symmetry

Article

An Evaluation of Power Consumption Gain and Security of Flexible Green Pool Mining in Public Blockchain Systems

Miodrag J. Mihaljević [1], Milan Todorović [1,2] and Milica Knežević [1,2,*]

[1] Mathematical Institute, Serbian Academy of Sciences and Arts, Kneza Mihaila 36, 11000 Belgrade, Serbia; miodragm@turing.mi.sanu.ac.rs (M.J.M.); mtodorovic@mi.sanu.ac.rs (M.T.)
[2] Faculty of Technical Sciences, University of Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia
* Correspondence: mknezevic@mi.sanu.ac.rs

Abstract: This paper proposes a variant of the recently reported pool mining approach and provides a reduction in the energy that is consumed by the blockchain consensus protocol. The novelty of the proposed architecture lies in the employment of an innovative cryptographic puzzle that is based on stream ciphering. This enables flexibility in setting the difficulty parameter of the protocol, and allows for the separation of the energy and memory resources that are required for the puzzle solving. The proposed approach provides high resistance against the following malicious activities of miners in public blockchain systems: (i) the submission of fake work and fictitious computation results; and (ii) some well-known attacks that target the blockchain incentive mechanism. We experimentally evaluate the power consumption of the proposed consensus protocol and compare it with the traditional proof-of-work protocol based on hashing. The obtained results point out the gain that the proposed pool mining provides compared with the traditional types.

Keywords: blockchain; consensus protocol; pool mining; energy reduction; security evaluation; selfish mining; block withholding attack

check for updates

Citation: Mihaljević, M.J.; Todorović, M.; Knežević, M. An Evaluation of Power Consumption Gain and Security of Flexible Green Pool Mining in Public Blockchain Systems. *Symmetry* **2023**, *15*, 924. https://doi.org/10.3390/sym15040924

Academic Editor: José Carlos R. Alcantud

Received: 28 February 2023
Revised: 11 April 2023
Accepted: 13 April 2023
Published: 16 April 2023

1. Introduction

Pool mining represents a predominant mode of miners' participation in blockchain systems, especially in public blockchains that use proof-of-work-based consensus protocols. Miners primarily join mining pools in order to ensure a more stable income compared with solo mining, where the rewards are earned sporadically and irregularly. The mining process can be seen as a type of lottery, where the probability of winning as individual miners with small computational power is almost negligible. More precisely, it would take an extremely long time for an individual miner to solve a block and win the reward. Mining within a mining pool provides a solution for this problem and enables a steadier income for miners, rewarding them proportionally for the work that they invested in finding a block. On the other hand, pool mining introduces new issues related to reward distribution and miners' incentives. Two well-known attacks related to mining in pools are block withholding [1] and selfish mining [2]. There are many variants and extensions of these attacks, and a significant amount of research addresses the problem of block withholding and selfish mining, proposes prevention techniques, and analyzes the impacts that the attacks have in blockchain networks.

The consensus protocol in [3] uses energy and memory resources, and allows for a trade-off between them. It is similar to traditional PoW protocols but has differences in the cryptographic puzzle and solution-finding technique. Recently, a "green" pool mining approach based on this protocol was proposed in [4], which reduces energy consumption. The reported architecture uses pseudo-symmetric, two-dimensional energy-memory allocation for executing the consensus protocol, which provides resistance against certain attacks launched against pool mining approaches as well as a reduction in energy spending.

Instead of separating the blockchain consensus protocol and monitoring the integrity of pool miners' work, this method utilizes a dedicated application of the consensus protocol to guarantee the honesty of both miners and pool managers.

Motivation for the Work. It is interesting to extend the flexibility of the pool mining proposed in [4], in particular regarding the flexible setting of the consensus puzzle difficulty. Furthermore, in addition to honest miners, some malicious miners can join public pools. Thus, an interesting open issue is to consider advanced pool mining architectures as providing higher resistance against malicious actions. Specifically, it is interesting to address blockchain consensus protocols that can straightforwardly support the detection of deliberate incorrect miners' work and be highly secure against attacks such as block withholding and selfish mining in a pool mining setting.

Summary of the Results. This paper proposes a variant of the pool mining approach recently reported in [4] and an evaluation of the security and power consumption. The architecture employs a cryptographic puzzle based on stream ciphering; it separates the energy and memory resources that are necessary for finding the puzzle's solution, and gives the pool manager control over the memory resources, i.e., tables, that enables the substantial efficiency of the blockchain puzzle-solving process. We discuss the security of the proposed approach against the following malicious activities in public blockchains: (i) fake work of the miners and spamming with fictitious computation results where, instead of the evaluated data, miners provide the pool manager with random values; and (ii) certain attacks targeting the consensus protocol layer and incentive mechanism in public blockchain pool mining. The correctness control of miners' work is based on a random sample checking. We consider three different attack scenarios: (i) attack by a malicious miner without memory resources; (ii) attack by a malicious miner with certain memory resources; and (iii) attack by a group of malicious miners that share their hidden memory resources. The experimental evaluation shows a significant reduction in power consumption when compared with the standard PoW hash-based consensus protocol.

Organization of the Paper. The background and related works are presented in Section 2. A variant of the previously reported pool mining approach is proposed in Section 3. Certain security issues of the proposed pool mining are discussed in Sections 4 and 5. An experimental evaluation of the considered pool mining and the gain achievable in comparison with a traditional method based on hashing PoW is given in Section 6 and Appendix A. Discussion of the experimental results is conducted in Section 7. Finally, the conclusions are given in Section 8.

2. Background and Related Work

The main goal of this paper is to propose a pool mining approach that provides a reduction in energy that the blockchain consensus protocol requires and high resistance against the security treats that are related to the pool mining. Accordingly, this section provides an overview of: (i) pool mining; (ii) the related work regarding certain techniques for energy consumption reduction or meaningful spending, i.e., green blockchain consensus protocols; and (iii) blockchain attacks and corresponding countermeasures, with a focus on the attacks targeting the incentive mechanism.

2.1. Pool Mining

Pool miners are rewarded in accordance with their contribution to the probability of finding a block, i.e., solving the blockchain cryptographic puzzle. More precisely, the pool manager rewards the miners based on the work they completed in solving the sub-puzzles, which present a valid partial proof-of-work. Because the difficulty of a sub-puzzle is significantly lower than the challenging puzzle, the miners can solve the sub-puzzles with a reasonable amount of work. A solution of the sub-puzzle may as well be a solution of the blockchain puzzle, but not necessarily. If the solution of the sub-puzzle also satisfies the difficulty of the current blockchain puzzle, it is a complete solution; otherwise, it is a partial solution. It should be noted that the pool receives a reward if and only if it finds a complete solution. If none of the partial solutions found by the pool miners is a complete solution, the pool does not get the reward, but depending on the employed reward system, the miners might still receive compensation for their work (see pay-per-share [5]). A realistic scenario, especially for public blockchains, is that a pool, aside from honest miners, also contains dishonest ones. Dishonest mining covers any malicious activity directed against mining pools or competitors. The main goal of dishonest miners is usually to increase their reward and gain more revenue than their fair share. For example, a dishonest miner could keep any complete solution he finds and report to the pool only partial solutions. This way, a dishonest miner still shares the reward won by the entire pool without any relevant contribution.

One of the first papers analyzing the (dis)advantages and problems of the pool mining incentive mechanisms is [5], and more recent studies on the pool mining issues can be found in [6–10].

2.2. Green Consensus Protocols Based on Proof-of-Work

Many blockchain networks use the proof-of-work (PoW) consensus protocol, which consumes significant amounts of energy but provides no useful outcome other than specifying accounting rights among miners. However, several approaches have been proposed to either reduce the energy consumption or to use the energy to perform some useful tasks.

To address high energy consumption-associated mining, the authors of [11] proposed a solution that suggests compensating the runner(s)-up of a round of mining by giving them exclusive rights to solve the block in the following round. This reduces the number of competing nodes and, as a result, lowers energy consumption. The proposed solution involves dividing time in epochs, each consisting of the two mining rounds. In the first round, every network node can participate in mining, while only the runners-up can participate in the second round. This approach can reduce the overall energy consumption of the mining process by almost 50%. The analysis presented in [11] shows that this solution is effective at reducing energy consumption, the likelihood of fork occurrences, the degree of mining centralization, and the impact of a transaction censorship attack in the PoW algorithm.

The consensus protocol proposed in [3] introduces a novel approach to the proof-of-work (PoW)-based consensus that aims to reduce energy consumption by utilizing memory resources as a trade-off. This protocol has been used in a recent "green" pool mining strategy to further reduce energy consumption [4]. The protocol framework follows the traditional structure of PoW-based consensus protocols used in public blockchains, such as Ethereum and Bitcoin, with three main phases: block construction, puzzle solving, and block inclusion in the blockchain. However, the protocol differs from previous protocols in two areas: the cryptographic puzzle and the method used for constructing and solving it. These differences are discussed in detail in [3].

Ref. [12] proposed a new consensus protocol, proof of federated learning (PoFL), which is designed to recycle the energy that is wasted in PoW by redirecting it towards federated learning. Federated learning and pool mining structures in PoW are naturally compatible, but the separation of the data utilization and ownership in blockchain may raise data privacy concerns during training and verification of a model. To overcome this issue, the paper proposes a privacy-preserving mechanism for verifying models and a reverse game-based mechanism for trading data. The former is used to verify a trained model while preserving privacy of used data, while the latter helps to prevent data leakage during training. The proposed mechanisms are shown to be effective and efficient in extensive simulations based on artificial and real-world data.

The authors of [13] proposed a novel consensus protocol called combinatorial optimization consensus protocol (COCP), which is based off of the proof-of-useful-work (PoUW) family of protocols. The proposed protocol requires solving real-life combinatorial optimization (CO) problems. The complexity of these problems has led to the development of various problem-dependent stochastic heuristic or metaheuristic methods for COCP.

Like most consensus protocols, COCP is asymmetric in that finding a solution is a difficult task whereas verifying it is straightforward. The main benefit of COCP is the efficient use of computing resources for finding solutions for real-life CO problem instances and for providing incentives for various BC participants. The paper also presents an illustrative example based on a real-life blockchain network, highlighting the potential practical impacts and energy savings that can be achieved using COCP.

2.3. An Overview of Blockchain Attacks

The architecture of a blockchain system consists of the following six layers [14]: the data layer, network layer, consensus layer, incentive layer, smart contract layer, and application layer. Using this layering perspective, we can classify blockchain security threats and attacks based on which of the layers they target [14,15].

Data layer attacks: These attacks target different elements of the data layers, such as transaction signatures, timestamping, or cryptographic methods. The following attacks fall into this category: malleability attack, time hijacking, and quantum attacks.

Network layer attacks: Blockchain is basically a peer-to-peer network in which nodes communicate with each other following a predescribed set of rules. The attacks on the network layer affect the efficiency of information propagation and routing. The following attacks fall into this category: distributed denial of service (DDoS) [16], eclipse attack, Sybil attack, Border Gateway Protocol (BGP) attack, and consensus delay attack.

Attacks on the consensus and incentive layer: The blockchain consensus protocol and the underlying incentive mechanism are tightly related as the incentive mechanism ensures that the most profitable strategy for a miner is to follow the consensus protocol. Attacks on this layer are directed toward incompatibilities of miners' incentive, so that attackers benefit from their malicious and dishonest actions. Section 2.4 discusses this type of attack and the countermeasures.

Smart contract layer attacks: Some blockchains, such as Ethereum, Hyperledger Fabric, and EOSIO, enable programmability through smart contracts. Attacks on the smart contract layers exploit some characteristics of the blockchain platforms, smart contract programming languages, smart contract execution environment, or bad coding practices. Some examples of such attacks are: integer overflow attack, re-entrancy attack, and replay attack. A comprehensive overview of the attacks can be found in [17].

Application layer attacks: There are different blockchain application scenarios, and cryptocurrencies or decentralized finances (DeFi) are probably the most popular and recognizable ones. The attacks targeting the application layer are in fact directed at a specific application. Some of the attacks that fall into this category are cryptojacking and crypto wallet thefts. Another is the blockchain poisoning attack, which is based on embedding malicious or illegal files in the blockchain (see [18–20], for example). According to [18], the attack can be performed as follows: (i) The first step is the preparation of a malicious or illegal file. The employed files can be privacy information, malware, and any illegal contents (see [20]); (ii) the attacker broadcasts the transaction with the embedded malicious or illegal file in the blockchain network; (iii) the file is embedded into the blockchain through the mining process and appears as a component that is shareable among the network participants. The reasons why the poisoning attack is very dangerous include the fact that it provides a widespread sharing of malicious or illegal content within the blockchain network, and the embedded files are difficult to modify or cancel.

This section provided a brief overview of the attacks on blockchain systems; surveys on blockchain security, attacks, and countermeasures can be found in [21,22]. The attacks on the consensus and incentive layer are discussed in more detail in the next section, as they are the focus of this paper.

2.4. Attacks and Corresponding Countermeasures Related to the Incentive Mechanism and Pool Mining

An incentive mechanism is one of the core elements of any blockchain system. An adequate incentive mechanism motivates miners to follow the protocol, as any deviation from the protocol would be unprofitable. It ensures that a stable consensus between the miners is reached and guarantees the efficiency of the system. In this section, we review the attacks targeting incompatibilities of incentive mechanisms.

2.4.1. Selfish Mining

The selfish mining attack was first proposed in [23]. The strategy of selfish miners consists of the following: Instead of publishing their discovered blocks immediately upon mining it, they intentionally fork the chain. While the honest miners continue mining on the public chain, dishonest miners try to extend their private branch. The feasibility and effectiveness of the attack is a consequence of the blockchain difficulty adjustment algorithm. The main goal of this attack is to create a private branch that is longer than the public one mined by the honest miners and to publish it at an opportune moment to gain larger revenue than its fair share, while the work of the honest miners is wasted. This attack is commonly performed by a group of colluded miners and not individually. It is also observed that the rational miners, upon observing the group's profit, will join the group of selfish miners and finally become the majority. A number of extensions and modifications of the original variant from [23] have been proposed. Some recent results on selfish mining are discussed in [2,24].

Significant research efforts have been invested into the investigation and analysis of detection strategies, prevention and defense solutions, and the impact of selfish mining attacks in blockchain networks. Different approaches have been proposed, including changes in the consensus protocols regarding the selection of the blocks to be broadcasted, the introduction of a new block type besides the regular blocks, modification of the block or transaction structure, favoring the blocks with more recent timestamps, and forcing a block timely publication. For an overview of the detection and prevention mechanisms for selfish mining, we point to [14,25,26].

2.4.2. Block Withholding

The block withholding attack was first proposed in [5]. The attack is directed against mining pools with the following strategy: A pool miner withholds any block they successfully mine without submitting it to the pool manager, so that the pool cannot claim any reward for the mined block. Feasibility of the attack is a consequence of the pool mining protocol, as miners can determine if the share (sub-puzzle solution) they found is a complete solution as well. There are two main types in this attack: "sabotage" and "lie-in-wait" attacks [5]. Sabotage is simpler and implies that the miner never reports the block. It does not bring any direct benefit to the attacker, but it causes financial harm to the pool manager, even causing bankruptcy. Lie-in-wait is another type of block withholding where a miner reports the mined blocks with intentional delay while focusing their mining work where it is most beneficial. Unlike sabotage, lie-in-wait is a profitable attack. The block withholding attack can form the basis for many different attacks. Some recent results related to the block withholding are presented in [1,27–29].

A number of research papers address the issue of block withholding and propose different approaches to prevent this kind of attack. According to [30], the defense methods can be divided in the following categories: methods that adjust reward distribution, methods that adjust the internal mining mechanism, and methods relying on credit level classification. For an overview of the defense mechanisms against block withholding, we point to [14,30–32].

3. A Variant of the Green Pool Mining

This section proposes a variant of the green pool mining reported in [4]. The proposed variant supports adjusting the difficulty of the consensus protocol puzzle and preserves reduced energy consumption and resistance against certain malicious activities of the miners within public pool mining. The approach is based on an idea to separate the puzzle-solving process between miners and the pool manager, which is feasible due to the two-dimensional nature of the required resources, so that neither miners nor the pool manager are able to solve the puzzle independently. We assume that: (i) the consensus puzzle requires both energy and memory resources in order to be efficiently solved; (ii) miners contribute the energy during the puzzle solving; and (iii) the pool manager contributes the memory for the puzzle solving.

The proposed pool mining approach is based on the following:

- The consensus puzzle is based on the cryptanalytic recovering of an internal state of a stream cipher;
- Correctness of the miners' work is controlled using random sampling of the computations submitted to the pool manager (PM);
- The resistance against certain attacks is based on the splitting of the resources that is necessary for efficient solving of the puzzle problem, i.e., a miner should employ only the energy resources;
- The lightweight implementation of the iterative recalculation is achieved through a lightweight stream cipher.

3.1. Cryptographic Puzzles for Consensus Protocol and Control of Its Difficulty

As the cryptographic puzzle for the consensus protocol, the proposal employs recovering the internal state of a keystream generator given its output segment. In stream cipher systems, the considered keystream generator yields the sequence, called a keystream sequence, for encryption of plaintext. The considered keystream generator is a finite state machine that consists of the following three parts [33]: (i) the internal state $\mathbf{X}_t$, (ii) state transition function $f(\cdot)$, and (iii) output function $g(\cdot)$. Assuming that the internal state is an L-dimensional binary vector and that the keystream sequence is a binary sequence $\{y_i\}_i$, we have the following:

$$X_t = [x_i^{(t)}]_{i=1}^{L}, \;\; t = 0, 1, \ldots,$$

$$X_{t+1} = f(X_t), \;\; t = 0, 1, 2, \ldots. \tag{1}$$

where X_0 denotes the initial internal state and

$$Y_t = g(X_t) = g(f(X_{t-1}), \;\; t = 1, 2, \ldots. \tag{2}$$

The considered puzzle problem is as follows: given an L-dimensional segment Y_t of the keystream sequence and knowing certain bits of the internal state X_t, recover the entire internal state.

The difficulty of the puzzle depends on the number of known bits of X_t, and assuming that the keystream generator is such that any pattern of the same number of known bits implies the same difficulty, for simplicity we can assume that the known bits are the first d bits. In order to control the difficulty of the puzzle, we pre-specify a certain set $\mathcal{D}$ of the parameter d. Figure 1 illustrates the puzzle problem. As a particular stream cipher, the lightweight ones, such as Grain [34] or some security-enhanced ones (see [35–37] for example), could be employed.

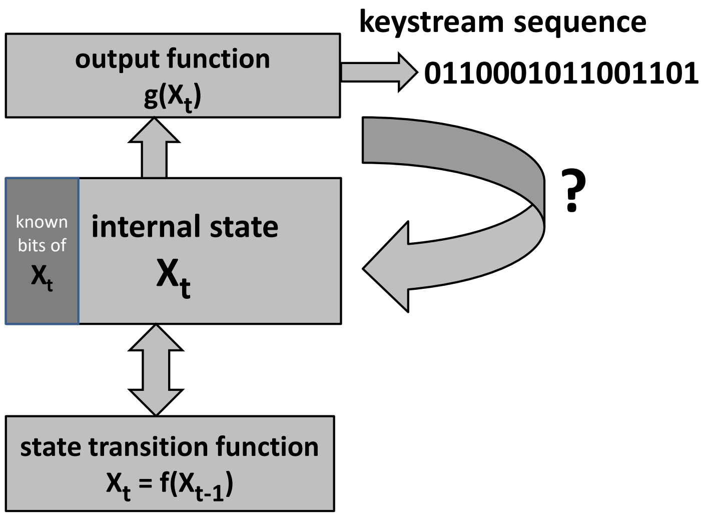

Figure 1. Puzzle problem: recovering the internal state of a keystream generator given the keystream sequence.

3.2. Control of the Miners

An important issue regarding the pool mining is the PM's control of the miners' work correctness. Without appropriate control, a dishonest miner could claim fake hard dedicated work. As an appropriate approach, this paper proposes a simple method for checking the correctness of a miner's work through periodical audits of a random sample of the results that a miner submits to the pool manager. We assume that a miner submits incorrect data at a rate of $r < 1$ and that PM checks the submitted data at a rate of $p < r$. We will show that this approach provides efficient detection of miners that submit incorrect data to the PM.

Design

The pool mining system consists of two entities: the pool manager (PM) and pool miners. The PM possesses computational and memory resources. The computational resources of the PM are for the PM's evaluations, and memory resources are rented out to the pool miners. The memory resources of the PM contain certain advances in specified table. The computational and memory resources of the PM could be considered as the PM cloud. The PM communicates with the pool miners through the dedicated gate. The main role of the pool miners is to perform certain evaluations that are required for solving the consensus protocol puzzle and, when considering the pool mining setting, assumes that only the PM could complete puzzle solving. Accordingly, pool miners require only computational resources but, as the pool mining participants, they also should support the PM by renting non-overlapping parts of the memory resources of the PM. The design setting assumes that certain miners could have own tables in order to perform malicious activities.

A mining pool is formed and initialized during the registration phase when the pool manager registers all miners interested in joining and contributing their mining work to the pool. In this phase, each miner declares the size of the sub-table they support, i.e., they rent it from the pool manager, and they get informed about the reward for the pool mining contribution. The architectural framework is provided in Figure 2.

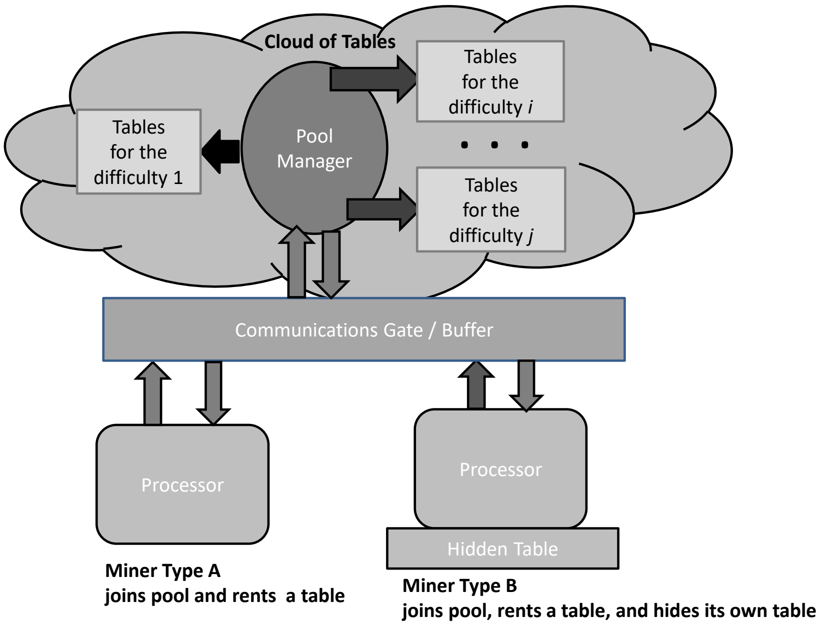
Cloud of Tables
Tables for the difficulty 1
Pool Manager
Tables for the difficulty *i*
• • •
Tables for the difficulty *j*
Communications Gate / Buffer
Processor
Processor
Hidden Table
Miner Type A joins pool and rents a table
Miner Type B joins pool, rents a table, and hides its own table

Figure 2. Pool mining architectural framework.

Communication between the pool manager and the miners is carried out through the communications gate with a buffer for accepting the results of the iterative re-encryption that miners send. More precisely, a miner writes the re-encryption result to the buffer, and afterwards, the pool manager reads this value for further processing, clearing the buffer. The pool manager periodically checks the validity of the results submitted by the miners as the way of controlling the correctness of their work. If the analyzed results are indeed obtained through recursive evaluations, the pool manager grants a share as a reward for the miner's work on the current block.

Table 1 provides an overview of the notations used throughout the paper. The mining procedures for the pool miner and pool manager are described in Algorithms 1 and 2, respectively.

Table 1. Table of frequently used notation.

Notation	Definition
$\mathcal{V} = \{v_1, v_2, ..., v_N\}$	Pool of N miners
$\mathcal{V}_H$	Sub-pool of honest miners
$\mathcal{V}_D$	Sub-pool of dishonest miners
q_i	Computing rate (power) of the miner v_i, $i = 1, 2, \ldots, N$
$d \in \mathcal{D}$	Puzzle difficulty
PM	Pool manager
$m_i^{(d)}$	Memories at PM, $i = 1, 2, \ldots, N^{(d)}$, $d \in \mathcal{D}$
$t_i^{(d)}$	Parameter of the puzzle-solving problem (time-memory trade-off parameter) employing memory $m_i^{(d)}$
b	Block of transactions
n	Nonce
$Enc_{pub}(\cdot)$	Encryption with the public key *pub* of PM
$Enc_{sec}^{-1}(\cdot)$	Decryption with the secret key *sec* of PM
$h(\cdot)$	hash function
$X_t = [x_i^{(t)}]_{i=1}^{\ell}$	ℓ-bit state of keystream generator
Y_t	ℓ-bit segment of the keystream sequence
$f(\cdot)$	The state transition function that maps X_t into X_{t+1}
$g(\cdot)$	The output function that maps X_t into Y_t
δ	Computing (energy) cost of the encryption employed in the puzzle

Table 1. *Cont.*

Notation	Definition
Δ	Expected time slot block verification
F_0	Flag indicating that mining should continue
F_{share}	Flag indicating a share solution is detected
$F_{solution}$	Flag indicating a puzzle solution is detected
F_A	Flag indicating that mining of the current block should be canceled

Algorithm 1 depicts the steps performed by a miner during the mining process. Initially, the miner selects a random nonce value to compute the hash value of the block. Subsequently, the miner chooses a prefix of the hash value with length ℓ to serve as the initial segment of the keystream sequence. After these preliminary steps, the miner performs the following operations repeatedly: The miner initializes the first d bits of the previous keystream sequence; then, with probability r, the miner either initializes the current keystream sequence with a random binary vector or uses the previous keystream sequence along with the state transition function and output function to initialize the current sequence. Then, the miner encrypts the sequence with the public key of the pool manager and sends it to the manager. Based on the response received from the manager, the miner takes the following actions:

- If the answer is $F_{solution}$, the miner sends the encrypted nonce to the manager;
- If the answer is F_0 or F_{share}, the miner continues with the mining process;
- If the answer is F_A, the miner terminates the current mining session and starts a new one.

Algorithm 1 The mining procedure of a MinerPool.

```
1:  procedure MinerMining(block $\mathbf{b}$, difficulty $d$)
2:      $\mathbf{n} \leftarrow$ RandomNonceVector( )
3:      $\mathbf{Y} \leftarrow Prefix_\ell(h([\mathbf{b}||\mathbf{n}]))$
4:      $\mathbf{Y}_0 \leftarrow \mathbf{Y}$
5:      $t \leftarrow 1$
6:      while $t \leq max_i\{t_i^{(d)}\}$ do
7:          Set the first $d$ bits of $Y_{t-1}$ to 1
8:          $rnd \leftarrow$ Random$(0, 1)$
9:          if $rnd < r$ then
10:             $\mathbf{Y}_t \leftarrow$ RandomBinaryVector( )
11:         else
12:             $\mathbf{Y}_t \leftarrow g(f(Y_{t-1}))$
13:         end if
14:         $\mathbf{c}^{(enc)} \leftarrow Enc_{pub}(Y_t)$
15:         $flag \leftarrow$SendToPM($\mathbf{c}^{(enc)}$)
16:         if $flag = F_{solution}$ then
17:             SendNonceToPM($Enc_{\mathbf{pub}}(\mathbf{n})$)          ▷ Mining is complete
18:             break
19:         end if
20:         if $flag = F_0 \vee flag = F_{share}$ then
21:             continue                                              ▷ Continue mining
22:         end if
23:         if $flag = F_A$ then
24:             abort                                                 ▷ Mining is aborted
25:         end if
26:     end while
27: end procedure
```

Algorithm 2 describes the actions performed by the pool manager during the mining process. The pool manager waits for messages from the miners and decrypts them using their private key. If it is not the first message from the miner, the manager checks the correctness of the received value with probability p. If the value is invalid, the manager makes a note of it. The manager then searches for the received value in the second columns of all TMTO tables.

Algorithm 2 The mining procedure of a pool manager.

```
1: procedure PoolManagerMining(block $\mathbf{b}$, difficulty $d$)
2:     $t \leftarrow 1$
3:     while $t \leq max_i\{t_i^{(d)}\}$ do
4:         $\mathbf{c}^{(enc)} \leftarrow$ ReceiveFromMiner( )
5:         $Y_t \leftarrow Enc_s^{-1}(\mathbf{c}^{(enc)})$
6:         if $t > 1$ then
7:             $rnd \leftarrow$ Random$(0,1)$
8:             if $rnd < p$ then
9:                 if $Y_t \neq g(f(Y_{t-1}))$ then        ▷ Check correctness of miner's work
10:                    $w \leftarrow w + 1$
11:                end if
12:            end if
13:        end if
14:        $t \leftarrow t + 1$
15:        Compare $Y_t$ using all tables $\{m_i^{(d)}\}_i$
16:        if $Y_t$ equal to the second column in $j$-th row of table $m_i^{(d)}$ then
17:            SendFlagToMiner($F_{solution}$)
18:            $Enc_{pub}(\mathbf{n}) \leftarrow$ ReceiveNonceFromMiner(())
19:            Get $\mathbf{n}$ from $Enc_{pub}(\mathbf{n})$
20:            $\mathbf{c} \leftarrow Prefix_\ell(h([\mathbf{b}||\mathbf{n}]))$
21:            $X_0 \leftarrow$ the $j$-th element of the first column of $m_i^{(d)}$
22:            $t \leftarrow 0$
23:            repeat
24:                $t \leftarrow t + 1$
25:                $X_t = g(f(\hat{X}_{t-1}))$
26:            until $X_t = Y_0$
27:            Publish block with the solution $X_{t-1}$        ▷ Because $Y_0 = g(f(X_{\tau-1}))$
28:            break
29:        end if
30:        if $w = 0$ then
31:            SendFlagToMiner($F_{share}$)
32:            continue
33:        end if
34:        if A block was published on the network then
35:            SendFlagToMiner($F_A$)
36:            break
37:        end if
38:        SendFlagToMiner($F_0$)
39:    end while
40: end procedure
```

If the value is found in a table, the miner has found the solution and the row where the value was found contains the solution. The manager sends the $F_{solution}$ flag to the miner and waits for the nonce value. Once the manager receives the nonce, they look for the solution in the specific row by obtaining the first element of the row, denoted as X_0, and repeatedly apply the state transition and output functions on the previous X value (starting with X_0) to reconstruct the elements of the row. If the found value X_t equals the value received from the miner, then the solution for the mining problem is X_{t-1}. Using that solution, the pool manager completes the block and publishes it.

If the manager does not find the received value in any of the tables, they check if the miner sent an invalid value. If no invalid values were received, the manager sends the F_{share} flag to the miner. If a new block was published on the network in the meantime, the manager sends the F_A flag to the miner, indicating that the current mining job must be aborted. Otherwise, the manager sends the F_0 flag, instructing the miner to continue mining.

3.3. Comparison with PoW

This section compares certain underlying ideas of the proposal with traditional PoW-based consensus protocols.

3.3.1. Cryptographic Puzzle and the Difficulty

The cryptographic puzzle in PoW consensus protocols is based on the cryptographic hashing problem, and the difficulty of this problem is determined by the required length of the pre-specified prefix of the hash result. It is assumed that the most efficient way to solve a hash puzzle is the exhaustive search. The approach in this paper is different. The puzzle problem is a crypt-analytic problem of recovering the internal state of a keystream generator with the assumption that the best technique for finding the puzzle solution is the time-memory trade-off type that is an adaptation of the one reported in [38]. The puzzle difficulty is controlled by pre-setting the internal state by a certain number of known bits.

3.3.2. Architecture

Traditional public blockchain architectures are not designed to provide specific resistance against certain malicious activities of miners or groups of colluding miners. On the other hand, this paper proposes a method for pool mining that ensures a higher level of resistance against some well-known blockchain attacks that target the incentive layer, i.e., selfish mining and block withholding attacks. The proposed pool mining method splits the resources necessary for solving the consensus puzzle between the pool manager and the miners, and protects the system against the aforementioned attacks in that way.

3.3.3. Control of Miner's Work

In standard PoW pool mining, the correctness control of miners' work is based on the specification of a sub-puzzle of lower difficulty, which is solved during solving the full PoW puzzle. The correctness control in our approach has to be different, as it does not rely on a sub-puzzle solving. Instead, the pool manager periodically checks some random sample of the results miners submitted as their contribution.

4. Probability of Puzzle Solving and Detection of Dishonest Miners

A malicious miner could submit some random values to the pool manager instead of valid results obtained through iterative computations, and so the PM should audit miners' work and submissions. We suppose that the PM selects outputs from a miner for a validity check with probability p and that a miner submits incorrect data to the PM with probability r.

Assumption 1. *Table dimensions $|m_i^{(d)}| << 2^\ell$ and the parameters $t_i^{(d)}$ are such that the collisions that are consequence of the birthday paradox do not exist in tables $m_i^{(d)}$, $i = 1, 2, ..., N^{(d)}$, $d \in \mathcal{D}$.*

Lemma 1. *Assuming that all miners from the sub-pool of honest miners v_H always send valid results to the pool manager and that the sub-pool of dishonest miners v_D sends valid values with rate $1 - r$, the probability Pr_{pool}^{win} that the pool will accomplish the inversion with the parameter ℓ within the time slot Δ is:*

$$Pr_{pool}^{win} = \frac{\Delta \sum_{i=1}^{N^{(d)}} |m_i^{(d)}| t_i^{(d)}}{2^{\ell} \delta} [(\sum_{v_i \in v_H} q_i) + (1-r) \sum_{v_i \in v_D} q_i] \,. \tag{3}$$

Proof. The candidate that can be inverted could appear within the set of candidates submitted by honest miners, or it can appear in the set of candidates submitted to the PM from the sub-pool of dishonest miners. Within time slot Δ, a miner v_i sends q_i/δ candidates, $i = 1, 2, \dots, N$ to the pool manager. Provided that the miners from the sub-pool v_H send genuine candidates, the pool manager collects a total of $\frac{\Delta \sum_{v_i \in V_H} q_i}{\delta}$ candidates. On the other hand, the sub-pool of dishonest miners provides $(1-r)\frac{\Delta \sum_{v_i \in V_D} q_i}{\delta}$ candidates. The pool manager tries to perform an inversion of each received candidate and for that purpose uses a joint table implicitly containing $\sum_{i=1}^{N^{(d)}} |m_i^{(d)}| t_i^{(d)}$ inversion pairs. Thus, the probability of finding the inverse of a candidate is $2^{-\ell} \sum_{i=1}^{N^{(d)}} |m_i^{(d)}| t_i^{(d)}$. Accordingly, we have the lemma statement. □

Proposition 1. *Assuming that a malicious miner submits incorrect data at rate r and that the PM checks validity of the data at rate p, the probability p_{detect} that the PM will detect malicious activity of the miner after n submissions is equal to*

$$p_{detect} = 1 - (1-r)^{np} \,. \tag{4}$$

Proof. Receiving n candidates for the puzzle solution implies that the PM has performed np random checks of the data validity. The probability that all checks have been performed on valid data and none on invalid data is $(1-r)^{np}$. Consequently, the probability that one or more invalid values are detected is equal to $1 - (1-r)^{np}$. □

5. Defense Mechanism against Block Withholding and Selfish Mining

In this section, we discuss the defense mechanism and resilience of the proposed pool mining approach against a dishonest miner or colluded group of dishonest miners that engage in block withholding or selfish mining attacks against the pool. We analyze the probability of a successful attack Pr_{attack} and the corresponding success rate ρ, as well as the probability Pr_{pool}^{win} that the pool will find the inverse within the given time slot Δ.

The pool of miners could contain a number of malicious ones. These miners could be with or without their own memory. Furthermore, they could act autonomously or as a group of attackers. Accordingly, this section considers the resistance against three types of attackers: (i) miners without memory; (ii) miners with memory; and (iii) groups of miners that could contain miners without memory and miners with memory.

Providing that the parameters are adequately chosen, the following analysis shows that the resistance of the pool against the two attacks can be arbitrarily high.

For simplicity of notation, in this section we assume that m_i, t_i, and N correspond to $m_i^{(d)}$, $t_i^{(d)}$, and $N^{(d)}$, respectively, $d \in \mathcal{D}$.

5.1. Attack of a Miner without Own Table

An attacker without their own memory could not use the rented one because it is under the exclusive control of the PM. We assume that the consensus puzzle is hard and that it can not be efficiently solved by applying an exhaustive search; instead, it requires the employment of a TM-TO approach. Nevertheless, a miner without their own memory could try to solve the puzzle using an exhaustive search. Accordingly, we should estimate the probability that the miner could solve the puzzle within the expected time slot. The miner could employ the following attacking approach.

Attacking Approach 1

- Scenario 1.1: After each evaluation of $\mathbf{c}^{enc}$, the miner first checks whether the puzzle solution has been reached. If the answer is positive, the miner keeps the solution and sends to the PM the next iteration of the evaluation;
- Scenario 1.2: If the miner receives the flag $F_{solution}$ from PM, they perform $t^* \leq t$ iterative re-evaluations trying to guess the puzzle solution before sending the nonce to the PM.

Theorem 1. *The success rate of a miner $v_i \in \mathcal{V}_D$ without their own table after Attacking Approach 1 is upper-bounded by*

$$\rho_i < \frac{q_i(\Delta + t^*)}{\Delta(\sum_{i=1}^{N} |m_i|t_i)[(\sum_{v_i \in v_H} q_i) + (1-r)\sum_{v_i \in v_D} q_i]} \tag{5}$$

Proof. The probability that any single hypothesis on the inversion appears as the correct one is equal to $2^{-\ell}$. Taking into account the computational power of the miner that performs the attack and the computational cost of each hypothesis check, the number of the hypotheses that could be checked within a time slot of duration τ is equal to $q_i\delta^{-1}\tau$. Accordingly, the probabilities of the success of Attacking Approach 1 are upper-bounded by $q_i\delta^{-1}\Delta 2^{-\ell}$ and $q_i\delta^{-1}t^*2^{-\ell}$ for Scenario 1.1 and 1.2, respectively, considering as well that the additional time slot t^* is not always available. Accordingly, we obtain:

$$Pr_{attack_1} < \frac{q_i(\Delta + t^*)}{\delta 2^{\ell}} \tag{6}$$

Combining the previous with Lemma 1, we obtain

$$\rho_i < \frac{q_i(\Delta + t^*)}{\delta 2^{\ell}}[\frac{\Delta \sum_{i=1}^{N} |m_i|t_i}{2^{\ell}\delta}[(\sum_{v_i \in v_H} q_i) + (1-r)\sum_{v_i \in v_D} q_i]]^{-1}$$

which yields the theorem statement. □

5.2. Attack of a Miner with Own Memory

A malicious miner $v_i \in \mathcal{V}_D$, which hides their own table m_i^*, could employ the following approach.

Attacking Approach 2

- Scenario 2.1: After each evaluation of $\mathbf{c}^{enc}$, the miner first checks whether the puzzle solution exists in their own table. If the answer is positive, the miner keeps the solution and sends to the PM the next iteration evaluation.
- Scenario 2.2: If the miner receives the flag $F_{solution}$ from the PM, the miner performs $t^* \leq t$ iterative re-evaluations and checks their own table to try and find the puzzle solution before sending the nonce to the PM.

Theorem 2. *The success rate of a miner $v_i \in \mathcal{V}_D$ with their own table after Attacking Approach 2 is upper-bounded as follows:*

$$\rho_i < \frac{(\Delta + t^*)q_i|m_i^*|t_i^*}{\Delta(\sum_{i=1}^{N} |m_i|t_i)[(\sum_{v_i \in v_H} q_i) + (1-r)\sum_{v_i \in v_D} q_i]} \tag{7}$$

Proof. The two-column table m_i is generated in the following way. For each table row: (1) the first element is randomly selected; (2) the value is iteratively encrypted t times; and (3) the second element is set to the value obtained after t re-encryptions performed in step (2). Provided that $|m_i| << 2^{\ell}$ and t are such that they ensure there are no repetitions appearing as a consequence of the birthday paradox, table m_i can be used to find $|m_i|t$ inverses. Thus, the inversion capacity α_i of the miner v_i is:

$$\alpha_i = 2^{-\ell}|m_i^*|t_i^* \,, \quad i = 1, 2, ..., N$$

Consequently, if a miner v_i employs ν different nonces, the probability of successfully performing at least one inversion is $\alpha_i \nu$. If the computing power is equal to q_i and the computing cost is δ, for the time slot Δ, the following applies:

$$\nu = \frac{q_i \Delta}{\delta}$$

It follows that the probability Pr^{win} that miner v_i successfully finds the inverse within the time slot Δ is equal to:

$$Pr^{win} = 2^{-\ell} \frac{q_i \Delta}{\delta} |m_i^*|t_i^* \,. \tag{8}$$

Thus, the probability of success in Scenario 2.1 and 2.2 is

$$Pr < \frac{\tau q_i |m_i^*| t_i^*}{2^\ell \delta} \tag{9}$$

where $\tau = \Delta$ and $\tau = t^*$ for Scenario 2.1 and 2.2, respectively, considering as well that the additional time slot t^* is not always available. The sum of these probabilities yields the following probability of success Pr_{attack_2} of a miner $v_i \in \mathcal{V}_\mathcal{D}$ with their own table after Attacking Approach 2 is upper-bounded as follows:

$$Pr_{attack_2} < \frac{(\Delta + t^*) q_i |m_i^*| t_i^*}{2^\ell \delta} \tag{10}$$

Combining the above with Lemma 1 gives the theorem statement. □

5.3. Attack of a Group of Miners

A group of malicious miners could establish a sub-pool in which they join their own tables and share outcomes of the re-evaluations. Note that this group of miners could consist of the ones with their own tables and the miners without their own tables. Figure 3 illustrates this attacking approach.

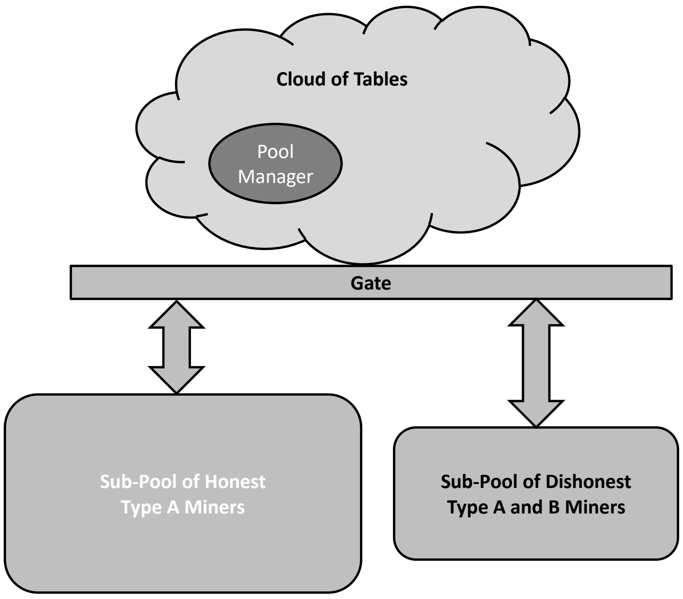
Cloud of Tables
Pool Manager
Gate
Sub-Pool of Honest Type A Miners
Sub-Pool of Dishonest Type A and B Miners

Figure 3. Model of the attack performed by a group of malicious miners.

Attacking Approach 3

Assumption 2. *A group of malicious miners establish a sub-pool v_D of attackers by joining their own tables and opening the joint table to all attackers. The malicious sub-pool consists of the group of miners v_{D_1} with their own tables and a group v_{D_0} of the ones without their own tables, $v_D = v_{D_1} \bigcup v_{D_0}$.*

- Scenario 3.1: After each evaluation of $\mathbf{c}^{enc}$, the miner first checks whether the puzzle solution exists in the joint table of the pool attackers. If the answer is positive, the miner keeps the solution and sends to PM the next iteration evaluation;
- Scenario 3.2: If the miner receives the flag $F_{solution}$ from the PM, the miner performs $t^* \leq t$ iterative re-evaluations and checks the joint table, trying to guess the puzzle solution before sending the nonce to the PM.

Theorem 3. *The success rate of a sub-pool of dishonest miners $v_i \in \mathcal{V}_D$ after Attacking Approach 3 is upper-bounded is as follows:*

$$\rho_{attack_3} < \frac{(\Delta + t^*)(\sum_{j \in v_D} q_j) \sum_{j \in v_D} |m_j^*| t_j^*}{\Delta(\sum_{i=1}^{N} |m_i| t_i)[(\sum_{v_i \in v_H} q_i) + (1-r) \sum_{v_i \in v_D} q_i]}$$

Proof. A group of dishonest miners can generate $\frac{\Delta \sum_{v_i \in v_D} q_i}{\delta}$ candidates within the time slot Δ. We assume that colluded malicious miners make all their tables available to the group for the inversion check and that a dishonest miner contributes with computation power q_i^*. The tables used in the inversion process implicitly contain $\sum_{v_i \in v_D} |m_i^*| t_i^*$ inversion pairs, and each candidate generated by the group is a subject of an inversion attempt. Therefore, the probability of finding the inverse for a candidate is equal to $2^{-\ell} \sum_{v_i \in v_D} |m_i^*| t_i^*$. Furthermore, the probability $Pr_{v_D}^{win}$ that the group of dishonest miners successfully finds an inverse within the time slot Δ expanded for t^* is equal to:

$$Pr_{v_D}^{win} < 2^{-\ell} \frac{(\Delta + t^*) \sum_{v_i \in v_D} q_i}{\delta} \sum_{v_i \in v_D} |m_i^*| t_i^* \; , \qquad (11)$$

where q_i, m_i^*, t^* are the parameters of the resources that a dishonest miner $v_i \in v_D$ contributes to malicious tasks, considering as well that the additional time slot t^* is not always available. Finally, employment of Assumption 2 and Lemma 1 gives the theorem statement. □

6. Experimental Evaluation and Comparison

6.1. Implementation of the Mining Pool

We developed a proof-of-concept implementation of the proposed consensus protocol and pool mining using the Go language-based *Geth* client [39], one of the three original implementations of the Ethereum protocol. Our implementation added the proposed protocol to the existing consensus protocols in *Geth* without altering any other elements. It includes the evaluation function E that serves as the core of the consensus protocol, as well as the creation and initialization of the TMTO tables with given dimensions, and verification of the blocks generated using the proposed consensus protocol. We implemented function E using the lightweight stream cipher Grain [34] for the experimental evaluation. It is worth noting that the other functionalities of the *Geth* client, such as block publishing, receiving new blocks from dedicated miners, and communication with the blockchain network, were not affected by our integration of the proposed consensus protocol.

The pool manager implementation consists of three components. The first component facilitates communication with the pool's miners through remote procedure calls, enabling the receipt of newly calculated values and nonce values when a solution is found. It also sends the appropriate flags to miners. The second component communicates with the blockchain network through standard *Geth* client functionalities, including sending mined blocks and receiving new blocks. The third component uses *Geth* functionalities to create new blocks, look up the received values in the TMTO table, verify the correctness of miners' work, and complete mined blocks.

The miners' implementation uses remote procedure calls to communicate with the pool manager. Periodically, the miners ask the manager if there is a new block that needs mining. When the miner receives a block, it selects a random nonce value for the block, calculates the hash value of the block (with the nonce), and starts calculating new values using the evaluation function E. The miner then sends the calculated values to the pool manager. The miners also wait for specific flags from the pool manager and react accordingly by continuing the value calculation, sending the nonce value to the manager, or aborting the mining for the current block.

Software for both miners and the pool manager was implemented using the Go language.

6.2. Experimental Results

We conducted the experiments on a Linux 64-bit platform running on an *Intel® Core™ i7-4710MQ CPU at 2.50 GHz* with eight cores. To simulate the proposed mining pool, we utilized *Docker* [40] software to create three different types of containers: pool manager, miner, and blockchain network container. The pool manager container comprises the pool manager software and a *Geth* instance that the manager uses. The miner container contains the software that represents the miner, while the blockchain network container only contains an instance of the *Geth* client, which simulated the rest of the blockchain network. Our mining pool in the experiments had 10 miners as members, so we used 10 instances of the miner container to simulate the miners. Depending on the experiment, the containers were configured to use the proof-of-work protocol or the consensus protocol proposed in this paper.

We aimed to compare the energy consumption of Ethereum's standard proof-of-work consensus protocol with the protocol proposed in our solution. To accomplish this, we identified the core methods of each consensus protocol that are the most CPU-intensive and used them to represent the energy consumption of the protocols.

In the proof-of-work algorithm, the core method is *HashimotoFull*, which calculates the hash value of the block. This method is repeatedly called during the mining process, making it the most CPU-intensive part of the protocol. On the other hand, the evaluation function E that implements the $g(\cdot)$ function mentioned earlier is the core method of the consensus protocol used in our proposed method. This is because it is the most CPU-intensive method in the protocol and is repeatedly called by miners during the mining procedure.

The first part of the experiments included the measuring of the CPU time needed to execute both of these methods. We have performed the measuring by utilizing the benchmark feature of the Go programming language. Both of these functions were executed 500,000 times each in order to calculate the average CPU time needed for their execution. Unlike the proof-of-work consensus where miners simply calculate hash values repeatedly, the proposed consensus mechanism requires miners to search the last column of the TMTO table for the most recent value that they have calculated. This represents an additional usage of the CPU resource that is not present in the proof-of-work consensus. The benchmark used to calculate the average CPU time of the evaluation function E included the function used to perform a binary search of the last column of the TMTO table. Therefore, the evaluation of the function E was conducted in conjunction with the search function. Table 2 shows the benchmark results for both the *HashimotoFull* method and evaluation function E. The results make it evident that the *HashimotoFull* function, which forms the fundamental component of the proof-of-work algorithm, consumes nearly 2.5 times the CPU time during execution compared with evaluation function E.

Because both methods are repeatedly called during the execution of consensus protocols, the second part of our experiments involved counting the number of times each method was invoked while mining a block using the aforementioned consensus protocols.

Table 2. Benchmark results for the *HashimotoFull* function and evaluation function *E*.

Function	Iterations	Average CPU Time
HashimotoFull	500,000	6365 ns
Evaluation function E	500,000	2454 ns

To determine the frequency at which these specific methods were invoked during block mining, we conducted tests with varying fixed difficulty levels for each consensus protocol. To evaluate the proof-of-work algorithm, we mined 100 blocks at fixed difficulties of 25, 26, and 27 each. However, for the second consensus algorithm, simply counting the function calls at different difficulty levels was inadequate, as the number of calls is also influenced by the dimension of the TMTO table used by miners. To address this issue, we conducted experiments using four tables of varying dimensions for each of the aforementioned difficulty levels. As a result, we conducted experiments for the second consensus protocol by mining 100 blocks at each of the fixed difficulty values (25, 26, and 27) for every corresponding fixed table dimension.

Figure 4 shows a histogram depicting the distribution of the number of calls to the *HashimotoFull* method for the proof-of-work consensus protocol with the difficulty set to 26. The main observation from the histogram is that the number of calls to the *HashimotoFull* method is in the order of 10^6 per block. During the experiment for proof-of-work with a difficulty of 26, the majority of mined blocks (67 out of 100) required less than 650×10^6 calls to the *HashimotoFull* method. As shown on the histogram, the number of blocks that require a larger number of calls to the *HashimotoFull* method is decreasing.

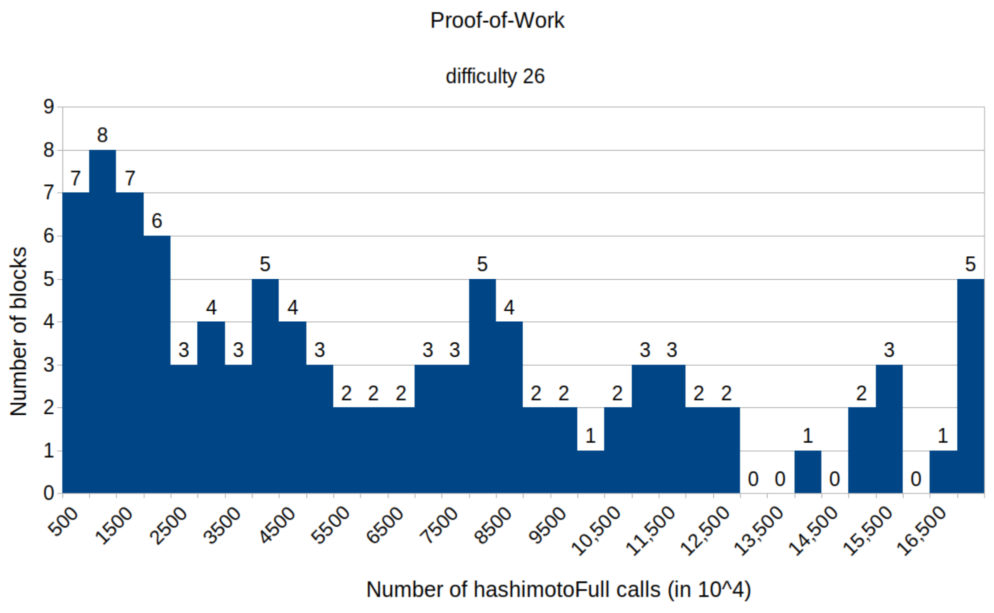

Figure 4. Histogram representing the number of mined blocks compared with the calls of the *HashimotoFull* method.

Table 3 presents the minimum, maximum, average, and median number of calls to the *HashimotoFull* function required for mining blocks using the proof-of-work algorithm with fixed difficulties. Initially, the minimum number of calls per block may appear perplexing as the number of calls for difficulty level 26 is smaller than the number for difficulty level

25. However, this result only indicates that the miner was more fortunate with a block of difficulty level 26 and found the correct nonce value with the least number of calls to the *HashimotoFull* method. The experimental results show that the maximum number of calls to the *HashimotoFull* function increases with the difficulty level, which is not unexpected given that higher difficulties require more computational effort. Moreover, the average and median number of calls also exhibit an increasing trend with the difficulty level, indicating that blocks with higher difficulty levels are more challenging to mine.

The results clearly demonstrate that there is a significant increase in the number of calls to the *HashimotoFull* function with an increase in mining difficulty, which is unsurprising.

Table 3. Results of the experiments regarding the number of calls of the *HashimotoFull* function in the proof-of-work per mined block.

Difficulty	Minimum	Maximum	Average	Median
25	388,686	169,457,010	35,826,915.65	25,652,903
26	64,203	238,039,725	64,099,263.27	50,874,196
27	445,186	536,247,913	116,922,745.37	73,640,447.5

The histogram in Figure 5 shows the distribution of specific number of calls to the evaluation function E for a fixed difficulty of 26 and TMTO table dimensions $2^{11} \times 2^{15}$. The first difference that can be seen when comparing with the previous histogram is that the number of calls to the evaluation function E is on the order of 10^4 calls, which is significantly lower than the number of calls to the *HashimotoFull* method in the proof-of-work consensus protocol. The second difference relates to the sizes of categories in the histogram. The categories in Figure 5 are smaller compared with the ones in the previous histogram in order to show a more detailed distribution. If the histogram for the proposed consensus used the same categories as the one for the proof-of-work consensus, all of the 100 mined blocks would fall into the first category. The experiment for the proposed consensus with a fixed difficulty and table dimension showed that the majority of the mined blocks (79 out of 100) required less than 34×10^4 calls for the evaluation function E. Similar to the proof-of-work consensus, the histogram shows that the number of blocks decreases as the number of calls to E increases.

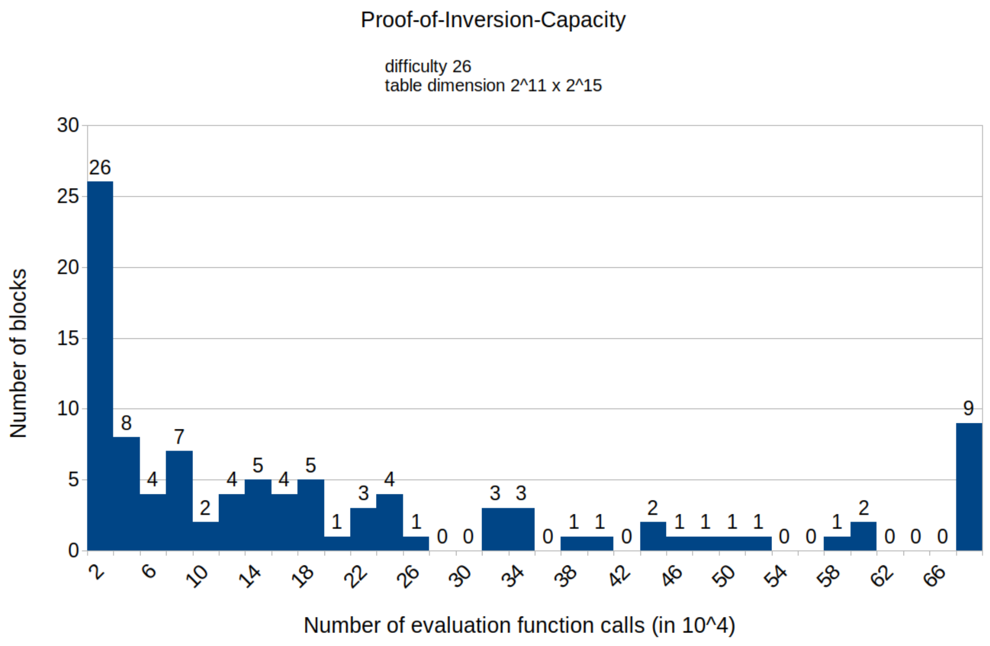

Figure 5. Histogram representing the number of mined blocks compared with calls of the evaluation function E.

Table 4 displays the dimensions of the TMTO tables that were utilized for each of the fixed difficulties, along with the minimum, maximum, average, and median number of calls in the evaluation function E in relation to the fixed difficulties and table dimensions. The minimum values for different difficulty levels and table dimensions may seem unexpected, but this is due to the fact that miners had more luck finding the correct nonce value with fewer calls in the evaluation function E for certain blocks, similar to what was observed in the proof-of-work consensus protocol. It can be observed that the maximum, average, and median values for the specific difficulties increase with the increase in table width and decrease in table height. This is in line with the consensus protocol, as a wider table requires more CPU usage for the mining process. Similarly, decreasing the table height increases the CPU usage as fewer values can be stored in the TMTO table, which requires miners to perform more calculations. Therefore, an increase in the table width or a decrease in the table height both result in an increase in CPU usage, which is in accordance with the consensus protocol. Furthermore, an increase in difficulty generally leads to an increase in the maximum, average, and median number of calls in the evaluation function E. One observation is that blocks with a difficulty of 26 and a table size of $2^{10} \times 2^{16}$ were mined faster than blocks with a difficulty of 25 and a table size of $2^{10} \times 2^{15}$, which may appear to be counter-intuitive. However, this can be explained by the fact that the table used for difficulty 26 stores twice as many values as the table used for difficulty 25.

Table 4. Results of experiments regarding the number of calls of evaluation function E per mined block.

Difficulty	Width	Height	Minimum	Maximum	Average	Median
25	2^{10}	2^{15}	6712	28,649	46,108.07	30,967
	2^{11}	2^{14}	5097	953,868	181,479.96	107,082
	2^{12}	2^{13}	5963	1,698,475	587,631.42	615,419.5
	2^{13}	2^{12}	4108	3,151,632	1,482,511.68	1,652,551.5
26	2^{10}	2^{16}	5638	194,241	39,992	28,915.5
	2^{11}	2^{15}	11,983	1,119,677	209,686.83	117,719.5
	2^{12}	2^{14}	13,030	2,112,467	739,251.28	650,157
	2^{13}	2^{13}	25,120	4,691,539	1,946,675.24	1,722,151
27	2^{10}	2^{17}	6710	155,303	45,492.31	31,970
	2^{11}	2^{16}	13,285	886,329	159,210.19	75,151
	2^{12}	2^{15}	26,940	3,148,764	794,431.39	672,646
	2^{13}	2^{14}	53,845	9,222,343	2,453,612.45	1,486,439

7. Discussion of Experimental Results

The experimental results presented in Section 6 show that the consensus protocol proposed in this paper requires less computation energy than the traditional hashing-based PoW. Note that the energy reduction in the considered green mining is the result of the employment of certain memory resources besides the computational ones. More precisely, the consensus protocol reduces the computational overhead, and therefore the energy consumption, as a trade-off with the employment of certain pre-computed tables for solving the consensus puzzle (see [3,4]) that belongs to a class of the inversion problems. In particular, if no memory is employed, the protocol puzzle should be solved using an exhaustive search, i.e., in the same manner as the traditional PoW puzzle. It is important to notice that the traditional hashing-based puzzle does not provide the possibility of reducing the exhaustive search operations in exchange for the employment of some memory. In our approach, the nature of the consensus puzzle is different, and memory employment provides a substantial reduction of the space that should be exhaustively searched. This results in a reduction in the energy consumption. In the experiments, we analyzed the settings with different amounts of allocated memory resources.

First, we identified the most CPU-intensive core methods of each of the consensus protocols, which are *HashimotoFull* for the PoW protocol and evaluation function E for our protocol. We used them to represent and compare the energy consumption of the protocols. The experiments show that *HashimotoFull*, which is the core function for PoW, is nearly 2.5 times more CPU intensive and thus more energy demanding than the evaluation function E. As the overall energy consumption depends on the number of calls to *HashimotoFull* and evaluation function E, we analyzed how many times these functions were executed during the block mining process. On average, PoW required 36×10^6 calls to *HashimotoFull* in to order to mine a block when the difficulty parameter was set to 25. With difficulty 26, the average number of calls was 64×10^6, and with a difficulty of 27, the average number of calls was 117×10^6. The experiments show that, for block difficulty 25, depending on the invested memory, our consensus protocol required on average between 46×10^3 and 1.5×10^6 calls for the evaluation function E. For difficulty 26, it required on average between 40×10^3 and 2×10^6, and for difficulty 27, between 45×10^3 and 2.5×10^6. Keeping in mind that *HashimotoFull* requires more CPU time (see Table 2) and thus more energy we obtain that the block mining process in PoW is between two and three orders of magnitude more expensive than our protocol for the considered block difficulties. Table 5 summarizes the results of the experiments regarding the energy reduction of the protocol proposed in this paper in comparison with the traditional hashing-based PoW.

Table 5. Comparison of CPU times for PoW and PoIC per mined block.

Difficulty	PoW	PoIC	Table Size	PoW CPU Time (ms)	PoIC CPU Time (ms)
25	36×10^6	46×10^3	$2^{10} \times 2^{15}$	2.3×10^5	1.1×10^2
		181×10^3	$2^{11} \times 2^{14}$		4.5×10^2
		588×10^3	$2^{12} \times 2^{13}$		1.4×10^3
		1.5×10^6	$2^{14} \times 2^{12}$		3.6×10^3
26	64×10^6	40×10^3	$2^{10} \times 2^{16}$	4.1×10^5	0.9×10^2
		210×10^3	$2^{11} \times 2^{15}$		5.1×10^2
		739×10^3	$2^{12} \times 2^{14}$		1.8×10^3
		2×10^6	$2^{14} \times 2^{13}$		4.8×10^3
27	117×10^6	45×10^3	$2^{10} \times 2^{17}$	7.4×10^5	1.1×10^2
		159×10^3	$2^{11} \times 2^{16}$		3.9×10^2
		794×10^3	$2^{12} \times 2^{15}$		1.9×10^3
		2.5×10^6	$2^{14} \times 2^{14}$		6×10^3

8. Conclusions

This paper proposes a variant of the recently reported blockchain pool mining [4], which employs energy and memory resources, and yields high resistance against certain malicious activities of miners within public pool mining. The proposed variant provides adaptability to a number of different execution scenarios that require a different level of difficulty in the consensus protocol puzzle and a reduction in energy consumption in comparison with standard PoW blockchains at the expense of using some additional memory resources. The main features of the proposed pool mining approach are: (i) the consensus puzzle is based on the crypt-analytic recovering of the internal state of a stream cipher that also supports different difficulties of the puzzle; (ii) control of miners' work is based on checking random samples of the results that are sent to the pool manager; (iii) the resiliency against block withholding attacks and selfish mining appears as a consequence of the separation of the resources required for efficient puzzle solving, i.e., a miner should employ only the energy resources and the PM controls the memory necessary for efficient puzzle solving; (iv) The lightweight implementation of the iterative recalculation is achieved through the employment of a lightweight stream cipher. The experimental evaluation of the energy consumption shows that it is substantially reduced in comparison with a traditional PoW hashing-based consensus protocol.

Author Contributions: Conceptualization, M.J.M.; methodology, M.J.M.; software, M.T.; validation, M.J.M., M.T. and M.K.; formal analysis, M.J.M.; writing—original draft preparation, M.J.M., M.T. and M.K.; writing—review and editing, M.J.M., M.T. and M.K.; supervision, M.J.M.; project administration, M.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the project AI4TrustBC from the Science Fund of Serbia and the project EU H2020 SMART4ALL-BC4GRID grant 872614.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The appendix contains histograms that display the results of all experiments that we conducted. Figures A1 and A2 illustrate the distribution of the number of calls to the *HashimotoFull* method for the proof-of-work protocol with the difficulties set to 25 and 27, respectively. The remaining histograms (from Figures A3–A13) show the distribution of the specific number of calls to the evaluation function *E* for various fixed difficulties and TMTO table dimensions of the proposed protocol. These results align with our paper's findings, indicating that the proposed solution requires fewer computational resources than the proof-of-work protocol. Notably, the results also confirm that the required computational power increases as the TMTO tables' height decreases and width increases.

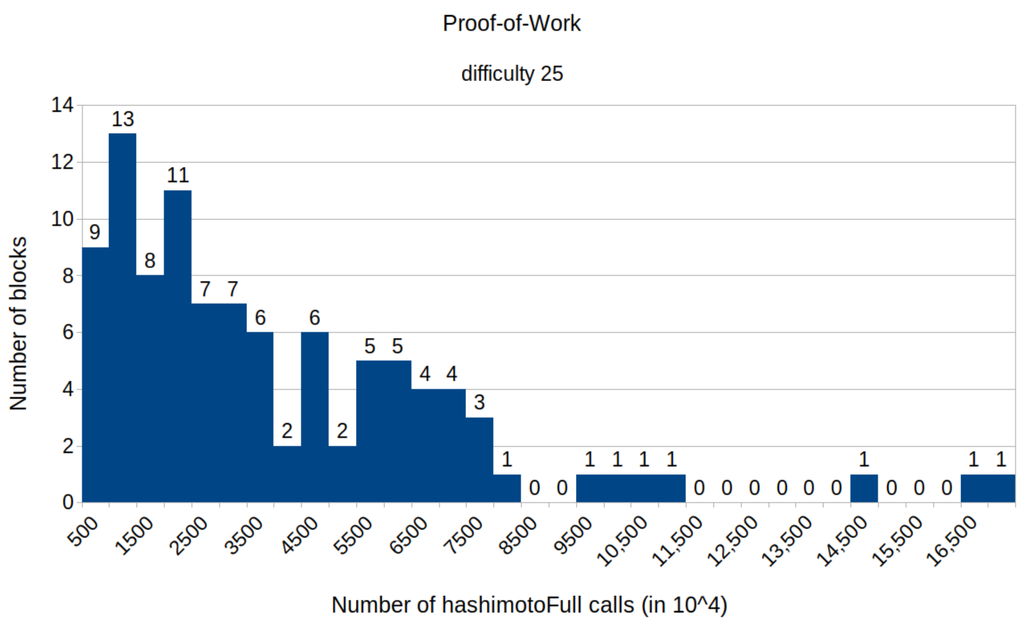

Figure A1. Histogram representing the number of mined blocks compared with the calls of the *HashimotoFull* method for a fixed difficulty of 25.

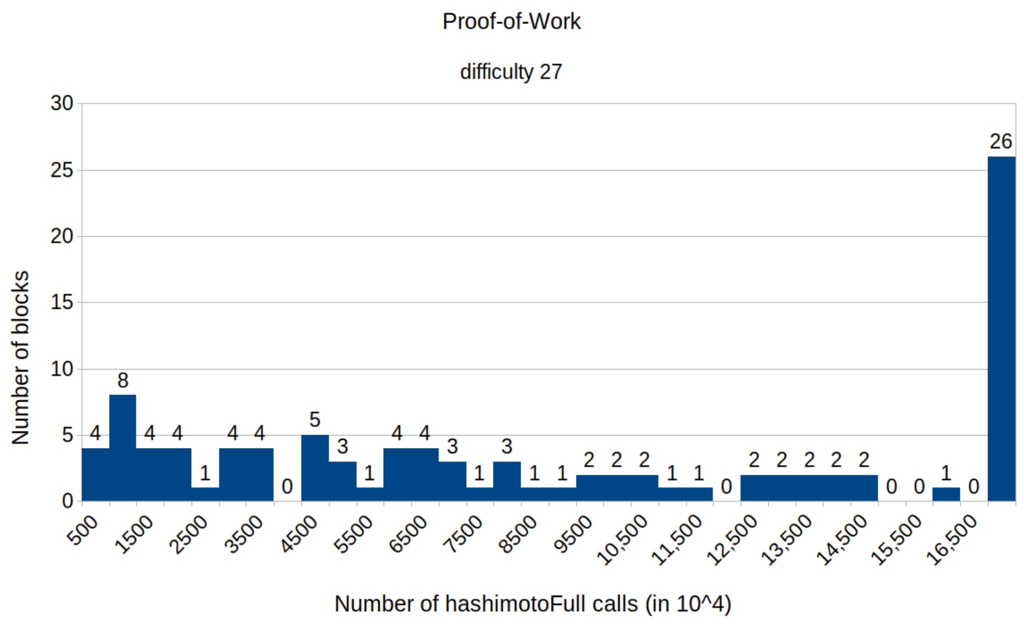

Figure A2. Histogram representing the number of mined blocks compared with the calls of the *HashimotoFull* method for a fixed difficulty of 27.

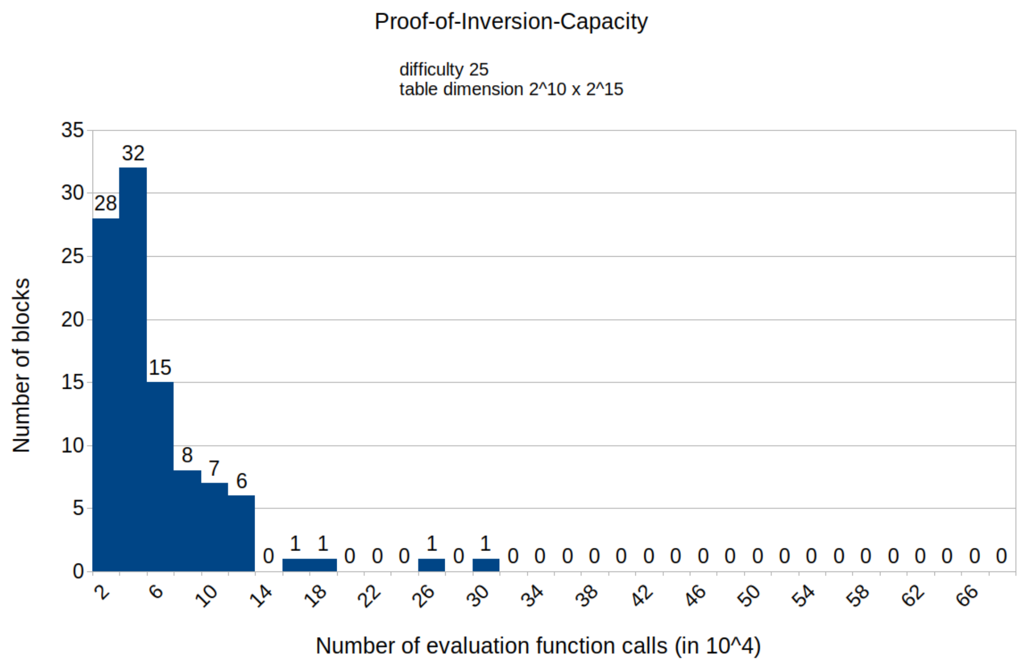

Figure A3. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 25 and table dimension $2^{10} \times 2^{15}$.

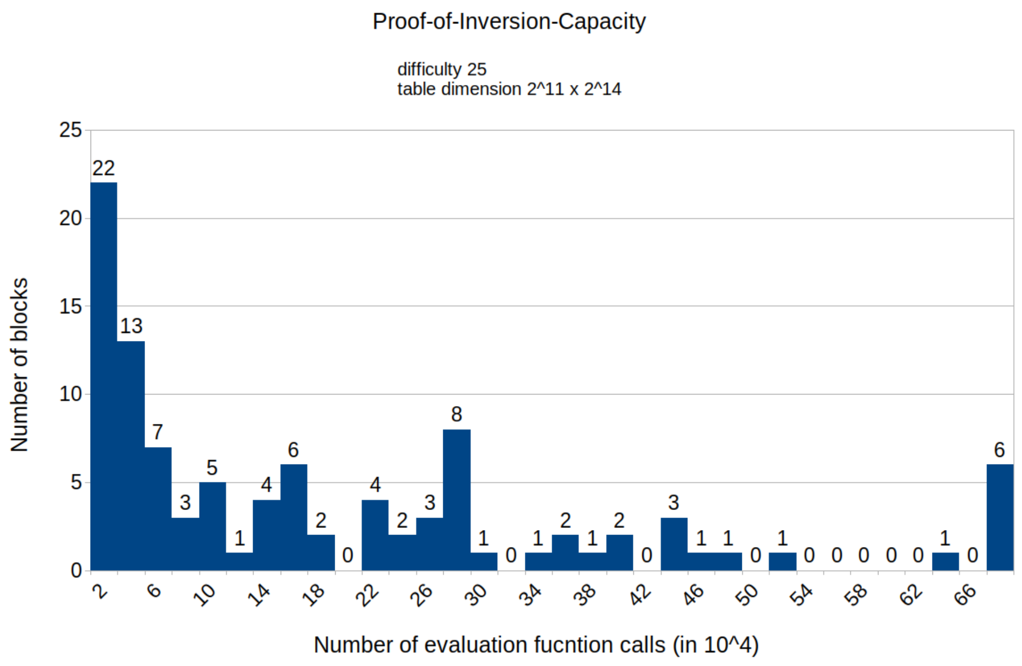

Figure A4. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 25 and table dimension $2^{11} \times 2^{14}$.

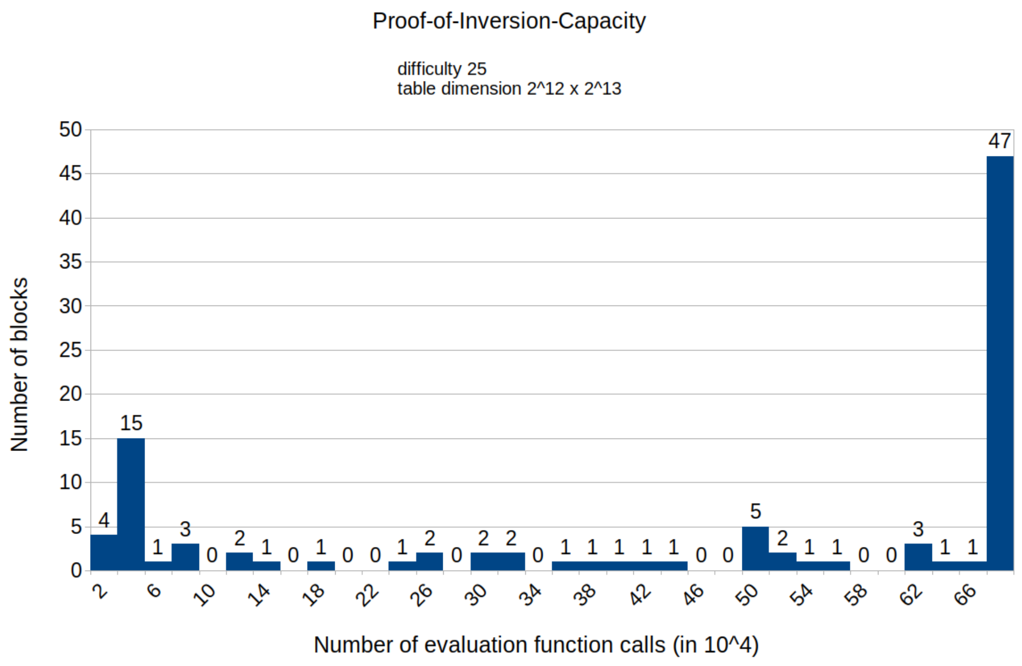

Figure A5. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 25 and table dimension $2^{12} \times 2^{13}$.

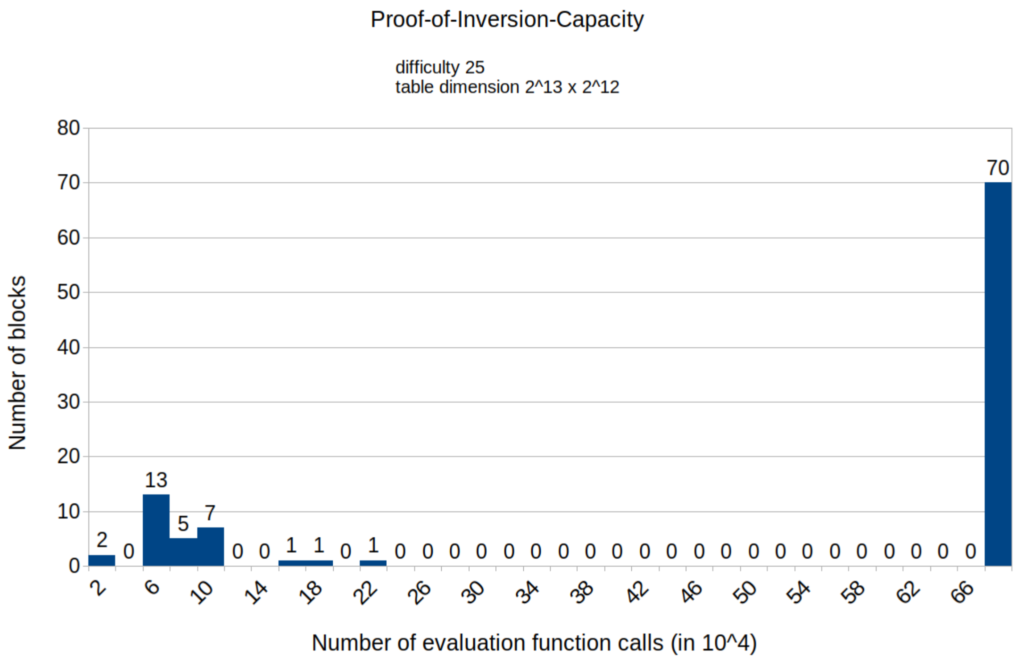

Figure A6. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 25 and table dimension $2^{13} \times 2^{12}$.

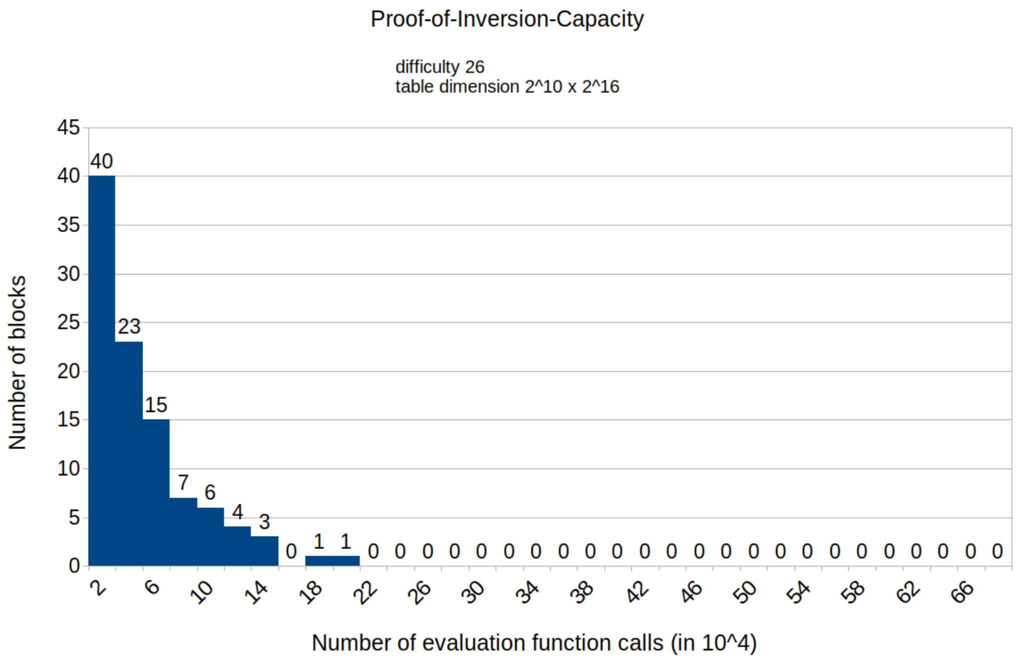

Figure A7. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 26 and table dimension $2^{10} \times 2^{16}$.

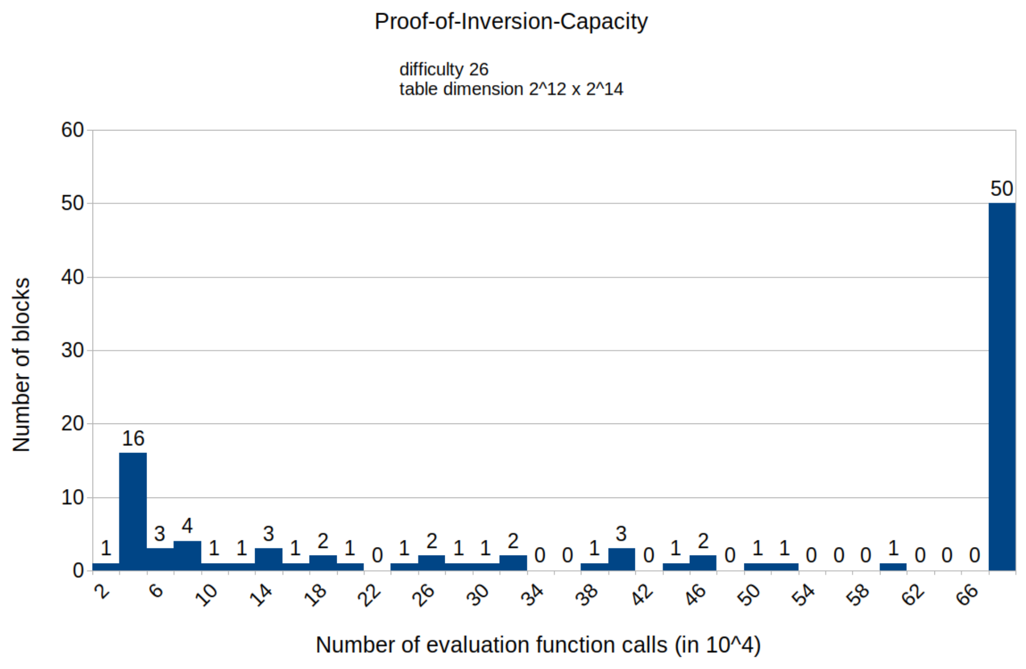

Figure A8. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 26 and table dimension $2^{12} \times 2^{14}$.

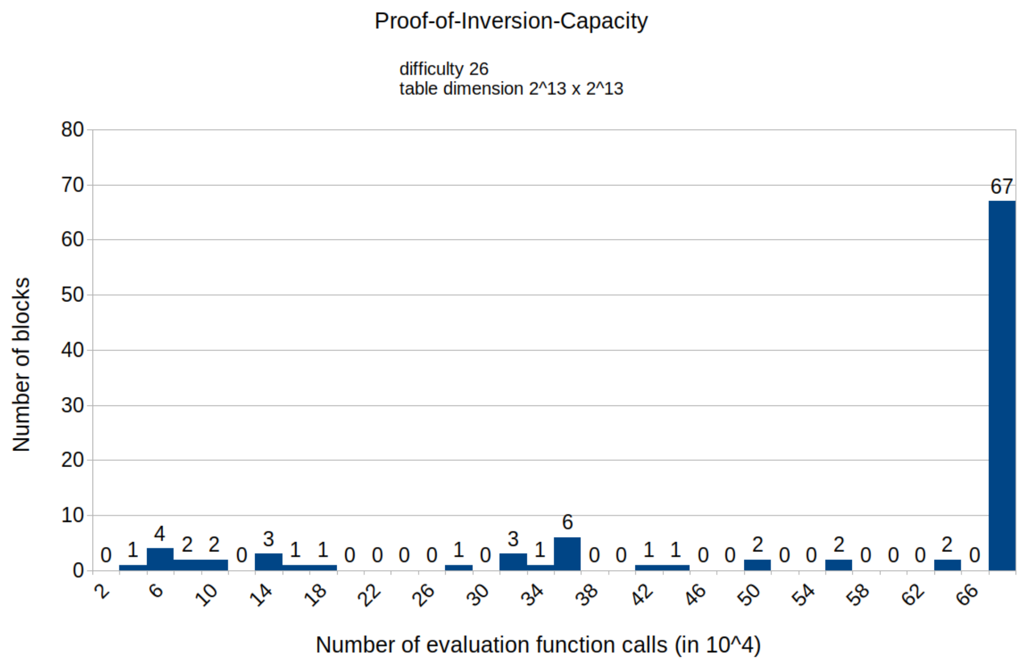

Figure A9. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 26 and table dimension $2^{13} \times 2^{13}$.

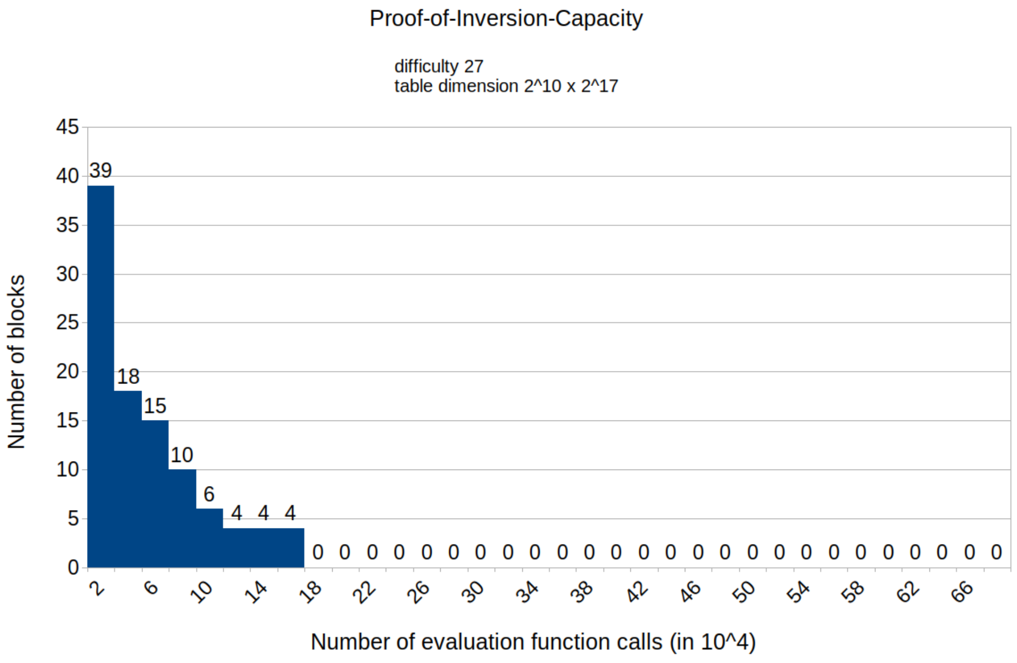

Figure A10. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 27 and table dimension $2^{10} \times 2^{17}$.

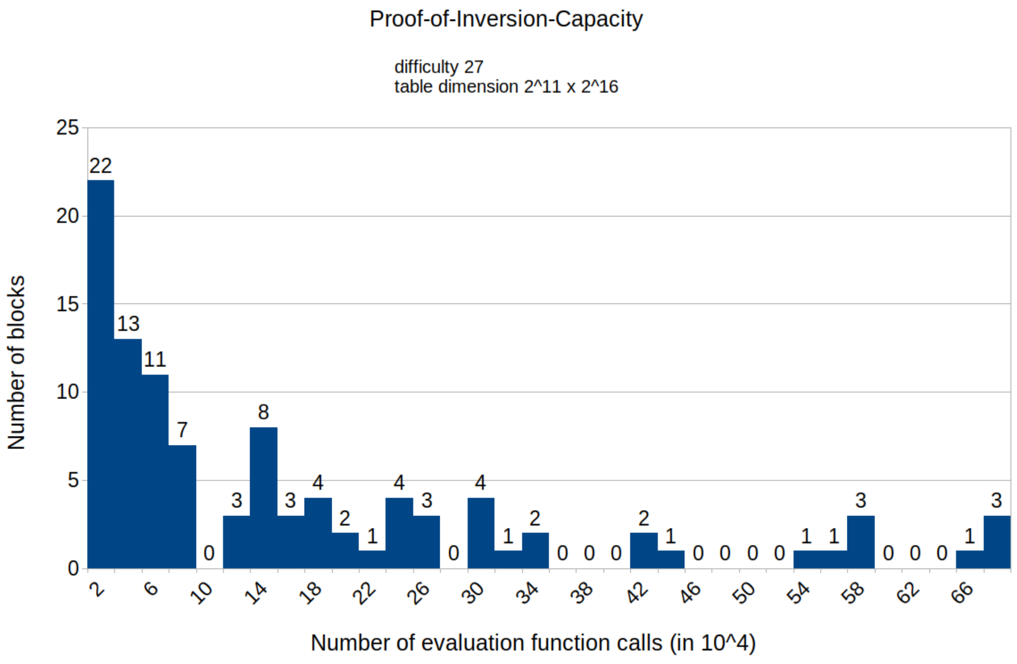

Figure A11. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 27 and table dimension $2^{11} \times 2^{16}$.

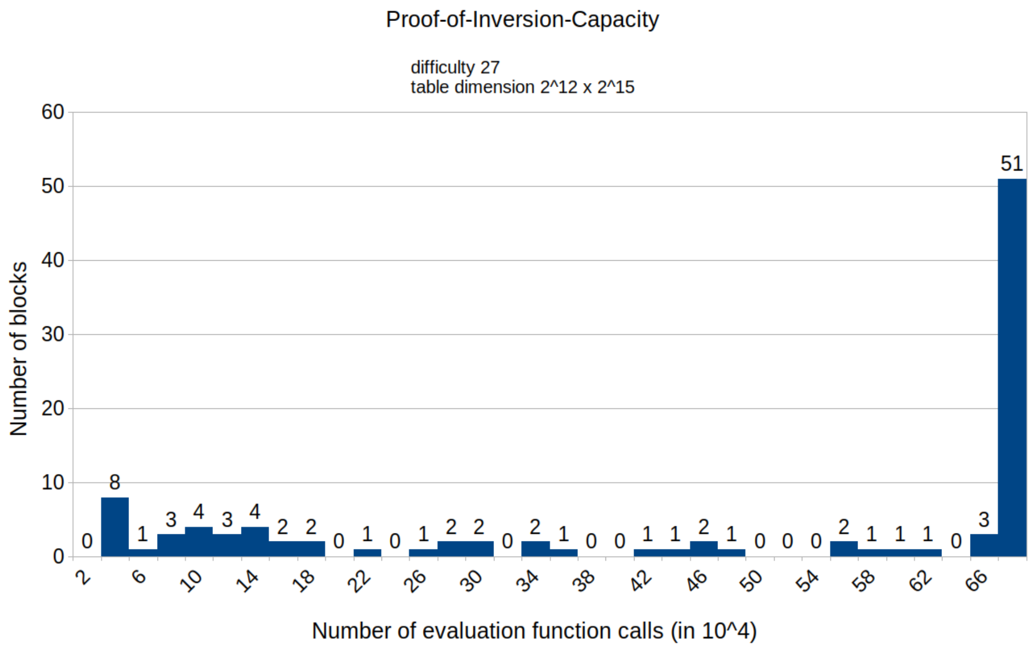

Figure A12. Histogram representing the number of mined blocks compared with the calls of the evaluation function E for a fixed difficulty of 27 and table dimension $2^{12} \times 2^{15}$.

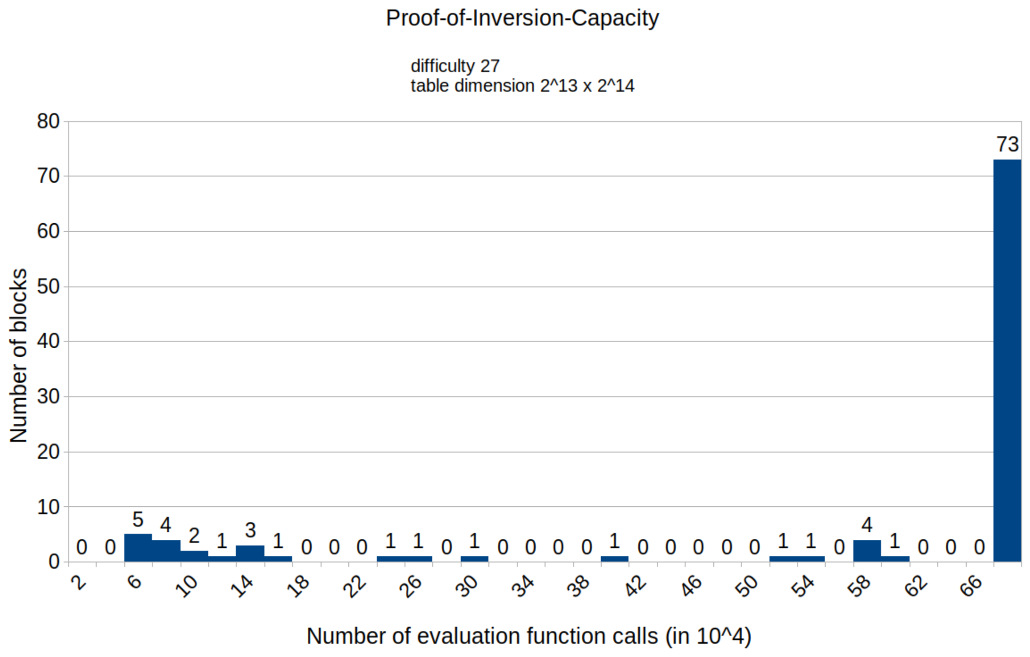

Figure A13. Histogram representing the number of mined blocks compared to calls of evaluation function E for fixed difficulty 27 and table dimension $2^{13} \times 2^{14}$.

References

1. Qin, R.; Yuan, Y.; Wang, F.-Y. Optimal Block Withholding Strategies for Blockchain Mining Pools. *IEEE Trans. Comput. Soc. Syst.* **2020**, *7*, 709–717. [CrossRef]
2. Kang, H.; Chang, X.; Yang, R.; Misic, J.; Misic, V.B. Understanding Selfish Mining in Imperfect Bitcoin and Ethereum Networks with Extended Forks. *IEEE Trans. Netw. Serv. Manag.* 2021, *early access*. [CrossRef]
3. Mihaljevic, M.J. A Blockchain Consensus Protocol Based on Dedicated Time-Memory-Data Trade-Off. *IEEE Access* **2020**, *8*, 141258–141268. [CrossRef]

4. Mihaljević, M.J.; Wang, L.; Xu, S.; Todorovixcx, M. An Approach for Blockchain Pool Mining Employing the Consensus Protocol Robust against Block Withholding and Selfish Mining Attacks. *Symmetry* **2022**, *14*, 1711. [CrossRef]
5. Rosenfeld, M. Analysis of bitcoin pooled mining reward systems. *arXiv* **2011**, arXiv:1112.4980.
6. Li, C.T.C.; Zheng, X.Y.Z.; Chen, Z. Cooperative Mining in Blockchain Networks With Zero-Determinant Strategies. *IEEE Trans. Cybern.* **2020**, *50*, 4544–4549.
7. Li, W.; Cao, M.; Wang, Y.; Tang, C.; Lin, F. Mining Pool Game Model and Nash Equilibrium Analysis for PoW-Based Blockchain Networks. *IEEE Access* **2020**, *8*, 101049–101060. [CrossRef]
8. Tang, C.; Wu, L.; Wen, G.; Zheng, Z. Incentivizing Honest Mining in Blockchain Networks: A Reputation Approach. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 117–121. [CrossRef]
9. Yu, J.; Kozhaya, D.; Decouchant, J.; Esteves-Verissimo, P. RepuCoin: Your Reputation is Your Power. *IEEE Trans. Comput.* **2019**, *68*, 1225–1237. [CrossRef]
10. Chen, Z.; Sun, X.; Shan, X.; Zhang, J. Decentralized Mining Pool Games in Blockchain. In Proceedings of the 2020 IEEE International Conference on Knowledge Graph (ICKG), Nanjing, China, 9–11 August 2020; pp. 426–432.
11. Lasla, N.; Al-Sahan, L.; Abdallah, M.; Younis, M. Green-PoW: An energy-efficient blockchain Proof-of-Work consensus algorithm. *Comput. Netw.* **2022**, *214*, 109118. [CrossRef]
12. Qu, X.; Wang, S.; Hu, Q.; Cheng, X. Proof of Federated Learning: A Novel Energy-Recycling Consensus Algorithm. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 2074–2085. [CrossRef]
13. Todorović, M.; Matijević, L.; Ramljak, D.; Davidović, T.; Urošević, D.; Krüger, T.J.; Jovanović, D. Proof-of-Useful-Work: BlockChain Mining by Solving Real-Life Optimization Problems. *Symmetry* **2022**, *14*, 1831. [CrossRef]
14. Wen, Y.; Lu, F.; Liu, Y.; Huang, X. Attacks and countermeasures on blockchains: A survey from layering perspective. *Comput. Netw.* **2021**, *191*, 107978. [CrossRef]
15. Cheng, J.; Xie, L.; Tang, X.; Xiong, N.; Liu, B. A survey of security threats and defense on Blockchain. *Multimed. Tools Appl.* **2021**, *80*, 30623–30652. [CrossRef]
16. Chaganti, R.; Boppana, R.V.; Ravi, V.; Munir, K.; Almutairi, M.; Rustam, F.; Lee, E.; Ashraf, I. A Comprehensive Review of Denial of Service Attacks in Blockchain Ecosystem and Open Challenges. *IEEE Access* **2022**, *10*, 96538–96555. [CrossRef]
17. Kannengießer, N.; Lins, S.; Sander, C.; Winter, K.; Frey, H.; Sunyaev, A. Challenges and common solutions in smart contract development. *IEEE Trans. Softw. Eng.* **2021**, *48*, 4291–4318. [CrossRef]
18. Sato, T.; Imamura, M.; Omote, K. Threat Analysis of Poisoning Attack Against Ethereum Blockchain. In Proceedings of the IFIP International Conference on Information Security Theory and Practice WISTP 2019: Information Security Theory and Practice, Paris, France, 11–12 December 2019; LNCS: Berlin, Germany, 2020; Volume 12024, pp. 139–154.
19. Mallah, R.A.; Lopez, D. Blockchain-based Monitoring for Poison Attack Detection in Decentralized Federated Learning. In Proceedings of the International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Malé, Maldives, 16–18 November 2022.
20. Matzutt, O.; Hiller, R.; Henze, J.; Ziegeldorf, M.; Mullmann, J.H.; Hohlfeld, D.; Wehrle, K. A quantitative analysis of the impact of arbitrary blockchain content on bitcoin. In Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC), Nieuwpoort, Curacao, 26 February–2 March 2018; Springer: Berlin/Heidelberg, Germany, 2018.
21. Leng, J.; Zhou, M.; Zhao, J.; Huang, Y.; Bian, Y. Blockchain Security: A Survey of Techniques and Research Directions. *IEEE Trans. Serv. Comput.* **2022**, *15*, 2490–2510. [CrossRef]
22. Guru, A.; Mohanta, B.K.; Mohapatra, H.; Al-Turjman, F.; Altrjman, C.; Yadav, A. Survey on Consensus Protocols and Attacks on Blockchain Technology. *Appl. Sci.* **2023**, *13*, 2604. [CrossRef]
23. Eyal, I.; Sirer, E.G. Majority is not enough: Bitcoin mining is vulnerable. In Proceedings of the International Conference on Financial Cryptography and Data Security, Christ Church, Barbados, 3–7 March 2014; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8437, pp. 436–454.
24. Dong, X.; Wu, F.; Faree, A.; Guo, D.; Shen, Y.; Ma, J. Selfholding: A combined attack model using selfish mining with block withholding attack. *Comput. Secur.* **2019**, *87*, 101584. [CrossRef]
25. Zhou, C.; Xing, L.; Liu, Q.; Wang, H. Effective Selfish Mining Defense Strategies to Improve Bitcoin Dependability. *Appl. Sci.* **2022**, *13*, 422. [CrossRef]
26. Azimy, H.; Ghorbani, A.A.; Bagheri, E. Preventing proof-of-work mining attacks. *Inf. Sci.* **2022**, *608*, 1503–1523. [CrossRef]
27. Chen, Y.; Chen, H.; Han, M.; Liu, B.; Chen, Q.; Ren, T. A Novel Computing Power Allocation Algorithm for Blockchain System in Multiple Mining Pools Under Withholding Attack. *IEEE Access* **2020**, *8*, 155630–155644. [CrossRef]
28. Fujita, K.; Zhang, Y.; Sasabe, M.; Kasahara, S. Mining Pool Selection under Block WithHolding Attack. *Appl. Sci.* **2021**, *11*, 1617. [CrossRef]
29. Yu, L.; Yu, J.; Zolotavkin, Y. *Game Theoretic Analysis of Reputation Approach on Block Withholding Attack*; NSS 2020; LNCS: Berlin, Germany, 2020; Volume 12570, pp. 149–166.
30. Chen, H.; Chen, Y.; Xiong, Z.; Han, M.; He, Z.; Liu, B.; Ma, Z. Prevention method of block withholding attack based on miners' mining behavior in blockchain. *Appl. Intell.* **2022**, 1–19. [CrossRef]
31. Zhang, Y.; Lv, X.; Chen, Y.; Ren, T.; Yang, C.; Han, M. FAWPA: A FAW Attack Protection Algorithm Based on the Behavior of Blockchain Miners. *Sensors* **2022**, *22*, 5032. [CrossRef]

32. Chen, Y.; Chen, H.; Han, M.; Liu, B.; Chen, Q.; Ma, Z.; Wang, Z. Miner revenue optimization algorithm based on Pareto artificial bee colony in blockchain network. *J. Wirel. Com. Netw.* **2021**, *2021*, 146. [CrossRef]
33. Katz, J.; Lindell, Y. *Introduction to Modern Cryptography*; CRC PRESS: Boca Ratton, FL, USA, 2007.
34. eSTREAM Portfolio of ECRYPT Project. Available online: https://www.ecrypt.eu.org/stream/announcements.html (accessed on 12 April 2023).
35. Oggier, F.; Mihaljević, M.J. An Information-Theoretic Security Evaluation of a Class of Randomized Encryption Schemes. *IEEE Trans. Inf. Forensics Secur.* **2014**, *9*, 158–168. [CrossRef]
36. Mihaljevic, M.J.; Oggier, F. Security Evaluation and Design Elements for a Class of Randomized Encryptions. *IET Inf. Secur.* **2019**, *13*, 36–47. [CrossRef]
37. Mihaljevic, M.J. A Security Enhanced Encryption Scheme and Evaluation of Its Cryptographic Security. *Entropy* **2019**, *21*, 701. [CrossRef]
38. Hellman, M.E. A Cryptanalytic Time-Memory Trade-Off. *IEEE Trans. Inf. Theory* **1980**, *IT-26*, 401–406. [CrossRef]
39. Ethereum Go Implementation—Geth. Available online: https://github.com/ethereum/go-ethereum (accessed on 12 April 2023).
40. Docker. Available online: https://www.docker.com/ (accessed on 12 April 2023).