Sharding PoW-based Blockchains via Proofs of Knowledge

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

Jens-Matthias Bohli
NEC Laboratories Europe
JensMatthias.Bohli@neclab.eu

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

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

***Abstract*—Blockchains based on proofs of work (PoW) currently account for more than 90% of the total market capitalization of existing digital cryptocurrencies. The security of PoW-based blockchains requires that new transactions are verified, making a proper replication of the blockchain data in the system essential. While existing PoW mining protocols offer considerable incentives for workers to *generate* blocks, workers do not have any incentives to *store* the blockchain. This resulted in a sharp decrease in the number of full nodes that store the full blockchain, e.g., in Bitcoin, Litecoin, etc. However, the smaller is the number of replicas or nodes storing the replicas, the higher is the vulnerability of the system against compromises and DoS-attacks.**

In this paper, we address this problem and propose a novel solution, EWoK (Entangled proofs of WOrk and Knowledge). EWoK regulates in a decentralized-manner the minimum number of replicas that should be stored 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—which is likely to increase its adoption by the existing PoW ecosystem. EWoK plugs an efficient in-memory hash-based proof of knowledge and couples them with the standard PoW mechanism. We implemented EWoK and integrated it within commonly used mining protocols, such as GetBlockTemplate and Stratum mining; our results show that EWoK can be easily integrated within existing mining pool protocols and does not impair the mining efficiency.

I. Introduction

In spite of their huge energy consumption and low performance, Proof of Work based blockchains currently account for more than 90% of the total market capitalization of existing digital cryptocurrencies [15] and are being used in a number of open blockchains such as Bitcoin, Litecoin, Dogecoin, and Ethereum.

These blockchains implement a novel distributed consensus scheme based on Proof of Work (PoW) that scales to a large number of nodes. Here, participants "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.

Recall that the current difficulty level of PoW mining is prohibitively high 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, rewards are shared between members of the mining pools, so called *workers*, based on the computational resources that they commit. There are currently a number of models for profit sharing in mining pools, such as PPS, PPLNS, among others. In all these models, 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).

Although profitable, mining pools offer a clear departure from the original model outlined in Nakamoto's whitepaper [39]. Namely, workers in a mining pool are mostly incentivized by obtaining the block reward and do not have strong incentives to store the blockchain, nor to operate a full Bitcoin node. In fact, most existing workers run dedicated mining protocols, such as Stratum mining [43] or GetBlockTemplate [24], 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 that store the full blockchain (cf. Figure 4) and a considerable surge in the number of workers and lightweight blockchain clients (which only store a small subset of transactions).

Given that blockchain security relies on the availability of all events (and their order of execution), we argue that the lack of incentives to store the blockchain data poses a serious threat to the security and sustainability of the PoW-based blockchains. In theory, ensuring few "trusted" replicas in the system suffices for security; however, the essence of blockchain is not to rely on "trusted nodes". Moreover, the smaller is the number of replicas, the higher is the vulnerability of the system against compromises and DoS-attacks.

In this paper, we address the problem and we propose a solution, EWoK (Entangled proofs of WOrk and Knowledge), to incentivize storing the blockchain data in the network. Our solution effectively divides the blockchain into shards and requires workers to locally store different shards in order to correctly solve the standard hash-based PoW. EWoK increases security by regulating in a decentralized-manner the *minimum* number of replicas that should be stored. EWoK achieves this by tying replication to the only directly-incentivized process in PoW-blockchains—which is PoW itself. In this sense, EWoK does not aim to increase the number of PoW-blockchains replications but distributes the responsibility of storing the blockchain ledger within those parties invested in making PoW-blockchains successful, even against abuse [3].

Notice that the literature features a number of contributions

that propose a re-purposing of the PoW to e.g., prove storage of archival data [19], [30], [35], [40]. Such re-purposing—although beneficial—however obviates the need for hash-optimized equipment (e.g., ASICs and FPGAs that have limited RAM and storage capabilities) and would require investments in different types of machinery (e.g., storage and exponentiation-optimized machinery). Namely, one of the (many) reasons that led to the sustainability of the Bitcoin blockchain was the large investments made by several mainstream companies in large data centers that are equipped with dedicated PoW mining capabilities [32], [33]. Due to their large market cap, it is clear that drastic changes to the rules governing the dynamics of the PoW ecosystem will be widely resisted by the backing industry.

Unlike these approaches, EWoK does not require any modification to PoW headers, is fully compliant with the specifications of existing mining hardware, and only incurs minor changes to the existing PoW protocols—which is likely to increase its adoption within the existing PoW ecosystem. EWoK plugs an efficient in-memory proof of knowledge into standard hash-based PoW mechanism to force workers to store (in-memory) a modest (but fixed) shard of the blockchain data. We integrated EWoK within existing mining pool protocols, such as GetBlockTemplate (GBT) and Stratum mining (STM); our results show that EWoK does not require any modification to these protocols and incurs marginal computational overhead to the existing PoW protocol adopted in existing PoW blockchains.

By ensuring that individual workers store *a small part* of the blockchain, EWoK emerges as the first workable attempt to incentivize storage in the blockchain and to practically amortize the costs of dispersing the blockchain data amongst various workers. In summary, we make the following contributions in this work:

- **Formal Framework:** We propose the first formal framework and a security model for a PoW that incentivizes storage of parts of the blockchain data. Our model, dubbed p-covering blockchain, extends the existing model of PoW and requires that each worker *independently* stores a fraction p of the blockchain data.
- **Concrete Instantiation:** We describe a concrete scheme, dubbed EWoK, which extends existing mining protocols with efficient in-memory proofs of knowledge and couples them with the standard PoW mechanism. EWoK leverages two interconnected 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 of transactions that is entangled with a proof of work over their specific shard. We show that EWoK is secure in our generic p-covering model.
- **Prototype Implementation:** We implement and evaluate a prototype based on EWoK using GPU mining and integrate it with STM and GBT. Our results show that EWoK does not require nor solicit changes in existing mining hardware, and achieves a 2% higher hash rate than GBT and only deteriorates the hash rate of STM by 1%.

The remainder of this paper is organized as follows. In Section II, we briefly recall basic information about blockchains based on Proofs of Work (PoW) and shed lights on their shortcomings. In Section III, 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 IV, we present EWoK and analyze its security. In Section V, we evaluate a prototype implementation based on the integration of EWoK with GPU mining based on STM and GBT. In Section VI, we review related work in the area, and we conclude the paper in Section VII.

II. Background & Problem Statement

A. PoW-based Cryptocurrencies

PoW-based blockchains[1] leverage Proofs of Work (PoW) as a public timestamping mechanism in order to prevent double-spending attacks. Namely, digital transactions are included in *blocks* that are broadcasted in the entire network. Note that when miners do not share the same view in the network (e.g., due to network partitioning), they might work on different block chains, thus resulting in "forks" in the block chain. Block forks are inherently resolved by the system; the longest block chain will eventually prevail. Although perfect synchronization within the network is not required, the parties are assumed to be loosely synchronized.

To prevent double-spending of the same coin, PoW-based cryptocurrencies rely on the synchronous communication assumption along with a hash-based PoW concept. More specifically, to generate a block, *miners* must find a block header that represents the solution of a PoW, i.e, when hashed, the result fulfills a certain criterion; for instance, it must be below a given target value (informally $H(\text{blockheader}) \leq target$.) If such a block header is found, miners then include it (as well as the additional fields) in a new block thus 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 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"[2], then blockchain nodes append it to their previously accepted blocks. Since each block links to the previously generated block, the Bitcoin block *chain* grows upon the generation of a new block in the network.

An example of a Bitcoin block header is depicted in Figure 8 in the Appendix. Notice that the hash function H and most of the values in the block header as the hash of the previous block LB and the (current) target $target$ are 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 only input parameters to the hash function that the worker can vary are directly a 4-byte nonce $nonce$ and indirectly the 32-byte root MR of the Merkle hash tree. The Merkle tree is built over the transactions where the first leaf plays a special role by

[1] In Section III-B, we present a formal model for PoW blockchains.

[2] That is, the block contains correctly formed transactions that have not been previously spent, and has a correct PoW.

Algorithm 1 Work flow for solving the PoW.

Input: Non-changeable block header parameters π_{blckhdr}

while PoW not solved **do**
  Choose a new value for $\mathsf{CB}_{\mathsf{nonce}}$ to specify the coinbase CB
  Compute the Merkle root MR over the coinbase CB and the set of transactions T
  for $nonce \in \{0,1\}^{32}$ **do**
    Compute $h := H(\pi_{\text{blckhdr}}, \mathsf{MR}, nonce)$
    if $h \leq target$ **then**
      break; {Solution found.}
    end if
  end for
end while

Output: The solution $\text{sol} = (\mathsf{CB}_{\mathsf{nonce}}, nonce)$.

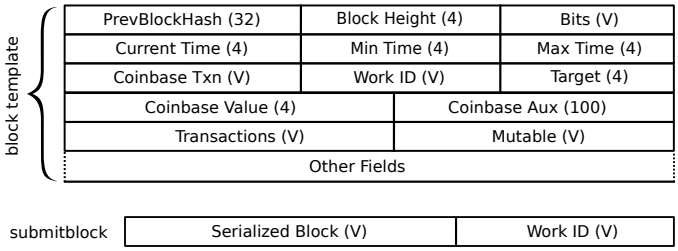

Fig. 1: Sketch of a GBT work template and corresponding worker's response. The pool operator outsources to its workers a block template that also contains the field CoinbaseAux. This field contains auxiliary information given by the pool operator which is used by the worker to populate parts of the coinbase transaction. Once a PoW solution is found, the worker constructs and serializes the corresponding block.

containing the coinbase transaction. The coinbase contains on the one hand information about the operator/worker (so that the mining reward can be claimed afterwards) and also a field (typically of 100 bytes) that can be fully changed depending on the underlying mining pool protocol. This field is used as an "extra nonce" to expand the search space available to workers (since the 32-bit nonce space can be exhausted very fast by dedicated mining hardware) without the need to frequently contact the pool operator for additional parameters. We refer to this field in the following by $\mathsf{CB}_{\mathsf{nonce}}$. Note that $\mathsf{CB}_{\mathsf{nonce}}$ is not directly input to the hash function but affects it implicitly as it influences MR. Thus, we summarize the process of solving the PoW in Algorithm 1. Notice that validation here is straightforward: given the block header parameters π_{blckhdr} and the value $\mathsf{CB}_{\mathsf{nonce}}$, one can compute the full block header and check whether its hash value is below the target $target$.

The main intuition behind PoW is that for peers to double-spend a given coin, they would have to replace the transaction where the coin was spent and the corresponding block where it appeared in, otherwise their misbehavior would be detected immediately. This means that for malicious peers to double-spend without being detected, they would not only have to redo all the work required to compute the block where that coin was spent, but also recompute all the subsequent blocks in the chain.

B. Mining Pool Protocols

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.

Namely, mining pools offer a way for miners to contribute their resources to generate a block and to split the reward between all the pool members following a certain reward payment scheme. Shares of the reward are assigned by the mining pool to its members (referred to as workers in the sequel) who presents 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 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. Appendix A outlines popular revenue sharing models adopted by existing mining pools.

Most existing mining pools adopt two main mining protocols: GetBlockTemplate [24] (cf. Figure 1) and Stratum mining [43] (cf. Figure 2). These protocols are supported by all existing hash-optimized mining hardware, such as ASICs, FPGAs, and GPUs. Due to lack of space, we give a detailed overview of these protocols in Appendix B.

C. Lack of Storage Incentives

PoW mining is strongly incentivized by coinbase transactions and fee collections in the system. For example, each block generation in Bitcoin currently awards miners a fixed revenue of 12.5 BTCs and a variable profit of 0.7 BTCs. This has lead to a considerable surge in the number of miners/workers over the last couple of years (cf. Figure 3).

Unlike mining, where participants are rewarded for confirming transactions, running a full node and storing the full blockchain does not provide any incentive. The only benefit to run a node is to help protect the network. As shown in Figures 3 and 4, in the early years of adoption of PoW blockchains, the system started in a well-balanced state; the majority of miners were basically full nodes that store the full blockchain. Later on, with the proliferation of dedicated mining

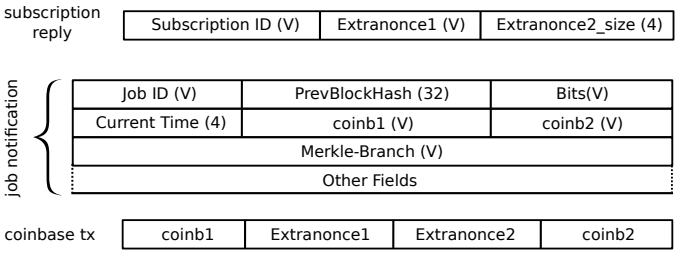

Fig. 2: Sketch of an STM work template and corresponding worker's response. Here, workers first receive the value of Extranonce1 and the *size* of Extranonce2 from the pool operator right after the initial subscription to the mining pool. Unlike GetBlockTemplate, the pool operator fixes all the transactions 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, which is enough to compute the Merkle root once the coinbase transaction is determined.

hardware, the surge of mining pools and lightweight clients, and the increase of the blockchain storage size, the mining process was almost completely decoupled from serving as a full node. This ecosystem change then outlined a lack of foresight in designing incentive mechanisms in existing PoW blockchains.

For instance, as shown in Figure 4, the number of full Bitcoin nodes dropped from 200,000 in 2014 to approximately 5,000 in 2016. Given lack of incentives, this number is only expected to decrease in the future. Although developers allow nodes to "prune" the blockchain—thereby freeing considerable space on their hard drive, nodes still need to download the full blockchain before pruning actually happens. Most users nowadays run lightweight clients which do not store the blockchain and only seldomly verify transactions/blocks of interest [21].

D. Problem Statement

The decrease of the number of full nodes directly impacts the security of the blockchain. As already stated, it is essential that the full and correct blockchain is available within the system at any time for validating new transactions. While in theory one "trusted" copy would suffice, this would contradict the fundamental decentralization principle of existing blockchains. Moreover, the less replicas are stored within the system, the more susceptible it becomes for compromise and (DoS) attacks.

A promising idea is to tie the replication to the only directly-incentivized process in PoW-blockchains—which is mining itself. In fact, literature features a number of interesting proposals for re-purposing PoW to prove the storage of archival data, but as we show in the coming paragraphs, none of these are applicable to the considered problem.

Most PoW-based blockchains, such as Bitcoin, use a hash-based PoW that has the task to find an input that is hashed to a value with certain properties (see Example 1 in Section III-B). That is, a PoW usually requires *many* hash executions over *randomly chosen* inputs. In Permacoin [30], the PoW is replaced by a Proof of Retrievability (PoR) over a large archival data file. This PoR essentially requires *few* iterations of a hash function on *selected* inputs (parts of the data file). Hence, Permacoin would require a drastic change in current mining hardware and consequently is less likely to be adapted by the community. Moreover, 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, Permacoin does not support a dynamically increasing file—such as the blockchain ledger. In Retricoin [40], the authors adopt the basic approach of Permacoin but aim for a more efficient solution in terms of storage overhead, network bandwidth, and verification time. For the same reasons as above, Retricoin is not applicable to the problem at hand.

Spacemint [35], Burstcoin [1], and Filecoin [19] 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.

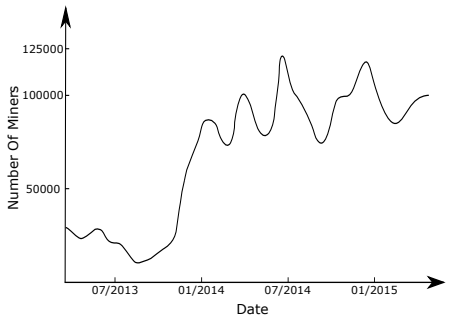

Fig. 3: Evolution of the number of Bitcoin miners over time. Estimates are adapted from [31].

This immediately raises the following question. *How can we ensure a minimum number of blockchain replicas without relying on any trusted parties and while being fully compliant to the current mining processes (cf. Section II-B)?* An ideal solution here should allow in a flexible manner to adjust the system-wide minimum number of replicas without wasting computational and storage resources.

III. p-Covering Blockchain

In this section, we introduce a formal model that extends PoW-based blockchains, dubbed p-covering blockchain. In a nutshell, a p-covering blockchain ensures that the blockchain is stored among the workers with a specified level of redundancy. We show at the end of this section how this solves the problem stated in Section II-D. In Section IV, we build on this model and propose a concrete instantiation of a p-covering blockchain which is fully compatible with existing hash-based mining hardware.

A. Notations

Throughout the rest of the paper, let λ denote the security parameter that determines the level of security and we denote by H a cryptographically secure hash function.

For a probabilistic algorithm A, we denote by $y \leftarrow A(x)$ the event that A on input x outputs y. To capture the notion of effort, we use the notation of steps as follows. Let $\mathrm{Steps}_A(x)$ denote 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.

B. Puzzle-Based Blockchains

We proceed with a short explanation of the basics of *puzzle-based blockchains*. A *blockchain* refers to a sequence of data entries, called *blocks*. A blockchain is maintained by a set of stateful parties $\mathcal{P}$ and represents a data storage where the content is trusted by anyone in $\mathcal{P}$ without the need of a centrally trusted instance. A blockchain is dynamic in the sense that new data, i.e., blocks, are continuously appended (within each time epoch). Formally, we see a blockchain BC as an array and denote the i-th block of BC by $BC[i]$.

Definition 1 (Blockchain System): A *blockchain system* is composed of a blockchain BC, a set $\mathcal{W}$ of workers (i.e., differ-

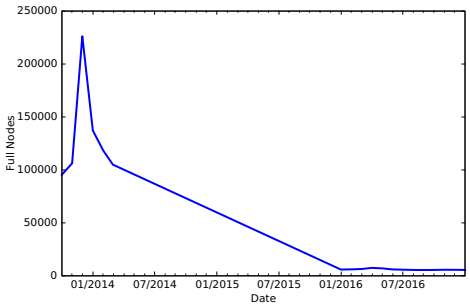

Fig. 4: Evolution of the number of Bitcoin full nodes over time. Estimates are adapted from [20], [21].

ent machines) who maintain the blockchain[3], and an interactive algorithm Consensus. At each point in time, the blockchain is a sequence of blocks where $s = \text{length}(BC)$ denotes its current length. The Consensus protocol is executed between the parties in $\mathcal{W}$ to commonly agree on the next block Bl of the blockchain. To increase the costs for appending incorrect content to the blockchain, most existing (open) blockchains (e.g., Bitcoin, Litecoin, Ethereum) require the worker to solve a *cryptographic puzzle* first before appending a block.

Cryptographic Puzzle: Roughly speaking, a *cryptographic puzzle* should be easy to generate, hard to solve, and easy to verify. We adapt and extend the model from [6].

Definition 2 (Puzzle System): Let λ denote the security parameter. A *puzzle system* is defined by several spaces and procedures. It comprises a puzzle parameter space $\mathcal{PP}_{\mathsf{PS}}(\lambda)$, a puzzle space $\mathcal{PS}_{\mathsf{PS}}(\lambda)$, a solution space $\mathcal{SS}_{\mathsf{PS}}(\lambda)$, and a hardness space $\mathcal{HS}_{\mathsf{PS}}(\lambda)$. These spaces specify a publicly known verification parameter π_{veri}. Moreover, there exist three algorithms $\mathsf{Sample}_{\mathsf{PS}}$, $\mathsf{Solve}_{\mathsf{PS}}$, and $\mathsf{Verify}_{\mathsf{PS}}$ defined as follows:

- $\mathsf{Sample}_{\mathsf{PS}}(\lambda,h,\pi_{\text{puz}})$ is a (possibly probabilistic) puzzle instance sampling algorithm. On input the security parameter λ, a hardness factor $h \in \mathcal{HS}_{\mathsf{PS}}$, and a puzzle parameter $\pi_{\text{puz}} \in \mathcal{PP}_{\mathsf{PS}}$, it outputs a puzzle instance $\text{puz} \in \mathcal{PS}_{\mathsf{PS}}$.
- $\mathsf{Solve}_{\mathsf{PS}}(\lambda, h, \text{puz})$ is a probabilistic puzzle solving algorithm. On input the security parameter λ, a hardness factor $h \in \mathcal{HS}_{\mathsf{PS}}$ and a puzzle instance $\text{puz} \in \mathcal{PS}_{\mathsf{PS}}$, it outputs a potential solution $\text{sol} \in \mathcal{SS}_{\mathsf{PS}}$.
- $\mathsf{Verify}_{\mathsf{PS}}(\lambda, h, \pi_{\text{veri}}, \text{puz}, \text{sol})$ is a deterministic puzzle verification algorithm. On input the security parameter λ, a hardness factor $h \in \mathcal{HS}_{\mathsf{PS}}$, a puzzle instance $\text{puz} \in \mathcal{PS}_{\mathsf{PS}}$ and a potential solution $\text{sol} \in \mathcal{SS}_{\mathsf{PS}}$ it outputs TRUE or FALSE.

A value puz is called a *valid puzzle* if there exists a puzzle parameter $\pi_{\text{puz}} \in \mathcal{PP}_{\mathsf{PS}}$ such that:

$$\Pr[\text{puz} \leftarrow \mathsf{Sample}_{\mathsf{PS}}(\lambda,h,\pi_{\text{puz}})] > 0.$$

For a given puz, one can efficiently, i.e., with an effort polynomial in λ, decide whether puz is a valid puzzle.
The hardness factor h specifies the effort to solve a puzzle using the $\mathsf{Solve}_{\mathsf{PS}}$ algorithm while λ specifies the difficulty to cheat on the system, i.e., to violate the soundness. To make the notation more simple, we omit in the sequel the security parameter λ and the hardness factor h if they are clear from the context. That is, we write $\mathsf{Sample}_{\mathsf{PS}}(\pi_{\text{puz}})$ instead of $\mathsf{Sample}_{\mathsf{PS}}(\lambda,h,\pi_{\text{puz}})$ and so on.

Naturally, a puzzle system should be *complete* and *sound*. Completeness means that a solution found by $\mathsf{Solve}_{\mathsf{PS}}$ for a correctly sampled puzzle will always be accepted:

Definition 3 (Completeness): We say that a puzzle system PS is *complete*, if for every hardness factor $h \in \mathcal{HS}_{\mathsf{PS}}$ it holds that:

$$\Pr\left[\begin{array}{l}\text{puz} \leftarrow \mathsf{Sample}_{\mathsf{PS}}(\pi_{\text{puz}}); \text{sol} \leftarrow \\ \mathsf{Solve}_{\mathsf{PS}}(\text{puz}) : \mathsf{Verify}_{\mathsf{PS}}(\pi_{\text{veri}}, \text{puz}, \text{sol}) \\ = \texttt{FALSE}\end{array}\right] = \text{negl}(\lambda)$$

While the notion of completeness is the same for all puzzle systems that we consider, the definition of soundness may vary. Recall that most public blockchains are currently based on cryptographic puzzles that require a certain amount of *work* to be solved, i.e., to provide so-called *proof of work*.

Proof of Work: A *Proof of Work* (PoW) system is a puzzle system as defined in Definition 2 that requires a certain amount of work, i.e., computational effort, to be solved. Here, 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_{PoW}.

Definition 4 (PoW Soundness): A PoW system is *g-sound* if for every PPT adversary $\mathcal{A}$, any $h \in \mathcal{HS}_{PoW}$ and every $\pi_{\text{puz}} \in \mathcal{PP}_{PoW}$, it holds that:

$$\Pr\left[\begin{array}{l}\text{puz} \leftarrow \mathsf{Sample}_{PoW}(\pi_{\text{puz}}); \text{sol} \leftarrow \\ \mathcal{A}(\text{puz}) : \mathsf{Verify}_{PoW}(\pi_{\text{veri}}, \text{puz}, \text{sol}) = \\ \texttt{TRUE} \wedge \text{Steps}_{\mathcal{A}}(h, \text{puz}) \leq \\ g(\text{Steps}_{\mathsf{Solve}_{PoW}}(h, \text{puz}))\end{array}\right] = \text{negl}(\lambda).$$

The definition of PoW-based blockchains is straightforward:

Definition 5 (PoW-based Blockchain): A g-sound PoW-based blockchain is a blockchain system where the process of appending a new block requires to solve at least one g-sound PoW.

We stress that the underlying PoW system is part of the system parameters of the blockchain system. That is, we assume that all parties participating into the blockchain system use the same PoW system and in particular the same puzzle sampling algorithm. Consequently, a potential solution sol provided by one worker for a puzzle puz is only accepted if $\mathsf{Verify}_{PoW}(\pi_{\text{veri}}, \text{puz}, \text{sol}) = \texttt{TRUE}$. However, any worker can select the puzzle parameter π_{puz} from a given range.

Example 1 (Hash-Based PoW—HPoW): Most PoW-based blockchains, such as Bitcoin, use a hash-based PoW as follows. Given a cryptographically secure hash function H and a bitstring puz, the task is to find a bitstring sol such that $H(\text{puz}, \text{sol}) \leq target$ for a pre-defined target $target$. We dub this PoW as HPoW and we give a more formal description of HPoW in Appendix C. In particular, we argue that HPoW is id-sound where id refers to the identity mapping. We will refer to HPoW multiple times within this paper.

C. Rational Workers

In the remainder of this paper, we assume *rational* workers in the following sense. We model a worker as a machine that

[3]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 dynamic sets of workers.

comprises one or several processing units and one local memory. That is, a worker can fetch data and store data to the local memory and can process data units. In addition, a worker can also request data from external data sources but as this requires network communication, downloading data takes significantly more time than simply reading it directly from local memory. The overall goal of a rational worker (i.e., the operator of the machine) is to maximize his probability in solving the PoW while minimizing his effort (which translate into costs).

Moreover, a worker is unlikely to alter its mining hardware characteristics unless such a move would drastically increase its advantage in solving PoW. Existing solutions that require the re-purposing of PoW, such as Permacoin, Filecoin, Retricoin [19], [30], [35], [40] are less likely to be adopted by workers since they obviate the need for hash-optimized ASICs and FPGAs equipment (that have limited RAM and storage capabilities) and would require investments in different types of machinery (e.g., storage and exponentiation-optimized machinery). For example, the hardness of solving a puzzle used in Permacoin [30] and Retricoin [40] relies on the assumption that a worker can only store part of the file—directly putting machines with larger RAM into strong advantage.

D. p-Covering Blockchain

As mentioned earlier, PoW *mining* is strongly incentivized by monetary rewards, gives no direct incentives to *store* the blockchain data. In this respect, we argue that any secure PoW-based blockchains should ensure a minimum level of replication of blockchain data in the system. Otherwise, there might be not enough information to verify PoW solutions.

A promising approach would be to couple the notion of blockchain data storage with the well-incentivized PoW mining process in order to offer an inherent incentive for workers to continuously support the system by storing parts of the blockchain. We capture this by requiring each worker to store at least parts of the blockchain data in its local memory in order to produce a possible PoW solution.

To capture the notion of storage, we adopt the common concept of *knowledge extractors*. A knowledge extractor algorithm $\mathcal{K}$ is given access to an algorithm A, represented by $\mathcal{K}^A$. Note that this access is non-black-box, e.g., $\mathcal{K}$ is allowed to rewind A. It is important to stress that both, $\mathcal{K}$ and A, have only access to the state of A, i.e., no external inputs are allowed. The interpretation is that $\mathcal{K}$ extracted the produced output from algorithm A.

Definition 6 (p-Covering Blockchain): Consider a PoW-based blockchain system with respect to a g-sound PoW system and a blockchain BC. This blockchain system is said to be *p-covering* for $0 \leq p \leq 1$ if it holds for any block that it can be reconstructed from a uniformly selected worker with a probability $\geq p$. Formally, there exists a knowledge extractor $\mathcal{K}$ such that it holds any block $BC[i]$ that:

$$\Pr\left[W \xleftarrow{\text{uniform}} \mathcal{W} : BC[i] \leftarrow \mathcal{K}^W\right] \geq p. \tag{1}$$

where $W \xleftarrow{\text{uniform}} \mathcal{W}$ means that W is uniformly sampled from $\mathcal{W}$. Note that when interacting with $\mathcal{K}$, a worker has no access to external inputs and has access to local memory only.

Notice that the case where $p = 0$ corresponds to existing PoW-based blockchain systems (see Definition 5). That is, a p-covering blockchain clearly generalizes PoW-based blockchain systems. Our goal in this paper is to devise an efficient p-covering blockchain where $p > 0$ while *(i)* requiring minimal changes to existing PoW blockchains and *(ii)* while being compliant with the specifications of existing mining hardware.

Such a p-covering blockchain would provide a solution to the problem stated in Section II-D as follows. A p-covering blockchain ensures for any randomly selected block to be stored by a p-fraction so that the expected number of replicas per block is $p \cdot |\mathcal{W}|$ where $|\mathcal{W}|$ is the number of workers. Moreover, the probability that a block is not included in any shard is at most $(1-p)^{|\mathcal{W}|}$. This ensures for any block a minimum number of decentralized replicas distributed over randomly selected miners. Note that these replicas are not meant to be frequently involved into the verification process but should be seen as a backup copy.

IV. EWoK: A Practical p-covering Blockchain

In this section, we present our solution, EWoK, and analyze its security. We do this iteratively; namely, we start by presenting and formalizing our approach, before detailing the specifications of EWoK.

A. A Strawman Solution

We start by considering the following strawman design which combines the notion of proofs of knowledge [19], [30], [35], [40] with the standard PoW mechanism. To this end, we first introduce an extension of the PoW-notion that binds a PoW to a file. The core idea is that an algorithm can only efficiently solve a PoW if it has stored a certain file $\mathbf{F}$.

Definition 7 (File-Bound PoW): A PoW is *g'-bound to some file* $\mathbf{F}$ if there exists a knowledge extractor $\mathcal{K}$ such that it holds for every PPT adversary $\mathcal{A}$ and every puzzle parameter $\pi_{\text{puz}} \in \mathcal{PP}_{PoW}$ that:

$$\Pr\left[\begin{array}{l} \text{puz} \leftarrow \mathsf{Sample}_{PoW}(\pi_{\text{puz}}); \text{sol} \leftarrow \mathcal{A}(\text{puz}); \\ \mathsf{Verify}_{PoW}(\pi_{\text{veri}}, \text{puz}, \text{sol}) = \texttt{TRUE}; \\ \text{Steps}_{\mathcal{A}}(\text{puz}) \leq g'(\text{Steps}_{\mathsf{Solve}_{PoW}}(\text{puz})): \\ \mathbf{F}^* \leftarrow \mathcal{K}^{\mathcal{A}} : \mathbf{F}^* \neq \mathbf{F} \end{array}\right] = \text{negl}(\lambda).$$

Note the difference to the g-soundness of a PoW. There, the definition actually gives a lower bound specified by a function g on the effort of any attacker. Here, the definition says that if an attacker aims to have an effort that is below a certain upper bound, specified by g', it has to store the file $\mathbf{F}$. Namely, we see different types of thresholds with respect to the time effort an attacker can achieve:

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

A strawman solution to realize a file-bound PoW 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. Since checking knowledge of the full blockchain is infeasible, a reasonable approach is to challenge a random selection of blocks of the blockchain such that the response can only be computed if these blocks are presented. For coupling both concepts, a PoW and a PoK, one can proceed as follows:

1) A worker has to solve a PoW puzzle puz. We denote the solution found by the worker by sol.
2) Compute from sol a PoK-challenge over BC, e.g., by means of a pseudorandom function.

Since the "file", i.e., the blockchain, is publicly known, conceptually simple proofs of knowledge are possible. For instance, a straightforward approach is to require the hash over a pseudorandom selection of blocks. A summary of this strawman solution is given in Algorithm 4 in the Appendix.

Note that existing schemes that re-purpose PoW to prove storage of a file F [19], [30], [35], [40] follow a similar approach but cannot be directly used in our case where F is the entire blockchain. Namely, in these solutions, any worker that stores the full blockchain can immediately solve the puzzle and produce a new block. Given the still moderate size of the blockchain, machines equipped with large RAM space would have more advantage over a fast Gigahash worker. Moreover, 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 the blockchain that constantly grows in size.

Although the aforementioned strawman solution already overcomes these limitations, we stress at this point that it does not necessarily satisfy Definition 6. Namely, although this solution ensures that it is g-bound to the blockchain in case that the underlying PoW is g-sound, it does not enforce that different workers store a *replicated* version of the blockchain. Namely, 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, this solution only ensures that the pool operator (or any other central entity) stores the full blockchain. In fact, one may consider one party that is only dedicated to solving the PoK at the end of the mining process. In any case, there is no guarantee that the workers store any parts of the blockchain. Notice that there are currently a handful of mining pools [22] whose operators already typically store the full blockchain.

In addition, any solution should be compatible with existing mining protocols. For example, in PoW blockchains, miners are rewarded by the transaction specified in the coinbase (cf. Section II-B)—which overcomes the need for a cumbersome key management process. Solutions such as Permacoin however deploy specific types of keys that eventually are used for signing the rewarding transaction. Note that these keys need to be updated regularly, incurring further computational effort incompatible on the hashing machines.

B. Sharding the Blockchain Storage

To overcome the limitations of the aforementioned strawman solution, we now introduce a further extension. Namely, we demand that each worker has to store a part of the file only, called a *shard*, which is determined by a sharding function. In EWoK, the shard 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). For the sake of generality, we define a sharding function over an abstract input space $\mathcal{S}$.

Definition 8 (Sharding Function): Consider a file $\mathbf{F}$, divided into ℓ blocks $\mathbf{F}[i]$. A *sharding function* shrd is a mapping that accepts inputs x from some input space $\mathcal{S}$ and outputs a fraction of the file, i.e., $\mathrm{shrd}(s)=[\mathbf{F}[i_1],...,\mathbf{F}[i_{\ell'}]]$ where $0\leq i_1<...<i_{\ell'}$ and $\ell'\leq\ell$. Slightly abusing notation, we say for a block $\mathbf{F}[i]$ that $\mathbf{F}[i]\in\mathrm{shrd}(s)$ if $\mathrm{shrd}(s)=[\mathbf{F}[i_1],...,\mathbf{F}[i_{\ell'}]]$ and there exists an index i_j such that $i_j=i$.

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

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

where $x\xleftarrow{\$}\mathcal{S}$ means that x is uniformly sampled from $\mathcal{S}$.

We consider now a variant of file-bound PoW where efficiently solving a puzzle puz requires to solve a shard of the file $\mathbf{F}$ only. This is covered by requiring 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 shard.

Definition 9 (Shards-Bound PoW): Consider a PoW that is file-bound to some file $\mathbf{F}$. Moreover, assume a sharding function shrd that is defined on $\mathcal{PS}_{PoW}$. That is, the input space of shrd is the puzzle space of PoW. The PoW is said to be g'-bound to ($\mathbf{F}$,shrd) (or *shards-bound* for short) if there exists a knowledge extractor $\mathcal{K}$ such that it holds for every PPT adversary $\mathcal{A}$, every $\pi_{\mathrm{puz}}\in\mathcal{PP}_{PoW}$, and any $h\in\mathcal{HS}_\lambda$ that:

$$\Pr\left[\begin{array}{l}\mathrm{puz}\leftarrow\mathsf{Sample}_{PoW}(\pi_{\mathrm{puz}});\mathrm{sol}\leftarrow\mathcal{A}(\mathrm{puz});\\ \mathsf{Verify}_{PoW}(\mathrm{puz},\mathrm{sol})=\mathtt{TRUE};\\ \mathrm{Steps}_{\mathcal{A}}(\mathrm{puz})\leq g'(\mathrm{Steps}_{\mathsf{Solve}_{PoW}}(\mathrm{puz})):\\ \mathbf{F}^*\leftarrow\mathcal{K}^{\mathcal{A}}:\mathbf{F}^*\neq\mathrm{shrd}(\mathrm{puz})\end{array}\right]=\mathrm{negl}(\lambda).$$

Notice that by setting $\mathrm{shrd}(x)=\mathbf{F}$ for all $x\in\mathcal{S}$, a shards-bound PoW becomes a file-bound PoW. The definition does not prevent $\mathcal{A}$ to access external data sources, but this would imply higher time effort. Hence, a rational worker (cf. Section III-C) has strong incentives to store the whole shard locally.

Definition 10 (Independent Shards): Consider a shards-bound PoW-based blockchain system. We say that this blockchain system ensures *independent shards* if it holds for any block that the probability of being part of one shard is practically independent of whether it is part of another shard. More precisely, let $\mathbb{E}(W,\mathsf{Bl})$ denote the event that block Bl is part of shard stored by worker W. Ensuring independent shards

means formally that any two different rational $W,W' \in \mathcal{W}$ and any block $\mathsf{Bl} := BC[i]$ it holds that:

$$\begin{aligned} &|\Pr[\mathbb{E}(W,\mathsf{Bl}) \wedge \mathbb{E}(W',\mathsf{Bl})] \\ &- \Pr[\mathbb{E}(W,\mathsf{Bl})] \cdot \Pr[\mathbb{E}(W',\mathsf{Bl})]| \quad = \quad \mathrm{negl}(\lambda). \end{aligned}$$

Our main goal is to devise a PoW-based blockchain system that is shards-bound with respect to a p-covering sharding function and with independent shards. This is motivated by the following result:

Theorem 1: A PoW-based blockchain system that is shards-bound for the identity function $g' = \mathrm{id}$ with respect to a p-covering sharding function and that ensures independent shards is a p-covering blockchain (see Definition 6).

We show the theorem informally. Recall that we consider only rational workers that aim to solve the underlying PoW as fast as possible to compete with other workers. As the PoW is id-bound to the shard, a rational worker practically realizes the Solve_{PoW} algorithm with continuous access to the shard to his specific puzzle. Notice that the best strategy for the worker is to have the shard loaded in RAM in order to minimize memory access overhead (from the network or from disk). The fact that the system ensures independent shards with respect to a p-covering sharding function ensures that each worker is storing a different p-fraction of the blockchain and hence is p-covering.

Notice that maintaining "correct replicas" within the RAM of individual miners in EWoK does not necessarily give more control to mining pools. Individual miners have shown great awareness of the dangers of centralization in the past and they are likely to change mining pools (to preserve their investments) whenever they feel that the pool operator is not acting in the interest of the system: see [3] for example. As such, we believe that EWoK places the additional responsibility of storing parts of the blockchain data within those nodes who are truly invested in making PoW blockchains successful.

In what follows, we present a concrete instantiation, EWoK, that practically instantiates a shard-bound PoW.

C. EWoK: Protocol Specification

We start by describing the sharding function (see Definition 8) that are used in EWoK, i.e., that are part of the specification of EWoK. Afterwards, we explain how to turn the Bitcoin PoW into a shards-bound PoW (see Definition 9).

1) Specification of the Sharding Function:

Recall that BC denotes the blockchain of size s and $BC[i]$ the i-th block. For the specification of the sharding function, EWoK divides the blockchain into *chunks* with equal chunk size N, i.e., the first N blocks form the first chunk and so on. The shard function selects pseudorandomly (with H) exactly one block per chunk. A complete description is given in Algorithm 2. The description is generic based on some input x; in the following paragraphs, we discuss the specification of the input x.

This sharding function provides various advantages:

- It is p-covering (see Section IV-D).
- It is independent of the current size of the blockchain.
- Although the shards grow over time (this is a consequence of the p-covering requirement), this does not affect or modify the storage of blocks already contained in the shard.
- The parameters can be reasonably chosen to ensure that shard's size fit within the available memory of existing dedicated mining hardware (see Section V).
- 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 V.

Algorithm 2 Sketch of the sharding function shrd in EWoK.

Input: Value $x \in \mathcal{S}$

1: $shard := []$ {Initialize an empty shard}
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 shard.}
4: Compute $ind := (i-1) \cdot N + j$ {Gives the total index of the block to be appended.}
5: $Append(shard, BC[ind])$ {Appends the selected block to the shard. If $BC[ind]$ does not exist yet, this command is ignored.}
6: **end for**

Output: Shard $shard$.

2) Specification of the Shards-Bound PoW:

The shards-bound PoW in EWoK is composed of two phases, each using a variant of the hash-based PoW HPoW (cf. Section II-A) with targets $target_1$ and $target_2$, respectively. 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 variant of HPoW based on this set that entangles the proof of work with a proof of knowledge on their specific shard. While this entanglement of PoW with PoK ensures that the PoW is shard-bound, it does not guarantee that different workers store independent shards. 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 shards is not possible[4]. Instead, EWoK gives economic incentives to the workers to store independent shards by ensuring that two workers operating over the same shard effectively decrease their 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.

In what follows, we explain EWoK in detail. We focus here on the description of the algorithms and discuss concrete suitable choices in subsequent sections.

Fixing the shard: Recall that the transactions exclude the coinbase transaction CB. In the sequel, we assume that the coinbase transaction is divided into two parts $CB = (CB_1, CB_2)$ such that CB_1 contains all information that specify the rewarding transaction for the miner. This is conforming with the operation of GBT and STM (cf. Section II). Each of these

[4]Notice that permission-based blockchains (e.g.,Hyperledger Fabric [4]) (where workers have a clear identity) allow to directly enforce the property of independent shards, i.e., by coupling them to the identity of the worker.

parts of CB plays a different role in EWoK. The first part, CB_1, determines the shard to which the PoW is bound. We have $shard := shard_{CB_1} := \mathrm{shrd}(H(CB_1))$. The second part, CB_2, 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 reach consensus amongst workers of the same mining pool on the set of transactions (including order) to be used in the second phase. We will show in Section IV-D that this phase particularly *enforces* workers to operate all over the same set of transactions. To achieve such consensus while leveraging existing mining hardware/software, EWoK deploys HPoW with target $target_1$. That is, a set of transactions T together with a specified order is said to be *valid* with respect to a nonce $nonce_{trans}$ if it holds:

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

In summary, the task of a worker in the first phase is to find a valid set of transactions together with the corresponding nonce $nonce_{trans}$.

Second phase — mining a block: Given 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 that solves a specific hash-based proof of work with respect to T. To this end, EWoK adopts the core idea from HPoW but entangles it with in-memory proofs of knowledge of the shard $shard_{CB_1} := \mathrm{shrd}(H(CB_1))$. Let $\mathsf{MR}(CB, T)$ denote the root of the Merkle tree where CB defines the first leaf and the following leaves are specified by T. Recall that CB_1 (part of CB) determines the shard per worker. The task is now to find a value CB_2 and a value $nonce$ such that:

$$H(\pi_{\mathrm{blckhdr}}, \mathsf{MR}((CB_1, CB_2), T), nonce) \leq target_2. \tag{3}$$

Here, the value CB_2 has the form $CB_2 = (nonce_{trans}, \mathsf{CB}_{\mathsf{nonce}})$, where $nonce_{trans}$ is output by the first phase and only the second value, $\mathsf{CB}_{\mathsf{nonce}}$, can be varied. To couple this PoW with a PoK over the shard $shard_{CB_1}$, the values of $\mathsf{CB}_{\mathsf{nonce}}$ are derived from the blocks of the shard as follows.

Let f_{trans} denote a function (e.g., a cryptographic hash) that derives from a given set of transactions T a characteristic value $v_T = f_{\mathrm{trans}}(T)$. We assume that v_T is computed once initially. The computation of $\mathsf{CB}_{\mathsf{nonce}}$ involves v_T, the blocks comprising the shard, and a value pn called the *pre-nonce* (to reflect the fact that it will be used as a seed to determine another nonce):

$$\mathsf{CB}_{\mathsf{nonce}} := (H(v_T, shard_{CB_1}[index], pn), pn). \tag{4}$$

Analogous to the first phase, we restrict the space of possible pre-nonces to enforce independent shards (cf. Definition 10). As the PoW is tied to a shard, workers who operate over the same shard share the same restricted nonce space and hence reduce their probability of success. As we show in Section IV-D, this gives incentives to rational workers to choose different shards. To restrict the nonce space, EWoK enforces an upper bound $\delta_{\mathrm{pre-nonce}}$ when choosing pn:

$$pn \text{ valid} \Leftrightarrow pn \leq \delta_{\mathrm{pre-nonce}}. \tag{5}$$

It remains to explain how the index $index$ that determines which block of the shard of length ℓ_{shrd} is involved in the computation of $\mathsf{CB}_{\mathsf{nonce}}$ (see Equation 4) is computed. To this end, we simply set

$$index := f_{\mathrm{ind}}(v_T, pn) := H(v_T, pn) \mod \ell_{\mathrm{shrd}}. \tag{6}$$

An overview of these steps is given in the last part of Algorithm 3. If $h \leq target_2$, 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 straightforward and given in Algorithm 5 in the Appendix.

Algorithm 3 The algorithm $\mathsf{Solve}_{\mathrm{EWoK}}$.

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

Compute Shard
Split the coinbase $CB = (CB_1, CB_2)$
Compute $shard := \mathrm{shrd}(H(CB_1))$.

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

Solve Proof of Work (second phase)
Compute $v_T = f_{\mathrm{trans}}(T)$
while Proof of Work not solved **do**
  Choose a pre-nonce $pn \leq \delta_{\mathrm{pre-nonce}}$
  Compute index $index := f_{\mathrm{ind}}(v_T, pn)$
  Compute $\mathsf{CB}_{\mathsf{nonce}} := (H(v_T, shard[index], pn), pn)$
  Set $CB = (CB_1, (nonce_{trans}, \mathsf{CB}_{\mathsf{nonce}}))$
  for $nonce \in \{0,1\}^{32}$ **do**
    Compute $h := H(\pi_{\mathrm{blckhdr}}, \mathsf{MR}(CB, T), nonce)$.
    if $h \leq target_2$ **then**
      PoW-solution found. Break WHILE-loop.
    end if
  end for
end while
Output: Solution $(T, CB, nonce)$.

3) Practical Considerations:

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. 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 for the second phase such that, on average, only one solution can be expected in the second phase for two reasons. First, this approach discourages different workers to operate over the same shards and hence

gives incentives for storing independent shards. Second, one needs to take into account that two-phase PoW protocols tend to favor large mining pools when implemented in a naive way [16], [34]. The reason is that finding a solution to the two-phase puzzle is no longer a Poisson process, where every miner has in every moment the same probability to succeed. Instead, a solution can only be found in the second phase, and large mining pools will typically progress faster to the second stage. For example, assuming a 2-stage PoW (where both phases have equal difficulty) and only two miners where one miner has the double computing power of the other, the winning probability of the more powerful miner increases to 74% instead of the default 66.6%. Similar to [34], we mitigate this by ensuring that for any Phase 1 solution, there is only one possible Phase 2 solution. For example, assuming the difficulty in the Bitcoin network on the 10th of August 2017, this can be achieved by restricting the nonce space to approximately 70 bits. Note that a worker can control two values in phase 2, the 32-bit value $nonce$ and the pre-nonce value pn, so that we restrict the freedom of the latter to 38 bits. We validate this choice of parameters experimentally in Section V.

Note that while a worker could in addition also vary the timestamp, this is possible within a very small range only as otherwise the block becomes outdated. Thus, the impact of varying the timestamp can be neglected. Clearly, other tradeoffs are possible as long as the total effort of both phases equals approximately the effort of the HPoW and as long the set of allowed pre-nonces in phase 2 is not too large.

Work Flow: We stress that, in practice, the computation of the shard will be done only once by each worker. To ensure efficient execution of phases 1 and 2, the workers are likely to load their respective shards in RAM. We show in Section V that this is a reasonable assumption; namely, we show how to realize modest shard sizes (with approximately few hundred MBs) that can easily fit into the available RAM of most existing mining hardware. Notice, however, that the shard size will grow with the length of the blockchain. In this case, workers will continuously update their shards using the non-interactive deterministic sharding function specified in Algorithm 2. However, although the shards grow over time, EWoK ensures that this does not affect or modify the storage of blocks already contained in the shards stored by each worker.

Moreover, as we show in Section IV-D, 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 3 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.

As motivated in Section IV-A, we assume that the verification of the blocks is done by parties that store the full, public blockchain. Recall that pool operators currently store the full blockchain in order to verify new blocks and outsource new puzzles.

Instantiation: 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 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 9 in the Appendix. We stress at this point that this process does not necessarily require a large mining pool and can be equally applied e.g., by a solo miner.

While phase 2 of EWoK is compatible with all current mining hardwares, 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 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.

D. Security Analysis

In this section, we analyze the security of our solution with respect to the model outlined in Section III. More precisely, we argue that EWoK is a shards-bound blockchain system (Lemma 2) with respect to a p-covering sharding function (Lemma 1) and that it provides independent shards (Lemma 3). Based on this, it follows from Theorem 1 that EWoK is a p-covering blockchain.

We base our arguments on the random oracle model with respect to the deployed hash function H. That is, the outputs of the hash function for pairwise different inputs are independent and uniformly distributed in $\{0,1\}^n$. Notice that, in EWoK, the hash function is used in different contexts, e.g., for checking the validity of a set of transactions (cf. Equation 2) or when a solution for the PoW in the second phase has been found (cf. Equation 3). We assume a distinguished input message format in each context to have formally independent H-calls in the different phases.

Lemma 1: The sharding function in Algorithm 2 is $(\frac{1}{N} - \frac{N}{2^n})$-covering, where N denotes the chunk size and n the output size of the hash function. If $2^n \gg N$, the sharding function is practically $\frac{1}{N}$-covering.

Proof: Recall that the sharding function splits the blockchain into chunks of size N and selects exactly one block per chunk. If the number is a multiple of the chunk size, then the probability for each index $j \in \{0,...,N-1\}$ 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, the higher indexes have a smaller probability of at least $\frac{1}{N} - \frac{N}{2^n}$ to be selected. ■

Lemma 2: EWoK is id-bound to $(shard_{CB_1}, \mathrm{shrd})$.

Proof: We now show that the PoW used in the second phase of EWoK is id-bound to $(shard_{CB_1}, \mathrm{shrd})$. Let Solve_2 denote the algorithm that solves this PoW of the second phase, i.e., the last part of Algorithm 3. Notice that this PoW is actually a variant of HPoW (cf. Section III-B) that we showed to be id-sound. More precisely, the considered PoW is based on the task to find an input of a certain form to the hash function such that the output fulfills a certain condition. As the hash function invocations are independent and hence the outputs unpredictable, uniformly distributed values, the success probability of any solution-finding algorithm can be expressed by the number of hash queries based on different inputs. That is, the optimal solving algorithm consists of checking as many hash inputs as possible, being exactly the work flow of Solve_2.

Now, let Solve_W denote the solution-finding algorithm implemented by some worker W. In case of a rational attacker, it requires finding an appropriate value $\mathsf{CB}_{\mathsf{nonce}}$ such that $H(\pi_{\mathrm{blckhdr}},\ \mathsf{MR}(CB,T),\ nonce) \leq target$. Recall that the values for $\mathsf{CB}_{\mathsf{nonce}}$ are indirectly given by

$$\mathsf{CB}_{\mathsf{nonce}} := (H(v_T, shard[index], pn),\ pn)$$

where only the value pn is under full control of the worker. Notice 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 but 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 $shard[index]$. Similar to the above, a worker who does not use $shard[index]$ will not be able to eventually produce a valid solution. While in principle the worker could fetch the block $shard[index]$ from external sources, this increases the total number of steps due to additional idle steps when the algorithm waits for $shard[index]$. Thus, the most efficient strategy for a worker is to store the necessary shard blocks $shard[index]$ in its local memory. Moreover, we point out that the indexes $index$ are pseudorandomly generated, i.e., $index := f_{\mathrm{ind}}(v_T, pn)$; these indexes also differ between epochs due to v_T. Therefore, the indexes $index$ cannot be predicted in the random oracle model, making all blocks of the shard equally necessary. ■

Lemma 3: EWoK ensures independent shards.

Proof: (Sketch) The mining process of EWoK includes two phases and each phase requires to solve a PoW. Consider an worker W implementing algorithms Solve_1 and Solve_2 to solve the PoW in phase 1 and 2, respectively. Using the same arguments as in the proof of Lemma 2, it holds that both Solve_1 and Solve_2 execute continuously different attempts to solve the respective PoW by selecting and trying different possible solutions. We call an attempt in Solve_1 to solve the PoW of the first phase a *type-1-attempt* and define *type-2-attempts* analogously. As the probability for successful mining grows linearly with the number of type-2-attempts only, the optimal strategy for any worker is to minimize the number of type-1-attempts and to maximize the number of type-2-attempts. This encourages the strategy that once one worker has found a solution for the PoW in phase 1, it is shared with the other workers among the mining pool so that all workers proceed to phase 2.

Notice that the nonce space restriction deployed in phase 2 actually limits the number of possible type-2-attempts with respect to one choice of CB_1. Once all nonces have been used with respect to CB_1, one can either try to change to another valid set of transactions or change CB_1. The first requires to run further type-1-attempts—contradicting the optimal strategy. The second makes it necessary to determine and load another shard (as $shard = \mathrm{shrd}(CB_1)$ into local memory (see arguments in the proof sketch of Lemma 2)—preventing the worker to run further type-2-attempts. Thus, the best strategy is to allow for as many type-2-attempts as possible; this means that different workers should operate on different choices of CB_1. As CB_1 fixes pseudorandomly the shard, different values CB_1 lead to shards that are computationally indistinguishable from independently randomly selected shards. Moreover, since CB_1 specifies the rewarding transaction, a miner cannot outsource this computation to a different party as this party would have no incentives to run the computation on behalf of this miner. This ensures that the shards are stored independently in different RAM. ■

It follows from Lemma 1, 2, and 3 that the prerequisites of Theorem 1 are given and hence that EWoK is an instantiation of a p-covering blockchain.

V. Implementation & Evaluation

In what follows, we evaluate a prototype implementation of EWoK integrated within GPU mining. The purpose of our evaluation is to validate the feasibility of integrating EWoK within existing mining protocols such as STM and GBT. By showing that EWoK can be instantiated within these protocols without modifications, we stress that EWoK can be immediately instantiated 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.

A. Implementation Setup

We integrate EWoK within the popular open-source mining tool BFGMiner [8] 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 implementations 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 IV-C3. Namely, we assume that the operator distributes the search for a valid set of

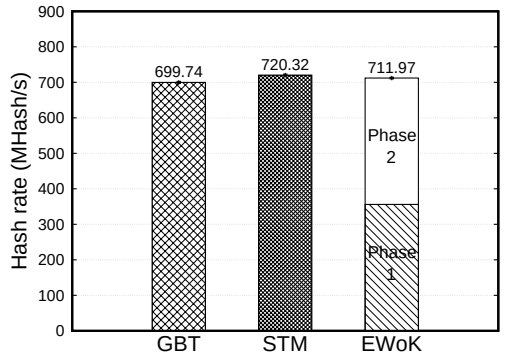

Fig. 5: Effective hash rate performance of EWoK when compared to GBT and STM.

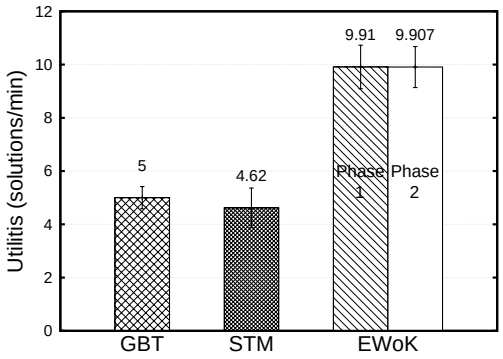

Fig. 6: Number of solutions mined in EWoK when compared to GBT and STM.

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 [11]. Each data point in our plots is averaged over 10 independent measurements; where appropriate, we include the corresponding 95% confidence intervals.

B. Performance Evaluation

We evaluate the performance of EWoK with respect to the achieved hash rate, the number of PoW solutions found per minute, and the GPU usage, when compared to existing mining protocols based on GBT and STM. We also evaluate the effective number of blockchain replicas enforced by EWoK within existing PoW-based blockchain deployments, such as Bitcoin, Litecoin, Dogecoin, and Ethereum.

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

Blockchain	Size (GB)	Growth (GB/year)	Shard size			
			Initial	50MB	200MB	500MB
Bitcoin	136.34	38.24	Replicas	35.81	143.25	358.13
			Growth (MB/year)	14.02	56.10	140.25
Litecoin	46.27	1.63	Replicas	105.53	422.11	1055.29
			Growth (MB/year)	1.77	7.06	17.66
Dogecoin	21.57	2.68	Replicas	226.37	905.48	2263.71
			Growth (MB/year)	6.21	24.82	62.05
Ethereum	7.7	3.69	Replicas	634.13	2536.53	6341.31
			Growth (MB/year)	23.93	95.72	239.3

TABLE I: Number of blockchain replicas incurred by EWoK when integrated with Bitcoin, Litecoin, Dogecoin, and Ethereum assuming 100,000 workers.

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 [17] to be 2 (target 0x1d007fff) for GBT and STM. Following EWoK's specification in Section IV-C, 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 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 [38] 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

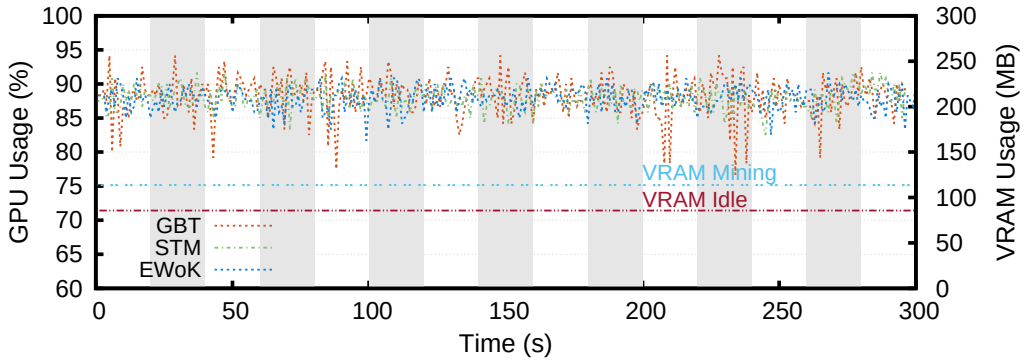

Fig. 7: GPU usage of EWoK over time when compared to GBT and STM. Here, the white shaded areas in correspond to phase 1 of EWoK, while the gray shaded areas in the figure correspond to phase 2.

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 I shows the number of effective blockchain replicas achieved by EWoK with respect to the shard 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 [31]) and we vary the shard 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 sharding routine described in Algorithm 2 which results in a pseudorandom distribution of shards amongst workers. We then compute the shard size averaged over all considered 100,000 workers. Our results show that assuming shard sizes of 200 MB and 500 MB results in the replication factor of almost 143 and 358 times, respectively, in the case of the Bitcoin blockchain, and almost 2500 and 6300 times (respectively) in the case of the Ethereum blockchain. Even small shard sizes of 50 MB result in a reasonable replication by 35 times in Bitcoin, 105 times in Litecoin, 226 times in Dogecoin, and 634 times in Ethereum.

As mentioned earlier, EWoK enables workers to grow their shard sizes over time without affecting their existing stored shard. For example, given that the growth rate of the Bitcoin network is 38 GB per year, each worker's shard grows almost by 140 MB per year in EWoK when current shard size is set to 500 MB. Table I summarizes the shard growth rate in the investigated blockchains. 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 shard sizes in EWoK is slow enough to easily fit into planned growth in RAM capacities in the foreseeable future.

VI. Related Work

Alternatives to PoW-based blockchains: In the recent years, a number of alternatives to PoW have been proposed [23]. For instance, *Proof of Stake* (PoS) [37] suggests to bias the vote impact of participants based on the amount of "stake" they own in the respective blockchain system. The literature features a number of additional proposals [18], [28], [29], [36], [44] that rely on classical Byzantine fault tolerant consensus protocols in the hope to increase the consensus efficiency and achieve high transactional throughput.

Some proposals [14], [42] aim to replace the linear structure of a blockchain by other structures, e.g., a directed-acyclic-graph. PieceWork [2], [16] leverages a 2-phase PoW protocol similar to EWoK; none of these schemes are designed to incorporate proofs of knowledge over blockchain data shards.

A number of other proposals propose to re-purpose PoW to prove storage of archival data [19], [30], [35], [40]. As discussed in Section II-D, all of these schemes are however not suited for ensuring the storage of the blockchain in existing PoW blockchains since they obviate the need for hash-optimized ASICs and FPGAs equipment (that have limited RAM and storage capabilities).

Proofs of Knowledge: Proofs of Knowledge (PoK) are cryptographic protocols in which a verifier is convinced that the prover knows some secret [7]. PoK are often based on zero-knowledge proofs, which have the property that the proof does not disclose any information not already known to the verifier [25]. Another line of work provides proofs that a certain file is stored without enforcing the verifier to keep a local copy, e.g. PDP and POR [5], [13], [26], [41]. Notice that the requirements in EWoK differ from these. One the one hand, we can assume that the knowledge, i.e. the blockchain, is fully known to the verifier. On the other hand, EWoK plugs a PoK that does not prove the *availability* of the knowledge but rather the existence of a *local copy*.

VII. Conclusion

In this paper, we propose a solution, EWoK, to incentivize storing the blockchain data in the network. Our solution plugs an efficient proof of knowledge into standard hash-based PoW mechanism to force workers to store (in-memory) a modest (but fixed) shard 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 shard.

We analyzed the security of EWoK and showed that it only incurs small modifications of existing PoW protocols, and is fully compliant with the specifications of existing mining hardware. Our implementation results show that EWoK can be easily integrated within GBT and STM and does not impair their mining efficiency. This makes EWoK as one of the few economically-viable and workable solutions that strongly incentivize the sharding of the blockchain data among mining pool workers. We therefore hope that our results motivate further research in this area.

REFERENCES

[1] Burstcoin. [Online]. Available: http://burstcoin.info

[2] How to disincentivize large bitcoin mining pools. [Online]. Available: http://hackingdistributed.com/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools/

[3] "Bitcoin Miners Ditch Ghash.io Pool Over Fears of 51% Attack," https://www.coindesk.com/bitcoin-miners-ditch-ghash-io-pool-51-attack/, 2014.

[4] "Hyperledger – Blockchain Technologies for Business," https://www.hyperledger.org/, 2016.

[5] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song, "Provable data possession at untrusted stores," in *ACM Conference on Computer and Communications Security*, 2007, pp. 598–609.

[6] F. Baldimtsi, A. Kiayias, T. Zacharias, and B. Zhang, "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*, 2016, pp. 902–933. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-53890-6_30

[7] M. Bellare and O. Goldreich, "On defining proofs of knowledge," in *Annual International Cryptology Conference*. Springer, 1992, pp. 390–420.

[8] "BFGMiner," 2017, Available from: https://github.com/luke-jr/bfgminer.

[9] "BIP22: getblocktemplate - Fundamentals," 2012, available from https://github.com/bitcoin/bips/blob/master/bip-0022.mediawiki.

[10] "BIP23: getblocktemplate - Pooled Mining."

[11] "Bitcoin average block size," 2017, Available from: https://blockchain.info/charts/avg-block-size.

[12] 2015, block hashing algorithm - Bitcoin Wiki, Available from https://en.bitcoin.it/wiki/Block_hashing_algorithm.

[13] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: theory and implementation," in *CCSW*, 2009, pp. 43–54.

[14] X. Boyen, C. Carr, and T. Haines, "Blockchain-free cryptocurrencies: A framework for truly decentralised fast transactions," Cryptology ePrint Archive, Report 2016/871, 2016, http://eprint.iacr.org/2016/871.

[15] 2017, cryptoCurrency Market Capitalizations, Available from https://coinmarketcap.com/.

[16] P. Daian, I. Eyal, A. Juels, and y. Emin Gün Sirer, "(short paper): Piecework: Generalized outsourcing control for proofs of work."

[17] "Bitcoin Difficulty," 2017, Available from: https://en.bitcoin.it/wiki/Difficulty.

[18] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse, "Bitcoin-ng: A scalable blockchain protocol," *arXiv preprint arXiv:1510.02037*, 2015.

[19] "Filecoin: A Cryptocurrency Operated File Network," 2014, available from http://filecoin.io/filecoin.pdf.

[20] "BitNodes," 2017, available from https://bitnodes.21.co/.

[21] "The Decline in Bitcoin Full Nodes," 2015, available from https://bravenewcoin.com/news/the-decline-in-bitcoins-full-nodes/.

[22] A. Gervais, G. Karame, S. Capkun, and V. Capkun, "Is bitcoin a decentralized currency?" in *IEEE Security and Privacy*, 2014.

[23] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "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*, 2016, pp. 3–16. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978341

[24] "Getblocktemplate - Bitcoin Wiki," 2015, available from https://en.bitcoin.it/wiki/Getblocktemplate.

[25] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989.

[26] A. Juels and B. S. K. Jr., "PORs: Proofs Of Retrievability for Large Files," in *ACM Conference on Computer and Communications Security*, 2007, pp. 584–597.

[27] G. Karame and E. Audroulaki, "Bitcoin and blockchain security," 2016.

[28] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 279–296. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias

[29] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," 2016, Available from: https://www.stellar.org/papers/stellar-consensus-protocol.pdf.

[30] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 475–490. [Online]. Available: http://dx.doi.org/10.1109/SP.2014.37

[31] "Number of Bitcoin Miners Far Higher Than Popular Estimates," 2015, available from http://bravenewcoin.com/news/number-of-bitcoin-miners-far-higher-than-popular-estimates/ .

[32] "Largest Cloud Mining Company - Genesis Mining," 2014, available from https://www.genesis-mining.com/.

[33] "The 18 companies that control Bitcoin in 2016," 2016, available from http://www.businessinsider.de/bitcoin-pools-miners-ranked-2016-6?r=UK&IR=T.

[34] "How to Disincentivize Large Bitcoin Mining Pools," 2014, available from http://hackingdistributed.com/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools/.

[35] S. Park, K. Pietrzak, A. Kwon, J. Alwen, G. Fuchsbauer, and P. Gazi, "Spacemint: A cryptocurrency based on proofs of space," *IACR Cryptology ePrint Archive*, vol. 2015, p. 528, 2015.

[36] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," *IACR Cryptology ePrint Archive*, vol. 2016, p. 917, 2016. [Online]. Available: http://eprint.iacr.org/2016/917

[37] QuantumMechanic, "Proof of stake," 2011, Available from: https://bitcointalk.org/index.php?topic=27787.0.

[38] "RadeonTop - GPU utilization morning," 2017, Available from: https://github.com/clbr/radeontop.

[39] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[40] B. Sengupta, S. Bag, S. Ruj, and K. Sakurai, "Retricoin: Bitcoin based on compact proofs of retrievability," in *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ser. ICDCN '16. New York, NY, USA: ACM, 2016, pp. 14:1–14:10. [Online]. Available: http://doi.acm.org/10.1145/2833312.2833317

[41] H. Shacham and B. Waters, "Compact Proofs of Retrievability," in *ASIACRYPT*, 2008, pp. 90–107.

[42] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "SPECTRE: A fast and scalable cryptocurrency protocol," *IACR Cryptology ePrint Archive*, vol. 2016, p. 1159, 2016. [Online]. Available: http://eprint.iacr.org/2016/1159

[43] "Stratum mining - Bitcoin Wiki," 2015, available from https://en.bitcoin.it/wiki/Stratum_mining_protocol.

[44] M. Vukolic, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *Proceedings of the IFIP WG 11.4 Workshop iNetSec 2015*, 2015.

APPENDIX

A. Mining Pool Revenue Sharing

As mentioned earlier, most existing PoW-based blockchains are populated by mining pools. Namely, mining pools offer a profitable alternative for individual workers to contribute their computing resources and to split the reward between all the pool members following a certain reward payment scheme. Shares of the reward are assigned by the mining pool to its members who presents valid proof-of-work. More specifically,

Fig. 8: Sketch of the PoW block header structure in Bitcoin adapted from [12].

a mining pool sets a difficulty level between 1 and the currency's difficulty. Subsequently, a share is assigned to those miners that provide a block header that scores a difficulty level between the pool's difficulty level and the currency's difficulty level. The main purpose of this step is to show that the miner is contributing with a certain amount of processing power [27].

The most basic reward payment scheme is the *Pay Per Share* that offers an instant, guaranteed payout for each share that is solved by a miner. On the other hand, *Equalized Shared Maximum Pay Per Share* requires that payments are distributed equally among all miners in the pool. *Recent Shared Maximum Pay Per Share*, on the other hand, gives priority to the most recent Bitcoin workers.

A different approach is that offered by the *Proportional* scheme which proposes a proportional distribution of the reward among all workers when a block is found—based on the number of shares they have each found. A well-known variation of this technique is the *Pay Per Last N Shares*, where rather than counting the number of shares in the round, only the last submitted correct N shares are taken into account. Additional details on pool revenue sharing techniques can be found in [27].

B. Mining Protocols

Most existing mining pools adopt two main mining protocols: GetBlockTemplate [24] and Stratum mining [43]. In what follows, we give a brief overview of these protocols.

1) GetBlockTemplate (GBT): GetBlockTemplate (GBT) [9], [10], [24] 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 GetBlockTemplate 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 1) 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 (cf. Section II-A) 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 8), 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) 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 2). 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 proceeds to searching 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.

C. HPoW

HPoW is a reformulation of the PoW used in Bitcoin. It deploys a cryptographically secure hash function $H:\{0,1\}^* \rightarrow \{0,1\}^n$. The core idea is that the hardness factor h determines a threshold δ, called the *target*, with $0 \leq \delta < 2^n$. For instance, one could define the hardness space to be a set of integers $\mathcal{HS}_{HPoW} = \{0,...,2^n\}$ and set the target as $target := 2^n - h$.

The sampling algorithm Sample_{HPoW} takes as input π_{puz} which are the non-flexible parts of the coinbase and combines these with the remaining public, fixed block header values to formulate a puzzle puz. The solving algorithm Solve_{HPoW} is depicted in Algorithm 1. It repeatedly tries (previously

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

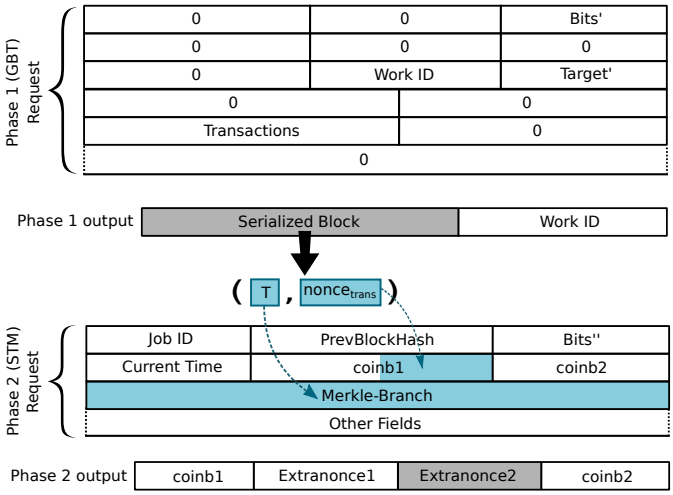

Fig. 9: Instantiating EWoK using GBT and STM work templates.

untried) values $\mathrm{sol} \in \mathcal{SS}_{HPoW} := \{0,1\}^r$ with r being the size of $\mathsf{CB}_{\mathsf{nonce}}$ until

$$H(\mathrm{puz},\mathrm{sol}) \leq \delta. \tag{7}$$

Notice that HPoW is a id-sound proof of work system in the random oracle model (ROM) where id refers to the identity function, i.e., $\mathrm{id}(x) = x$ for all inputs x.

To see why, observe that the probability for any $\mathrm{sol} \in \mathcal{SS}_{HPoW}$ to be a valid solution is $\frac{\delta+1}{2^n}$, independent of the hash values of the other values in the solution space. As an adversary can only win if she queries the ROM oracle, it follows that an adversary that the success probability of any solving algorithm is directly tied to the number of hash function calls. Thus, the optimal strategy is to retry different hash inputs—which is exactly the procedure of Solve_{HPoW}.

Note that the description of $HPoW$ is not a direct one-to-one mapping with the procedure given in Section II as we omitted in the description the step of computing the Merkle tree. Security-wise, this does not have any impact. To see why, one could define a procedure H^* based on a hash function H where in H^*, first the Merkle tree is computed and then H is applied. It is easy to see that if H is indistinguishable from a random oracle, so is H^*. Otherwise, one could use the construction of H^* to run a distinguishing attack on H.

D. Additional Pseudocodes

Algorithm 4 Informal work flow for solving the strawman solution.

Input: Non-changeable block header parameters π_{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 MR over the coinbase CB and the set of transactions T
4: **for** $nonce \in \{0,1\}^{32}$ **do**
5: Compute $h := H(\pi_{\mathrm{blckhdr}}, \mathsf{MR}, nonce)$
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** $i = 1,...,\ell$ **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 $IND = [i_1,...,i_\ell]$
17: Compute $h := H(BC[i_1],...,BC[i_\ell])$
Output: The solution $\mathrm{sol} = (\mathsf{CB}_{\mathsf{nonce}}, nonce, h)$.

Algorithm 5 The verification algorithm $\mathsf{Verify}_{\mathrm{EWoK}}$.

Input: Block header parameters π_{blckhdr}, possible solution $(T, CB, nonce)$

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

Check Proof of Work solution (second phase)
if $H(\pi_{\mathrm{blckhdr}}, \mathsf{MR}(CB,T), nonce) > target$ **then**
Solution invalid. Abort.
end if

Check validity of parameters (second phase)
Check if nonce restriction applies
Parse from CB the value $\mathsf{CB}_{\mathsf{nonce}}$
Parse pn from $\mathsf{CB}_{\mathsf{nonce}}$
if $pn > \delta_{\mathrm{pre-nonce}}$ **then**
Solution invalid. Abort.
end if
Check if correct shard block involved.
Compute $v_T = f_{\mathrm{trans}}(T)$
Compute index $index := f_{\mathrm{ind}}(v_T, pn)$
if $\mathsf{CB}_{\mathsf{nonce}} \neq (H(v_T, shard_{CB_1}[index], pn), pn)$ **then**
Solution invalid. Abort.
end if