Sustainable Blockchain through Proof of eXercise

Internal Draft

Ali Shoker
HASLab, INESC TEC & Minho University
Braga, Portugal
ali.shoker@inesctec.pt

***Abstract*—Cryptocurrency and blockchain technologies are recently gaining wide adoption since the introduction of Bitcoin, being distributed, authority-free, and secure. Proof of Work (PoW) is at the heart of blockchain's security, asset generation, and maintenance. Although simple and secure, a hash-based PoW like Bitcoin's puzzle is often referred to as "useless", and the used intensive computations are considered "waste" of energy. A myriad of *Proof of "something"* alternatives have been proposed to mitigate energy consumption; however, they either introduced new security threats and limitations, or the "work" remained far from being really "useful". In this work, we introduce *Proof of eXercise* (PoX): a sustainable alternative to PoW where an *eXercise* is a real world matrix-based scientific computation problem. We provide a novel study of the properties of Bitcoin's PoW, the challenges of a more "rational" solution as PoX, and we suggest a comprehensive approach for PoX.**

***Index Terms*—Cryptocurrency; Blockchain; Bitcoin; Distributed Systems;**

I. Introduction

Since the introduction of Bitcoin [37], cryptocurrency is increasingly drawing the attention of business, industry, and academia [8], [24], [11], [3], [26], [49], and the exchange rate of Bitcoin currency is still going steep[1]. The concept is based on using cryptographic tamper-proof public ledger, called *blockchain*, to protect the generation and transfer of "digital" money in a fully distributed peer-to-peer (P2P) fashion. The goals of cyrptocurrenices are mainly to avoid central authorities (like banks), reduce transaction delays and fees, and preserve the real value of money by backing the currency with some "work", done through *mining* — to the contrary of gold-backed or fiat currencies (e.g., USD, Euro, etc.). At the heart of blockchain, mining maintains the security and correctness of the system and generates (a.k.a., mines) money as a reward for the miner's work, namely, adding new blocks (of *transactions*) to the blockchain, and verifying the protocol's invariants [44]. Being a critical part of these systems, the "work" is made credible through providing a tamper-proof Proof of Work (PoW). The properties of PoW are discussed further in Sections II and III.

In most cryptocurrencies, and mainly Bitcoin, the "work" a miner must do is to solve a cryptographic puzzle: to find a random nonce that once (cryptographically) hashed with a perspective block header, returns a 32 Bytes number having a leading pre-defined number of zeros (called *difficulty*) [50]. This puzzle represents the PoW, and lives forever in the blockchain (together with the block), allowing for future verifications. Unfortunately, solving the puzzle is a very computation-hungry process that manifests in very high energy consumption, controversial to the recent trend and demands of sustainable and environment-friendly technology [25], [41]. For instance, recent studies have shown that the annual electricity consumption of Bitcoin system is equivalent to that of Ireland in 2014 [38], and is expected to be similar to that of Denmark in 2020 [14]. This raised the voices referring to Bitcoin's hash-based puzzle as "useless" work; whereas, Bitcoin proponents consider this a legitimate price for maintaining the system. The latter claim being (partially) sound, we argue that the work can be more *rational* if the puzzle itself is useful, rather than being random. Therefore, we propose an approach to replace the hash-based puzzle with solving scientific computation problems [39], [40].

In the same direction as ours, several attempts have been made to reduce the "wasted" energy following two main approaches. (We omit Byzantine Fault Tolerance approaches being not scalable in public settings [49].). The first is to introduce new forms of mining proofs like *Proof of Stake* (PoS) and its variants [30], [33], [45], [6]. Creating blocks in PoS is based on *coin age*: a function of coin balance and earning time. The proposal is often criticized that coin age accumulates even when the node is not connected to the network, and being non-democratic solution — biased to wealthy peers. The other variants like Delegated Proof-of-Stake [33] and Proof of Stake Velocity [45] tried to address each issue aside, leaving the other open and inducing new limitations or security threats [48]; whereas, Proof of Activity (PoA) [6] is a hybrid solution of PoS and PoW, where computation is still considered wasted on a useless nonce.

The second class tried to simply replace the puzzle with a more useful real world problem, as we do. However, the proposals fall short at addressing a wide range of real interesting problems. For instance, Primecoin [29] suggested finding prime numbers instead of a random useless nonce; Permacoin [35] tried to use have the miners to invest on the system's storage and memory through *Proof of Retrievability*; while PieceWork [42] tried to outsource work like spam

The author is supported by SMILES track of TEC4Growth project (NORTE-01-0145-FEDER-000020), and the EU H2020 LightKone project (732505).

[1]According to https://www.coindesk.com/, one Bitcoin is worth 3600 USD — by the time of writing this paper.

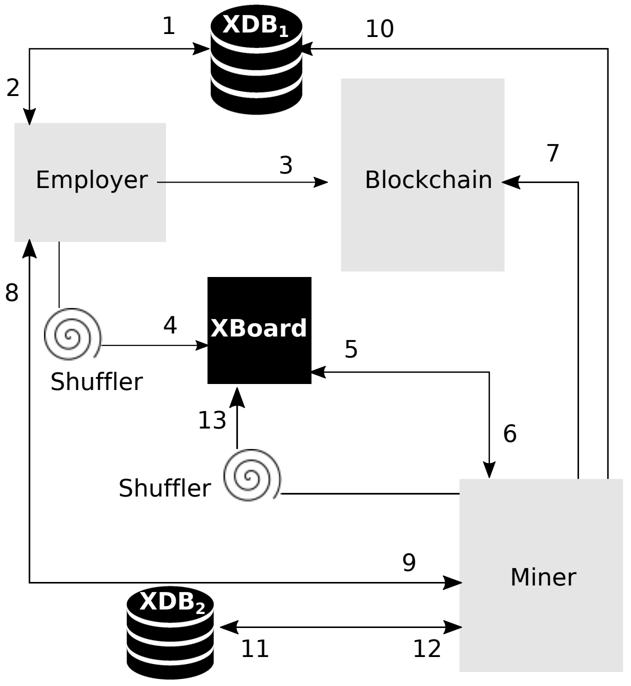

Fig. 1. The workflow of PoX without verification. Verification occurs in a similar manner to the steps from 5 through 13 on a verified instance. Refer to Section V for more details.

deterrence and Denial of Service defense.

In this work, we introduce *Proof of eXercise* (PoX), an approach to rationalize mining in cryptocurrencies — focusing on Bitcoin — through solving a real eXercise: a scientific computation matrix-based problem. The choice behind matrix-based problems is two-fold: (1) matrices have interesting composability properties that help in tuning difficulty, collaborative verification, and *pool-mining* (see later); and (2) matrix-based problems span a wide range of useful real world problems, being a principle abstraction for most scientific computation problems, among them: DNA and RNA sequencing and data comparison [1], [7], protein structure analysis, image comparison, object superposition, surface matching [15], [2], collaborative-filtering recommendation, data mining [28], computational geometry [19], face detection, image comparison, object superposition, surface matching [15], and many others [28], [40].

To emphasize on the challenges we faced in PoX, we adopt a down-top presentation approach: we first drive a novel and comprehensive analysis on the properties of Bitcoin's hash-based PoW (in Section III). Confronted with PoW's properties, we then show that the challenges facing a real puzzle like eXercise is indeed far beyond that of PoW due to: (1) the lack of tunable hardness and easy verification methods, (2) required commitments to solve and maintain an eXercise and its solution, (3) preventing collude, etc. We analyze these challenges in Section IV and propose corresponding solutions. After that, we make use of the discussed solutions to build a comprehensive Proof of eXercise solution in Section V.

In a nutshell, the workflow of PoX (shown in Fig. I) is as follows: an *employer* has an eXercise (e.g., a huge matrix-based scientific problem) to solve in Bitcoin; it stores it in a highly available store and keeps a hash digest and access details. It then deposits some credit in a blockchain transaction to guarantee the availability of eXercise until the process ends. The hash digest is then shuffled (i.e., randomized) before being presented on a public board (XBoard) for miners *bidding*, which occurs randomly (to avoid collude). A miner commits to solve eXercise (also through a blockchain deposit transaction) to be granted access details to its storage place. Once the miner solves eXercise, it stores the solution and provides access guarantees (as before), and publishes the (shuffled) digest to verifiers (to avoid collude).

Similarly, verifiers also commit to verify an eXercise (before access rights are granted) and to store the verified data instance (for further audit). Verifiers use a *probabilistic verification* scheme to collaboratively verify — possibly different — parts of the solution. Once a sufficiently high number of verifiers approve the solution, it becomes valid, and the PoX is created based on all the signed verifications as well as the block header digest (to prevent using this PoX for many blocks). The PoX lives forever with the block, but can no longer be verified once the stored eXercise, the solution, and the verified instance become unavailable. This is arguably sufficiently safe given the tradeoff of probabilistic verification, i.e., to reduce the huge storage overhead of scientific data — more details in Sections IV and V.

To get the reader more familiar with the subject, we overview Bitcoin in Section II, and we postpone related works to Section VI, just before the conclusion (Section VII).

II. Background on Bitcoin

As defined by Satoshi Nakamoto — an anonymous author — in the Bitcoin seminal paper [37], Bitcoin is a peer-to-peer (P2P) electronic cash system often referred to as a cryptocurrency [8], [6], [32], [29], [35]. The currency itself (i.e., the asset) is made of digital bits secured via intensive cryptographic techniques making it extremely hard to invent "bit-coins" out of thin air, steal them, or modify confirmed transfers. Being P2P, Bitcoin brings many advantages over classical banking-based currencies: no authorities are needed to make transfers, transfers are faster and cheaper, transparency, credible smart contracts, and others [43].

Bitcoin payments, transfers, and smart contracts are implemented though transactions. To transfer money (a.k.a., Bitcoins) from a spender to a receiver, the corresponding transaction must include the sources of the money transferred, i.e., hash digests (representing unique identifiers) of previous transactions through which the current spender earned the Bitcoins earlier. Transactions are digitally signed by the spender to prevent masquerade and ensure that transactions are issued by their owner, and are protected by the public key of the receiver to guarantee that only the true receiver can spend the money later.

However, given the underlying distributed model, deprived from central authorities, the described cryptographically protected P2P transaction scheme cannot prevent the spender from creating two "concurrent" transactions to spend the same amount (that refers to the same source transactions) twice, a problem known as *double spending*. Consequently, transactions are usually collected in a chain of blocks, representing a *public ledger*, i.e., the blockchain. Block creation is carefully tuned (see later) to enforce global consensus on the order of blocks across the entire Bitcoin network. Blocks are chained together in a similar fashion to transactions, ensuring that no block can get squeezed through previously confirmed blocks — the current design unfortunately allows for some forks in the blockchain when two blocks are created at "the same time", which leads to adopting the "longer" chain.

The order of blocks in the chain is globally defined by challenging block-creators, called miners, through solving a very hard cryptographic puzzle: search for a random *nonce* that once hashed with the digest of the *block header* generates a hash of a predefined number of leading zeros (see details in Section III). This hard "work" has two main roles. The first is to stand as a Proof of Work to generate, i.e., mine, Bitcoins as a reward to miners for maintaining the blockchain and verifying the protocol's correctness. The second is to tune the rate of creating blocks by forcing a global consensus through periodically calibrating the hardness of the puzzle — by simply changing the required number of leading zeros in the hash. The rate of Bitcoin's block creation is usually tuned to around six blocks per hour.

The interesting properties of Bitcoin's PoW, i.e., being hard to solve and easy to verify (among others discussed next), come at the price of high computation demands, reflected as a steep energy consumption similar to the electricity consumption of entire countries [38], [14]. Consequently, despite the essential guarantees that Bitcoin's PoW provides, like security and maintaining the blockchain, many researchers consider finding the nonce a *waste of energy* being "useless". This motivated more sustainable alternatives like Proof of Stake [30], Proof of Activity [6], Proof of Retrievability [35], and Proofs of Useful Work [5].

III. Bitcoin's PoW Properties and Challenges

The several attempts to make the hash-based PoW in [30], [6], [35], [5], [33], [42] were either limited to a narrow range of concrete "useful work" or introduced new undesired properties. Importantly, the proposed solutions are often based on meta-data that may be retrieved from the system in a secure and efficient way, keeping the challenges somehow close to those of the current Bitcoin's hash-based PoW (follow next). Indeed, addressing concrete scientific problems incurs new significant challenges — that are not thoroughly studied by the community — like efficiency, commitment, anonymity, etc., which require non-trivial solutions and trade-offs. To identify these challenges and address them, we need a reference baseline properties to compare against. Since we are unaware of such a comprehensive study, we found it intuitive to first analyze the properties of Bitcoin's PoW first, and try to meet them in our solution (in the following sections).

To analyze the properties of current hash-based PoW in Bitcoin, we need to understand the structure and the role of PoW in more details. As described in Section II, the miner collects the transactions (usually paid by the spender) to be included in the prospective block it is trying to commit, and it constructs the block header. The block header at this stage is not complete as it retains information about the included transactions — together with a digest of the previous block, current difficulty, timestamp, protocol's version — but missing the nonce. We refer to the block header prior to adding the nonce as B_h. Then, the miner tries to solve a cryptographic puzzle that is composed of B_h and a random $n \in \mathbb{N}$ such that:

$$H_{B_h}(n) = \mathsf{SHA\text{-}256}^2(B_h \mid n) \leq \tau$$

where H is a $\mathsf{SHA\text{-}256}$ [52] hashing function that once applied twice to the concatenation of B_h with the nonce n, returns a positive integer not greater than a predefined target τ. The role of the miner is to find the nonce n that satisfies this inequation, which leads to the first property:

(P1) Puzzle Hardness Solving the puzzle must have a notion of hardness that manifests in the PoW itself.

Hardness in PoW is crucial for security reasons: it deters the adversary from constructing new blocks to possibly incur conflicting transactions and succeed in double spending. In Bitcoin, the creation of a block (i.e., solving the puzzle) is rewarded by Bitcoins, which is believed to be an incentive for an adversary to solve the puzzle and provide the corresponding PoW, rather than attacking the system. Therefore, the hardness must be reflected in the PoW.

Indeed, cryptographic hashing is interesting as the inverse function $H^{-1}_{B_h}$ does not exist. Consequently, to find the nonce n, miners keep incrementing it until the function $H_{B_h}(n) \leq \tau$ holds true. Since n is implemented as a 32 Byte (i.e., 256 bits) integer in Bitcoin, the possible values of n are 2^{256} which is an extremely huge number to cover in a short time. (Given the current difficulty $\tau \approx 10^{12}$, specialized hardware having a hash rate of Terahashs/s [51] need several years to solve the puzzle [50].) Despite the fact that this randomness my lead to finding the nonce quickly, on average, a single miner will not often succeed in solving the puzzle in 10 minutes, which is the typical maintained time to create a new block by the entire network — and thus the miner has to restart the process from scratch as B_h will change. Maintaining this time frame is guaranteed by modifying τ; and here comes the second property:

(P2) Tunable Hardness The hardness of the work must be deterministically tunable.

Tuning the hardness of the work is required for two reasons: a business reason that ensures transactions are being committed within acceptable delays, and a technical reason which helps reaching consensus, i.e., by inducing an explicit delay required to enforce a global order on the blockchain. Consequently, τ is a tradeoff between the business and technical demands. In fact, it is clearly desirable to commit transactions faster from a business perspective (which is a main reason behind using cryptocurrencies), but unfortunately, doing this makes it almost impossible to construct a single chain of blocks. Indeed, despite maintaining a 10-minutes interval, the blockchain is still subject to forks, which can be resolved by adopting the *longer chain*[2]. This comes at the price of waiting more time once higher security, called *confirmation level*, is required [44].

In addition, tuning hardness must be deterministic across all nodes for correctness (i.e., to impose a fixed delay of mining rate) and fairness. In particular, tuning hardness in Bitcoin is done in a deterministic way by increasing/decreasing τ every 2,016 blocks. This is done by using the block's timestamps to calculate the number of seconds elapsed between the generation of the first and the last of those recent 2,016 blocks. The target is to maintain an approximate of two weeks rate (which leads to an average creating six blocks per hour).

Another form of tuning the difficulty of a problem is to break it down to smaller sub-problems, leading to the following property:

(P3) Embarrassing Parallelization The work shall easily be broken down into smaller (i.e., easier) problems.

This property is not essential for PoW's correctness, but it is rather important for business reasons (i.e., pool mining). Indeed, a puzzle may require years to solve by a miner, yielding delays in the expected profit, and most likely losing mining efforts if the difficulty τ got higher or the transactions got committed in another block. Consequently, it is crucial for miners to join forces and solve the puzzle faster. In a manner similar to parallel computing, the puzzle should be easily divided into *embarrassingly parallel* [23] sub-problems where miners in a *mining pool* [44] can work in parallel to solve the puzzle. Indeed, a hash-based PoW is embarrassingly parallel as different miners can search for the nonce within different sub-domains.

On the other hand, despite tunable hardness and parallelization, the PoW must be easy to verify, and thence the next property:

(P4) Easy Verification The PoW must be "easy" to verify.

The entire Bitcoin's protocol and underlying invariants must be verified by the peers themselves, otherwise the entire system and the currency will be untrustful. Consequently, the verification must be easy otherwise peers will likely avoid it and try to solve a new puzzle instead (which is rewarding). We do not accurately specify how easy verification is, but in principle, it must be orders of magnitude easier than solving the puzzle to be able to verify the always-growing blockchain (that is estimated by 2,016 every two weeks). Bitcoin's PoW is interestingly very easy to verify; it only requires verifying if $H_{B_h}(n) \leq \tau$ holds, knowing that n is already known by now (and remember that the hash rate of current mining hardware is several Terahashs/s).

[2]The term "longer chain" is misleading as it refers to the longer work (in time) done rather than the number of blocks in the fork.

Notice that we did not explain the use of B_h in H_{B_h} so far, which is crucial for the fifth property:

(P5) Block Sensitivity The PoW must be sensitive to the committed block.

A PoW must correspond to a particular block otherwise one can simply use the same puzzle to commit two blocks in the blockchain, thus creating a double spending attack. In Bitcoin, this is prevented by having B_h tight to the current block. Indeed, the structure of B_h in $H_{B_h}(n)$ is as follows:

$$B_h = [V \mid Bhash_{prev} \mid MTree_{txns} \mid utime \mid \tau]$$

which guarantees that the PoW exactly corresponds to the block of: protocol version V, exact predecessor block hash $Bhash_{prev}$, exact set and order (i.e., Merkle tree hash) of transactions $MTree_{txns}$, at *Unix epoch time* $utime$, and with a predefined difficulty τ. Given all these details, it is almost impossible to associate the PoW to another block.

IV. Proof of eXercise (PoX)

We now define the eXercise problem and discuss the challenges to build a PoX by addressing the aforementioned properties P1-5, and proposing corresponding potential solutions. At the end, we introduce a comprehensive solution for PoX based on these suggestions. (We adopt this down-top presentation style to emphasize on the challenges).

A. Matrix, as an eXercise

In PoX, we try to solve computation-intensive scientific problems [39], [40], [57], in contrast to hash-based puzzle in PoWs. A nice observation is that many scientific problems can typically be reduced to matrix-solving problems (matrix product, determinant, eigen vectors, orthogonal vectors, etc.) as in DNA and RNA sequencing and data comparison [1], [7], protein structure analysis, image comparison, object superposition, surface matching [15], [2], collaborative-filtering recommendation, data mining [28], computational geometry [19], etc. Consequently, devising a matrix-based PoX can address a wide spectrum of scientific problems. Now, we formulate the problem as follows.

Consider a number of matrices $X_1, X_2, \ldots, X_p$ and Y such that:

$$X_1 \circ X_2 \circ \cdots \circ X_p = Y$$

where $\circ$ is a matrix operator, e.g., product, sum, Schur product, etc. (Although this can be generalized further to include more complex combination of operators, we opt to keep the presentation easier.) Since an eXercise is supposed to be hard to solve, we require the matrices X_i, as well as Y, to have a very high dimension (e.g., an order of millions or more), and all matrices X_i not to be highly sparse (further discussion later).

A miner is assigned an eXercise $X_1 \circ X_2 \circ \cdots \circ X_p$ to solve, and return Y'. The role of the PoX is to prove that $X_1 \circ X_2 \circ \cdots \circ X_p = Y'$ holds, and importantly, that the exact miner (or mining pool) has solved it. This raises several challenges we address in the following.

For ease of presentation, and without loss of generality, we reduce the eXercise to a *matrix product* AB of two square matrices A and B of dimension n, where:

$$AB_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$

B. The Challenges of PoX

Now we discuss the challenges of PoX by referring to the five properties P1-5 of hash-based PoWs. Our analysis shows that PoX does not naturally guarantee the aforementioned properties, and it thus requires special techniques, we introduce, to satisfy them. Some of these techniques will be used later to assemble a comprehensive PoX solution. Furthermore, considering PoX allowed us to observe salient properties for hash-based PoWs that were not explicitly mentioned in literature.

1) Proof of hardness: Solving the matrix product is known to be $O(n^3)$. Although more efficient algorithms of $O(2.373)$ were recently found in theory, current implementations are less efficient than the naive solution [54]; and thus we don't consider them here for simplicity. In computational science, the matrix dimension n can grow up to Billions [57], [36], [7], [28], and thus a *petaflops* machine may take several months to solve the matrix product, which may serve as an PoX for cryptocurrencies. However, unfortunately, the algorithm's hardness is not accurate in reality once the values of the matrix are considered. For instance, a sparse matrix can be solved faster than a dense one [55], and the matrix product of the identity matrix $I_{n \times n}$ is trivial.

Consequently, the hardness of PoX should be instance-based and not algorithm-based. In other terms, it should be eXercise-based rather than matrix-product-based, which requires including a "proof of hardness" (PoH) in each PoX eXercise considered. In the case of matrix product, a PoH can be a computationally efficient *script* to estimate the sparseness or structure of a given matrix [34]; whereas different methods may be required once the eXercise is not matrix product. The PoH must be associated with the PoX eXercise which is only accepted (by miners) as valid eXercise if the sparseness level exceeds a pre-defined threshold. Since this paper focuses on the technical challenges of PoX, an interesting complementary research is to precisely identify what factors may affect problem hardness in each eXercise, and what efficient scripts can be included within PoH to guarantee a certain level of PoX eXercise hardness.

2) Hardness tuning through batching: Tuning hardness is not as simple as in hash-based PoW since the eXercise may not be modified, being the main target. In some cases [5], [53], a matrix can be modified or extended with fake numbers to increase the difficulty accordingly, whereas in others, as in our matrix product eXercise, the result will be irrelevant. To address this, it can be useful to use compositions of matrix problems, batched in a single eXercise. We envision two methods. The first can be to add N complete fake matrices F_k where the miner is required to solve all binary matrix product combinations. For example, the product of two matrices A and B can be transformed into multiple binary product of all matrices A, B, $F_1, F_2, \ldots, F_N$. In this scheme, the product AB is the only interesting combination for an *employer* (the entity who proposes an eXercise). This literally means some work is being wasted — a bit against the soul of "rational" work — and can be mitigated if the matrices F_k are real meaningful problems as well.

A second more interesting scenario is to batch different eXercises in a single one to achieve a required hardness level. The batched eXercise is then solved by miners through providing corresponding solutions for all embedded individual problems (whose number can be pre-defined a priori). Only in the case where all batched problems are verified (see later), a PoX is considered correct, and a corresponding block can be committed. Notice that batching can also be used in the current hash-based PoW through dividing B_h into m parts and finding m nonces to commit the block.

Reducing the difficulty is however intuitive since matrix product is naturally parallizable via *divide and conquer* [23]. Therefore, the hardness of a matrix product can be reduced by transforming it into a *block matrix product* where each block product stands as an independent eXercise (assembled latter by the employer).

3) Parallelization through Map-Reduce: Parallelization in PoX is straightforward as matrices are natively parallizable. In fact, matrix-based algorithms are the typical use-cases for *map-reduce* parallel computing [23], [57]. For our matrix product example, in a mining pool, a pool operator can *map* the matrix product AB into n different A_iB matrices (where A_i is a row matrix), and thus assigning them to available miners in the pool. We do not address the pool mining problem in details in this paper, but it would be interesting to discuss if some guarantees as those in Byzantine fault-tolerant Map-Reduce [12] can be provided. The operator can then *reduce* the maps and commit the block. In the case of PoX batching, the operator can simply assign miners individual problems in the batch. The challenge here is that the pool operator doesn't necessarily — and should not — trust the solutions provided by pool miners, which requires an efficient way to verify correctness (and obviously not re-solving the eXercise itself). A possible solution is to use a "probabilistic verification" method as described next; nevertheless, we believe that this problem requires further research, and it's recently a hot research topic in mathematics and cryptography [53], [5], [48].

4) Probabilistic verification: PoX verification is considered the most challenging aspect due to the lack of a fast, e.g.,

$O(n)$, and accurate formula, as SHA-256, that guarantees the solution uniqueness (i.e., multiple inputs can return the same output). The problem is that even if such a formula exists, e.g., based on the matrix properties or structure, the miner may find it easier to "guess" the values that satisfy the formula, rather than solving the eXercise itself. This urged finding probabilistic alternatives as in *holographic proofs* [4], [53], [5], in which multivariate expressions are efficiently expressed as univariate ones. In this scheme, a matrix problem can be transformed into a probabilistic univariate polynomial, making it hard (but possible) to construct the polynomial. To make it even harder to guess, the authors in [5] proposed inducing random fake numbers through an interactive protocol to make sure that the eXercise was really solved using the exact expected algorithm. Again, this unfortunately comes at the price of additional wasteful work by the miner.

The most credible way to verify an eXercise, as in matrix product, is to simply recompute the eXercise itself only a pre-defined number of times (e.g., by solving the entire eXercise again), and attach them as proofs within the same PoX. This means that the eXercise shall not be verified forever, as currently done in Bitcoin, which is also required due to eXercise storage and availability reasons as described next. (In principle, even in current Bitcoin PoW, it is unreasonable to keep verifying the old PoWs over and over for the entire blockchain; however, this is indistinguishable and cheap since the PoW is simply the block header hash that is used to verify the subsequence of blocks in the chain.) We believe that this tradeoff is reasonable as long as the eXercise is sufficiently verified. This verification scheme can be done by assigning the same eXercise to several miners such that the first PoX is empowered by the consecutive ones. For fairness and to avoid collude, miners that solve the same eXercise must share the rewarded Bitcoins (otherwise one miner may sell the solutions to others).

Again, this may look impractical being costly, and since the same work is being done several times, a bit controversial to the idea of "rational" PoWs. A more viable solution is to use a probabilistic verification method in which verifiers can compute random parts of the matrix product. In particular, a verifier can choose $r < n$ random vector products $A_i \times B_j$ to compute a single entry A_{ij}, and check if these values exist in the solution. Assuming that v random verifications are required per eXercise, the fraction of verified entries in the solution matrix AB is less than $\frac{v \times r}{n^2}$ (since different verifications may overlap). If the r entries are completely random, e.g., based on $\mathsf{SHA\text{-}256}(time) \bmod n^2$, and different (for simplicity), the chance that the miner can fool a single verifier is $\frac{1}{n^{2r}}$. Depending on whether verifiers chose different entries or not, the chance that the miner can generate a valid PoX is very low: between $\frac{1}{n^{2r}}$ and $\frac{1}{n^{2pr}}$. As long as the miner cannot tamper with his provided solution (ensured by disclosing the corresponding hash digest beforehand), he will simply refrain from cheating at a high risk of being caught.

5) PoX commitments: The main target of a miner is to commit a block and win the reward through the *coinbase* transaction of a mined block [44]. However, the miner in Bitcoin's PoW can give up solving a specific puzzle (e.g., to increase the number of transactions in a block) since this very miner is the sole owner and creator of the puzzle. In contrast, this is no longer true in the case of PoX since the employer is basically interested in the solution of the eXercise itself, correct and timely. Therefore, there should be a commitment from the miner to solve the eXercise before a given time t (sufficiently long, but not infinite), and importantly, make it available and accessible, e.g., by storing it in a highly available storage service (at some cost), and provide the needed access details.

To guarantee this, the miner has to deposit a hostage credit, e.g., a smart contract similar to *Micropayment Channels* or *Arbitrated Contracts* [44], that can be claimed once the PoX is verified before the deadline t, otherwise the deposit is lost. The deposit amount (1) must be greater than the reward in the coinbase and the storage cost to enforce the commitment, and (2) must not be destined to a specific peer to avoid collude (the employer and the miner can simply be the same entity). Similarly, the miner needs to guarantee that during and after solving the eXercise, the employer guarantees the availability and accessibility to the eXercise, otherwise it may lose the work done. For this reason, the employer also deposits a hostage credit as above; but this time, with an amount greater than the storage cost. Furthermore, verifiers must also commit to store the verified instance values for further potential audit in a similar way (see next section for details).

6) Shuffling eXercises: Once again, due to the dependency of the eXercise and its solution, collude can easily occur between the employer and the miner, or the miner and the verifier. Consequently, the assignment of an eXercise must be completely randomized and anonymous. To do this, a hash digest of the eXercise (similarly, the solution, or the verified instance) is shuffled before getting published for miners' "bidding". Shuffling can be done via a shuffling service that can be implemented as an *onion routing* [21] service on Bitcoin's network peers themselves, or using an external service like TOR [16]. After that, the miner or verifier commits (through the deposit transaction) to a randomly chosen eXercise before it knows any details about it or its owner. Once this information is unveiled, the miner will no longer withdraw at the risk of losing his deposit.

7) Block-sensitive eXercises: Analogous to property P5 of hash-based PoW, PoX must also be sensitive to a unique block, i.e., a PoX shall not be used to commit two blocks. In the case of hash-based PoW, this is not problematic since the puzzle's solution itself, i.e., the nonce, has no meaning beyond solving the puzzle, and can thus be completely randomized to match B_h. In contrast, the very solution of an eXercise in PoX is what the employer really cares about. Consequently, the eXercise cannot be simply modified to match the block via B_h. Instead, this can be solved by tying the eXercise to a specific block through imposing a random selection criteria based on the block header. In particular, a miner that already computed a hash of B_h, gets (randomly) assigned the unsolved eXercise whose hash is "matching". The matching criteria can be a minimum number of consecutive matching characters, or a hash with certain property similar to the notion of *difficulty* in Bitcoin. To avoid matching — possibly many — old eXercise, a time window for matching, based on the timestamp in the block header, must be respected.

V. PoX: All Put Together

Now, we introduce a comprehensive solution for PoX, based on the above discussion, and briefly shown in Fig. I.

1) Task proposals: Consider an employer E having a scientific problem, a.k.a., an eXercise X, that requires computing a matrix product. E stores X in a highly available database XDB, and gets the corresponding credentials and hash digest H(X). For simplicity, assume that XDB is an external paid DB service. Then, E creates an eXercise Transaction XT that comprises the PoX version, H(X), meta-data about X, e.g., "type:matrix product; Proof of Hardness: OK; dimension: 1 Billion, etc". Then, it deposits a credit (in Bitcoins) for a tolerated period of time after which E can give up (i.e., E is only interested in the solution before that time expires). This guarantees the availability and correctness of X, otherwise the miner may lose (part of) his work. This credit may only be claimed once the eXercise X is solved and verified or the tolerated time has expired.

After that, E computes a hash digest H(XT) and submits it to a shuffling service (discussed in Section IV) that shuffles H(XT) several times to make it impossible to relate H(XT) to E, and thus prevent collude. The shuffling service then publishes SH(XT), i.e., the shuffled H(XT), to the eXercise Board (XBoard). Only SH(XT)s that were published for a predefined time (e.g., one day) may be selected by miners to avoid forks in XBoard — which will require expensive handling as in the blockchain — since delays are not critical at this level.

2) eXercise bidding and mining: On the other side, a miner M collects a set of (paid) transactions to be committed and added to the blockchain. To do so, M needs to solve an eXercise chosen from the XBoard and provide a corresponding PoX. To prevent collude, M gets assigned an eXercise X in a random way, e.g., through matching the hash of block header H(B_h) to the eXercises in XBoard. (Matching can succeed via a pre-defined size of a matching string, or using the hash of H(B_h) and hash digests in XBoard in a similar scheme to Bitcoin's difficulty.)

At this stage, M promises to solve X in the eXercise Transaction XT' through creating a Deal Transaction (DT) that contains: PoX version, SH(XT'), and H(B_h); and then deposits a credit (in Bitcoin's) for a defined period of time — sufficiently long enough — to guarantee its commitment to solve the assigned eXercise. In a similar way to the employer E, the miner M can claim the credit in case the eXercise X is incorrect or became unaccessible. Once the DT is issued, the shuffling service uncovers the onion such that M and E know each other. Consequently, E unveils the meta-data of the eXercise in XT' and gives the credentials of X in the XDB to start working on it.

3) PoX Audit: Once the miner M finds Y', i.e., the solution of X, it follows the same process of the eXercise proposing above, making it available for verifiers, called Auditors. In particular, M stores Y' in highly available store, e.g., XDB, and gets a corresponding hash digest H(Y') and access credentials; it creates a corresponding Verify eXercise Transaction (VXT') which is similar to XT, but without requiring a credit this time since M has already deposited a credit through Deal Transaction above. The auditor submits the VXT' to a shuffling service which publishes SH(VXT') — a shuffled version of VXT' to be verified. Again, this is required to remove any bias in verification.

Auditors follow the same bidding procedure as well to choose a random solution Y" to verify, retrieve access details from M and E after the SH(VXT') onion is unshielded, and start auditing Y" through the probabilistic verification scheme — described in the previous section. If the verification *Passed*, the auditor submits a Passed Report through creating an Audit Transaction (AT) that includes the (random) *verification instance* this auditor used for its report, otherwise a Failed Report is submitted. The verification instance is also stored in XDB, and is made available for future audits (within a predefined time frame). Auditors have no interest in submitting false reports since they are at the risk of being caught by other honest auditors in case the same verification instance is repeated. To the contrary, malicious auditors may try submit Failed Reports to compromise the system. This can be prevented by having auditors deposit a credit as a guarantee against false reports — only in the case of submitting Failed Reports.

4) Committing the block: Once M notices a pre-defined number of Audit Transactions with Passed Reports, it collects the references of all XT, DT, VXT, and AT transactions together with H(X) and H(Y'), and attaches them as a PoX to the block header, that is confirmed by now and can thus safely be added to the blockchain. Finally, all credit deposits are claimed using the PoX of the confirmed block, and the stored data in XDB can be removed. Recall that this verification scheme is important to reduce the overhead of repeated verification of the entire blockchain as well as the data storage and availability costs — which are expensive in the case of PoX.

VI. Related Work

Bitcoin was introduced by an anonymous author, called Satoshi Nakamoto, as a fully functional Peer-to-Peer cryptocurrency system [37]. The security, i.e., generation, transfer, and maintenance of Bitcoins, is mainly guaranteed by a tamper-proof public ledger called blockchain: the structure where transactions are safely retained [44], [43]. To guard against malicious behaviors, creating a block is made expensive through solving a cryptographic puzzle: finding a *nonce* that once hashed together with the block header returns an integer having a leading number of zeros. Solving the puzzle serves as a *Proof of Work* (PoW) asserting that this very miner has done the hard job, and is thus worth some Bitcoins from the system. PoW also imposes a maintained rate of generating new blocks as an attempt to impose total order across all system transactions (to prevent *double spending*). The interesting properties of Bitcoin's PoW are rarely studied in a comprehensive way in literature. In this work, we show that these properties are beyond what is usually mentioned in literature.

Despite its nice properties, Bitcoin's PoW is very energy hungry, and it has been shown in [38], [14] that Bitcoin may consume as energy as Ireland or Denmark. Consequently, there are continuous attempts from researchers and practitioners to provide more "useful" alternatives to PoW to justify the energy consumed. Among the famous proposals is Proof of Stake [30] (PoS) in which creating blocks is based on the *coin age*. The proposal is often criticized being non-democratic solution — biased to richer peers, and that coin age accumulates even when the node is not connected to the network. Delegated Proof-of-Stake (DPoS) [33] and Proof of Stake Velocity [45] (PoSV) tried to solve the two issues independently without solving the other, and with leaving new limitations. To the contrary, Proof of Activity (PoA) [6] adopted a hybrid solution of PoW and PoS to address both issues, however being using PoW, the computation is still considered "wasted" on unless nonce computation. Our work avoids these issues by sticking to the nice properties of PoW, however, computing something more valuable — like matrix-based scientific problems — rather than a random nonce.

Similar to PoX, another class of proposals tried to replace the "work" in Bitcoin by a more "useful" one that has some other real use. In particular, Primecoin [29] suggested finding prime numbers instead of finding the nonce. Although this systems achieves similar properties to Bitcoin's PoW, the usefulness of finding prime numbers remains questionable. Proof of Retrievability [35] is another suggestion where miners provide a proof that they are investing on maintaining the system through providing storage and memory resources. Similarly, PieceWork [42] tried to outsource work like spam deterrence and Denial of Service without showing how this is done in detail. In all of these works, the work is limited to few selected services, whereas PoX allows solving a wide range of scientific problems.

The problem of finding proof of works has also attracted other communities in industry and academia. Recently, Intel introduced a CPU extension called Intel Software Guard Extensions (Intel SGX) SDK that permits the execution of trustworthy code in an isolated tamper-free environment [11]. This allowed to reduce the waste by computing real world problems; however, it induced security threats through using the partially-decentralized (Intel as authority) Proof of Elapsed Time (PoET) model to force an idle elapsed time before signing a block [56]. REM [56] addressed these security challenges without providing a fully decentralized scheme. Nevertheless, we believe that such hardware technology can be exploited to improve the proof of work, e.g., to enforce the use of a given algorithm, and to use more credible timestamps.

Theoreticians in the Computation Complexity area also addressed the PoW problem since the nineteens [17], [47], [18], [20], [53]. The closest work to PoX is Proofs of Useful Work (uPoW) [5] in which the authors introduce a "usefulness" property of a probabilistic PoW algorithm for matrix-based problems (on Orthogonal Vectors). The authors explicitly address blockchain in the last section, without going deep into the technical design and integration challenges in cryptocurrencies. Our work reveals that these challenges are significant, and thus worth a dedicated research. Interestingly, the uPoW work supports our idea that matrix-based problems have a high potential to serve as PoWs, and it has a high potential to fit in our PoX model.

Finally, ensuring the security of blockchain is also being studied in academia, e.g., [9], [13], [32], [49], [46], following Byzantine fault tolerant (BFT) approaches [10], [31], [22], [27]. However, as shown in [49], BFT-based approaches are not scalable to public settings as in cryptocurrencies, and are thus only used in private blockchain.

VII. Conclusions

We introduced Proof of eXercise (PoX): a new proof of work for cryptocurrencies, where the work is a real matrix-based scientific computation problem. This work shows that the inherit "magical" properties of Bitcoin's hash-based PoW (i.e., the puzzle), make it even more interesting than what is documented in literature; and thus, we presented a comprehensive analysis to the propoerties of PoW. This was only possible via thoroughly considering a real alternative problem as eXercise.

As our work shows, the complexity of designing and implementing PoX is much higher than PoW, and therefore, as long as no cheaper alternatives that do not sacrifice the genuine properties of PoW are proposed, it is wise to explore the feasibility of PoX by studying individual scientific computation use-cases, and discussing potential extensions, e.g., as those based on computational complexity [5]. Otherwise, one may opt to stick to cheaper Proof of Stake [30], [33], [45], [6] methods as long as the limitations and constrains are tolerated.

Finally, an empirical evaluation that compares the difficulty levels of PoW versus PoX matrices (e.g., dimension, sparseness, etc.) is an interesting future work.

References

[1] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 39–51. Springer, 2014.

[2] Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995.

[3] Angel.co. Blockchains startups. https://angel.co/blockchains. Accessed: 2017-09-15.

[4] László Babai, Lance Fortnow, Leonid A Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 21–32. ACM, 1991.

[5] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of useful work. 2017.

[6] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y. *ACM SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.

[7] Philip Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1):217–239, 2005.

[8] Vitalik Buterin. A next-generation smart contract and decentralized application platform. *white paper*, 2014.

[9] Christian Cachin, Simon Schubert, and Marko Vukolic. Non-determinism in byzantine fault-tolerant replication. In *20th International Conference on Principles of Distributed Systems, OPODIS 2016, December 13-16, 2016, Madrid, Spain*, pages 24:1–24:16, 2016.

[10] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[11] Intel Corporation. Intel software guard extensions (intel sgx) sdk. https://software.intel.com/en-us/sgx-sdk. Accessed: 2017-08-29.

[12] Pedro Costa, Marcelo Pasin, Alysson N Bessani, and Miguel Correia. Byzantine fault-tolerant mapreduce: Faults are not just crashes. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 32–39. IEEE, 2011.

[13] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. (leader/randomization/signature)-free byzantine consensus for consortium blockchains. *CoRR*, abs/1702.03068, 2017.

[14] Sebastiaan Deetman. Bitcoin Could Consume as Much Electricity as Denmark by 2020. https://motherboard.vice.com/en_us/article/aek3za/bitcoin-could-consume-as-much-electricity-as-denmark-by-2020. Accessed: 2017-09-05.

[15] Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of Distances*, pages 1–583. Springer, 2009.

[16] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[17] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[18] Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 501–512. ACM, 2012.

[19] Anka Gajentaan and Mark H Overmars. On a class of o (n2) problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

[20] Jiawei Gao and Russell Impagliazzo. Orthogonal vectors is hard for first-order properties on sparse graphs. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 23, page 53, 2016.

[21] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[22] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European conference on Computer systems*, pages 363–376. ACM, 2010.

[23] Maurice Herlihy and Nir Shavit. *The art of multiprocessor programming*. Morgan Kaufmann, 2011.

[24] IBM. Hyperledger fabric. https://www.ibm.com/blockchain/hyperledger.html. Accessed: 2017-09-15.

[25] COST: "European Cooperation in Science and Technology" NESUS Action IC1305. Network for sustainable ultrascale computing. http://www.nesus.eu/.

[26] Bitmain Technologies Inc. Antminer hardware. https://shop.bitmain.com/main.htm?lang=en#spec-para. Accessed: 2017-09-15.

[27] Jean-Paul Bahsoun, Rachid Guerraoui, and Ali Shoker. Making BFT Protocols Really Adaptive. In *In the Proceedings of the 29th IEEE International Parallel & Distributed Processing Symposium*, IPDPS'15. IEEE-CS, May 2015.

[28] Maja Kabiljo and Aleksandar Ilic. Recommending items to more than a billion people. https://code.facebook.com/posts/861999383875667/recommending-items-to-more-than-a-billion-people/. Accessed: 2017-08-29.

[29] Sunny King. Primecoin: Cryptocurrency with prime number proof-of-work. *July 7th*, 2013.

[30] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.

[31] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 45–58. ACM, 2007.

[32] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 2014.

[33] Daniel Larimer. Delegated proof-of-stake (dpos). *Bitshare whitepaper*, 2014.

[34] Miles Lopes. Estimating unknown sparsity in compressed sensing. In *International Conference on Machine Learning*, pages 217–225, 2013.

[35] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 475–490. IEEE, 2014.

[36] Cleve Moler. The worlds largest matrix computation. The MathWorks, Inc., 2002.

[37] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.

[38] Karl J O'Dwyer and David Malone. Bitcoin mining and its energy footprint. 2014.

[39] University of California. BOINC projects. http://boinc.berkeley.edu/projects.php. Accessed: 2017-08-29.

[40] University of Zurich. The Center for Theoretical Astrophysics & Cosmology Program. http://www.ctac.uzh.ch/research/index.html. Accessed: 2017-09-9.

[41] UNITED NATIONS:Framework Convention on Climate Change. Paris climate change agreement. http://unfccc.int/paris_agreement/items/9485.php, October 2016.

[42] Ittay Eyal Philip Daian, Emin Gn Sirer and Ari Juels. Piecework: Generalized outsourcing control for proofs of work. In *BITCOIN Workshop*. Springer, 2017.

[43] Bitcoin Project. Bitcoin. https://bitcoin.org/.

[44] Bitcoin Project. Bitcoin documentation. https://bitcoin.org/en/developer-reference. Accessed: 2017-08-29.

[45] Larry Ren. Proof of stake velocity: Building the social currency of the digital age. 2014.

[46] João Sousa, Alysson Bessani, and Marko Vukolić. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. *arXiv preprint arXiv:1709.06921*, 2017.

[47] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology–CRYPTO 2013*, pages 71–89. Springer, 2013.

[48] Florian Tschorsch and Björn Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials*, 18(3):2084–2123, 2016.

[49] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015.

[50] Bitcoin Wiki. Difficulty. https://en.bitcoin.it/wiki/Difficulty. Accessed: 2017-09-06.

[51] Bitcoin Wiki. Mining hardware comparison. https://en.bitcoin.it/wiki/Mining_hardware_comparison. Accessed: 2017-09-06.

[52] Bitcoin Wiki. Sha-256. https://en.bitcoin.it/wiki/SHA-256. Accessed: 2017-09-06.

[53] Ryan Williams. Strong eth breaks with merlin and arthur: Short non-interactive proofs of batch evaluation. *arXiv preprint arXiv:1601.04743*, 2016.

[54] Virginia Vassilevska Williams. Multiplying matrices in o (n2. 373) time. *preprint*, 2014.

[55] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms (TALG)*, 1(1):2–13, 2005.

[56] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert van Renesse. Rem: Resource-efficient mining for blockchains. In *USENIX Security Symposium*. Springer, 2017.

[57] Albert Y Zomaya. *Parallel computing for bioinformatics and computational biology: models, enabling technologies, and case studies*, volume 55. John Wiley & Sons, 2006.