM-TO technique reported in [6] and later-on generalized in a number of papers. TM/TMD-TO is a generic technique for inversion of certain one-way functions which could be easily evaluated in forward direction but are hard for the inversion. The inversion is based on joint employment of a memory and restricted exhaustive search. TM/TMD-TO based inversion consists of two phases: the preparation (preprocessing) phase and the processing phase. The memory employed should be initialized in the preprocessing phase. The initialized memory consists of certain suitably constructed table/tables - these tables should be constructed only once in advance, and later on used for all inversions of the considered one-way function. In the processing phase a restricted exhaustive search and the tables are employed.

1) PREPROCESSING

The tables constructed in the preprocessing phase are as follows. Each table contains just two columns: The second column row element is evaluated through a number of recursive recalculations with the first element of the row as the

starting argument according to the following:

$$[x_{i,j+1}||r_{i,j+1}] = f([x_{i,j}||r_{i,j}]),$$

$$j = 0, 1, \dots, t - 1, \quad i = 1, 2, \dots, m,$$

where $[x_{i,j}||r_{i,j}]$ is a binary vector with the prefix $x_{i,j}$ and the suffix $r_{i,j}$, $f(\cdot)$ is a function under the inversion processing, and t, m are the parameters. In each row of the constructed two-column table the first column element is $[x_{i,0}||r_{i,0}]$ and the second one is $[x_{i,t}||r_{i,t}]$, $i = 1, 2, \dots, m$. An illustration of the table generation is displayed in Fig. 4.

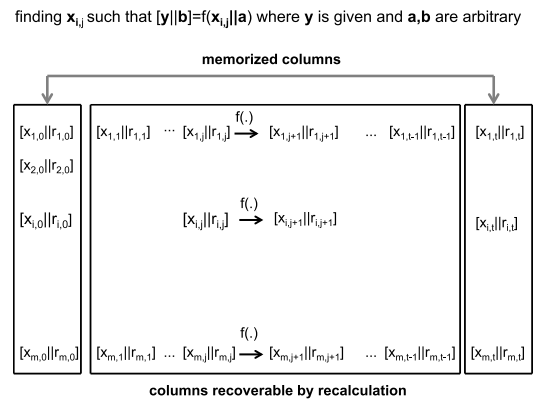


FIGURE 4. Time-memory trade-off framework for the inversion.

TM-TO Based Inversion

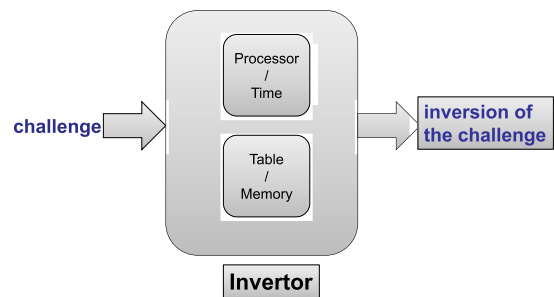


FIGURE 5. The processing phase: Paradigm for solving the puzzle.

2) PROCESSING

The processing phase yields a solution for the given inversion problem, and it is informally explained as follows, as well as in Fig. 5. In a general setting, the goal is to find an argument $x_{i,j}$ such that $[y||b] = f([x_{i,j}||a])$ where y is a given image and a, b are arbitrary. The inversion is based on a number of iterative recalculations, and after each recalculation a search over the second column of the table constructed in the preprocessing phase. The iterative recalculation follows the same approach as the one which has been employed for generation of the table. Assuming that, after certain recalculation, the result corresponds to the element in the i -th row of the second column, we finalize the inversion as follows: Take the

i -th row element of the first column and perform the iterative recalculations until we obtain the result which corresponds to the given y - the inversion result is the argument $x_{i,j}$ which has generated y .

Consequently, in a general case, we employ a dedicated constrained re-evaluation where the input for the next iteration step is output of the previous step modified in a deterministic manner. Note that, because in a general case the given image belong to a subset of all possible images, the inversion results is not a unique one, i.e. we obtain one of possible arguments which map into the given image.

The method for solving the puzzle problem, i.e. the inversion problem, follows the approaches illustrated in Figs. 4 and 5, and will be in details explained in the next two sections. This approach employs some underlying ideas reported in [8] and [9], as well.

III. THE BASIC APPROACH FOR THE CONSENSUS

This section yields a basic, unconstrained, version of the consensus protocol based on a dedicated TMD-TO.

The consensus protocol for inclusion of a new block into the blockchain, we propose, consists of the preprocessing algorithm and the protocol execution algorithms described in the next two subsections.

For given encryption scheme $E_K(\cdot)$, the inversion problem addressed is the following one: Recover a binary n -dimensional key K which encrypts an n -dimensional all ones vector 1^n into a binary ciphertext with certain ℓ -dimensional prefix where $\ell \leq n$. Note that in this setting the space of the possible challenges has dimension 2^ℓ and the inversion result is a point in the space of dimension 2^n , implying that the solution is not unique. Note that, in an alternative setting it is possible to consider inversion problem over a subspace of the (constrained) inversion results where dimensions of the both spaces are the same and equal to 2^ℓ - this setting is considered in the next section.

For simplicity of explanation, we assume that each (elementary) node operates using a single table for any hardness level of the puzzle. We also assume that the so called federated nodes could appear and that they consist of multiple elementary nodes, and each of the elementary nodes employs a single table.

The main notations employed in this article are summarized in Table 1.

A. PREPARATION PHASE

The preparation phase should be performed only once and in advance, before a node begins participation in the consensus process. The output of this phase is a two-columns table which is constructed in a dedicated manner (to be described later on) and appears as a tool for an efficient attempt for solving the blockchain puzzle during the consensus protocol execution. This table contains, in an implicit way, a lot of the pairs (ciphertext, secret key). The dedicated construction of the table is based on a generic approach for inversion of one-way functions employing a time-memory trade-off approach

TABLE 1. Summary of the main notations.

K	secret key
$N = 2^m$	space of secret keys
$E_K(\cdot)$	encryption algorithm
1^m	all ones m -dimensional vector
$H(\cdot)$	hash function
d	dimension of all zeros prefix of the hash value
\mathcal{J}	set of the parameters which specify the inversion hardness
$[a b]$	concatenation of vectors a and b
ℓ^j	dimension of a suffix corresponding to the parameter $j \in \mathcal{J}$ of n -dimensional binary vector
\mathbf{M}_i^j	memory employed at the node $i, j \in \mathcal{J}$
M_i^j	dimension of memory at the node $i, j \in \mathcal{J}$
t_i^j	the processing time at the node $i, j \in \mathcal{J}$
C	the inversion challenge; a binary vector of dimension ℓ^j
D_i^j	number of the inversion challenges employed
δ	time for generating the inversion challenge
Δ	time slot between two blockchain updates
V_i	i -th verification node
\mathcal{V}_H	number of the honest nodes
\mathcal{V}_M	number of the malicious nodes
P_i^j	the probability that a node V_i solves the puzzle when $j \in \mathcal{J}$

Algorithm 1 Initialization of the Two-Columns Table \mathbf{M}_i

- 1) *Input:* Parameters n, t_i , and M_i (number of the rows in the table).
- 2) Randomly select M_i different n -dimensional vectors X_0 and save them as the first column elements of the table \mathbf{M}_i .
- 3) For each row of the table \mathbf{M}_i :
 - perform the following recursive evaluation t_i times

$$X_t = E_{X_{t-1}}(1^n), \quad t = 1, \dots, t_i.$$
 - Memorize the element X_{t_i} as the second column element of the table \mathbf{M}_i .
- 4) *Output:* Two-columns table \mathbf{M}_i .

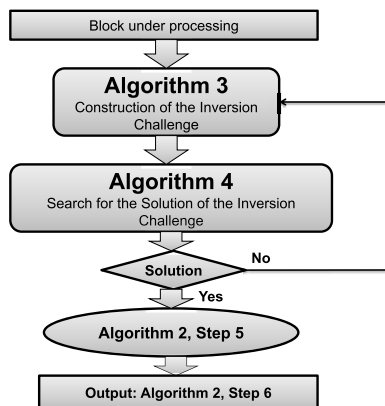
described in Section II. In the basic approach only one table \mathbf{M}_i is required for any difficulty parameter $j \in \mathcal{J}$ at any node i , i.e. $\mathbf{M}_i^j = \mathbf{M}_i$ assuming appropriate selection of the table parameters. The goal of the preparation phase is to generate certain two columns table \mathbf{M}_i with M_i rows employing the following Algorithm 1.

B. THE CONSENSUS PROTOCOL

Proposal of the novel protocol for inclusion of a new block into the blockchain follows the traditional paradigm which involves construction of a candidate block and solving certain puzzle in order to follow the consensus achieving approach. The final verification of the blockchain updates assumes that the longest extension of the previously verified blockchain is accepted as the verified extension for the further updates. The protocol we propose contains an alternative puzzle which should be performed over each candidate block of the updates in order to achieve the verification consensus. Each node, which generates candidate blocks for inclusion into the blockchain, should execute the following Algorithm 2.

Algorithm 2 Procedure for Inclusion of a New Block Into the Blockchain

- 1) *Input*: New block of the transactions.
- 2) Construction of the inversion challenge for considered block of transactions.
- 3) Searching for solution of the given inversion challenge; If the solution has been found go to the Step 5.
- 4) If execution of the Step 3 does not provide a solution for the inversion, and if processing of the considered block is still relevant go to Step 2; otherwise go to Step 6 (b).
- 5) Check the position for inclusion of the considered block into the blockchain after the last already included valid block if no one transaction of the block has already been included in the blockchain; the validity of the predecessor includes correctness check of the inversion problem solution, as well.
- 6) *Output*: (a) New block inclusion into the blockchain; (b) the failure flag that the block cannot be included into the blockchain.

**FIGURE 6.** Flowchart of Algorithm 2.

Two core components of the above algorithm for inclusion of a new block into the blockchain are: (i) Step 2) of the algorithm which specify construction of a challenge for the inversion; and (ii) Step 3) of the algorithm for the inversion search. A flowchart of Algorithm 2 is displayed in Fig. 6, and proposals of the Algorithms 3 and 4 are given below.

A flowchart of Algorithm 4 main components is displayed in Fig. 7.

IV. THE CONSENSUS PROTOCOL BASED ON THE CONSTRAINED INVERSION PROBLEM

This section points out to an alternative version of the previously proposed consensus protocol which corresponds to the setting where the inversion solutions should be from certain subspace of the entire space of possible solutions. This setting appears as important one because it provides an additional dimension for controlling the puzzle difficulty.

Algorithm 3 Construction of the Inversion Challenge

- *Input*: The considered block of transactions, and the parameters d, ℓ^j
- *Construction of the challenge*:
 - 1) Randomly select a nonce;
 - 2) Evaluate the hash value of the binary representation of the considered block concatenated with a nonce;
 - 3) If the evaluated hash value has d -zeros prefix select its ℓ^j suffix bits as the inversion challenge and go to the Output; otherwise go to the Step 1.
- *Output*: Employed nonce and the inversion challenge C .

Algorithm 4 Search for a Solution of the Inversion Challenge at the Node i for Given Difficulty Parameter j

- *Input*: The challenge C for inversion, the difficulty parameter ℓ^j , the table \mathbf{M}_i , and the parameter t_i .
- *Search for the Challenge Inversion*:
 - 1) Check whether the challenge C is equal to ℓ^j -dimensional suffix of certain second column element of the table \mathbf{M}_i ; If “yes” record the row where the overlapping has appeared and go to Step 5; otherwise go to Step 3;
 - 2) Set $t = 0$ and $X_0 = 1^{n-\ell^j} || C$, i.e., X_0 is a concatenation of $n - \ell^j$ ones and the challenge C ;
 - 3) Set $t = t + 1$; if $t > t_i$ go to the Output (b); Otherwise go to the Step 4;
 - 4) Evaluate

$$X_t = E_{X_{t-1}}(1^n)$$

and check whether X_t is equal to certain second column element of the table \mathbf{M}_i ; If “yes” record the row index where the overlapping has appeared and go to Step 5; otherwise go to Step 3;

- 5) Set X_0 to the first column element corresponding to the recorded row index, and perform iterative evaluation $X_t = E_{X_{t-1}}(1^n)$ until X_t has the suffix C , and go to the Output (a);
- *Output*: (a) Set the inversion result as X_{t-1} ; (b) the inversion has not been performed.

In this case, the main inputs are: (i) $N = 2^n$ - number of the points in the space of all possible solutions of the inversion problem; (ii) \mathcal{J} - set of the parameters which specify the inversion problem hardness, i.e. the puzzle difficulty. For each level of the hardness/difficulty $j \in \mathcal{J}$, a node with the index i selects the following two appropriate (according to the node’s criteria) space and energy budgets: a memory of dimension M_i^j and the processing time parameter t_i^j . Note that, in a general case, a node could select, for certain difficulty levels, that the memory and time budgets are equal to zero. Accordingly, this section proposes the alternative versions of

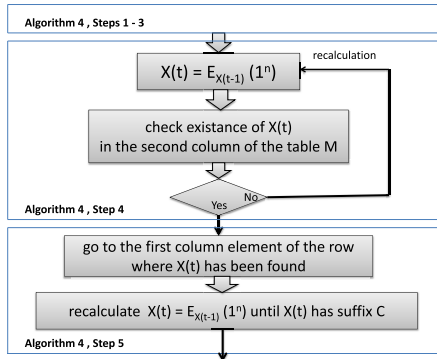


FIGURE 7. Flowchart of Algorithm 4 main components.

Algorithm 1a Initialization of the Two-Columns Table M_i^j

- 1) *Input*: Parameters n, j, ℓ^j, t_i^j , and M_i^j (number of the rows in the table M_i^j).
- 2) Randomly select M_i^j different n -dimensional vectors X_0 ; preset the first $n - \ell^j$ bits of each vector to all ones, and save them as the first column elements of the table M_i^j .
- 3) For each row of the table M_i^j :
 - for $t = 1, \dots, t_i^j$ perform the following:
 - if $t = 1$ go to the next (evaluate) step, otherwise preset the first $n - \ell^j$ bits of X_{t-1} to 1 and go to the next step;
 - evaluate
$$X_t = E_{X_{t-1}}(1^n).$$
 - Memorize the element $X_{t_i^j}$ as the second column element of the table M_i^j .
- 4) *Output*: Two-columns table M_i^j .

the Algorithms 1 and 4 which correspond to the consensus protocol based on the constrained inversion problem.

Recall that each node i should execute Algorithm 1a for all selected values of the difficulty parameter $j \in \mathcal{J}$.

Note that the obtained inversion result should begin with $1^{n-\ell^j}$ prefix, i.e., its prefix should contain a run of all ones of length which corresponds to the considered difficulty.

V. SECURITY EVALUATION

Let V_i denotes i -th verification node and let \mathcal{V} be the set of all verification nodes. For the security evaluation, we assume that the set \mathcal{V} consists of two non-overlapping sets: \mathcal{V}_H and \mathcal{V}_M corresponding to the honest and malicious nodes, respectively. Also, let the chain is updated after each Δ -time slot. In order to be the node which provides the next block of the blockchain, a node should fulfill the following: (i) face a puzzle solvable by the table in its possession; (ii) be the first of solving the puzzle for a block with transactions which

Algorithm 4a Search for a Solution of the Inversion Challenge at the Node i for the Given Difficulty Parameter j

- *Input*: The challenge C for inversion, the table M_i^j , and the parameter t_i^j (corresponding to the difficulty parameter j).
- *Search for the Challenge Inversion*:
 - 1) Check whether the challenge C is equal to ℓ^j -dimensional suffix of certain second column element of the table M_i^j ; If “yes” record the row where the overlapping has appeared and go to Step 5; otherwise go to Step 3;
 - 2) Set $t = 0$ and $X_0 = 1^{n-\ell^j} || C$, i.e., X_0 is a concatenation of $n - \ell^j$ ones and the challenge C ;
 - 3) Set $t = t + 1$; if $t > t_i^j$ go to the Output (b); Otherwise go to the Step 4;
 - 4) for $t = 1, \dots, t_i^j$ perform the following:
 - if $t = 1$ go to the next step (evaluate); otherwise preset the first $n - \ell^j$ bits of X_{t-1} to 1, and go to the next step;
 - evaluate

$$X_t = E_{X_{t-1}}(1^n),$$

and check whether X_t is equal to certain second column element of the table M_i^j ; If “yes” record the row index where the overlapping has appeared and go to Step 5; otherwise go to Step 3;

- 5) Set X_0 to the first column element corresponding to the recorded row index, and perform iterative evaluation $X_t = E_{X_{t-1}}(1^n)$, with the presetting as in the Step 4), until X_t has the suffix C , and go to the Output (a);
- *Output*: (a) Set the inversion result as X_{t-1} ; (b) the inversion has not been performed.

are not included into the blockchain. In order to be secure, the consensus protocol should be such that the chain version generated by the nodes from \mathcal{V}_H is always longer than the chain generated by the nodes from the set \mathcal{V}_M .

For the security evaluation, without losing a generality, we assume that each node is an elementary node which employs a single table: In this model, a system node which employs multiple tables could be considered as a consortium of elementary nodes. Also, we assume that each table is generated according to the requirements given in [6] and [2], for example, so that we expect that all elements of the table in the visible two columns, as well as in the hidden ones are different. Let $P_i^j, i \in \mathcal{V}$, be the probability that a node could solve the puzzle, and let P_H^j and P_M^j denote the probabilities that a node from the sets \mathcal{V}_H and \mathcal{V}_M , respectively, could solve the puzzle of given difficulty $j \in \mathcal{J}$.

Lemma 1: Assuming random and mutually independent constructions of the tables for TM-TO corresponding to the

given difficulty $j \in \mathcal{J}$, we have the following:

$$P_i^j = \frac{D_i^j M_i^j t_i^j}{N}, \quad i \in \mathcal{V}, \quad (1)$$

$$P_J^j = \sum_{i \in \mathcal{V}_J} \frac{D_i^j M_i^j t_i^j}{N}, \quad J = H, M, \quad (2)$$

where N is equal to 2^{ℓ^j} and 2^n in the unconstrained and constrained cases, respectively, and D_i^j is the number of employed challenges, i.e. number of the returns from Step 4 to Step 2 during an execution of the Algorithm 2.

Proof: Assuming the random model, we could expect that $2^{n-\ell^j}$ arguments X fulfils $E_X(1^n) = C$ where we have mapping $\{0, 1\}^n \rightarrow \{0, 1\}^{\ell^j}$. Accordingly, the probability that inversion of the given C can be performed by the given table is equal to $\frac{M_i^j t_i^j}{2^{\ell^j}}$. On the other hand, when we consider the constrained inversion where the first $n - \ell^j$ bits of the inversion result should be ones, the probability of successful inversion reduces for the factor $2^{-n+\ell^j}$, i.e. the probability of success becomes $\frac{M_i^j t_i^j}{2^n}$. Finally, when there are D_i^j attempts for the inversion employing D_i^j different challenges, the both probabilities increase for the factor D_i^j . Q.E.D.

According to Algorithm 4/4a, note that a node V_i for given difficulty $j \in \mathcal{J}$ requires time t_i^j for each attempt to perform the requested inversion, and we assume that $t_{min} \leq t_i^j \leq t_{max}$, $V_i \in \mathcal{V}_J$, $J = H, M$. Also, we assume that δ denotes the time required for generation of a challenge in the step 2 of Algorithm 2.

Theorem 1: Assuming that all nodes performs the elementary operations with the same time complexity, a conservative security condition which provides security of the blockchain consensus protocol requires the following: For each value of the difficulty parameter $j \in \mathcal{J}$,

$$\lfloor \frac{\Delta}{\delta + t_{max}^j} \rfloor \sum_{i \in \mathcal{V}_H} M_i^j t_i^j > O(\lfloor \frac{\Delta}{\delta + t_{min}^j} \rfloor \sum_{i \in \mathcal{V}_M} M_i^j t_i^j). \quad (3)$$

Proof: For each $j \in \mathcal{J}$, we have:

$$D_i^j = \lfloor \frac{\Delta}{\delta + t_i^j} \rfloor, \quad i \in \mathcal{V} \quad (4)$$

and when $t_{min}^j \leq t_i^j \leq t_{max}^j$, $t_i^j \in \mathcal{V}_J$, $J = H, M$, for each i and j ,

$$\frac{\Delta}{\delta + t_{max}^j} \leq D_i^j \leq \frac{\Delta}{\delta + t_{min}^j} \quad (5)$$

Consequently, within the time period Δ , each node from \mathcal{V}_H could consider the puzzle solving for inclusion a block into the blockchain at least $\lfloor \frac{\Delta}{\delta + t_{max}^j} \rfloor$ times, and each node from \mathcal{V}_M could consider the puzzle solving for inclusion a block into the blockchain at most $\lfloor \frac{\Delta}{\delta + t_{min}^j} \rfloor$ times. Accordingly, Lema 1

TABLE 2. Time and space complexities of the proposed consensus protocol at an elementary node i for a given difficulty j .

	time complexity	space complexity
pre-processing complexity	$O(M_i^j \cdot t_i^j)$	$O(M_i^j)$
processing complexity	$O(\frac{\Delta}{\delta + t_i^j} \cdot t_i^j)$	$O(M_i^j)$

TABLE 3. Comparison of the execution complexities at a node assuming that the expected probability of success is P and that the complexity of solving the puzzle corresponds to N , and M is a parameter $M \ll N$.

	time complexity	space complexity
PoW	$O(N \cdot P)$	$O(1)$
PoM	$O(1)$	$O(N \cdot P)$
proposed approach	$O(\frac{N}{M} P)$	$O(M)$

implies that the expected number of the new blocks included into the blockchain by the nodes from \mathcal{V}_H is at least:

$$\lfloor \frac{\Delta}{\delta + t_{max}^j} \rfloor \sum_{i \in \mathcal{V}_H} \frac{M_i^j t_i^j}{N}, \quad j \in \mathcal{J}. \quad (6)$$

Also, Lema 1 implies that the expected number of the new blocks included into the blockchain by the nodes from \mathcal{V}_M is at most:

$$\lfloor \frac{\Delta}{\delta + t_{min}^j} \rfloor \sum_{i \in \mathcal{V}_M} \frac{M_i^j t_i^j}{N}, \quad j \in \mathcal{J}. \quad (7)$$

Consequently, a conservative condition that the number of new blocks added to the chain by the nodes from the set \mathcal{V}_H is always greater than the number of the new blocks added by the nodes from the set \mathcal{V}_M can be specified as:

$$\lfloor \frac{\Delta}{\delta + t_{max}^j} \rfloor \sum_{i \in \mathcal{V}_H} \frac{M_i^j t_i^j}{N} \gg \lfloor \frac{\Delta}{\delta + t_{min}^j} \rfloor \sum_{i \in \mathcal{V}_M} \frac{M_i^j t_i^j}{N}, \quad (8)$$

for each $j \in \mathcal{J}$. The above non-equality can be directly rewritten obtaining a form claimed in the theorem statement. QED.

The above discussion of the security shows resistance of a consensus protocol against misbehavior of a number of nodes which participate into the consensus protocol, and this resistance is also called fault tolerance of a consensus protocol. For example, under conventional assumptions Bitcoin blockchain is fault tolerant if honest participants have a majority of hash-power which provides higher probability that a honest player solve the hash-puzzle in comparison with a malicious one.

VI. IMPLEMENTATION ISSUES

A. IMPLEMENTATION COMPLEXITY

Implementation complexity of the proposed protocol consists of the following two components: (i) the pre-processing

TABLE 4. Comparison of the execution complexities at a node assuming that the the expected probability of success is > 0.99 and that the complexity of solving the puzzle corresponds to finding the solution within the space of dimension 2^{30} . Performances have been considered when Ethereum with traditional PoW and the modified one, where PoW is replaced with the approach proposed in Section III, are employed and when the parameter d of Algorithm 3 takes values 10 and 15 (reported in [12]).

	PoW	proposed approach the parameter $d = 10$	proposed approach the parameter $d = 15$
average time execution (in seconds)	3067.153	0.132	15.224
minimum time execution (in seconds)	123.473	0.001	0.106
maximum time execution (in seconds)	7782.805	1.554	82.401
required dedicated memory (in 32-bit words)	/ (negligible)	2^{24}	2^{21}

complexity which is required for the protocol initial setting; and (ii) the processing complexity corresponding to each execution of the protocol.

The pre-processing complexity is complexity of generation the tables employing Algorithm 1/1a, and it is just one-time overhead.

The protocol execution (processing) complexity is determined by time and space complexities of the processing for generation of a candidate block employing Algorithms 2-4/4a. This processing consists of the complexity of mini-PoW and the complexity of solving the inversion problem.

The implementation and execution complexities of the proposed consensus protocol at a node are summarized in the following table.

Consequently, the cumulative energy and space complexities of the protocol within the entire system for adding one block into the blockchain are as follows:

$$O\left(\sum_i \frac{\Delta}{\delta + t_i^j} \cdot t_i^j\right)$$

$$O\left(\sum_j \sum_i M_i^j\right)$$

respectively, where the summation is over all elementary nodes i in the system, and the difficulties $j \in \mathcal{J}$.

Beside the complexity, scalability, throughput per second (TPS), and block-size are important features of a blockchain system. On the other hand, scalability and block-size usually are not affected by the employed puzzle, but TPS could strongly depend on the puzzle. For example, PoW and PoM have good scalability, but TPS of Bitcoin blockchain with PoW is low, which greatly limits the application of PoW. The consensus puzzle proposed in this article is such that it provides good scalability, does not imply limitations on the block-size and could provide appropriate TPS, but in-details discussion of these issues is out of the scope of this article.

```
func generate(w, h uint64, b, tct uint) *tableSet {
    if tct==0 { tct = uint(math.Pow(2,math.Ceil(math.Log2(float64(runtime.NumCPU)))) ) }
    h /= uint64(tct)
    tset := &tableSet {
        width:    w,
        height:   h,
        block_size: b,
        tblct:    tct,
        table_set: make([]table, tct),
    }
    sad = time.Now().Unix()
    var wg sync.WaitGroup
    wg.Add(int(tset.tblct))
    patterns = generatePatterns(tset.tblct, tset.block_size)
    for i:uint(0); i<tset.tblct; i++ {
        go tset.buildMiniTable(i, patterns[i], &wg)
    }
    wg.Wait()
    return tset
}

func(t *tableSet) buildMiniTable(ti uint, pattern uint32, wg *sync.WaitGroup) {
    defer wg.Done()
    t.table_set[ti].pattern = pattern
    dc := tdes.New(t.block_size, pattern)
    randGen := randNew(rand.NewSource(time.Now().UnixNano()))
    for i:uint64(0); i>t.height; i++ {
        first := dc.MakeKey(dc.EncryptFixedPlaintext(k))
    }
    last := dc.EncryptFixedPlaintext(k)
    t.table_set[ti].tbl[last >> uint(64-slice_ind_len)] = append(
        t.table_set[ti].tbl[last >> uint(64-slice_ind_len)], tblrow(first: first, last: last)
    )
}
}
```

FIGURE 8. Implementation, [12], of Algorithm 1 in Go language.

B. IMPLEMENTATION IN GO LANGUAGE

The proposed consensus approach has been implemented and embedded into Ethereum platform as an alternative consensus protocol. The implementation has been performed employing Go (or Golang) programming language (<https://golang.org>) which is also employed in Ethereum platform (see <https://github.com/ethereum/go-ethereum> for official Golang implementation of the Ethereum protocol). These implementations are reported in [12] and the illustrative codes are given in the Appendix. Experimental evaluation has been performed employing the system with processor Intel Core i7-4710MQ CPU at 2.50GHz with 8 CPU cores and 16 GB of RAM. For the illustrative purposes, as the encryption in the consensus puzzle, well known DES block cipher is employed.

```

func (t *Tmto) verifySeal(chain consensus.ChainReader, header *types.Header) error {
    if header.Difficulty.Sign() <= 0 {
        return errInvalidDifficulty
    }
    var {
        hash      = t.SealHash(header).Bytes()
        difficulty = uint(header.Difficulty.Uint64())
        extra     = header.Extra
    }
    he_sum := Keccak256(hash,extra)
    half_difficulty := uint(math.Ceil(float64(difficulty)/2))
    if !checkMiniPoW(he_sum, half_difficulty) {
        return errInvalidMiniPoW
    }
    tmto_challenge := binary.BigEndian.Uint64(he_sum[24:32])
    tmto_solution := header.Nonce.Uint64()
    des_cipher := tdes.New(tmto_block_size,0)
    check := des_cipher.EncryptFixedPlaintext(tmto_solution)
    mask := ^(uint64(1) << uint(64-difficulty) - 1)
    if (check & mask) != (tmto_challenge & mask) {
        return errInvalidTMTOsolution
    }
    Return nil
}

```

FIGURE 9. Implementation, [12], of Algorithm 2, Step 5, in Go language.

```

func miniPoW(hash []byte, difficulty uint, res chan []byte, stop <-chan struct{}) {
    randGen := rand.New(rand.NewSource(time.Now().UnixNano()))
    extra := make([]byte, 8)
    for {
        select {
        case <-stop:
            return
        default:
            randGen.Read(extra)
            sum := sumKeccak256(hash,extra)
            if checkMiniPoW(sum[:8], difficulty) {
                res <-extra
                return
            }
        }
    }
}
func checkMiniPoW(hash []byte, difficulty uint) bool {
    sum := binary.BigEndian.Uint64(hash)
    mask := ^(uint64(1) << uint(64-difficulty) - 1)
    return 0 == (sum & mask)
}

```

FIGURE 10. Implementation, [12], of Algorithm 3 in Go language.

C. COMPARISON

For the comparison, we assume that all the considered puzzles are related to a problem with a solution within a space of N (different) points. We consider a single node and our goal is to compare time and space complexities of PoW, PoM and the consensus approach proposed in this article assuming that all three approaches have the same probability P of success for solving the corresponding puzzles.

Note that Algorithm 4 implies that the time complexity of solving the puzzle is Dt where D is number of different challenges employed and t corresponds to the time employed for solving the puzzle. On the other hand, according to Lemma 1, the time complexity of the approach proposed in this article is: $Dt = \frac{N \cdot P}{M}$ where M corresponds to the space complexity of the proposed approach.

According to the table, for example, when $M = 2^{30}$:

- the energy consumption is reduced about 2^{30} times in comparison with PoW requirement at the expense of employing a memory $O(2^{30})$;

```

func (t *table) search(h uint64, d, b uint, width uint64, res chan Search_result, stop <-chan struct{}) {
    dc := tdes.New(b, t.pattern)
    h_orig := h
    f.key := t.searchLastCol(h,d)
    if f {
        mask := ^(uint64(1) << uint(64-d) - 1)
        search := h_orig & mask
        for i:=uint64(0); i<=width; i++ {
            c := dc.EncryptFixedPlaintext(key)
            if search == (c&mask) {
                res <- Search_result{Found: true, Key: key}
                return
            }
        }
    }
    for i:=uint64(0); i<=width; i++ {
        select {
        case <- stop:
            return
        default:
            h = dc.EncryptFixedPlaintext(dc.MakeKey(h))
            f.key := t.searchLastCol64(h)
            if f {
                mask := ^(uint64(i) << uint(64-d) - 1)
                search := h_orig & mask
                for j:=uint64(0); j<=width; j++ {
                    c := dc.EncryptFixedPlaintext(key)
                    if search == (c&mask) {
                        res <- Search_result{Found: true, Key: key}
                        return
                    }
                }
                key = dc.MakeKey(c)
            }
        }
    }
    res <- Search_result{Found: false}
}

```

FIGURE 11. Implementation, [12], of Algorithm 4 in Go language.

- the required space is reduced for a factor $\sim \frac{N \cdot P}{2^{30}}$ times in comparison with PoM requirement at the expense of increase of the required energy for the same factor.

Suitable experiments have been performed with the developed modified Ethereum platform where PoW consensus protocol has been replaced with the one proposed in Section III of this article. Performances have been considered when Ethereum employing traditional PoW and the modified consensus protocol were running at the same node. According to the experimental evaluation results, illustrative numerical findings are displayed in Table 4 where a comparison of the execution complexities is given at a node assuming that: (i) the expected probability of success is >0.99 , and (ii) the complexity of solving the puzzle corresponds to finding a solution within the space of dimension 2^{30} .

VII. CONCLUDING NOTES

The proposed consensus protocol provides a flexibility for selection of the energy and space resources which should be employed by a participating entity in the process of verification of the blockchain updates. Security and complexity of the protocol could be controlled by a number of the parameters: the processing time, dimension of the memory and the difficulty of the considered puzzle, i.e. hardness of the inversion problem. These parameters should be adjusted to the particular scenario where the consensus protocol is employed. Accordingly, a player of the consensus protocol (a miner) could adjust the (mining) strategy in order to fit the expenses into a desired budget.

An interesting issue for the further work could be a consideration of the time vs. memory issues employing [1] as a background.

APPENDIX IMPLEMENTATION OF THE ALGORITHMS IN GO LANGUAGE

This section shows implementations reported in [12] of the core components of the consensus protocol, given in Section III, in Go language which provides easy inclusion of the novel puzzle into Ethereum platform. Fig. 8 displays implementation of Algorithm 1. Fig. 9 displays implementation of Algorithm 2, Step 5, where the validity of the predecessor of the new block includes check of the inversion problem solution correctness, as well. Note that according to the flowchart displayed in Fig. 6, Algorithm 2 employs Algorithms 3 and 4, so that the remaining core part of the algorithm is the procedure in Step 5, and so Fig. 9 displays only implementation of this step. Fig. 10 displays implementation of Algorithm 3, and Fig. 11 displays implementation of Algorithm 4.

REFERENCES

- [1] H. Abusalah, J. Alwen, B. Cohen, D. Khilko, K. Pietrzak, and L. Reyzin, "Beyond Hellman's time-memory trade-offs with applications to proofs of space," in *Proc. ASIACRYPT*, in Lecture Notes in Computer Science, vol. 10625, 2017, pp. 357–379.
- [2] A. Biryukov and A. Shamir, "Cryptanalytic time/memory/data tradeoffs for stream ciphers," in *Proc. ASIACRYPT*, in Lecture Notes in Computer Science, vol. 1976, 2000, pp. 1–13.
- [3] T. Dryja, Q. C. Liu, and S. Park, "Static-memory-hard functions, and modeling the cost of space vs. time," in *Proc. TCC*, Lecture Notes in Computer Science, vol. 11239, 2018, pp. 33–66.
- [4] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Proc. CRYPTO*, in Lecture Notes in Computer Science, vol. 740, Heidelberg, Germany: Springer, 1993, pp. 139–147.
- [5] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Proc. CRYPTO*, in Lecture Notes in Computer Science, vol. 9216, 2015, pp. 585–605.
- [6] M. Hellman, "A cryptanalytic time-memory trade-off," *IEEE Trans. Inf. Theory*, vol. IT-26, no. 4, pp. 401–406, Jul. 1980.
- [7] L. Ismail and H. Materwala, "A review of blockchain architecture and consensus protocols: Use cases, challenges, and solutions," *Symmetry*, vol. 11, no. 10, p. 1108, 2019.
- [8] M. Mihaljevic, M. Fossorier, and H. Imai, "Security evaluation of certain broadcast encryption schemes employing a generalized time-memory-data trade-off," *IEEE Commun. Lett.*, vol. 11, no. 12, pp. 988–990, Dec. 2007.
- [9] M. J. Mihaljevic, S. Gangopadhyay, G. Paul, and H. Imai, "Internal state recovery of grain-v1 employing normality order of the filter function," *IET Inf. Secur.*, vol. 6, no. 2, pp. 55–64, Jun. 2012.
- [10] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
- [11] S. Park, A. Kwon, G. Fuchsbaauer, P. Gazi, J. Alwen, and K. Pietrzak, "SpaceMint: A cryptocurrency based on proofs of space," in *Financial Cryptography and Data Security (Lecture Notes in Computer Science)*, vol. 10957, Berlin, Germany: Springer, 2018, pp. 480–499.
- [12] M. Savić, M. Todorović, and M. J. Mihaljević, "Implementation of a novel blockchain consensus protocol and a modification of Ethereum platform," Math. Inst. Serbian Acad. Sci. Arts, Belgrade, Serbia, (in Serbian), Tech. Rep., Dec. 2019.
- [13] A. Shahaab, B. Lidgley, C. Hewage, and I. Khan, "Applicability and appropriateness of distributed ledgers consensus protocols in public and private sectors: A systematic review," *IEEE Access*, vol. 7, pp. 43622–43636, 2019.
- [14] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, "A survey on consensus mechanisms and mining strategy management in blockchain networks," *IEEE Access*, vol. 7, pp. 22328–22370, 2019.
- [15] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1432–1465, 2nd Quart., 2020.



MIODRAG J. MIHALJEVIĆ is currently a Research Professor and the Project Leader with the Mathematical Institute, Serbian Academy of Sciences and Arts, Belgrade. His main research interests include cryptology and information security. He has published more than 100 research articles in the leading international journals and conference proceedings and over 200 publications in total. He is co-inventor of eight granted patents in Japan, U.S., and China. He has participated in over ten international research projects. Since 1997, he has been holding long-term visiting positions at the universities and research institutes in Japan, including The University of Tokyo, Sony Research Labs, the National Institute AIST, and Chuo University, Tokyo. Since 2014, he has been an Elected Member of the Academia Europaea. In 2013, he received the National Award of the Serbian Academy of Sciences and Arts for ten years achievements. He has been a Guest Editor of few journals. He is an Associate Editor of *SN Computer Science* (Springer). For more information, please visit <http://www.mi.sanu.ac.rs/cv/cvmihaljevic.htm>.

• • •