Regulating Storage Overhead in Existing PoW-based Blockchains

Frederik Armknecht
University of Mannheim
Germany
armknecht@uni-mannheim.de

Jens-Matthias Bohli*
Mannheim University of Applied Sciences
Germany
j.bohli@hs-mannheim.de

Ghassan O. Karame
NEC Laboratories Europe
Germany
ghassan@karame.org

Wenting Li
NEC Laboratories Europe
Germany
wenting.li@neclab.eu

ABSTRACT

Proof of Work (PoW) blockchains regulate the frequency and security of extensions to the blockchain in a decentralized manner by adjusting the difficulty in the network. However, analogous decentralized measures to regulate the replication level of the associated transactions and blocks data are completely missing so far. We argue that such measures are required as well. On the one hand, the smaller the number of replicas, the higher the vulnerability of the system against compromises and DoS-attacks. On the other hand, the larger the number of replicas, the higher the storage overhead, and the higher the operational blockchain cost are.

In this paper, we propose a novel solution, EWoK (Entangled proofs of WOrk and Knowledge), that regulates in a decentralized manner the minimum number of replicas that should be stored by miners in the blockchain. EWoK achieves this by tying replication to the only directly-incentivized process in PoW-blockchains – which is PoW itself. EWoK only incurs small modifications to existing PoW protocols and is fully compliant with the specifications of existing mining hardware. Our implementation results confirm that EWoK can be easily integrated within existing mining pool protocols, such as GetBlockTemplate and Stratum mining, and does not impair the mining efficiency.

1 INTRODUCTION

Blockchains realize a conflict-free, "extend-only", distributed data structure without the need for a central root of trust. Proof of work (PoW) based blockchains such as Bitcoin and Ethereum account for more than 90% of the total market capitalization of existing digital cryptocurrencies [10]. These blockchains enable miners to "vote" with their computing power on the next set of transactions to be added by "mining" blocks—which effectively limits the power of individual users and makes Sybil attacks difficult. The difficulty in PoW mining is self-adjusted by the network to ensure that the block generation time (and therefore the system *throughput*) lies within reasonable bounds [16].

However, similar mechanisms that attempt to regulate the *storage* of the associated blocks/transactions are missing so far. This resulted in a sharp variance in the number of blockchain replicas over time. Namely, during the early years of PoW blockchains, every miner was also a "full-node" and stored a full copy of the blockchain. As a result, the blockchain witnessed an unprecedented level of replication (>200,000 replica) until early 2014 [14, 15]. Nowadays, the current difficulty level of PoW mining is prohibitively high enough that miners do not have incentives to operate solo. Instead, joining a mining pool emerges as an attractive option for miners to receive a portion of the block reward on a consistent basis. Here, workers do not connect directly to the blockchain; instead, they only connect to the pool operator which defines their search space parameters (e.g., workers are typically required to solve a PoW with a reduced difficulty). In fact, most existing workers run dedicated mining protocols, such as Stratum mining (STM) [25] or GetBlockTemplate (GBT) [17], in which they solve specific outsourced PoW puzzles without having to store any parts of the blockchain. This resulted in a sharp decrease in the number of full nodes from hundreds of thousands to a few thousands of nodes [14, 15].

While, in theory, ensuring that the system has a few "trusted" replicas suffices for security, the essence of blockchain is not to rely on such "trusted nodes". This makes it essential that a *sufficient* number of replicas are stored within the system. However, this number depends on many aspects of the system and could dynamically change in the future. On the one hand, the smaller the number of replicas, the higher the vulnerability of the system against compromises and DoS-attacks. On the other hand, the larger the number of replicas, the higher the storage overhead, and the higher the operational blockchain cost. *In many ways, this is analogous to the need of regulating block generation times using PoW difficulty: a large difficulty increases the block generation times and lowers system throughput, while a low difficulty decreases the generation time and increases the throughput.*

In this paper, we propose a solution, EWoK (Entangled proofs of WOrk and Knowledge), to regulate in a decentralized manner the level of ledger replication amongst miners (and mining pools) in the network. EWoK ties the replication to the only directly-incentivized process in PoW-blockchains—which is mining itself. To this end, EWoK effectively divides

*Work done while author was affiliated with NEC Laboratories Europe the blockchain into dynamically adapting partitions and requires mining pool workers to use different partitions in order to correctly solve the standard hash-based PoW. EWoK allows to freely scale the size of the partitions to fit them into the memory of typical miner hardware. This encourages pool workers, i.e., parties that aim to maximize their efficiency, to store the individual partitions locally for faster access instead of fetching the information from a remote external resource. In doing so, EWoK ensures a minimum bound on the number of replicas in the system as mandated by a system-wide parameter. We implemented and evaluated a prototype based on EWoK using GPU mining and integrated it with STM and GBT. Our results show that EWoK does not require changes in existing mining hardware, and achieves a 2% higher hash rate than GBT and only deteriorates the hash rate of STM by 1%.

Related Work. The literature features a number of contributions that propose a re-purposing of the PoW to e.g., prove storage of archival data, but require either drastic changes or are not applicable in practice. Note that most PoW-based blockchains, such as Bitcoin, use a hash-based PoW to find an input that is hashed to a value with certain properties (see Example 2.2 in Section 2.2). That is, a PoW usually requires *many* hash executions over *randomly chosen* inputs. In Permacoin [18] and Retricoin [23], the PoW is replaced by a Proof of Retrievability (PoR) over a large archival data file. Here, the archival data file needs to be known a-priori in its entirety to get extended by a maximum-distance-separable code and to allow the miners to pick their own selection of file shares to be stored in their local storage. That is, these systems do not support a dynamically increasing file—such as the blockchain ledger.

Spacemint [21], Burstcoin [8], and Filecoin [13] aim to replace PoW by a Proof of Space. This means that a miner has to prove that a certain amount of storage has been invested (instead of computation as in the case of a PoW). This would require likewise drastic changes in existing mining hardware. PieceWork [2, 11] leverages a 2-phase PoW protocol similar to EWoK; it is however not designed to incorporate proofs of knowledge over blockchain data. BetterHash [9] is a newly proposed mining protocol that allows the workers to construct their own block template while benefiting from the payout as a pool worker. It aims to encourage miners to become full nodes by removing the restriction that only the pool coordinator decides the block template. However, BetterHash does not incentivize miners to store blockchain data.

Summing up, all existing approaches either do not solve the problem that we target or would require additional investments in different types of machinery, e.g., storage and exponentiation-optimized machinery. Due to their large market capitalization, it is clear that drastic changes to the rules governing the dynamics of the PoW ecosystem will be widely resisted by the backing industry. In contrast, EWoK does not require any modification to PoW nor to the underlying blockchain (it does not require any soft/hard-forks), is fully compliant with the specifications of existing mining hardware, and only incurs minor changes to the existing PoW protocols. Hence, EWoK emerges as the first workable attempt to regulate the dispersal of the blockchain data amongst various workers.

Outline. In Section 2, we briefly recall basic information about blockchains based on Proofs of Work (PoW) and shed lights on their shortcomings. In Section 3, we introduce our extension of PoW-based blockchains, p-covering blockchain, which ensures that the blockchain is stored among the workers with a specified level of redundancy. In Section 4, we detail EWoK and analyze its security. In Section 5, we evaluate a prototype implementation based on the integration of EWoK with GPU mining based on STM and GBT, and we conclude the paper in Section 6.

2 PRELIMINARIES

2.1 Notations.

λ denotes the security parameter and $\mathsf{negl}(\lambda)$ a function that is negligible in λ. We denote by H a cryptographically secure hash function that accepts bitstrings of arbitrary length. For the sake of readability, if the input x is the concatenation of several individual strings $x_1, x_2, \ldots$, we simply write $H(x_1, x_2, \ldots)$ instead of $H(x_1||x_2|| \ldots)$. For a probabilistic algorithm A, we denote by $y \leftarrow \mathsf{A}(x)$ the event that A on input x outputs y. In case that the random coins ρ need to be specified explicitly, we write $y \leftarrow \mathsf{A}(x, \rho)$. Otherwise, we omit these to keep the formalism short and assume that all probabilities are also over the random coins. To capture the notion of effort, we express by $\mathsf{Steps}_{\mathsf{A}}(x)$ the number of steps (i.e., machine/operation cycles) executed by algorithm A on input x. This includes also idle steps, e.g., when an algorithm has to wait for some data. $x \overset{\$}{\leftarrow} X$ means that an element x has been sampled uniformly from a set X. When we consider a function $g = \mathsf{id}$, we refer to the identity function.

2.2 PoW-Based Blockchains

A blockchain $BC = [BC[0], BC[1], \ldots]$ is a list where new data, i.e., blocks, are appended within each time epoch. A *blockchain system* is composed of a blockchain BC and a set $\mathcal{W}$ of workers who maintain the blockchain[1].

To regulate content addition to the blockchain, most existing (open) blockchains (e.g., Bitcoin, Litecoin, Ethereum) require the workers to solve a *proof of work* (PoW) first before appending a block. We adapt and extend the model from [3] to formalize the notion of PoW. Roughly speaking, a PoW system allows to generate *cryptographic puzzles* that are easy to generate, hard to solve, and whose potential solutions are easy to verify. More formally, it comprises a parameter space PP, a puzzle space PS, a solution space SS, and a publicly known

[1]For the sake of simplicity, we restrict to the case that the set of workers $\mathcal{W}$ is fixed. It is straightforward to extend the definition (and the follow-up discussion) to capture dynamically changing sets of workers $\mathcal{W}_t$.

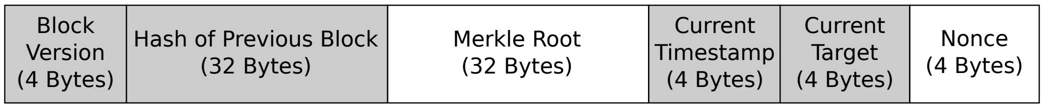

Figure 1: Sketch of the PoW block header structure in Bitcoin. The grey area indicates system-wide parameters.

verification parameter $\mathsf{pp_{veri}}$. Moreover, there exist three algorithms Sample, Solve, and Verify. $\mathsf{Sample(pp_{puz})}$ takes a puzzle parameter $\mathsf{pp_{puz}} \in \mathsf{PP}$ and outputs a puzzle instance $\mathsf{puz} \in \mathsf{PS}$. $\mathsf{Solve(puz)}$ is a probabilistic puzzle solving algorithm that on input a puzzle instance $\mathsf{puz} \in \mathsf{PS}$ outputs a potential solution $\mathsf{sol} \in \mathsf{SS}$. $\mathsf{Verify(pp_{veri}, puz, sol)}$ is a deterministic puzzle verification algorithm that validates whether sol is a solution to the puzzle puz.

Naturally, a PoW system should be *complete* and *sound*. Completeness means that a solution found by Solve for a correctly sampled puzzle will always be accepted. Soundness means that for any attacker $\mathcal{A}$, the effort to solve a puzzle cannot be lower (up to a function g) than the effort for running the "official" solving algorithm Solve. To simplify the presentation of the following definitions, we introduce

$$\mathsf{Event}(\mathsf{pp_{puz}}, \mathsf{A}, g) := \begin{cases} \mathsf{puz} \leftarrow \mathsf{Sample}(\mathsf{pp_{puz}}) \wedge \mathsf{sol} \leftarrow \mathsf{A}(\mathsf{puz}) \wedge \\ \mathsf{Verify}(\mathsf{pp_{veri}}, \mathsf{puz}, \mathsf{sol}) = \mathtt{TRUE} \wedge \\ \mathrm{Steps}_{\mathcal{A}}(\mathrm{puz}) \leq g(\mathrm{Steps}_{\mathsf{Solve}}(\mathrm{puz})) \end{cases}$$

$\mathsf{Event}(\mathsf{pp_{puz}}, \mathsf{A}, g)$ expresses the event that a puzzle has been sampled with puzzle parameter $\mathsf{pp_{puz}}$, some algorithm A produced a correct solution, and the time effort deviated from the effort of Solve up to a function g.[2] The parameter g will be used to characterize the soundness of PoW in the sense that requiring less effort than given by g is unlikely:

Definition 2.1 (PoW Soundness). A PoW system is *g-sound* for some function g if for every PPT adversary $\mathcal{A}$ and every $\mathsf{pp_{puz}} \in \mathsf{PP}$, it holds that $\Pr\left[\mathsf{Event}(\mathsf{pp_{puz}}, \mathcal{A}, g)\right] = \mathrm{negl}(\lambda)$. A PoW-based blockchain is g-sound for some function g if appending new blocks requires to solve one g-sound PoW.

All parties in $\mathcal{W}$ use the same PoW system and the same puzzle sampling algorithm Sample but can select the puzzle parameter $\mathsf{pp_{puz}}$ from a given range.

Example 2.2 (Hash-Based PoW—HPoW). Most PoW-based blockchains, such as Bitcoin, use a hash-based PoW (short: HPoW) as follows. Omitting details, the puzzle parameter $\mathsf{pp_{puz}}$ essentially describes a set of possible bitstrings while the verification parameter specifies a cryptographically secure hash function H and a positive integer bound $target$. Given a a bitstring puz (= the output of Sample), the task is to find a bitstring sol such that $H(\mathsf{puz}, \mathsf{sol}) \leq target$ for a pre-defined target $target$. The corresponding Solve algorithm randomly picks values sol and checks if the condition is met (Verify). In case that H is idealized as a random oracle, all values sol are equally likely to be a solution – making HPoW id-sound.

[2]In this work, we will only consider puzzles that are id-sound with id being the identity function. Still, we decided to present the more generic definition based on some function g as $g \neq id$ may hold in other cases.

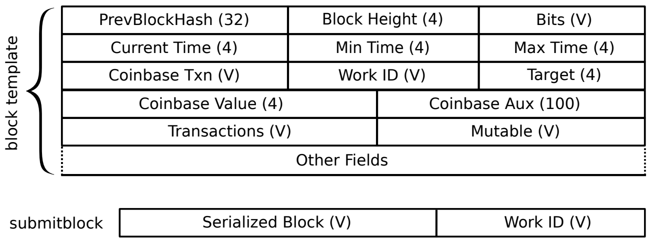

Figure 2: Sketch of a GBT work template and corresponding worker's response. The pool operator outsources to its workers a block template that also contains the field CoinbaseAux. This field contains auxiliary information given by the pool operator which is used by the worker to populate parts of the coinbase transaction.

2.3 Mining Pool Protocols.

To generate a block, *miners* must find a block header that represents the solution of a PoW and include it (as well as the additional fields) into a new block, allowing any entity to verify the PoW. Upon successfully generating a block, a miner is granted a monetary reward (i.e., in the form of a coinbase transaction CB[3] that specifies the ID of the recipient of the reward). The resulting block is forwarded to all peers in the network, who can then check its correctness by verifying the hash computation. If the block is deemed to be "valid"[4], then blockchain nodes append it to their previously accepted blocks. Notice that the hash function H and most of the values in the block header are either system-wide parameters or implicitly given and cannot be changed by the miner. In the sequel, we refer to these parameters as π_{blckhdr}.

The current difficulty level of mining valid blocks is so prohibitively high that it reduces the incentives for miners to operate alone. Joining a mining pool is an attractive option to receive a portion of the Bitcoin block reward on a consistent basis. Miners contribute their resources to generate a block and to split the reward between all the pool members (referred to as workers in the sequel) who present valid proof-of-work. In more detail, a mining pool sets $target$ between 1 and the blockchain's target difficulty (denoted in the sequel as pool target level). Subsequently, a share is assigned to those

[3]The coinbase CB contains on the one hand fixed information CB_{fix} about the miner (so that the mining reward can be claimed afterwards) and also a field CB_{var}(typically of 100 bytes) that can be fully changed depending on the underlying mining pool protocol.
[4]That is, the block contains correctly formed transactions that have not been previously spent, and has a correct PoW.

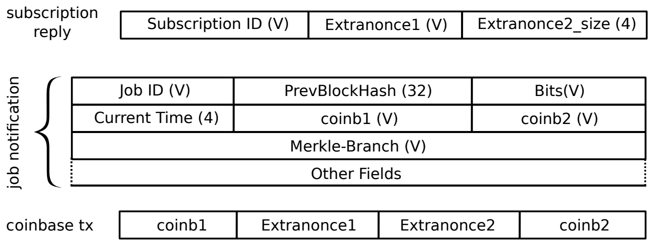

Figure 3: Sketch of an STM work template and corresponding worker's response. Here, workers first receive the value of Extranonce1 and the *size* of Extranonce2 from the pool operator right after the initial subscription to the mining pool. Unlike GetBlockTemplate, the pool operator fixes all the transactions to be confirmed in the PoW excluding the Extranonce2 field in the coinbase transaction. The workers only receive the sibling paths of the coinbase transaction in the transaction Merkle tree, using which they compute the Merkle root once the coinbase transaction is determined.

workers that provide a block header that scores a difficulty level between the pools difficulty level and the currency's difficulty level. The main purpose of these block headers is to show that the worker is contributing with a certain amount of processing power.

Algorithm 1 Work flow for solving the PoW.

```
Input: Non-changeable block header parameters $\pi_{\text{blckhdr}}$
 1: while PoW not solved do
 2:     Choose a new value for $\mathsf{CB}_{\text{nonce}}$ to specify the coinbase $CB$
 3:     Compute the Merkle root $\mathsf{MR}$ over the coinbase $CB$ and the set
        of transactions $T$
 4:     for $nonce \in \{0,1\}^{32}$ do
 5:         Compute $h := H(\pi_{\text{blckhdr}}, \mathsf{MR}, nonce)$
 6:         if $h \leq target$ then
 7:             break; {Solution found.}
 8:         end if
 9:     end for
10: end while
Output: The solution $\mathsf{sol} = (\mathsf{CB}_{\text{nonce}}, nonce)$.
```

To pool mining, most existing mining pools either adopt the GetBlockTemplate [17] protocol (cf. Figure 2) or Stratum mining [25] (cf. Figure 3). These protocols are supported by all existing hash-optimized mining hardware, such as ASICs, FPGAs, and GPUs.

2.3.1 GetBlockTemplate (GBT). GetBlockTemplate (GBT) [5, 6, 17] is a mining pool protocol natively supported in Bitcoin. In this protocol, the mining pool operator orchestrates task assignment to the various connected workers. The GetBlock-Template protocol gives workers some degrees of freedom in choosing some PoW parameters while still ensuring that no two workers work on towards the same PoW solution.

More specifically, upon request, the pool operator outsources to its workers a block template (cf. Figure 2) that contains the following fields: previous block hash, block height, list of transactions, the target, the coinbase transaction, and time, among others.

The field CoinbaseAux contains auxiliary information given by the pool operator which is used by the worker to populate parts of the coinbase transaction.[5] Notice that the worker can add up to around 100 bytes of arbitrary data to the coinbase transaction, which is commonly used by workers as an extra nonce when searching for PoW solutions.

The worker then constructs the Merkle root of all transactions (including the modified coinbase transaction) and searches for a 4 byte nonce to solve the PoW according to the pool difficulty specified in the field Target. If no solution can be found, the worker restarts as shown in Algorithm 1. Notice that a worker can freely modify or shuffle the transactions in the block and compute the corresponding Merkle tree accordingly; this gives workers additional flexibility when solving their tasks.

Once a solution is found, the worker constructs the corresponding block header (cf. Figure 1), appends it to the full corresponding block (including all transactions), serializes the result, and submits the serialized block along with his worker ID back to the pool operator.

2.3.2 Stratum Mining (STM). Stratum (STM) is one of the most commonly adopted mining pool protocols. In Stratum, workers first receive the value of Extranonce1 and the *size* of Extranonce2 from the pool operator right after the initial subscription to the mining pool. Subsequently, the workers are regularly notified of new mining work with templates consisting of coinbase data, the block version, the difficulty, the time, the prefix (coinb1), suffix (coinb2) of the coinbase transaction, and the Merkle tree branches among other information (cf. Figure 3). Unlike GetBlockTemplate, the pool operator here fixes all the transactions to be confirmed in the PoW excluding the coinbase transaction. The workers only receive the sibling paths of the coinbase transaction in the transaction Merkle tree, which is enough to compute the Merkle root once the coinbase transaction is determined.

The worker then searches for the value Extranonce2 which results in a Merkle root that solves the PoW. The resulting coinbase transaction is then formed by appending the following information: (coinb1|| Extranonce1|| Extranonce2|| coinb2), where Extranonce2 is generated locally by each worker.

As such, the Stratum protocols gives workers the flexibility to cycle through the 32-bit nonce and the Extranonce2 field in the coinbase transaction whose size is determined by the pool operator during the subscription stage. Similar to GBT, workers can additionally adjust the timestamp and the nonce in the block header to gain additional flexibility.

2.4 Rational Worker Model.

We model a worker as a machine that comprises one or several processing units and one local memory. It can fetch data and store data from/to the local memory and can process data units. In addition, a worker can also request data from external data sources. In practice, these workers refer to

[5]More specifically, CoinbaseAux contains auxiliary information guiding the worker in creating the scriptSig field of the coinbase transaction.

Algorithm 2 Informal work flow for solving the strawman solution.

```
Input: Non-changeable block header parameters $\pi_{\text{blckhdr}}$
1:  while PoW not solved do
2:      Choose a new value for $\mathsf{CB}_{\mathsf{nonce}}$ to specify the coinbase $CB$
3:      Compute the Merkle root $\mathsf{MR}$ over the coinbase $CB$ and the set of transactions $T$
4:      for $nonce \in \{0,1\}^{32}$ do
5:          Compute $h := H(\pi_{\text{blckhdr}}, \mathsf{MR}, nonce)$
6:          if $h \leq target$ then
7:              break the WHILE-loop; {Solution found.}
8:          end if
9:      end for
10: end while
11: Set $IND := []$; {Index set of challenges}
12: for $i = 1, \dots, \ell$ do
13:     Compute index $ind := H(\mathsf{CB}_{\mathsf{nonce}}, i) \mod s$ {$s$ denotes the current size of the blockchain}
14:     Append $ind$ to $IND$.
15: end for
16: Let $IND = [i_1, \dots, i_\ell]$
17: Compute $h := H(BC[i_1], \dots, BC[i_\ell])$
Output: The solution $\mathsf{sol} = (\mathsf{CB}_{\mathsf{nonce}}, nonce, h)$.
```

the multitude of nodes who commonly join mining pools to increase their mining advantage in the system.

We assume all workers to be *rational*, i.e., to maximize their probability in solving the PoW while minimizing their effort (which translate into costs). In this work, we consider the scenario that reading data from external sources requires network communication and that downloading data takes so much more time than simply reading it directly from local memory that a rational worker aims to work with local memory only.

3 *P*-COVERING BLOCKCHAIN

We start by considering a strawman design which combines the notion of proofs of knowledge [13, 18, 21, 23] with the standard PoW mechanism. The core idea is that an algorithm can only efficiently solve a PoW if it *knows* a certain file $\mathbf{F}$. We then discuss the shortcomings of this strawman design and present our solution, EWoK.

3.1 Overview

A strawman solution to the problem that we consider is to couple the PoW with a proof of knowledge (PoK) with respect to a file $\mathbf{F}$, being the blockchain BC in our case. To this end, one can proceed as follows:

(1) A worker has to find a solution sol for a PoW puzzle puz.
(2) The worker has to compute from sol a PoK-challenge over BC.

Since checking knowledge of the full blockchain is infeasible and since the "file", i.e., the blockchain, is publicly known, a straightforward approach is to require the hash over a pseudorandom selection of blocks as given in Algorithm 2.

Note that existing schemes that re-purpose PoW to prove storage of a file F [13, 18, 21, 23] follow a similar approach but cannot be directly used in our case where $F = BC$ is the entire blockchain. First of all, these solutions typically assume that the file is fixed (e.g., Permacoin extends the file with an erasure code)—which is not the case in existing blockchains that constantly grow in size. Moreover, workers are typically equipped with dedicated mining hardware that only possess limited storage capabilities (typically few GBs of storage and RAM) and cannot store the full blockchain. As a consequence, such a solution only ensures that the pool operator (or any other entity) stores the full blockchain and does not guarantee that the workers store any parts of the blockchain.

To overcome the limitations of the aforementioned strawman solution, we restrict the PoK to a *part* of the blockchain which is small enough to fit into local memory of typical workers. The part to be stored is called a *partition*, which is dynamically determined by a data partitioning function from the current blockchain and depends (randomly) on the worker.

In what follows, we formally model this concept, dubbed Partitions-Bound PoW. We then discuss the various design choices used in EWoK in Section 4 in greater detail.

3.2 Modeling a Partitions-Bound PoW

We now define a p-Covering Blockchain as a means to formally introduce the concept of a Partitions-Bound PoW. To this end, we adopt the common concept of *extractors*, e.g., see [24, Section 2]. An extractor algorithm Extr is given access to an algorithm A, represented by $\mathsf{Extr}^{\mathsf{A}}$. Note that this access is non-black-box, e.g., Extr is allowed to rewind A and is given access to the states of A. The interpretation is that the outputs generated by Extr are extracted from algorithm A.

Next, we introduce in Definition 3.1 an extension of the PoW-notion that binds a PoW to a file, i.e., a bitstring in general. For the definition, we introduce a procedure *create* for *creating* PoW that depends on a given bitstring and a security parameter λ. More formally, *create* takes as input a security parameter λ and a bitstring $\mathbf{F} \in \{0,1\}^*$ and outputs a PoW-system, i.e., it outputs descriptions of the spaces $(\mathsf{PP}, \mathsf{PS}, \mathsf{SS})$ and specifications of the algorithms $(\mathsf{Sample}, \mathsf{Solve}, \mathsf{Verify})$. Then, the notion of a PoW being bound to some file is as follows:

Definition 3.1 (File-Bound PoW). We say that a PoW-creating process *create* is g'-file-bound if there exists a knowledge extractor Extr such that the following holds for any file $\mathbf{F} \in \{0,1\}^*$ and any PoW that results from running *create* on λ and $\mathbf{F}$. For every PPT adversary $\mathcal{A}$ and every puzzle parameter $\mathsf{pp}_{\mathsf{puz}} \in \mathsf{PP}$, it holds that:

$$\Pr\left[\mathsf{Event}(\mathsf{pp}_{\mathsf{puz}}, \mathcal{A}, g') \wedge \mathbf{F}^* \leftarrow \mathsf{Extr}^{\mathcal{A}} : \mathbf{F}^* \neq \mathbf{F}\right] = \mathsf{negl}(\lambda).$$

We say that a PoW is *g'-bound to some file* $\mathbf{F}$ if it is the output of $create(\lambda, \mathbf{F})$ for a g'-file-bound PoW-creating process $create$. Note that if λ is clear from the context, we sometimes omit it and simply write $create(\mathbf{F})$ instead of $create(\lambda, \mathbf{F})$.

We stress that the function g' in Definition 3.1 is *different* from the function g to express g-soundness of a PoW g (cf. Definition 2.1.) However, there are some relations.

Both functions g and g' are positive, i.e., output non-negative values only, are both defined on integer values, and it holds $g(x) \leq g'(x')$ for any input x. The first function, g, is used to characterize the hardness of solving the PoW by expressing a threshold such that no PPT can solve the POW with an effort lower than this. Similarly, the function g' is used to express a threshold for the effort of an attacker but restricted to attackers who do not know the file $\mathbf{F}$ (in the sense of extractability). More precisely, Definition 3.1 says that if an attacker aims to have an effort that is below a certain upper bound, specified by g', then it has to store the file $\mathbf{F}$.

In consequence, we see different types of thresholds with respect to the time effort an attacker can achieve:

(1) $\mathsf{Steps}_{\mathcal{A}}(\mathrm{puz}) \leq g(\mathsf{Steps}_{\mathsf{Solve}}(\mathrm{puz}))$ is practically impossible if the blockchain system is g-sound.
(2) $g(\mathsf{Steps}_{\mathsf{Solve}}(\mathrm{puz})) < \mathsf{Steps}_{\mathcal{A}}(\mathrm{puz}) \leq g'(\mathsf{Steps}_{\mathsf{Solve}}(\mathrm{puz}))$ is possible but only if $\mathcal{A}$ stores the file $\mathbf{F}$ if the blockchain system is g'-bound to the file $\mathbf{F}$.
(3) $g'(\mathsf{Steps}_{\mathsf{Solve}}(\mathrm{puz})) < \mathsf{Steps}_{\mathcal{A}}(\mathrm{puz})$ could be possible even without storing the file $\mathbf{F}$.

As mentioned earlier, the idea is to use a file-bound PoW that is restricted to a *part* of the blockchain, which is small enough to fit into local memory of typical workers. The part to be stored is called a *partition*, which is determined by a data partitioning function on the blockchain.

Definition 3.2 (Partitioning Function). Consider a file $\mathbf{F}$, divided into ℓ blocks $\mathbf{F}[i]$. A *partitioning function* prt (with respect to the file $\mathbf{F}$) is a mapping that accepts inputs x from some input space $\mathcal{P}$ and outputs a fraction of the file, i.e., $\mathrm{prt}(x) = [\mathbf{F}[i_1], \ldots, \mathbf{F}[i_{\ell'}]]$ where $0 \leq i_1 < \ldots < i_{\ell'}$ and $\ell' \leq \ell$. Slightly abusing notation, we say for a block $\mathbf{F}[i]$ that $\mathbf{F}[i] \in \mathrm{prt}(x)$ if $\mathrm{prt}(x) = [\mathbf{F}[i_1], \ldots, \mathbf{F}[i_{\ell'}]]$ and $\mathbf{F}[i] \in \{\mathbf{F}[i_1], \ldots, \mathbf{F}[i_{\ell'}]\}$.

We say that the partitioning function is *p-covering* with $0 \leq p \leq 1$ if it holds for any block $\mathbf{F}[i]$ that

$$\Pr\left[x \xleftarrow{\$} \mathcal{P} : \mathbf{F}[i] \in \mathrm{prt}(x)\right] \geq p.$$

In EWoK (Section 4), the partition is determined by data that is connected to the worker (to realize a distributed storage of the blockchain) but is independent of the puzzle (to encourage the storage in local memory).

We consider now a variant of file-bound PoW where efficiently solving a puzzle puz requires to store a partition of the file $\mathbf{F}$ only. To this end, we require that a solving algorithm can only be time-optimal, i.e., the effort being below a certain threshold, if the algorithm has direct access to the involved partition. The following definition is a straightforward extension from Definition 3.1.

Definition 3.3 (Partitions-Bound PoW). We say that a PoW-creating process $create$ is g'-partition-bound if there exists a knowledge extractor Extr such that the following holds for any tuple $(\mathbf{F}, \mathrm{prt})$ where prt is a partition for $\mathbf{F}$ with $\mathcal{P} = \mathsf{PP}$, i.e., input space of prt equals the puzzle parameter space, for any PoW that results from running $create$ on $(\mathbf{F}, \mathrm{prt})$, and for any $\mathsf{pp}_{\mathrm{puz}} \in \mathsf{PP}$:

$$\Pr\left[\mathsf{Event}(\mathsf{pp}_{\mathrm{puz}}, \mathcal{A}, g') \wedge \mathbf{F}^* \leftarrow \mathsf{Extr}^{\mathcal{A}} : \mathbf{F}^* \neq \mathrm{prt}(\mathsf{pp}_{\mathrm{puz}})\right] = \mathrm{negl}(\lambda). \quad (1)$$

We say that a PoW is g'-bound to $(\mathbf{F}, \mathrm{prt})$ (or *partitions-bound* for short) if it is the output of $create(\mathbf{F})$ for a g'-partition-bound PoW-creating process $create$.

Notice that by setting $\mathrm{prt}(x) = \mathbf{F}$ for all $x \in \mathcal{P}$, a partitions-bound PoW becomes a file-bound PoW.

Theorem 3.4 explains how these concepts help to solve the problem stated in Section 1:

THEOREM 3.4. *Consider a PoW-based blockchain system and a p-covering partitioning function* prt' *with respect to the blockchain BC with input space $\mathcal{P}$. Moreover, let a hash function $H : \mathsf{PP} \to \mathcal{P}$ be given and we define* $\mathrm{prt}(\mathsf{pp}_{\mathrm{puz}}) := \mathrm{prt}'(H(\mathsf{pp}_{\mathrm{puz}}))$ *for each* $\mathsf{pp}_{\mathrm{puz}} \in \mathsf{PP}$. *Finally, we assume that the PoW is partitions-bound (cf. Definition 3.3) for the identity function* $g' = \mathrm{id}$ *with respect to* prt.

Let $\mathcal{W}$ refer to the set of rational workers in the system. If the following two conditions are met:

Condition 1: $\mathrm{prt}(\mathsf{pp}_{\mathrm{puz}})$ *fits completely into local memory of any worker for each* $\mathsf{pp}_{\mathrm{puz}} \in \mathsf{PP}$.
Condition 2: *The workers in $\mathcal{W}$ operate over different puzzle parameter* $\mathsf{pp}_{\mathrm{puz}}$.

Then, the following holds: Any randomly selected block is on average locally stored by a p-fraction of $\mathcal{W}$ so that the expected number of replicas per block is $p \cdot |\mathcal{W}|$. Moreover, the probability that a block is not locally stored by any worker in $\mathcal{W}$ is at most $(1-p)^{|\mathcal{W}|}$.

PROOF OF THEOREM 3.4. We model H as a random oracle. As the workers operate over different puzzle parameters $\mathsf{pp}_{\mathrm{puz}}$ (Condition 2), the inputs to prt' are uniformly and independently selected from $\mathcal{P}$. As prt' is p-covering by assumption and the blockchain system is partitions-bound, it follows that each block of the blockchain needs to be known on average by a p-fraction of $\mathcal{W}$. As any $\mathrm{prt}'(x)$ fits into local memory of any worker (Condition 1), a rational worker will store "its" partition in local memory to minimize access time. This shows that each block is on average locally stored by a p-fraction of $\mathcal{W}$. The other claim follows analogously. □

This ensures for any block a minimum number of decentralized replicas are distributed over randomly selected miners if p is accordingly chosen. Note that these replicas are not meant to be frequently involved in the verification process but merely constitute a backup copy. Our goal in this paper is to devise a partitions-bound blockchain system for a p-covering partitioning function such that the Conditions 1 and Condition 2 mentioned in Theorem 3.4 are fulfilled while

Algorithm 3 Sketch of the partitioning function prt in EWoK.

```
Input: Value $x \in \mathcal{P}$
1: $part := []$ {Initialize an empty partition }
2: for $i = 1, \ldots, \lceil \frac{s}{N} \rceil$ do
3:     Compute $j := H(x, i) \mod N$ {Specifies which block within the $i$-th chunk is to be appended to the partition.}
4:     Compute $ind := (i-1) \cdot N + j$ {Gives the total index of the block to be appended.}
5:     $Append(part, BC[ind])$ {Appends the selected block to the partition. If $BC[ind]$ does not exist yet, this command is ignored.}
6: end for
Output: Partition $part$.
```

(i) requiring minimal changes to existing PoW blockchains and *(ii)* being compliant with the specifications of existing mining hardware.

4 DETAILING EWOK

In what follows, we detail our solution, EWoK, and analyze its security.

4.1 Protocol Specification

We start by describing the partitioning function (see Definition 3.2) and explain afterwards how to turn the standard PoW into a partitions-bound PoW (see Definition 3.3).

4.1.1 Specification of the Partitioning Function. Recall that BC denotes the blockchain of size s and $BC[i]$ the i-th block. The partitioning function divides the blockchain into *chunks* with equal chunk size N, i.e., the first N blocks form the first chunk and so on, and selects pseudorandomly (with H) exactly one block per chunk. A complete description is given in Algorithm 3.

This partitioning function provides various advantages. It is p-covering (see Section 4.3) and is independent of the current size of the blockchain. Although the partitions grow over time (this is a consequence of the p-covering requirement), this does not affect those blocks that are already stored in the partition. Moreover, the parameters can be reasonably chosen to ensure that the partition's size fits within the available memory of existing dedicated mining hardware (see Section 5). Finally, it does not require any involved computation and supports mechanisms already contained in existing PoW-based blockchains. We confirm this by means of implementation in Section 5.

4.1.2 Specification of the Partitions-Bound PoW. The partitions-bound PoW in EWoK is composed of two phases, each using a variant of HPoW (cf. Example 2.2) with sampling algorithm Sample_1 and target $target_1$ resp. Sample_2 and $target_2$. In the first phase, the workers reach consensus on the exact set (including order) of transactions to be confirmed in a given block. In the second phase, the workers solve HPoW based on this set that entangles the proof of work with a proof of knowledge on their specific partition.

Notice that, while this entanglement of PoW with PoK ensures that the PoW is partition-bound, it does not guarantee that different workers store independent partitions. Since PoW (by design) does not embed any notion for coupling workers with a unique system-wide identity, the straightforward approach of coupling identities to partitions is not possible[6]. Instead, EWoK gives economic incentives to the workers to store independent partitions by ensuring that two workers operating over the same partition effectively decrease their combined mining capabilities. This is achieved by selecting the parameters such that on average, only one solution can be expected in the second phase. As a worker can vary both the set of transactions and the nonces, EWoK restricts the set of transactions in phase 1 and the set of allowed nonces in phase 2.

Fixing the partition: Recall that the transactions exclude the coinbase transaction CB. In the sequel, we assume that the coinbase transaction is divided into two parts $CB = (CB_{fix}, CB_{var})$ such that CB_{fix} contains all information that are fixed by a pool operator, including the specification of the rewarding transaction for the miner. This is conforming with the operation of GBT and STM (cf. Section 2). Each of these parts of CB plays a different role in EWoK. The first part, CB_{fix}, determines the partition to which the PoW is bound: $part := part_{CB_{fix}} := \mathrm{prt}(H(CB_{fix}))$. The second part, CB_{var}, is used as input for solving the PoW in the second phase.

First phase—fixing the transactions: Assume a set of transactions T^* that are candidate for inclusion in the next block. The goal of this first phase is to *enforce* workers to operate all over the same set of transactions (including their order). To achieve such consensus while leveraging existing mining hardware/software, EWoK deploys HPoW with target $target_1$ and a sampling algorithm Sample_1 that takes as input T^*, i.e., a *set* of transactions, and outputs a *sequence* of transactions T, i.e., with a specified order. T is said to be *valid* with respect to a nonce $nonce_{trans}$ if it holds:

$$H(T, nonce_{trans}) \leq target_1. \tag{2}$$

Second phase—mining a block: Given some block header parameters π_{blckhdr} and a valid set of transactions T together with an appropriate value $nonce_{trans}$, the second phase mainly deals with the generation of the appropriate block header. To this end, EWoK entangles HPoW with in-memory proofs of knowledge of the partition $part_{CB_{fix}} := \mathrm{prt}(H(CB_{fix}))$. The HPoW consists of finding values CB_{var} and $nonce$ such that

$$H(\pi_{\mathrm{blckhdr}},\ \mathsf{MR}((CB_{fix}, CB_{var}), T),\ nonce) \leq target_2. \tag{3}$$

Here, $\mathsf{MR}(CB, T)$ denotes the root of the Merkle tree where CB defines the first leaf and the following leaves are specified by T.

[6]Permission-based blockchains (where workers have a clear identity) directly enforce the property of independent partitions, i.e., by coupling them to the identity of the worker.

Formally, the sampling algorithm Sample_2 has the values $\pi_{\text{blckhdr}}, T, nonce_{trans}$ as fixed internal parameters and takes as input the $\mathsf{pp}_{\text{puz}} := CB_{fix}$ as puzzle parameter. Recall that CB_{fix} (part of CB) determines the partition per worker and has been fixed by the operator. CB_{var} has the form $CB_{var} = (nonce_{trans}, \mathsf{CB}_{\mathsf{nonce}})$, where $nonce_{trans}$ is output by the first phase and only the second value, $\mathsf{CB}_{\mathsf{nonce}}$, can be varied.

We now show how to couple this PoW with a PoK over the partition $part_{CB_{fix}}$. To this end, the values of $\mathsf{CB}_{\mathsf{nonce}}$ are derived as follows:

$$\mathsf{CB}_{\mathsf{nonce}} := \left(H\left(v_T, part_{CB_{fix}}[index], pn\right),\ pn\right). \tag{4}$$

Here, v_T is a fingerprint for T, e.g., its hash value. Note that this value does not change during phase 2 and hence needs to be computed only once per worker at the beginning of the phase. $part_{CB_{fix}}[index]$ are blocks involved in the partition that are pseudorandomly selected as follows:

$$index := H(v_T, pn) \mod \ell_{\text{part}}. \tag{5}$$

That is, $H(v_T, pn)$ determines which block of the partition of length ℓ_{part} is involved.

pn is a value called the *pre-nonce* (to reflect the fact that it will be used as a seed to determine the nonce $\mathsf{CB}_{\mathsf{nonce}}$). In EWoK, we restrict the space of possible pre-nonces to enforce independent partitions. As we will show in Section 4.3, this gives incentives to rational workers to store independent partitions. To restrict the nonce space, EWoK enforces an upper bound $\delta_{\text{pre-nonce}}$ when choosing pn:

$$pn \text{ valid} \iff pn \le \delta_{\text{pre-nonce}}. \tag{6}$$

If $h \le target_2$, a solution is found. The complete solution is then given by CB and $nonce$ together with the valid set of transactions T. The validation of a solution $(T, CB, nonce)$ is given in Algorithm 4. Notice that the verification of the blocks is done by parties that store the full, public blockchain, such as the mining pools or full nodes. Recall that pool operators currently store the full blockchain in order to verify new blocks and outsource new puzzles.

Algorithm 4 The verification algorithm $\mathsf{VerifyEWoK}$.

```
Input: Block header parameters $\pi_{\text{blckhdr}}$, possible solution $(T, CB, nonce)$

1: Check if set of transactions is valid (first phase)
2: Parse from $CB$ the value $nonce_{trans}$
3: if $H(T, nonce_{trans}) > target_1$ then
4:     Solution invalid. Abort.
5: end if

6: Check Proof of Work solution (second phase)
7: if $H(\pi_{\text{blckhdr}},\ \mathsf{MR}(CB, T),\ nonce) > target_2$ then
8:     Solution invalid. Abort.
9: end if

10: Check validity of parameters (second phase)
11: Check if nonce restriction applies
12: Parse from $CB$ the value $\mathsf{CB}_{\mathsf{nonce}}$
13: Parse $pn$ from $\mathsf{CB}_{\mathsf{nonce}}$
14: if $pn > \delta_{\text{pre-nonce}}$ then
15:     Solution invalid. Abort.
16: end if
17: Check if correct partition block involved.
18: Compute characteristic value $v_T$ from $T$
19: Compute index $index := f_{\text{ind}}(v_T, pn)$
20: if $\mathsf{CB}_{\mathsf{nonce}} \neq (H(v_T, part_{CB_{fix}}[index], pn),\ pn)$ then
21:     Solution invalid. Abort.
22: end if
```

4.2 Practical Considerations

4.2.1 Parameter Selection: To ensure a smooth migration from an HPoW-based blockchain to a EWoK-based blockchain, the overall computational effort for successful mining should remain the same, ensuring that the hash rate remains unchanged.

To this end, we suggest to set both $target_1$ and $target_2$ to $2 \times target$ where $target$ denotes the target used in HPoW. This ensures that the work efforts for the proofs of work in phase 1 and 2, respectively, are about half the effort of the HPoW each.

In addition, we restrict the nonce space within phase 2 to a set of size $2^{2 \times target}$. For example, assuming the difficulty in the Bitcoin network in April 2020, this can be achieved by restricting the nonce space to approximately 76 bits. Note that a worker can control two values in phase 2, the 32-bit value $nonce$ and the pre-nonce value pn. Thus, we restrict the freedom of the latter to 44 bits. The reason for restricting the nonce space is to give economic incentives for workers to store and operate over different partitions (cf. Claim 4 in Section 4.3 for more details). Note that while a worker could in addition also vary the timestamp, this is possible within a certain range only as otherwise the block becomes outdated. This can be accommodated by reducing $target_2$ accordingly.

4.2.2 Work Flow: We stress that, in practice, the computation of the partition is done only once by each worker. The process of fixing a shard for each miner in the mining pool can take place in agreement with the mining pool operator; indeed, we show in Claim 4 in Section 4.3 that a set of rational workers in the same mining pool are incentivized to work on different shards to maximize the pool rewards.

To ensure efficient execution of phases 1 and 2, the workers are likely to load their respective partitions in RAM. We show in Section 5 that this is a reasonable assumption; namely, we show how to realize modest partition sizes (with approximately few hundred MBs) that can easily fit into the available RAM of most existing mining hardware. Even when the partitions grow over time, EWoK ensures that this does not affect or modify the storage of blocks already contained in the partitions stored by each worker.

Moreover, as we show in Section 4.3, it is a good strategy for the operator to distribute the search for a valid set of transactions (phase 1) among the workers in the mining pool, e.g., by splitting the space of $nonce_{trans}$ values. Once one worker succeeded, the solution of phase 1 (i.e., the fixed transaction set) is forwarded to the others who can then immediately start with phase 2. In this sense, Algorithm 5 should not be seen as a representation of the work of a single worker; instead, it summarizes all single processes occurring within a mining pool.

Algorithm 5 The algorithm SolveEWoK.

Input: The block header parameters π_{blckhdr} including hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$, set of transactions T^*, thresholds $target_1$, $target_2$, and $\delta_{\text{pre-nonce}}$

```
 1: Find valid set of transactions (first phase)
 2: while No valid set of transactions found do
 3:   Choose a set of transactions $T$ from $T^*$ (including an order)
 4:   Choose value $nonce_{trans}$
 5:   if $H(T, nonce_{trans}) \leq target_1$ then
 6:     Valid set of transactions found. Break WHILE-loop.
 7:   end if
 8: end while

 9: Solve Proof of Work (second phase)
10: Split the coinbase $CB = (CB_{fix}, CB_{var})$
11: Compute $part := \text{prt}(H(CB_{fix}))$.
12: Compute characteristic value $v_T$ from $T$
13: while Proof of Work not solved do
14:   Choose a pre-nonce $pn \leq \delta_{\text{pre-nonce}}$
15:   Compute index $index := f_{\text{ind}}(v_T, pn)$
16:   Compute $\mathsf{CB}_{\text{nonce}} := (H(v_T, part[index], pn), pn)$
17:   Set $CB = (CB_{fix}, (nonce_{trans}, \mathsf{CB}_{\text{nonce}}))$
18:   for $nonce \in \{0,1\}^{32}$ do
19:     Compute $h := H(\pi_{\text{blckhdr}}, \mathsf{MR}(CB, T), nonce)$.
20:     if $h \leq target_2$ then
21:       PoW-solution found. Break WHILE-loop.
22:     end if
23:   end for
24: end while
```

Output: Solution $(T, CB, nonce)$.

4.2.3 Compatibility with existing mining protocols: This aforementioned process can be best instantiated practically by first requiring all workers to solve a GBT work template where all other fields—besides the worker ID, the target difficult, and the set of transactions—are set to zero. Once a worker finds a solution for phase 1, he reports it in the serialized block output in GBT. For all practical reasons, if a worker is able to find a nonce for the exact set (and order) of transactions reported in the GBT work template, he can only submit the nonce and his worker ID back—without the need to serialize the entire resulting block.

Given the output of GBT in phase 1, the operator quickly constructs the sibling paths associated with the Merkle root

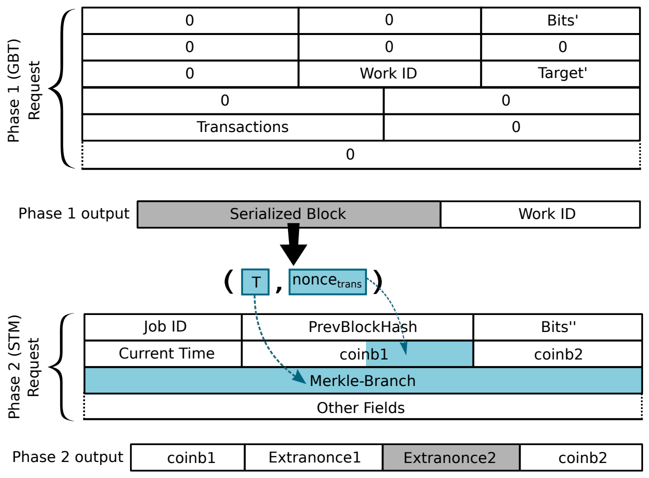

Figure 4: Instantiating EWoK using GBT and STM work templates.

and proceeds similarly to STM. Namely, the operator outsources a work template asking workers to solve the corresponding solution for phase 2. This process is detailed in Figure 4. We stress at this point that this process is not necessarily restricted to pooled mining but can be equally applied by a solo miner.

While phase 2 of EWoK is compatible with all current mining hardware, phase 1 might require some tailoring to make it compatible with existing non-programmable mining hardware in the market. For example, ASIC-based Gigahash machines are optimized to compute hashes on inputs of 80 bytes while the transaction set T has typically a larger size. To accommodate for this, one could first compute the hash of T before handing it over to the Gigahash machine and pad the 32-byte output along with $nonce_{trans}$ to be 80 bytes. The security analysis can be transported to this case in a straightforward manner.

4.2.4 Coping with large mining pools: The existence of large mining pools has been identified as a risk for PoW-based blockchains for a while. For instance, the pool *GHash.IO* held 54% of the hashrate for a day, which exceeds the theoretical attack threshold of 51% [1]. Prior work [11, 20] addresses this problem by suggesting the use of *non-outsourcable* puzzles. In a nutshell, this means that the puzzle can be solved safely only by the miner receiving the mining reward. This would limit the size of the pool to parties that are trusted by the pool operator and would particularly disincentivize large public mining pools. For instance, Daian *et al.*[11] suggest a 2-phase-PoW where outsourcing of puzzles of the second phase result in exposure to theft of mining rewards. Notice that the same holds for EWoK as well: the coinbase that specifies the recipient of the mining fee is fixed *after phase 1*, allowing workers that join a public mining pool to "steal" the solution of phase 1 and then proceed on their own.

4.3 Security Analysis

In this section, we show that EWoK fulfills all requirements of Theorem 3.4 by showing the following four claims. Claims 1 and 2 are the prerequisites mentioned at the beginning of Theorem 3.4 while Claims 3 and 4 shows Conditions 1 and 2, respectively, mentioned there. The arguments are based on the random oracle model with respect to the deployed hash function H. Moreover, we assume a distinguished input message format in each context to have formally independent H-calls in the different phases.

Claim 1: *The partitioning function* prt *in Algorithm 3 is* $(\frac{1}{N} - \frac{N}{2^n})$*-covering (cf. Definition 3.2), where* N *denotes the chunk size and* n *the output size of the hash function. If* $2^n \gg N$*, the term* $\frac{N}{2^n}$ *can be ignored and* prt *is practically* $\frac{1}{N}$*-covering.*

Recall that prt splits the blockchain into chunks of size N and selects exactly one block per chunk. The probability for each index $j \in \{0, \dots, N-1\}$ to be selected is exactly $\frac{1}{N}$ under the assumption that the outputs of the hash function are uniformly distributed. In case the number of possible outputs of the hash function is not a direct multiple of the chunk size, the higher indexes have a smaller probability of at least $\frac{1}{N} - \frac{N}{2^n}$ to be selected. □

Claim 2: *EWoK is* id*-bound to* $(part_{CB_{fix}}, \mathrm{prt})$.

We now show that the PoW used in the second phase of EWoK is id-bound to $(part_{CB_{fix}}, \mathrm{prt})$. Notice that this PoW is actually a variant of HPoW (cf. Section 2.2) that is id-sound in the random oracle model. That is, the optimal solving algorithm consists of choosing as many hash inputs as possible until it holds that

$H(\pi_{\mathrm{blckhdr}},\ \mathsf{MR}(CB, T),\ nonce) \leq target$. Recall that the values for $\mathsf{CB}_{\mathsf{nonce}}$, that is part of CB, are indirectly given by $\mathsf{CB}_{\mathsf{nonce}} := (H(v_T, part[index], pn),\ pn)$ where only the value pn is under full control of the worker. Notice also that a worker who does not select the values $\mathsf{CB}_{\mathsf{nonce}}$ as given above will eventually fail as pn is part of the solution to be published and as H is preimage-resistant. Whenever a new value $\mathsf{CB}_{\mathsf{nonce}}$ is to be determined based on a new pn, it requires the execution of the hash function H over $part[index]$. Similar to the above, a worker who does not use $part[index]$ will not be able to eventually produce a valid solution. Moreover, we point out that the values $index$ are generated by the random oracle, i.e., $index := H(v_T, pn)$; these indexes also differ between epochs due to v_T. Therefore, the values $index$ cannot be predicted, making all blocks of the partition equally necessary. □

Claim 3: *For appropriately chosen parameters,* $\mathrm{prt}(\mathsf{pp}_{\mathsf{puz}})$ *fits completely into local memory of any worker for each* $\mathsf{pp}_{\mathsf{puz}} \in \mathsf{PP}$.

We refer the readers to Section 5 for more details.

Claim 4: *Assuming claim 3, a set* $\mathcal{W}$ *of rational workers will operate over at least* $|\mathcal{W}|$ *different puzzle parameter* $\mathsf{pp}_{\mathsf{puz}}$.

For workers that operate in different mining pools, the rewarding transaction is different which in turn determines $\mathsf{pp}_{\mathsf{puz}}$. Thus, it remains to investigate Claim 4 for a set of workers that all operate in the same mining pool. Recall that $\mathsf{pp}_{\mathsf{puz}} = CB_{fix}$ and that CB_{fix} contains all information that are fixed by a pool operator, including the specification of the rewarding transaction for the miner. This ensures that workers belonging to different mining pools are going to use different parameters $\mathsf{pp}_{\mathsf{puz}}$ and we can restrict to the case that all workers in $\mathcal{W}$ belong to the same mining pool.

The mining process of EWoK includes two phases and each phase requires to solve an instantiation of HPoW. We call an attempt to solve the HPoW of the first phase a *phase-1-attempt* and define *phase-2-attempts* analogously. As the probability for successful mining grows linearly with the number of phase-2-attempts only, the optimal strategy for any worker is to minimize the number of phase-1-attempts and to maximize the number of phase-2-attempts. Therefore, once one worker has found a solution for the PoW in phase 1, it is incentivized to share it with the other workers among the mining pool so that all workers proceed to phase 2.

Now, consider the case that $\omega := |\mathcal{W}|$ workers are collaboratively trying to solve phase 2. Notice that the nonce space restriction deployed in phase 2 actually limits the number of possible phase-2-attempts with respect to a fixed choice of $\mathsf{pp}_{\mathsf{puz}}$. Let us denote this number by ν. Moreover, the parameter is chosen such that for any choice of $\mathsf{pp}_{\mathsf{puz}}$, one worker can exhaust the set of permitted nonces during one time period. That is, the average time period of phase 2 permits ω different workers to compute in total up to $\omega \cdot \nu$ phase-2-attempts.

Due to claim 3, we know that such a strategy can be realized: each worker can locally store a partition $\mathrm{prt}(\mathsf{pp}_{\mathsf{puz}})$ for pairwise different $\mathsf{pp}_{\mathsf{puz}}$. In case that $\mathcal{W}$ decides to operate over $\omega' < \omega$ puzzle parameters, then due to the nonce restriction at most $\omega' \cdot \nu < \omega \cdot \nu$ different phase-2-attempts are possibly - effectively lowering the success probability. This shows that any optimal strategy will involve at least ω different $\mathsf{pp}_{\mathsf{puz}}$.[7] □

Lemma 4.1. *In EWoK, if a rational worker aims to solve the PoW with respect to a certain partition, it will store it in local memory in its entirety if possible.*

Proof. Solving the PoW in the 2nd phase requires to repetitively compute $\mathsf{CB}_{\mathsf{nonce}} := (H(v_T, part[index], pn), pn)$ for different pre-nonces $pn \leq \delta_{\mathrm{pre-nonce}}$ and with $index := f_{\mathrm{ind}}(v_T, pn)$ (see also Algorithm 4). As one cannot predict if and which $part[index]$ will be successful, a worker cannot predict which blocks in $part$ will be required—each block is equally likely to be needed in phase 2. Thus, on average each block in $part$ that is not locally stored will add some time overhead for fetching it remotely (e.g., from the mining pool operator or some remote storage) with some probability. Since rational workers aim to maximize their success probability, they will opt to minimize the overall time effort on average without adding more delay is possible—by fully storing $part$ locally. □

[7]Neither the claim nor the proof excludes that good strategies may exist that involve more than ω different puzzle parameters.

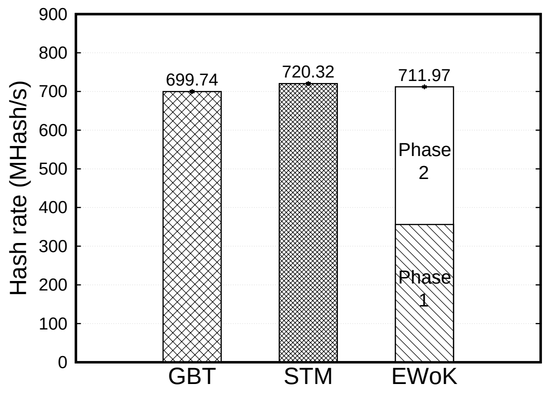

Figure 5: Effective hash rate performance of EWoK when compared to GBT and STM.

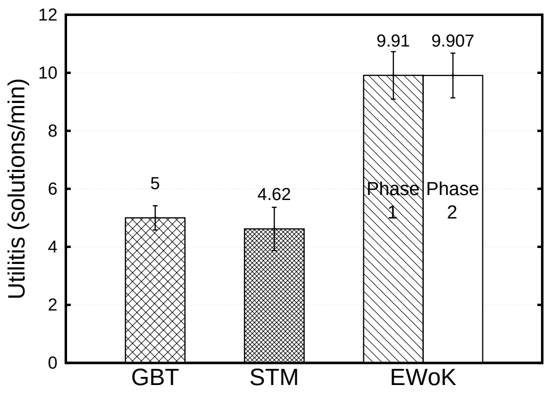

Figure 6: Number of solutions mined in EWoK when compared to GBT and STM.

5 IMPLEMENTATION & EVALUATION

We evaluate a prototype implementation of EWoK integrated within GPU mining. By showing that EWoK can be instantiated within existing mining protocols without modifications, we stress that EWoK can be immediately deployed on other mining hardware, such as ASIC and FPGA mining. Notice that, in our implementation, we only focus on GPU mining owing to its popularity and reasonable value for the money.

5.1 Implementation Setup

We integrate EWoK within the popular open-source mining tool BFGMiner [4] implemented in C. BFGMiner is a middleware which resides on a host machine and communicates with the mining pool(s) to retrieve the work template. More specifically, BFGMiner uses OpenCL to define and assign the mining tasks to GPU. To this end, it first pre-calculates the intermediate state of the SHA-256 computation over parts of the PoW input data (i.e., the block header) and then feeds this intermediate state to the GPU worker which finalizes the hash function computations over different nonces in parallel.

We deploy our implementation on an Intel Core i5-7400 equipped with 8GB of RAM which implements the BFGMiner orchestration, and on an AMD Radeon RX480 GPU featuring 1120 MHz of clock frequency and 4GB of RAM.

We instantiate EWoK by leveraging functionality from GBT and STM as described in Section 4.2. Namely, we assume that the operator distributes the search for a valid set of transactions (phase 1) among the workers using GBT. Given the output of GBT in phase 1, the operator quickly constructs the sibling paths associated with the Merkle root and proceeds similarly to STM to solve the corresponding solution for phase 2. When benchmarking EWoK, we prepare a local pool by leveraging the BFTMiner implementation of the work templates. We only measure the performance of EWoK witnessed by the worker and do not evaluate the overhead incurred by EWoK on the operator (since this overhead is exactly similar to that witnessed by operators executing GBT and STM). In our setup, we assume a 1 MB block size and transaction sizes of approximately 250 Bytes conforming with the Bitcoin blockchain [7]. Each data point in our plots is averaged over 10 independent measurements; where appropriate, we include the corresponding 95% confidence intervals.

5.2 Performance Evaluation

Hash rate performance: In Figure 5, we evaluate the hash rate performance of EWoK when compared to GBT and STM as follows. We measure the time that the worker spends in preparing the block headers (in GBT) and searches for the corresponding PoW solutions (GBT and STM), and we count the number of hashes that the worker computed for each puzzle. Notice that EWoK is composed of two phases: phase 1 which is instantiated with GBT and phase 2 which we instantiate with STM. The runtime for phases 1 and 2 are similar since the target difficulty in both phases is the same. Therefore, we compute the hash rate of EWoK as the average hash rate exhibited by phases 1 and 2. Our results show that EWoK achieves 712 hashes per second and is almost 2% faster than GBT. Notice that phase 1 in EWoK incurs less hash computations than GBT since the worker only needs to compute the hash over the ordered transaction set instead of building the entire Merkle tree. Recall that STM achieves a higher hash rate performance than GBT since it only outsources to the worker a small number of additional information for computations to construct the Merkle root. Despite the additional operations in phase 2 when compared to STM, EWoK is only 1.1% slower than STM; our results therefore show that EWoK emerges as a strong tradeoff between the performance of GBT and STM.

Number of solutions per minute: We measure the number of effective solutions computed by EWoK when compared to GBT and STM. Here, we set the pool target difficulty [12] to be 2 (target 0x1d007fff) for GBT and STM. Following EWoK's specification in Section 4.1, phases 1 and 2 in EWoK each exhibit half the difficulty (set to 1 with target 0x1d00ffff). Our results show that on average both phases 1 and 2 in EWoK achieve approximately double the number of effective

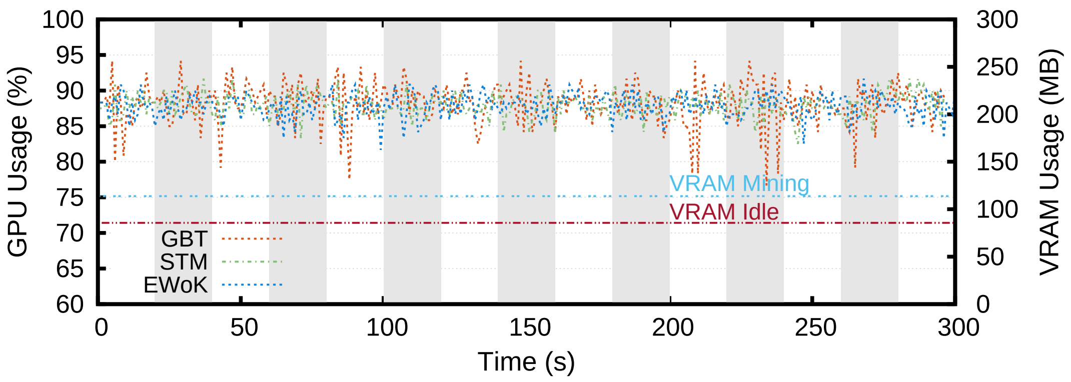

Figure 7: GPU usage of EWoK over time when compared to GBT and STM. Here, the white shaded areas in correspond to phase 1 of EWoK, while the gray shaded areas in the figure correspond to phase 2.

PoW solutions due to the higher target: 9.9 solutions/min compared to 5 and 4.6 solutions/min for GBT and STM.

GPU usage: In Figure 7, we evaluate the GPU usage incurred on a EWoK worker when compared to GBT and STM. We use the RadeonTop tool [22] to fetch the utilization of the GPU with a sampling interval of 1 second. Our results suggest that the switch between phases 1 and 2 in EWoK exhibit a smooth transition that mimics the work template transition exhibited by GBT and STM since the new work templates (for the various phases) are fetched in the background in parallel to the PoW mining. We additionally note that GBT exhibits the highest variance in GPU utilization when compared to EWoK and STM. We also point out that the VRAM consumption in EWoK, GBT, and STM is fixed at 113.80 MB, compared to the memory usage of 85.65 MB in the idle state. That is, in all investigated protocols, the worker only consumes approximately 30 MB even when mining.

Number of effective replicas: Table 1 shows the number of effective blockchain replicas achieved by EWoK with respect to the partition size when integrated with the Bitcoin, Litecoin, Dogecoin, and the Ethereum blockchains. Here, we assume that there are a total of 100,000 workers (adapted from [19]) and we vary the partition size in EWoK between 50 MB, 200 MB, and 500 MB by setting the chunk size accordingly. We measure the data growth per year for each of the investigated blockchains by analyzing their respective publicly available blockchain datasets. We implement the partitioning routine described in Algorithm 3 which results in a pseudorandom distribution of partitions amongst workers. We then compute the partition size averaged over all considered 100,000 workers. Our results show that assuming partition sizes of 200 MB and 500 MB results in the replication factor of almost 72 and 179, respectively, in the case of the Bitcoin blockchain, and almost 147 and 368 times (respectively) in the case of the Ethereum blockchain. Even small partition sizes of 50 MB result in a reasonable replication by 18 times in Bitcoin, 198 times in Litecoin, 143 times in Dogecoin, and 37 times in Ethereum.

Our results confirm that EWoK can be easily integrated within existing dedicated mining hardware; most dedicated hardware mining are equipped with at least 500 MB RAM capacity and the growth rate of the partition sizes in EWoK is slow enough to easily fit into planned growth in RAM capacities in the foreseeable future.

6 CONCLUSION

In this paper, we propose a solution, EWoK, to regulate the storage of the blockchain data among miners in the system. Our solution plugs an efficient proof of knowledge into standard hash-based PoW mechanism to force mining pool workers to store (in-memory) a modest (but fixed) partition of the blockchain data. EWoK leverages two sequential phases: in the first phase, the workers reach consensus on the exact set (including order) of transactions to be confirmed in a given block. In the second phase, the workers solve a proof of work instantiation based on this set and on their specific partition.

Our implementation results show that EWoK can be easily integrated within GBT and STM and does not impair their mining efficiency. This makes EWoK one of the few economically-viable and workable solutions that regulate, in a decentralized manner, the level replication of the blockchain data among mining pool workers.

ACKNOWLEDGEMENTS

This work was partly supported by the CyberSec4Europe project (G.A. no 830929), funded by the European Union (EU) under the Information and Communication Technologies (ICT) theme of the Horizon 2020 (H2020) research and innovation programme.

REFERENCES

[1] 2014. Bitcoin Miners Ditch Ghash.io Pool Over Fears of 51% Attack. https://www.coindesk.com/bitcoin-miners-ditch-ghash-io-pool-51-attack/.
[2] 2014. *How to Disincentivize Large Bitcoin Mining Pools.* http://hackingdistributed.com/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools/

Table 1: Number of blockchain replicas incurred by EWoK when integrated with Bitcoin, Litecoin, Dogecoin, and Ethereum assuming 100,000 workers. We use the reported sizes of these blockchains in April 2020.

Blockchain	Size (GB)	Growth (GB/year)	Partition size			
			Initial	50MB	200MB	500MB
Bitcoin	272.25	55.63	Replicas	17.94	71.74	179.74
			Growth (MB/year)	10.22	40.87	102.17
Litecoin	24.65	5.13	Replicas	198.09	792.34	1980.86
			Growth (MB/year)	10.41	41.62	104.06
Dogecoin	34.18	5.65	Replicas	142.86	571.42	1428.56
			Growth (MB/year)	8.27	33.06	82.65
Ethereum	132.58	50.21	Replicas	36.83	147.32	368.29
			Growth (MB/year)	18.94	75.74	189.36

[3] Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. 2016. Indistinguishable Proofs of Work or Knowledge. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*. 902–933. https://doi.org/10.1007/978-3-662-53890-6_30
[4] bfgminer 2017. BFGMiner. Available from: https://github.com/luke-jr/bfgminer.
[5] BIP22 2012. BIP22: getblocktemplate - Fundamentals. Available from https://github.com/bitcoin/bips/blob/master/bip-0022.mediawiki.
[6] BIP23 2012. BIP23: getblocktemplate - Pooled Mining. Available from https://github.com/bitcoin/bips/blob/master/bip-0023.mediawiki.
[7] bitcoin-bsz 2017. Bitcoin average block size. https://blockchain.info/charts/avg-block-size.
[8] BurstCoin [n.d.]. BurstCoin, Available from http://burstcoin.info, 2017.
[9] Matt Corallo. [n.d.]. BIP draft – BetterHash Protocol. https://github.com/TheBlueMatt/bips/blob/betterhash/bip-XXXX.mediawiki
[10] CryptoCurrency 2017. CryptoCurrency Market Capitalizations, Available from https://coinmarketcap.com/.
[11] Philip Daian, Ittay Eyal, Ari Juels, and Emin Gün Sirer. 2017. (Short Paper): PieceWork: Generalized Outsourcing Control for Proofs of Work.
[12] difficulty 2017. Bitcoin Difficulty. Available from: https://en.bitcoin.it/wiki/Difficulty.
[13] Filecoin 2014. Filecoin: A Cryptocurrency Operated File Network. Available from http://filecoin.io/filecoin.pdf.
[14] full1 2017. BitNodes. Available from https://bitnodes.21.co/.
[15] full2 2015. The Decline in Bitcoin Full Nodes. Available from https://bravenewcoin.com/news/the-decline-in-bitcoins-full-nodes/.
[16] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 3–16. https://doi.org/10.1145/2976749.2978341
[17] getblocktemplate 2015. Getblocktemplate - Bitcoin Wiki. Available from https://en.bitcoin.it/wiki/Getblocktemplate.
[18] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. 2014. Permacoin: Repurposing Bitcoin Work for Data Preservation. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP '14)*. IEEE Computer Society, Washington, DC, USA, 475–490. https://doi.org/10.1109/SP.2014.37
[19] minernumber 2015. Number of Bitcoin Miners Far Higher Than Popular Estimates. http://bravenewcoin.com/news/number-of-bitcoin-miners-far-higher-than-popular-estimates/ .
[20] naive 2014. How to Disincentivize Large Bitcoin Mining Pools. http://hackingdistributed.com/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools/.
[21] Sunoo Park, Krzysztof Pietrzak, Albert Kwon, Joël Alwen, Georg Fuchsbauer, and Peter Gazi. 2015. Spacemint: A Cryptocurrency Based on Proofs of Space. *IACR Cryptology ePrint Archive* 2015 (2015), 528.
[22] radeontop 2017. RadeonTop - GPU utilization morning. https://github.com/clbr/radeontop.
[23] Binanda Sengupta, Samiran Bag, Sushmita Ruj, and Kouichi Sakurai. 2016. Retricoin: Bitcoin Based on Compact Proofs of Retrievability. In *Proceedings of the 17th International Conference on Distributed Computing and Networking* (Singapore, Singapore) *(ICDCN '16)*. ACM, New York, NY, USA, Article 14, 10 pages. https://doi.org/10.1145/2833312.2833317
[24] Hovav Shacham and Brent Waters. 2008. Compact Proofs of Retrievability. In *ASIACRYPT*. 90–107.
[25] stratum 2015. Stratum mining - Bitcoin Wiki. Available from https://en.bitcoin.it/wiki/Stratum_mining_protocol.