Zk-SNARK Marketplace with Proof of Useful Work

Samuel Oleksak[0009−0008−9098−6171], Richard Gazdik[0000−0001−7777−7537], Martin Peresini[0000−0002−2875−9567], and Ivan Homoliak[0000−0002−0790−0875]

Brno University of Technology, Czech Republic
{ioleksak,igazdik,iperesini,homoliak}@fit.vut.cz

Abstract. Proof of Work (PoW) is widely regarded as the most secure permissionless blockchain consensus protocol. However, its reliance on computationally intensive yet externally useless puzzles results in excessive electric energy wasting. To alleviate this, Proof of Useful Work (PoUW) has been explored as an alternative to secure blockchain platforms while also producing real-world value. Despite this promise, existing PoUW proposals often fail to embed the integrity of the chain and identity of the miner into the puzzle solutions, not meeting necessary requirements for PoW and thus rendering them vulnerable. In this work, we propose a PoUW consensus protocol that computes client-outsourced zk-SNARKs proofs as a byproduct, which are at the same time used to secure the consensus protocol. We further leverage this mechanism to design a decentralized marketplace for outsourcing zk-SNARK proof generation, which is, to the best of our knowledge, the first such marketplace operating at the consensus layer, while meeting all necessary properties of PoW.

1 Introduction

Permissionless blockchain systems rely on consensus protocols to provide agreement among mutually distrusting nodes. The first and still one of the most popular blockchain consensus protocols is Proof of Work (PoW) [36]. PoW secures the network from Sybil attacks by requiring the nodes that participate in consensus to solve computationally intensive puzzles, such as finding a particular hash [36], solving memory-hard problems [39], or completing randomized computational tasks [32]. The results of these puzzles typically do not have any use outside the consensus protocol, which causes this protocol to be labeled as wasteful and environmentally unfriendly. Some estimates suggest that Bitcoin mining alone may be responsible for 65.4 megatonnes of CO_2 emitted per year, which is comparable to the country-level emissions of Greece [44].

To justify the energy consumption of PoW, the community has proposed puzzles that deliver value beyond securing the consensus protocol, with notable examples including Cunningham chains [32], Cuckoo cycles [28], DNA sequence alignment [29], and deep learning model training [4]. Protocols incorporating such useful computational puzzles are known as Proof of Useful Work (PoUW) protocols. These protocols have not yet gained significant traction in the blockchain space, mainly due to the challenges in designing meaningful tasks

that meet the core PoW requirements, such as the verifiability of solutions, adjustable hardness, the prevention of replay attacks of the solutions (freshness property), and chain integrity embedding [5,17].

Zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) [6] are a powerful cryptographic tool that enables verifiable computation and privacy preservation by allowing a prover to convince a verifier of the truth of a statement with no interaction nor revealing underlying data.

However, their usability is hindered by the fact that generating zk-SNARKs is both computationally and memory intensive, often requiring tens of gigabytes of RAM and taking tens of minutes even on high-end hardware, making them impractical to compute on resource-constrained devices. To address this, various off-chain proof generation frameworks and marketplaces have been proposed, allowing users to delegate proof generation to external providers. These solutions can be broadly grouped as (a) **centralized** [15] and **decentralized** [34,40], depending on whether a trusted operator manages task allocation and payments or whether matching is achieved through a peer-to-peer protocol, and (b) those that operate at the **application layer** [35], where proof outsourcing is an optional service, or those that are embedded at the **consensus layer**, where proof generation is inseparable from maintaining the blockchain's security.

In this paper, we tackle both the challenges of PoUW task design and zk-SNARK generation by proposing a novel consensus protocol where zk-SNARK generation constitutes the useful work, inheriting the security of PoW while producing useful cryptographic proofs as a byproduct.

To the best of our knowledge, no existing decentralized zk-SNARK marketplaces generate **general-purpose** zk-SNARKs at the **consensus layer**. Our proposed solution represents the first such system. While some protocols [8] utilize zk-SNARK at the consensus layer, their purpose is different, and they are limited to a **single specialized circuit**.

Contributions. The contributions of this work are as follows:

- We propose a novel approach to offload resource-intensive SNARK proof computation onto nodes using a blockchain based on the PoUW consensus protocol, creating the first decentralized marketplace for SNARK computation that provides security to the consensus protocol.
- We make an extension of the proposed approach to support zk-SNARK proofs computations as part of the PoUW consensus protocol. In particular, we address the challenge of delegating zk-SNARK proof computation involving private inputs in a public permissionless blockchain setting.
- We analyze and demonstrate the security properties of our proposed consensus protocol and its extension.

2 Background

In this section, we present the necessary preliminaries to facilitate understanding of the remainder of the paper, with particular focus on zero-knowledge proofs.

Proof of Useful Work (PoUW). This concept emerged to address the profound inefficiency of traditional Proof of Work consensus, where immense computational power is expended on puzzles with no intrinsic value outside of securing the blockchain. The core idea of PoUW is to replace arbitrary puzzles with computational tasks that produce valuable results, such as complex scientific simulations [29] or algorithmic challenges, e.g., k-orthogonal vectors problem [5]. PoUW covers a class of algorithms that decrease the energy wasted while still reaping the security benefits of PoW.

Zero-Knowledge Proofs (ZKPs). Zero-knowledge proofs [24] are a cryptographic system that enables one party (the prover) to convey the correctness of a statement to another party (the verifier) without disclosing any additional information about the statement.

Some ZKP systems also possess the property of *succinctness*, whereby both the proof size and verification time are significantly smaller than those of the original statement. This is especially useful in the blockchain environment, where proof storage and verification are replicated among all consensus nodes.

ZKPs have a wide range of applications across blockchain and cryptography, including cross-chain interoperability [46,45], lightweight blockchains [8], transaction mixing [43], L2 chain facilitation [12], and federated learning [30,18].

Zk-SNARK. Zk-SNARK (Zero-Knowledge Succinct Non-interactive Argument of Knowledge) [6] is a type of zero-knowledge proof system enabling the generation of proofs that are significantly shorter than the original statements, while still allowing for efficient verification time.

The process of obtaining zk-SNARKs typically starts with a program in a high-level domain-specific language (DSL), such as Circom [1] or Zokrates [48]. Arithmetic circuits are used to represent computations using mathematical operations, specifically addition and multiplication. R1CS (Rank-1 Constraint System) is an intermediate format that represents the program in a system of linear equations that capture the relationships between inputs, intermediate values, and outputs. The circuit takes two types of inputs: *public inputs*, which are known to both the prover and verifier, and *private inputs*, which are known only to the prover. These private inputs are used to demonstrate knowledge of a secret without revealing it. The *witness* is the assignment of values to all private inputs and intermediate variables that satisfy the circuit constraints for a given public input.

One of the drawbacks of SNARKs is the need for a *trusted setup*, which introduces a potential security risk. This setup involves generating a pair of cryptographic keys used for proof generation and verification. During this process, intermediate data known as *toxic waste* are created, and if not securely discarded, this data can be used to produce false proofs. Nevertheless, this problem can be alleviated by *secure multi-party computation* (SMPC), where multiple participants collaboratively contribute randomness and share responsibility for the setup. However, SMPC comes at the cost of increased coordination complexity and longer setup times.

Generating a zk-SNARK proof is computationally expensive and often impractical for resource-constrained devices. On the other hand, verification is fast and is often performed on resource-constrained devices or even on-chain. To address this imbalance, proof generation is often offloaded to specialized nodes with powerful hardware. To facilitate this outsourcing, proof generation marketplaces have emerged [34,40], enabling users to request specific proofs to be generated in exchange for fees.

3 Problem Definition

Developing a feasible PoUW protocol has the potential to significantly enhance the energy efficiency of PoW systems without compromising their security guarantees. Although several PoUW schemes have been proposed [5,4,22], none have yet reached a level of maturity and security for real-world deployment. The goal of our work is to design a PoUW protocol that will address the challenges of the previous designs and meet the required properties of PoW.

Required Properties of PoW. For any PoW protocol to be viable in an adversarial environment, its puzzle must retain critical security properties [2] which must hold true in PoUW as well:

- **Asymmetry**: Due to mutual distrust between the nodes, each node has to verify the correctness of a puzzle solution itself. To achieve reasonable scalability, the verification of a solution must be very quick while finding it should take significant effort and thus time.
- **Granularity and Parametrization**: The difficulty of a computational puzzle should be dynamically adjustable to achieve desired network parameters, e.g., block time.
- **Amortization-freeness**: Producing multiple puzzle solutions should be proportionally more difficult than producing a single solution.
- **Independence**: Solving one instance of the puzzle provides no computational advantage toward solving other instances.
- **Efficiency**: Additional overhead of operating PoW protocol in terms of memory and network bandwidth resources should be low.
- **Unforgeability**: A malicious prover should not be able to construct such puzzle tasks that allow him to precompute the solution.
- **Freshness**: The puzzle scheme is resistant to replay attacks, which would enable reuse of existing solutions or stealing of recently computed solutions.
- **Deterministic**: The cost of finding a solution should either be deterministic or have a predictable expected value.
- **Progress-free**: The probability of finding a solution should be independent of the effort already spent on solving the puzzle.
- **Publicly Verifiable**: The puzzle solution can be verified by any party without the participation of the prover (non-interactively).

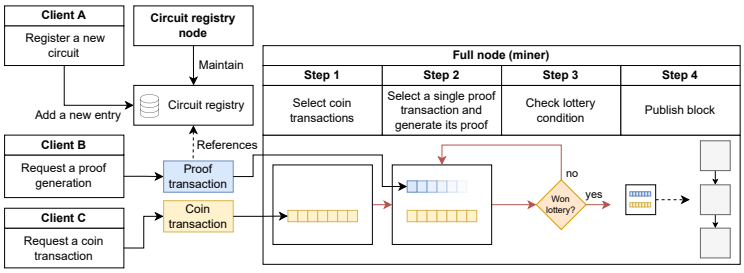

Fig. 1: Diagram illustrating the lifecycle of a single block, including the different roles of users within the protocol.

4 Proposed Approach

In this section, we propose a novel PoUW consensus protocol in which clients request outsourced SNARK proof (i.e., without zero-knowledge property for now) generation by providing an arithmetic circuit and proof parameters (see Figure 1). Consensus nodes compute these proofs, establishing a decentralized general-purpose SNARK marketplace that combines proof generation with consensus. The computation of these SNARK proofs forms the core puzzle of the consensus protocol, which secures the network. Each block within the chain contains two types of transactions: (1) **proof transactions**, which encapsulate SNARK generation and (2) **coin transactions** involving native currency operations.

4.1 Network Participants

In the proposed blockchain network, we distinguish three roles of participants: (1) **clients**, (2) **miners** and (3) **circuit registry nodes**.

Clients do not participate in the network's consensus process but create and broadcast coin transfer transactions and might utilize the marketplace properties of the network to request SNARK proof generation by publishing proof transactions. Coin transaction fees are fixed (adjusted by the network), while proof transaction fees scale with the complexity of the associated arithmetic circuit.

Miners (also referred to as *full nodes* or *workers*) participate in the consensus – they select coin and proof transactions from the corresponding mempools, generate the required SNARK proofs, publish new blocks, and validate incoming blocks. Miners, in turn, receive a block reward and transaction fees by producing a block that contains generated SNARK proofs and confirmed coin transactions.

The *circuit registry nodes* maintain the SNARK circuit registry, which enables decentralized registration and retrieval of arithmetic circuits and their metadata. They participate in SMPC for trusted setup during the initial circuit registration and store proving and verification keys. To ensure reliability, these nodes are required to stake native currency, which can be slashed for misbehavior or inactivity (as further detailed in Section 4.4).

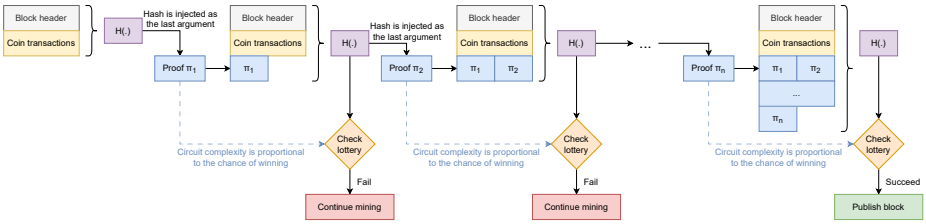

Fig. 2: The lottery mechanism during block mining (as described in Section 4.2). A lottery is performed every time a new proof is added to the block in progress.

4.2 Block Proposer Selection

The most important part of our approach is the selection of the block proposers, which is principally based on the idea of a weighted lottery in typical PoW consensus protocols. In the long term, the frequency with which a block leader is selected should be proportional to their computational power (fairness property). Nevertheless, some level of randomness must be incorporated into the system to prevent the most powerful miner from consistently winning each round by reaching the difficulty target first. We begin by presenting a strawman approach, which is then incrementally refined.

(A) Strawman Approach. Let us consider a system where (1) each coin and proof transaction can be included in only one valid block, (2) each computed proof is bound to the currently generated block by embedding the block hash as a public parameter of a proof (see Section 4.3), (3) the currently generated block is bound to the latest published block through direct hash reference forming a linear chain, and (4) miners must generate enough proofs to reach a minimum difficulty threshold κ for producing a block. The value of κ is periodically adjusted after a predetermined number of blocks (reflecting a certain amount of time, such as 2 weeks in Bitcoin) to account for changes in total mining power, maintaining an approximately constant expected block time.

Shortcomings. The approach misses the essential condition of progress-freeness, which mandates that the probability of successfully mining a block remains independent of the time already spent mining it. The absence of this property would lead to block production being centralized, since each block would consistently be created by the strongest miner, who always reaches the threshold first, causing all other miners to have to repeatedly start over. To improve it, we can introduce randomness into the block-building process.

(B) Addition of Lottery. To address the missing progress-freeness property of the Strawman, we replace condition (4) with a new condition (5) in which a lottery is triggered each time a miner adds a new proof to the current block being assembled, as illustrated in Figure 2 and described in Algorithm 2. Due to the accumulative nature of a block in progress, each generated proof must be cryptographically bound to the previous proof within the same block, forming

a *proof chain* that prevents the theft of useful work solutions and ensures the integrity of the chain (see Section 4.3 for a more detailed description). If a miner wins a lottery, the block is finalized and can be published. The difficulty threshold for a single block is no longer a fixed value that must be reached every time, but rather an expected value in the long term, reflecting the mean effort required for block creation. This mechanism is designed to give smaller miners a fairer chance to produce blocks while ensuring that, over time, the probability of block production aligns with each miner's proportional share of total mining power, under the assumptions of Poisson approximation, based on the law of rare events [21] (see $\mathbf{H_2}$ in Section 5). The greater the complexity C_i of the i-th proof π_i within the proof chain, the higher the probability of winning the lottery P_{win}, as described by the following formula:

$$P_{win}(i) = \frac{C_i}{\kappa}, \tag{1}$$

where κ denotes the current block difficulty, which is constant across the current block round. This satisfies the progress-free property of PoW protocols to a certain degree since the miner's chance of winning the lottery depends solely on the most currently generated proof.[1] Consequently, the time between block creations follows an exponential distribution (we elaborate on the expected block time in Section 6). The lottery is executed by comparing the hash of the potential block (as shown in Figure 2) with a target threshold derived from P_{win} similarly to the Bitcoin protocol.

It must be ensured that the computational complexity of each arithmetic circuit is proportional to its probability of winning the lottery, calibrated so that the expected difficulty remains consistent in the long term. We define the complexity C_i of a single proof as the number of gates (constraints) of the circuit associated with the proof transactions (see Section 6).

Shortcomings. This approach produces a fair amount of useful work without giving an undue advantage to more powerful miners, ensuring that the relationship between work and reward retains the fairness property (see $\mathbf{H_1}$ in Section 5). However, most of the work performed would be wasted, as miners who do not win the round must discard their partial block along with the generated proofs in accordance with conditions (2) and (3) defined in Strawman.

(C) Reducing Wasted Work – Adjusting Lottery Odds. To slightly decrease the amount of wasted work, one could implement a slight advantage in lottery odds for miners that have performed more useful work (constructed a proof chain with more accumulated work) within the current block round. This can be achieved by introducing an additional term into the lottery winning function that accounts for the sum of the complexities of all generated proofs in the currently produced block, where the influence of this term can be adjusted using

[1] The process of generating a single proof involves incremental progress, but it is reset once the proof is completed and the lottery executed.

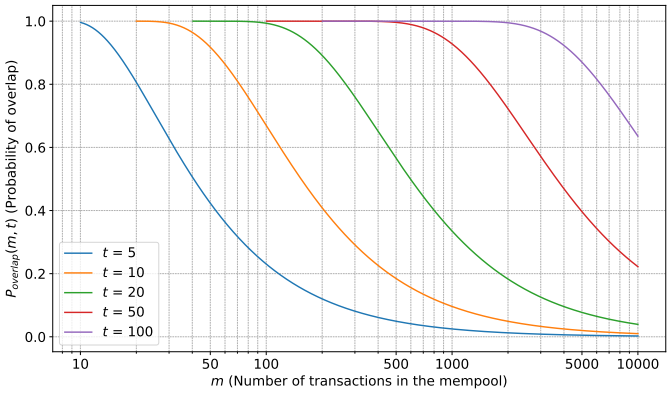

Fig. 3: The overlap probability $P_{overlap}(m,t)$ of two randomly selected transaction sets of size t from the mempool of size m.

the constant $\psi \in [0;1]$:

$$P_{win}(i) = \frac{C_i}{\kappa} + \psi \cdot \sum_{j<i} \frac{C_j}{\kappa}. \tag{2}$$

Shortcomings. Intuitively, this alteration provides a disproportionate advantage to the stronger miners that are able to produce stronger chains within a single block window, thereby increasing centralization (see $\mathbf{H_3}$ in Section 5).

(D) Reducing Wasted Work – Naive Parallel Block Production. By relaxing condition (3) of Strawman, which requires each newly generated block to be strictly bound to its immediate predecessor forming a linear chain, while still maintaining condition (1), that each proof and coin transaction can be included only once, we arrive at a scenario where parallel block production becomes possible if overlaps are avoided. If two miners were to concurrently produce a block of t transactions[2] with m unconfirmed transactions in the network (where $m \geq 2t$), then the probability $P_{overlap}$ of two miners selecting subsets of transactions with at least a single transaction overlap would be:

$$P_{overlap}(m,t) = 1 - \frac{\binom{m-t}{t}}{\binom{m}{t}}. \tag{3}$$

Shortcomings. Despite naively reducing wasted work by this approach, it still suffers from a large amount of wasted work caused by frequent overlaps of transactions, unless $m \gg t$, as can be seen in Figure 3. The probability of overlap decreases as the total number of transactions in the mempool grows and increases with a higher number of transactions per block. However, even with a large mempool and small blocks, the overlap probability remains significant, resulting in a substantial amount of wasted work.

[2] For simplicity, we assume that each proof transaction is equally difficult to compute and that transactions are selected at random.

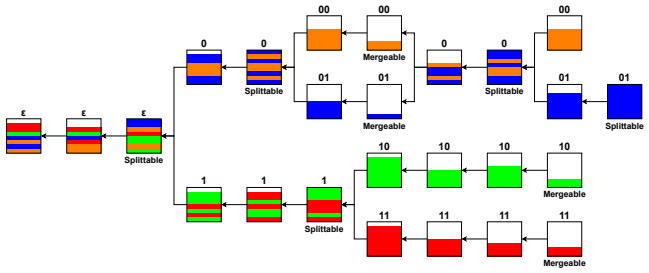

Fig. 4: Sycomore ledger creates a split in the block DAG in times of high traffic. Transactions here are color-coded based on their prefix, which determined their position after a split. Adapted from [3].

(E) Addition of Bucketing. A solution to alleviate the issue with overlapped transactions in the blocks is to divide the transactions into separate buckets based on the prefix of their hash.[3] It is noteworthy that clients have no incentive to manually alter the hash, as transactions are distributed uniformly across the buckets, resulting in constant fees. The length of the prefix, and thus the number of buckets, is dynamically adjusted based on the current number of pending proof transactions in the network (as seen in Figure 4). Miners can select a bucket of their choice, but the miner's inclusion set for the block is strictly limited to transactions originating from that singular bucket.

This is an approach inspired by Sycomore [3] that dynamically splits the blockchain into multiple parallel canonical branches according to network demand (Figure 4). The proposed adjustment would mean that the proof and coin transaction space would be logically partitioned. By partitioning the transaction space, overlap can occur only within the same bucket, thereby enabling parallelism and significantly reducing the likelihood of conflicts between concurrently mined blocks, thus reducing wasted work (see $\mathbf{H_4}$ in Section 5).

4.3 Integrity of the Blockchain

To prevent the mined solution (proof) from being stolen and used in another block, each arithmetic circuit must be slightly modified by introducing a new public parameter, which we call the *integrity parameter* η. The integrity parameter is initially calculated by hashing the header of the block being produced, which contains the previous block's hash, along with the included coin transactions. Next, the integrity parameter is injected along with client-provided parameters into the circuit (as shown in Figure 2) and a proof is generated. If a lottery fails, the integrity parameter is recalculated to also include the newly produced proof to prevent miners from discarding proofs that did not yield a lottery win, producing a cryptographically linked chain of proofs.

[3] We assume that this value is uniformly distributed and unique, even though certain miners might utilize multiple public keys.

Algorithm 1 shows an example source code of a program in ZoKrates DSL that proves knowledge of two factors of a certain number without revealing the factors to the verifier. To integrate this program into our system, we added an additional public parameter `integrity` to ensure integrity (as described in Section 4.3) along with a dummy `assert` statement to prevent the argument from being removed during optimization (both changes are highlighted in orange). Altering the value of the integrity parameter η causes the proof verification to fail.

Algorithm 1 An example of the arithmetic circuit with highlighted necessary modifications.

```
def main(
    private field factor1,
    private field factor2,
    public field product,
    public u32 integrity
) -> bool {
    assert (integrity != 0); // Dummy utilization
    assert (factor1 * factor2 == product);
    return true;
}
```

4.4 Circuit Registry Subnetwork

Our solution uses a subnetwork of nodes that manage the registered arithmetic circuits. The subnetwork provides functions for decentralized registration and retrieval of arithmetic circuits along with the corresponding proving and verification keys, circuit complexity, and the content of the original circuit source code file in domain-specific language. This enables clients to query the database for an existing circuit that meets their requirements, thereby avoiding the overhead of registering a new circuit and reducing unnecessary load on the subnetwork. The circuit subnetwork consists of nodes that stake a sufficient amount of native currency. In the event of node misbehavior (submitting an invalid SMPC share) or inactivity (failing to respond to circuit metadata requests or missing an SMPC contribution), the stake will be slashed.

A client requesting registration of a new circuit pays a fee that is distributed among the subnetwork nodes. The registration request contains a source code file that is compiled by subnetwork nodes, which then participate in the trusted setup by contributing randomness using SMPC. SMPC enables nodes to generate the circuit key pair such that security is preserved as long as at least one node is honest and deletes its randomness after the ceremony.

4.5 Extension: Proofs with Private Parameters

Our current solution can only generate SNARKs with public parameters, as clients must publicly share the requested proof parameters in the mempool. While some applications can operate without private parameters and rely on the network state as-is, many others (e.g., privacy-preserving ones) require proofs that incorporate private inputs. Therefore, we propose an approach to overcome this limitation by the witness obfuscation outsourcing technique; however, due to space constraints, we defer it to Section B of Appendix.

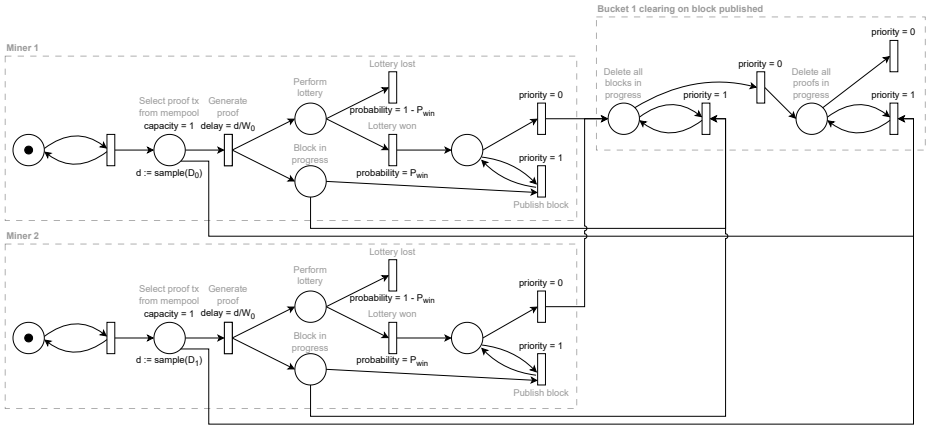

Fig. 5: Stochastic Time Colored Petri Net (STCPN) modeling the consensus protocol with two miners and a single bucket. The network can be extended to more miners or buckets by replicating the miner or bucket component.

5 Evaluation

We modeled the lottery portion of our proposed protocol using a Petri net (Figure 5) to empirically verify the following hypotheses:

- $\mathbf{H_1}$: Reward distribution is commensurate with the mining power of nodes.
- $\mathbf{H_2}$: The complexity of selected proofs during a certain time frame does not affect the reward.
- $\mathbf{H_3}$: Introduction of the ψ parameter (Section 4.2) reduces wasted work at the cost of increasing centralization.
- $\mathbf{H_4}$: Introduction of bucketing (Section 4.2) reduces wasted work.

Experiment Description. We implemented the Petri net (Figure 5) in Python using the `simpn` library.[4] We focused on the proof generation process and the resulting interactions among miners. Consequently, our model considered only proof transactions, as coin transaction confirmation is effectively instantaneous by comparison. Likewise, block verification time was negligible and therefore omitted. We assumed that each miner had already cached the data required to generate a proof from the circuit registry subnetwork (Section 4.4).

Plain Lottery. Plain lottery in its simplest form requires hypotheses $\mathbf{H_1}$ and $\mathbf{H_2}$ to provide the fairness property. Experiments for these hypotheses were run with two miners which differed in their computational power by a certain factor in each run, as shown in Figure 6 and Figure 7. Block rewards ratio for miners has an identity relationship with the computational power ratio, as expected. However, the proof rewards ratio exhibits a quadratic dependence, which is caused by the stronger miner not only producing more blocks, but also each block being stronger on average compared to the weaker miner. This quadratic

[4] `https://pypi.org/project/simpn/`

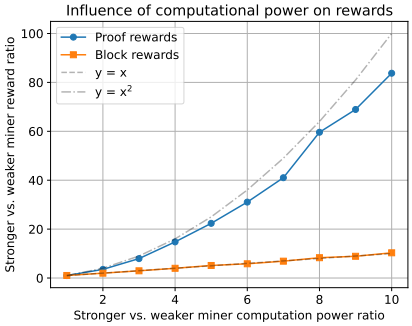

Fig. 6: Simulation results for $\mathbf{H_1}$ run with 2 miners, where the relative computational power of a stronger miner was increasing as indicated on the x-axis.

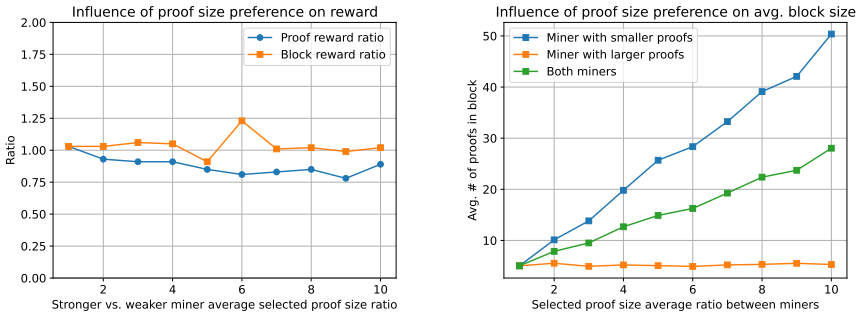

Fig. 7: Results of experiments for hypothesis $\mathbf{H_2}$ regarding the preference of proof complexity. Simulation was run with two miners of the same computational power, but with a different preference of proof size (as indicated by the x-axis).

trend would make the reward system unfair since increasing consensual power by a factor of two would increase proof rewards four-fold. Thus, the contribution of proof rewards in total miner rewards should be minimized to disincentivize the creation of mining pools. However, proof rewards cannot be removed completely because miners need to be motivated to continue mining the same block after an unsuccessful lottery instead of starting a new block from scratch.

Lottery Preferring Longer Chains. In Section 4.2 we introduced the parameter ψ into the lottery formula, which controls the influence of current proof chain complexity on the lottery chance, as a means to prefer longer proof chains. We anticipated that with the increasing value of this parameter the network would decrease the ratio of wasted work to all performed work at the cost of reducing fairness (hypothesis $\mathbf{H_3}$). Figure 8 shows the evolution of wasted work and rewards distribution with regard to the increasing parameter ψ.

Bucketed Lottery. Hypothesis $\mathbf{H_4}$ considers the influence of the number of buckets on the amount of wasted work (Figure 9). It is clear from the result that

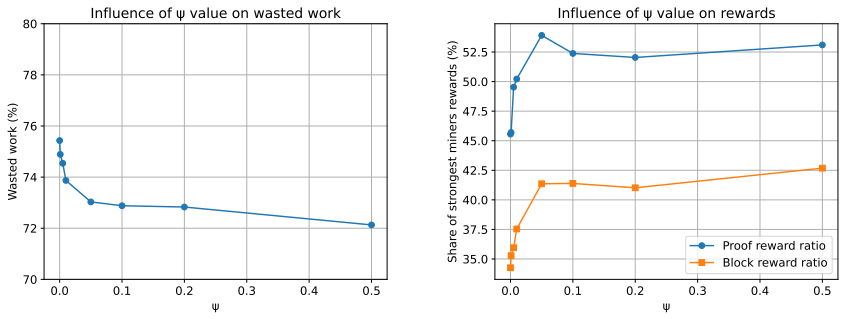

Fig. 8: The experimental results of ψ parameter addition ($\mathbf{H_3}$) with 5 miners with different computational power.

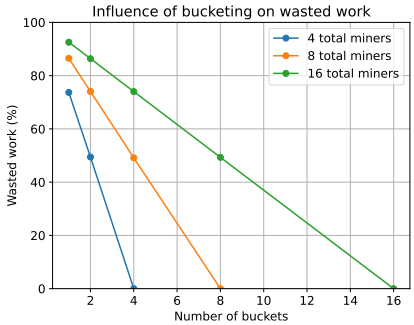

Fig. 9: Results from simulation of $\mathbf{H_4}$.

a high number of buckets is beneficial; however, their number is constrained by the number of proofs that are available in the mempool.

6 Discussion

In this section, we examine specific aspects of our proposed solution, focusing especially on how our consensus protocol fulfills the key PoW conditions.

Time to Generate a Proof. Proof generation time in Groth16 zk-SNARKs[25] is critical for the scalability of our consensus protocol and block time calibration. A linear relationship between the number of constraints and proof time is essential for predictable performance and efficient resource allocation. This linearity allows workers to handle increasingly complex circuits without exponential slowdowns, supporting the decentralized and scalable nature of the protocol. Theoretically, the proof time $T(n)$ for a circuit with n constraints follows $T(n) = a \cdot n + b$, where a is the per-constraint cost (e.g., polynomial evaluations) and b is the fixed overhead (e.g., constant-time cryptographic routines and I/O operations).

As confirmed by benchmarking data from [20] and the protocol comparison from [19], Groth16 exhibits a linear trend in proof time with increasing constraint count. With a precomputed trusted setup, proof generation becomes the dominant cost and scales predictably with circuit complexity.

Alternative Proving Schemes. In our proposal, we only explored the protocol variant which uses SNARKs with the Groth16 proving scheme. Groth16 is a good fit due to two properties – its constant proof size and its proving time being asymptotically linear with respect to the circuit's constraint count (see Section 6). The ability to verify block correctness necessitates the storage of proofs on-chain. With Groth16, the actual proof only takes up 160 B. Alternative schemes with a constant proof size could be used. The proving time does not necessarily need to be linear; it just needs to be predictable based on the constraint count in the circuit.

Puzzle Condition Analysis. In Section 3, we outlined the essential conditions for a secure and effective PoW protocol. The following discusses how our proposed protocol addresses each of these requirements:

- **Asymmetry**: SNARK proof generation and verification time fits very well into this condition. The verification time is in the order of milliseconds and is constant to the complexity of the related circuit (see Section 6).
- **Granularity and Parametrization**: Target difficulty can be adjusted to control the expected block time, allowing for fine-grained tuning of the block production rate.
- **Amortization-freeness**: While methods for batching the verification of Groth16 proofs exist [7], no analogous techniques have yet been developed for batching the proving process. Although batching methods for proof generation are being actively explored [27], these efforts currently focus on protocols other than Groth16.
- **Independence**: There is no work shared between the generation of two separate proofs, and no advantage is gained for generating one after the other, even if they use the same circuit with different parameters. Even if there were two proof transactions in the mempool with the same circuit and parameters, their proof generation would still constitute completely independent work, since they differ in the integrity parameter.
- **Efficiency**: The network introduces some inefficiencies, such as the use of SMPC for the trusted setup. However, these costs are amortized as circuits are reused with different parameters, which is a behavior encouraged by the high registration fee for new circuits. The actual proof computation is modified by introducing a single new parameter that ensures integrity, but this change has minimal impact on resource usage.
- **Unforgeability**: At first glance, it may appear that a miner could create their own pending proof transactions, withhold them from the network, and begin secretly precomputing proofs. However, because the integrity parameter of proofs depends on the previous block's hash, miners cannot precompute

or forge proofs in advance without knowing the future chain state, preventing secret manipulation of pending proofs.
- **Freshness**: As mentioned in Section 4.3, the integrity parameter of each proof serves as a binding commitment between the proof and its containing block, ensuring that the proof cannot be reused or hijacked in another block without invalidating the verification.
- **Deterministic**: As mentioned in Section 6 the constraint count constitutes a deterministic cost metric expressing the expected proving time.
- **Progress-free**: The chance for a miner to win the current round is reset after each proof is generated, thus creating a progress-free environment.
- **Publicly Verifiable**: Proofs are published on the blockchain for public verification, with the verifying key provided by any member of the circuit registry subnetwork (see Section 4.4).

7 Related Work

This section reviews existing literature in two domains: Proof of Useful Work (PoUW) protocols and marketplaces for SNARK proof generation.

Proof of Useful Work. Primecoin [32] introduced in 2013 pioneered practical PoUW by having miners search for prime number chains, contributing to number theory. Ball et al. [5] formalized PoUW using hard problems such as orthogonal vectors, addressing the usefulness through randomized challenges of prior block hashes. Later works applied NP-hard problems, such as combinatorial optimization in Ofelimos [22] and minimal dominating sets in Chrisimos [9]. Other works [42,10] have focused on creating frameworks for solving general real-life optimization problems. Other approaches use matrix multiplication [33] or machine learning, such as distributed deep learning in Coin.AI [4] and scalable ML training in PoGO [38] through quantized gradients and Merkle proofs. Generalized frameworks encompass PUPoW [10] for practical blockchains, Proofware [16] for dApps, COCP [14], and crowdsourcing paradigms [11] to integrate general real-life tasks into the mining process. Nevertheless, none of these works satisfy all the required properties of PoW (see Section 3). In particular, freshness is the most critical property, as its absence allows adversaries to appropriate recently computed solutions.

Proof Generation Marketplaces. The computational demands of the generation of the zk-SNARK proof have spurred marketplaces, divided into consensus-integrated and standalone models. Consensus-integrated systems [31] embed proof in blockchain consensus using PoUW SNARKs for state verification, although limited to fixed purpose-specific circuits without arbitrary proof support. In particular, Mina protocol [8] employs recursive SNARKs to maintain a constant state (as well as a constant size), but restricts recursion to block validation. Its Snarketplace [35] enables block producers to buy proofs from SNARK workers, operating above the consensus layer. In contrast, our approach integrates

PoUW directly at the consensus layer, requiring miners to fulfill SNARK proof generation requests for valid block creation. Standalone networks such as =nil; Foundation Proof Market [34] offer a decentralized exchange with SNARK order books, and RISC Zero's Bonsai [40] as a zkVM-based coprocessor for arbitrary programs in standard languages for dApps.

8 Conclusion

Due to PoW's high energy use and environmental impact, alternatives with their specific trade-offs have emerged, where PoUW offers a promising yet challenging approach to maintaining PoW's security without compromising its core properties. Additionally, zk-SNARKs are critical for scalability and privacy in many blockchain networks, but their generation is too resource-intensive for many users. Several zk-SNARK marketplaces emerged to offload their generation.

We design the PoUW protocol where SNARK proof generation constitutes the work in the PoUW consensus mechanism, providing chain security benefits of PoW while producing SNARK proofs valuable in broader cryptographic applications as a byproduct. To the best of our knowledge, our solution is the first one that meets all the necessary properties of PoW in contrast to prior work.

References

1. 0KIMS Association: Circom: A high-level language for zk-snark circuits (2021), https://github.com/iden3/circom, accessed: 2025-01-08
2. Ali, I.M., Caprolu, M., Pietro, R.D.: Foundations, properties, and security applications of puzzles: A survey. ACM Comput. Surv. **53**(4) (Aug 2020). https://doi.org/10.1145/3396374, https://doi.org/10.1145/3396374
3. Anceaume, E., Guellier, A., Ludinard, R., Sericola, B.: Sycomore: A permissionless distributed ledger that self-adapts to transactions demand. In: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA). pp. 1–8 (2018). https://doi.org/10.1109/NCA.2018.8548053
4. Baldominos, A., Saez, Y.: Coin.ai: A proof-of-useful-work scheme for blockchain-based distributed deep learning. Entropy **21**(8) (2019). https://doi.org/10.3390/e21080723, https://www.mdpi.com/1099-4300/21/8/723
5. Ball, M., Rosen, A., Sabin, M., Vasudevan, P.N.: Proofs of useful work. Cryptology ePrint Archive, Paper 2017/203 (03 2017), https://eprint.iacr.org/2017/203
6. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: Proceedings of the 23rd USENIX Conference on Security Symposium. p. 781–796. SEC'14, USENIX Association, USA (2014)
7. Blazy, O., Fuchsbauer, G., Izabachène, M., Jambert, A., Sibert, H., Vergnaud, D.: Batch groth-sahai. Cryptology ePrint Archive, Paper 2010/040 (2010), https://eprint.iacr.org/2010/040
8. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Mina: Decentralized cryptocurrency at scale. Tech. rep., O(1) Labs (2020), https://minaprotocol.com/wp-content/uploads/technicalWhitepaper.pdf

9. Chatterjee, D., Banerjee, P., Mazumdar, S.: Chrisimos: A useful proof-of-work for finding minimal dominating set of a graph. In: 2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 1332–1339 (2023). https://doi.org/10.1109/TrustCom60117.2023.00182
10. Chaurasia, Y., Subramanian, V., Gujar, S.: Pupow: A framework for designing blockchains with practically-useful-proof-of-work & vanitycoin. In: 2021 IEEE International Conference on Blockchain (Blockchain). pp. 122–129 (2021). https://doi.org/10.1109/Blockchain53845.2021.00026
11. Chen, C., Cheng, Z., Qu, S., Fang, Z.: Crowdsourcing work as mining: A decentralized computation and storage paradigm. In: Proceedings of the 7th Asia-Pacific Workshop on Networking. p. 174–175. APNet '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3600061.3603177, https://doi.org/10.1145/3600061.3603177
12. Chiesa, A., Koutsou, E., Williams, D.: zk-rollups: Scalable privacy-preserving layer-2 solutions for blockchain. arXiv **abs/2101.11134** (2021), https://arxiv.org/abs/2101.11134, accessed: 2025-03-09
13. Costan, V., Devadas, S.: Intel SGX explained. Tech. Rep. 2016/086, IACR Cryptology ePrint Archive (2016), https://eprint.iacr.org/2016/086
14. Davidović, T., Todorović, M., Ramljak, D., Krüger, T.J., Matijević, L., Jovanović, D., Urošević, D.: Cocp: Blockchain proof-of-useful-work leveraging real-life applications. In: 2022 Fourth International Conference on Blockchain Computing and Applications (BCCA). pp. 107–110 (2022). https://doi.org/10.1109/BCCA55292.2022.9922117
15. Domenech, A.A., Heiss, J., Tai, S.: Servicifying zk-snarks execution for verifiable off-chain computations (2024), https://arxiv.org/abs/2404.16915
16. Dong, Z., Lee, Y.C., Zomaya, A.Y.: Proofware: Proof of useful work blockchain consensus protocol for decentralized applications (2019), https://arxiv.org/abs/1903.09276
17. Dotan, M., Tochner, S.: Proofs of useless work – positive and negative results for wasteless mining systems (2021), https://arxiv.org/abs/2007.01046
18. Ebrahimi, E., Sober, M., Hoang, A.T., Ileri, C.U., Sanders, W., Schulte, S.: Blockchain-based federated learning utilizing zero-knowledge proofs for verifiable training and aggregation. In: 2024 IEEE International Conference on Blockchain (Blockchain). pp. 54–63 (2024). https://doi.org/10.1109/Blockchain62396.2024.00017
19. El-Hajj, M., Oude Roelink, B.: Evaluating the efficiency of zk-snark, zk-stark, and bulletproof in real-world scenarios: A benchmark study. Information **15**(8) (2024). https://doi.org/10.3390/info15080463, https://www.mdpi.com/2078-2489/15/8/463
20. Ernstberger, J., Chaliasos, S., Kadianakis, G., Steinhorst, S., Jovanovic, P., Gervais, A., Livshits, B., Orrù, M.: zk-bench: A toolset for comparative evaluation and performance benchmarking of SNARKs. Cryptology ePrint Archive, Paper 2023/1503 (2023), https://eprint.iacr.org/2023/1503
21. Falk, M., Hüsler, J., Reiss, R.: Laws of Small Numbers: Extremes and Rare Events. Birkhäuser Basel (2013), https://books.google.ca/books?id=LIP1BwAAQBAJ
22. Fitzi, M., Kiayias, A., Panagiotakos, G., Russell, A.: Ofelimos: Combinatorial optimization via proof-of-useful-work. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. pp. 339–369. Springer Nature Switzerland, Cham (2022)

23. Gentry, C.: A Fully Homomorphic Encryption Scheme. Dissertation, Stanford University (2009), https://crypto.stanford.edu/craig/craig-thesis.pdf
24. Goldwasser, S., Micali, S., Rackoff, C.: Knowledge complexity of interactive proof-systems. SIAM Journal on Computing **18**(6), 186–208 (1985)
25. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
26. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11, https://doi.org/10.1007/978-3-662-49896-5_11
27. Henry, R., Goldberg, I.: Batch proofs of partial knowledge. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) Applied Cryptography and Network Security. pp. 502–517. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
28. Hess, Z., Malahov, Y., Pettersson, J.: Æternity blockchain. Tech. rep. (2017), https://blockchainlab.com/pdf/%91ternity-blockchain-whitepaper.pdf, version 0.1
29. Ileri, A.M., Ozercan, H.I., Gundogdu, A., Senol, A.K., Özkaya, M.Y., Alkan, C.: Coinami: A cryptocurrency with DNA sequence alignment as proof-of-work. CoRR **abs/1602.03031** (2016), http://arxiv.org/abs/1602.03031
30. Jin, Y., Wang, T., Yang, Q., Shi, L., Zhang, S.: Zero-knowledge federated learning: A new trustworthy and privacy-preserving distributed learning paradigm (2025), https://arxiv.org/abs/2503.15550
31. Kattis, A., Bonneau, J.: Proof of necessary work: Succinct state verification with fairness guarantees. Cryptology ePrint Archive, Paper 2020/190 (2020), https://eprint.iacr.org/2020/190
32. King, S.: Primecoin: Cryptocurrency with prime number proof-of-work. https://primecoin.io/primecoin-paper.pdf (2013)
33. Komargodski, I., Schen, I., Weinstein, O.: Proofs of useful work from arbitrary matrix multiplication (2025), https://arxiv.org/abs/2504.09971
34. Komarov, M.: =nil; proof market. https://nil.foundation/blog/post/proof-market (Jan 2023), accessed: 2025-07-18
35. Mina Foundation: What are snark workers and the snarketplace? https://minaprotocol.com/blog/what-are-snark-workers-and-the-snarketplace (Aug 2021), accessed: 2025-07-18
36. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), http://www.bitcoin.org/bitcoin.pdf
37. Nakamura, M., Miyamae, T., Morinaga, M.: A privacy-preserving outsourcing scheme for zero-knowledge proof generation. Journal of Information Processing **30**, 151–154 (02 2022). https://doi.org/10.2197/ipsjjip.30.151
38. Orlicki, J.I.: Pogo: A scalable proof of useful work via quantized gradient descent and merkle proofs (2025), https://arxiv.org/abs/2504.07540
39. Percival, C.: Stronger key derivation via sequential memory-hard functions. In: Proceedings of the 2009 BSDCan Conference (May 2009), https://www.tarsnap.com/scrypt/scrypt.pdf
40. RISC Zero: Bonsai network blockchain litepaper. https://dev.risczero.com/litepaper, accessed: 2025-07-18
41. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176, https://doi.org/10.1145/359168.359176

42. Todorović, M., Matijević, L., Ramljak, D., Davidović, T., Urošević, D., Jakšić Krüger, T., Jovanović, D.: Proof-of-useful-work: Blockchain mining by solving real-life optimization problems. Symmetry **14**(9) (2022). https://doi.org/10.3390/sym14091831, https://www.mdpi.com/2073-8994/14/9/1831
43. Tornado Cash: Tornado cash: A privacy tool for ethereum (2021), https://github.com/tornadocash/tornado-core, accessed: 2025-03-09
44. de Vries, A., Gallersdörfer, U., Klaaßen, L., Stoll, C.: Revisiting bitcoin's carbon footprint (02 2022). https://doi.org/10.1016/j.joule.2022.02.005
45. Westerkamp, M., Eberhardt, J.: zkrelay: Facilitating sidechains using zksnark-based chain-relays. In: IEEE Security & Privacy on the Blockchain (IEEE S&B 2020) (2020), https://eprint.iacr.org/2020/433
46. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkbridge: Trustless cross-chain bridges made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 3003–3017 (2022). https://doi.org/10.1145/3548606.3560652, https://doi.org/10.1145/3548606.3560652
47. Yao, A.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (FOCS '86). pp. 162–167. IEEE Computer Society, Washington, DC, USA (1986). https://doi.org/10.1109/SFCS.1986.25, https://doi.org/10.1109/SFCS.1986.25
48. ZoKrates Team: ZoKrates: A toolbox for zkSNARKs on Ethereum (11 2023), https://zokrates.github.io/, accessed: 2024-05-08, version 0.8.8

A Block Production and Verification Pseudocode

Algorithm 2 shows pseudocode of the most important functions for block generation and verification of a block as well as registration of a proof within the circuit registration subnetwork.

Algorithm 2 Pseudocode of essential functions.

```
function produceBlock(κ)
    blkHdr ← createBlkHdr()                                   ▷ Timestamp, etc.
    coinTxs ← {createCoinbaseTx()} ∪ mempool.selectValidCoinTxs()
    proofTxs ← {}
    blkHash ← H(blkHdr || coinTxs || proofTxs)
    repeat                                                    ▷ Build the proof chain
        proofTx ← mempool.selectValidProofTx()
        proofTx.proof ← zkLib.genProof(proofTx, blkHash)
        proofTxs ← proofTxs ∪ {proofTx}
        blkHash ← H(blkHdr || coinTxs || proofTxs)
    until blkHash < κ * proofTx.complexity                    ▷ Lottery-weighted difficulty threshold
    return blkHdr, coinTxs, proofTxs
end function

function verifyBlock(blk)
    assert(isBlkHdrValid(blk.hdr))                            ▷ Timestamp, etc.
    assert(areCoinTxsValid(blk.coinTxs))
    checkedProofTxs ← {}
    ▷ Iteratively build and verify proof chain
    for proofTx in blk.proofTxs do
        iterHash ← H(blk.hdr || coinTxs || checkedProofTxs)
        assert(zkLib.verifyProof(proofTx, iterHash))
        checkedProofTxs ← checkedProofTxs ∪ {proofTx}
    end for
end function

function genProof(proofTx, integrity)
    hash ← proofTx.circuitHash
    circuit, complexity, provingKey ← registry.getCircuit(hash)
    args ← proofTx.parameters || integrity
    proof ← zkLib.generateProof(circuit, provingKey, args)
    return proof
end function

function registerCircuit(sourceCode, burnTxId)
    compiledCircuit ← zkLib.compile(sourceCode)
    complexity ← zkLib.getConstraintCount(compiledCircuit)
    circuitHash ← H(compiledCircuit)
    if registry.doesCircuitExist(circuitHash) then
        return circuitHash                                    ▷ Reject duplicate circuit registration
    end if
    ▷ Perform SMPC for trusted setup and register circuit
    feeTx ← blockchain.getTxById(burnTxId)
    assert(isValid(feeTx))
    assert(feeTx.amount >= complexity * pricePerGate)
    provingKey, verifKey ← registry.performSMPC(compiledCircuit)
    registry.store(circuitHash, complexity, provingKey, verifKey)
    registry.distributeRewards(feeTx.amount)                  ▷ To SMPC contributing nodes
    return circuitHash
end function
```

B Zk-SNARK Puzzle Extension

The current zk-SNARK marketplace proposal requires all inputs, including the private witness, to be publicly submitted to the blockchain, breaking the zero-knowledge property essential to zk-SNARKs. Supporting private inputs in outsourced proof generation is critical to maintaining privacy while leveraging external workers for computation. Several techniques have been explored, ranging from basic approaches to advanced cryptographic methods, evaluated based on key parameters: **confidentiality** (ensuring private inputs remain hidden from workers), **open-proving** (allowing any participant to act as a worker without restrictions), **non-interactivity** (requiring no communication between client and worker), and **feasibility** (practicality in terms of computational and integration complexity).

Naive Approach. A basic approach is to directly share private inputs with the external workers for zk-SNARK proof generation. The client submits a proof request on-chain and sends the inputs to any worker via a secure channel. If multiple workers are considered, inputs must be sent to each, increasing communication overhead. This method fully compromises confidentiality, as workers gain complete access to sensitive data, violating the zero-knowledge property.

Fully-Fledged Options. To overcome the limitations of a naive approach, we outlined several options, drawing from established methods in secure computing and cryptography:

- **Multiparty Computation (MPC)**: The client splits the witness using a secret-sharing scheme (e.g., Shamir's [41]) and distributes shares to multiple workers. A secure MPC protocol [47] enables joint proof generation without revealing the full input. The final proof is aggregated and submitted on-chain. MPC ensures confidentiality and supports open-proving, but the overhead increases with the circuit size and the worker count. Fewer nodes increase the risk of Sybil risks. Integration is complex due to consensus modifications.
- **Fully Homomorphic Encryption (FHE)**: The client encrypts the witness using FHE [23] and includes it in the proof request. Workers compute the proof homomorphically and return an encrypted result without seeing the inputs. FHE provides strong privacy and non-interactivity, enabling open-proving. However, it remains highly inefficient for large circuits. No end-to-end zk-SNARK FHE solution is practical as of now.
- **Trusted Execution Environments (TEEs)**: A TEE-enabled worker decrypts the witness inside a secure enclave (e.g., Intel SGX [13]) and generates the proof, submitting it with a hardware attestation. TEEs offer confidentiality via hardware isolation and enable non-interactive proofs, but they restrict proving to specific hardware, introduce the necessity for vendor trust, and add complexity due to attestation and environment adaptation.
- **Witness Obfuscating Outsourcing (WOO)**: The client blinds each private input s_{in} with a random value r_{in}, producing $s'_{\text{in}} = s_{\text{in}} + r_{\text{in}}$ [37]. The

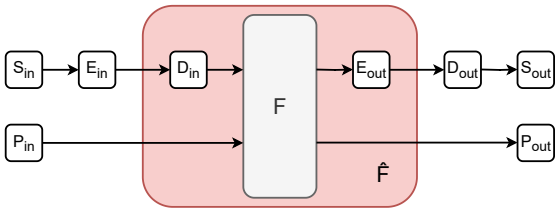

Fig. 10: Structure of the transformed circuit $\hat{F}$, where encrypted input $\hat{s}_{\text{in}}$ passes through the input decryption function D_{in}, and the output $\hat{s}_{\text{out}}$ is later decrypted using D_{out}. Adapted from [37].

obfuscated witness is included in the proof request, allowing the worker to generate the proof without accessing the original inputs. WOO provides strong confidentiality and enables open-proving without special hardware. It is non-interactive and integrates well with existing zk-SNARK tools, adding minimal overhead. The main drawback is the need to adapt the circuits to handle obfuscated inputs, requiring a light preprocessing step on the client side. This step remains compatible with standard zk-SNARK frameworks.

WOO stands out for its balance of confidentiality, open-proving, non-interactivity, and feasibility, making it the preferred solution for maintaining privacy in outsourced zk-SNARK proof generation within our decentralized marketplace.

B.1 Encrypting Private Inputs

To protect the confidentiality of the private inputs, the client uses additive masking. For each private input s_{in}, a random value r_{in} is sampled uniformly from a large finite field. The obfuscated input is calculated as:

$$\hat{s}_{\text{in}} = s_{\text{in}} + r_{\text{in}}. \tag{4}$$

These masked inputs form the obfuscated witness included in the proof transaction submitted to the blockchain mempool. The client keeps r_{in} secret, ensuring the obfuscated values reveal no information about the original inputs. This provides information-theoretic security, as $\hat{s}_{\text{in}}$ appears uniformly random to any observer without r_{in}. The process is computationally lightweight, making it highly suitable for integration into our PoUW protocol.

B.2 Circuit Compilation and Masking Parameters

The masking parameters r_{in}, controlling how random values are applied, are provided as function arguments during circuit compilation. Using frameworks like Zokrates [48] or Circom [1], the high-level computation is converted into an R1CS form suitable for zk-SNARK proving. To support masked inputs, the circuit includes an input decryption function D_{in}, which recovers the original value internally using:

$$s_{\text{in}} = \hat{s}_{\text{in}} - r_{\text{in}}. \tag{5}$$

Here, r_{in} is hard-coded during compilation to ensure the logic cannot be tampered with. The resulting circuit $\hat{F}$, shown in Figure 10, preserves the original computation but operates entirely on obfuscated inputs, enabling private and verifiable proofs.

B.3 Trusted Setup and Proving Key

After compiling the circuit, a trusted setup, typically via MPC among circuit registry nodes, generates the proving and verification keys. The client contributes initial entropy, embedding masking parameters into the proving key while keeping the R1CS and masks private. Only the proving key and obfuscated witness are sent to the worker, keeping inputs and the circuit hidden. The verification key is published on-chain, allowing standard proof verification without changes to the consensus protocol.

B.4 Proof Workflow

The WOO-based proof process is divided into two stages:

Client's Workflow.

1. **Write WOO Circuit**: Design a circuit with input decryption logic D_{in} using Circom or Zokrates.
2. **Generate Randomness**: Sample random $r_{\text{in}} \in \mathbb{F}$ for each private input s_{in}.
3. **Compile Circuit**: Compile into an R1CS circuit with r_{in} hard-coded for use with $\hat{s}_{\text{in}}$.
4. **Delegate Trusted Setup**: Launch an MPC-based trusted setup, contributing initial entropy.
5. **Generate Obfuscated Witness**: Apply additive masking to produce $\hat{s}_{\text{in}}$.
6. **Request Proof**: Submit a proof transaction containing $\hat{s}_{\text{in}}$, keeping r_{in} secret.

Worker's Workflow.

1. **Select Proof Request**: Retrieve a proof transaction from the mempool with the obfuscated witness and a reference to circuit $\hat{F}$.
2. **Generate zk-SNARK Proof**: Using the proving key, compute a standard zk-SNARK proof (e.g., via Groth16 [26]) from the obfuscated witness.
3. **Submit Block**: Broadcast a block containing the proof, meeting PoUW consensus difficulty along with other transactions.

B.5 Wrapping Metadata for Integrity and Authentication

To enforce contextual proof binding while preserving zero-knowledge, the system wraps the original zk-SNARK proof inside an outer circuit. The inner proof is

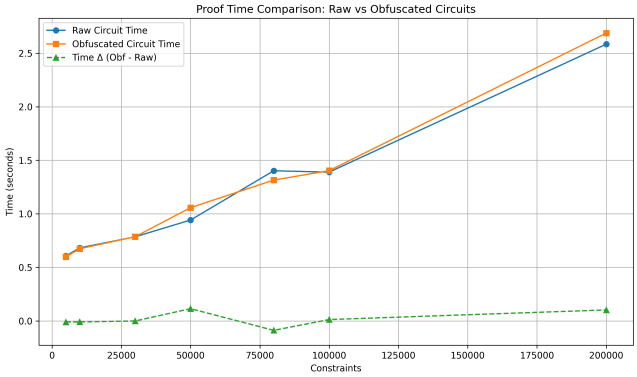

Fig. 11: Proof time comparison for raw vs. obfuscated Circom circuits. The green curve shows the difference between obfuscated and raw proofs ($\Delta = T_{\text{obf}} - T_{\text{raw}}$), which remains low across all constraint sizes. Benchmarked using Circom and snarkjs on an AMD Ryzen 5 7600X with 32 GB RAM.

a private input, while the outer circuit includes public inputs like the previous block hash and miner address. It verifies the inner proof without exposing its contents.

Only the outer proof is published on-chain, ensuring confidentiality while binding the computation to specific metadata and blockchain state. During block validation, the network verifies the outer proof and its references, securing consensus without revealing sensitive data.

Performance Evaluation. To assess the impact of Witness Obfuscating Outsourcing (WOO) on proof generation time, we benchmarked obfuscated and raw circuits using Circom and Groth16, across constraint sizes from 10,000 to 200,000 under identical conditions. As shown in Figure 11, WOO introduces a small, consistent overhead due to embedded decryption logic. This added latency remains below 0.1 seconds, even for large circuits, and does not scale significantly with circuit size.

Overall, WOO maintains high efficiency and scalability, making it well-suited for real-time, privacy-preserving proof generation in the PoUW marketplace.

Conclusion. The Witness Obfuscating Outsourcing (WOO) mechanism strengthens privacy in zk-SNARK proof generation by hiding private inputs from external provers using additive masking and embedded decryption logic. This enables open-proving without exposing confidential data, even in decentralized settings.

Compared to Nakamura et al. [37], who rely on a trusted third party (TTP) to transform the circuit and generate the common reference string (CRS), WOO eliminates the TTP by leveraging an MPC-based setup with client-initiated entropy. Furthermore, WOO keeps both the circuit (R1CS) and inputs hidden from workers, sharing only the proving key and obfuscated witness, which further reduces the trust assumptions and improves decentralization. However, this

enhanced privacy reduces the circuit reusability. Since obfuscation embeds client-specific randomness, circuits must be generated individually per client or transaction, increasing setup effort and reducing generality.

In summary, WOO improves confidentiality and trust minimization compared to prior approaches, but introduces a trade-off between privacy and flexibility in decentralized zk-SNARK outsourcing.