論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Primitives and Variants of the Proof-of-Work Mechanism in Blockchain-Based Consensus Protocols
著者(和文)	蘇翔宇
Author(English)	Xiangyu Su
出典(和文)	学位:博士(理学), 学位授与機関:東京工業大学, 報告番号:甲第12511号, 授与年月日:2023年9月22日, 学位の種別:課程博士, 審査員:田中 圭介,伊東 利哉,尾形 わかは,鹿島 亮,安永 憲司
Citation(English)	Degree:Doctor (Science), Conferring organization: Tokyo Institute of Technology, Report number:甲第12511号, Conferred date:2023/9/22, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Primitives and Variants of the Proof-of-Work Mechanism in Blockchain-Based Consensus Protocols

Xiangyu Su
Supervisor: Keisuke Tanaka

Department of Mathematical and Computing Science
Tokyo Institute of Technology

August 24, 2023

Acknowledgements

First and foremost, I would express my deepest gratitude to my supervisor Professor Keisuke Tanaka for all the support and encouragement since I joined his laboratory in 2016. I am really, really glad and honored to have the opportunity to be a member of his laboratory. I sincerely appreciate the time he and Prof. Mario Larangeira spent on our discussion and the valuable advice they gave us.

I want to express my great thanks to Prof. Xavier Defago and Dr. Yasumasa Tamura for their time and advice on Chapter 5.

I would thank Prof. Toshiya Ito, Prof. Wakaha Ogata, Prof. Ryo Kashima, and Prof. Kenji Yasunaga, who referees my doctoral thesis and thank for their precious feedback.

I would also like to thank all members and related members of Tanaka Laboratory for the delightful time we spent both in and out of the office. I would especially thank Dr. Maxim Jourenko, Dr. Masayuki Tezuka, and Pengfei Wang for the fruitful time in collaboration studies with them.

I would express my gratitude to my family for their support, and especially to my grandpa, who always felt proud of me.

At last, I would thank my beloved girlfriend Yue Zhang for her company, support, and the colorful life she brought to me. Also, I would like to thank my cute cat, Homura, for all the MEOWs.

Contents

Chapter 1

Introduction

1.1 Background

Consensus has been a long-standing research topic in the field of distributed systems for multiple decades. Participants in a consensus protocol seek to agree on a log of messages that satisfies two crucial properties: (1) persistence, *i.e.*, all honest participants' logs agree with each other except for some logs may progress faster than others; (2) liveness, *i.e.*, messages received as input by honest participants get confirmed in all honest participants' logs quickly [26,45]. Conventionally, this can be achieved by the Byzantine Fault-Tolerance (BFT) algorithms [13,37]. However, the high communication complexity of these algorithms prevents them from being deployed in large-scale networks, *e.g.*, decentralized digital currency systems, in which consensus is necessary for settling transactions.

In light of early attempts of digital currencies [14,15], Satoshi Nakamoto [42] proposed the Bitcoin system atop a proof-of-work (PoW) [20]-based blockchain protocol. On a high level, a blockchain is formed by chaining blocks of messages (or transactions in the case of digital currency) with hash functions, *i.e.*, by requiring the later block to include the hash of its previous block. In order to generate a block, the PoW mechanism requires its participants (usually called provers or miners) to solve a moderately hard computational task, *i.e.*, the difficulty of PoW. Hence, provers who successfully solve the task are eligible to collect transactions and generate a block.

The public verifiability of the hash chain and the task guarantees that any participants in a blockchain protocol can verify the validity of any given block or blockchain. Moreover, due to the difficulty of PoW, the number of valid blocks generated during a period of time is bounded by the provers' computing power. If the number is one, then participants will agree on the only valid block to extend the blockchain. Otherwise, there will be "forks". By assuming an honest majority, *i.e.*, more than half of the participants are honest, and applying the longest-chain rule [26], *i.e.*, honest participants only accept and extend the longest chain in their view, the shorter chain (fork) will eventually "die out". The reason is that honest computing power within the network will concentrate on the longest chain and outperform adversaries who intend to work on shorter forks.

Considering a concrete PoW, the most well-adopted construction is based on hash functions. Given a difficulty parameter $T > 0$, provers are required to find a nonce such that the hash of the previous block and the nonce is less than T. For a hash function $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^n$, finding a valid nonce is expected to need $2^n/T$ hash evaluations; whereas, the verification requires only one hash check.

Although it has been proven that the Nakamoto-style blockchain protocol from the aforementioned hash-based PoW satisfies persistence and liveness under the random oracle model [26], the hash evaluation itself is wasteful in energy and meaningless other than generating blocks. Therefore, numerous alternative PoW schemes, *i.e.*, proof-of-X (PoX), have been proposed to utilize more "useful" tasks, *e.g.*, proof-of-useful-work (PoUW) [6, 23], or other resource, *e.g.*, proof-of-stake [19, 35] and proof-of-space [22].

1.2 Our Results

In this thesis, we will present three results [51–53] related to these primitives and variants of PoW. We start from looking back into the origin [20], *i.e.*, a resource-demanding verifiable computation (for combating junk mail). Instead of hash functions, we show a generic construction based on computationally hard primitives that can be instantiated with different kinds of resources (*i.e.*, time and memory); Next, back in the sense of blockchain protocols, we investigate a subset of the PoUW, *i.e.*, with deep learning tasks as the useful work. We propose a distributed proof-of-deep-learning (D-PoDL) scheme and transform it into a provably secure blockchain protocol that can take different chain selection rules into consideration; Finally, in the last result, we consider a competitive PoX framework which is generalized from a score-based assignment problem abstracted from double auction systems. We show the design of a competitive PoX-based blockchain protocol that can also achieve consensus but requires significant modification compared to the existing approaches.

1.2.1 PoW from Computationally Hard Primitives

In contrast to the hash-based construction that requires the random oracle model in security analysis, one can better argue the security of a computationally hard primitive-based PoW [6]. This result proposes a generic PoW construction based on one-way trapdoor functions [29] and asymmetrically hard functions [8]. Intuitively, provers are required to invert a one-way function with the help of an instance from the hard function. The task is designed in a way that the evaluation of the hard function instance equals the trapdoor of the one-way function. Hence, the prover can compute the inversion by first evaluating the hard function to obtain the trapdoor. Additionally, we show two concrete constructions based on the RSA problem [49]. The first one takes time as the difficulty resource, which is built atop the RSW time-lock puzzle [48]; whereas, the second one considers memory and is built atop a memory-hard function, *i.e.*, the DIODON function [8]. Note that our task generation unavoidably incurs trust in the generator, which will limit the usefulness of our approach in distributed scenarios. However, since this result considers PoW as a resource-demanding verifiable computation, we argue that our constructions fulfill this purpose and show enough versatility for being capable of using either time or memory as the difficulty resource.

1.2.2 Blockchain from Deep Learning Tasks

For the second result, we investigate the potential of using deep learning tasks as useful work in PoUW. Existing works on deep learning-based PoUW[1] rely on strong assumptions, cannot act as a distributed solver among provers, and lack security proofs for the blockchain protocols built atop them. We overcome these drawbacks by designing a hash-train-hash structure in the solving algorithm and enabling provers to refer to others' pre-trained models with a model-referencing mechanism. Then, based on our scheme, we propose a generic blockchain protocol that is capable of two different chain selection rules: *i.e.*, the longest-chain rule [26] and the weight-based framework [27, 33]. Hence, we can derive two concrete blockchain protocols from the generic design. For security, we analyze the block generation rate and show both our concrete protocols satisfy the robust ledger properties [26,33], *i.e.*, chain growth, chain quality, and common prefix, which derive persistence and liveness as proven in [26].

[1] Details of related works can be found in Chapter 4.

1.2.3 Consensus from Competitive PoX

The starting point of this result has two aspects: (1) a general double auction system in which participants can bid and match their bids to form transactions; (2) the weight-based PoW has a rather constrained weight distribution [33] for satisfying consensus which cannot be applied for general (score-based) optimization problems. Therefore, we first refine the blockchain data structure to add support for bidding operation. Then, we abstract a bid-assignment problem (BAP) as the underlying task for a PoW. However, we observe that, unlike any existing PoX schemes, the difficulty of the BAP-based PoW lies in competition among provers. Hence, we further abstract it into a competitive *PoX* framework. Notice that we can drop the “work”, but only need to focus on the score distribution of blocks. Based on the framework, we design a protocol that achieves consensus on a blockchain but has intermediate states in the tree (forest) structure. By considering the accumulated score of each branch on the tree and adopting the “highest-scored-branch” rule, we conclude that under an appropriate accumulating function, our protocol can achieve consensus even assuming an arbitrary score distribution.

Chapter 2

Preliminaries

2.1 Notations

Throughout this thesis, we use λ for the security parameter. The negligible function of λ is denoted with $\mathsf{negl}(\lambda)$. For an integer $k \in \mathbb{N}$, $[k]$ denotes the set $\{1, \ldots, k\}$. Given a set $\mathcal{X}$, $x \overset{\$}{\leftarrow} \mathcal{X}$ denotes that x is randomly and uniformly sampled from $\mathcal{X}$. For an algorithm Alg, $x \leftarrow \mathsf{Alg}$ denotes that x is assigned the output of an algorithm Alg on fresh randomness. Let Hash denote a collision-free hash function.

2.2 Frameworks and Definitions

Execution model. General protocol executions are modeled by the standard Interactive Turing Machines (ITM) approach [12]. A protocol refers to algorithms for a set of nodes (participants) to interact with each other. All corrupted participants are considered to be controlled by an adversary $\mathcal{A}$ who can read inputs and set outputs for these nodes. We will present concrete settings with respect to each result.

Next, we show the general PoX framework.

2.2.1 General PoX Framework

Recall the hash-based PoW. Given a hash function $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^n$ and a difficulty parameter $T > 0$, provers are required to find a nonce nonce such that $\mathsf{Hash}(\mathsf{prevBK}, \mathsf{nonce}) < T$. We abstract this approach into a general PoX framework which involves a task generation algorithm $\mathsf{TaskGen}$, a solving algorithm Solve, and a verification algorithm Verify.

Definition 1 (Syntax of General PoX) *The tuple of algorithms* $(\mathsf{TaskGen}, \mathsf{Solve}, \mathsf{Verify})$ *in a general PoX scheme performs as follows:*

- $\mathsf{TaskGen}(1^\lambda, \mathsf{kind}, u)$ *takes as input the security parameter* λ*, the kind of resource* kind*, e.g., time, memory or storage, and the designed amount of resource units* u*. It outputs public parameters* pp *such that* $(\mathsf{kind}, u) \in \mathsf{pp}$*, and a task* task*;*

- $\mathsf{Solve}(\mathsf{pp}, \mathsf{task})$ *takes as input* pp*, a task* task*. It outputs a proof* π *for the given task* task*;*

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \pi)$ *takes as input* pp*, a task* task*, and a proof* π*. It outputs* 1 *if* π *is a valid according to* task*; Otherwise, it outputs* 0*.*

Here, we consider the correctness and efficiency of the general framework. Other security definitions, *e.g.*, difficulty, will be given with respect to concrete schemes.

Definition 2 (Correctness) *A general PoX satisfies perfect correctness if the following property holds for any* $\lambda, \mathsf{kind}, u$*, and* $(\mathsf{task}, \mathsf{aux}) \leftarrow \mathsf{TaskGen}(1^\lambda, \mathsf{kind}, u)$*:*

$$\Pr[\mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{Solve}(\mathsf{pp}, \mathsf{task}) = 1] = 1.$$

Efficiency requires that $\mathsf{TaskGen}$ and Verify should run in time $\tilde{\mathsf{O}}(\lambda)$[1], and should be logarithm of the run time of Solve.

2.2.2 Security Definitions

Here, we introduce security definitions of blockchain protocols in the following.

Persistence and liveness. Typically in distributed ledgers, each transaction changes the state of the protocol. Thus, it is convenient to introduce a notation to address this framework. Assume the existence of two states st_1 and st_2, let $\mathsf{st}_1 \overset{\mathsf{tx}}{\to} \mathsf{st}_2$ denote the state change from st_1 to st_2 introduced by a transaction tx. Moreover, let $\mathsf{st} \overset{*}{\to} \mathsf{st}'$ denote the transition between two states caused by a finite number of transactions. Under our protocol execution setting, we recall the ledger properties, *i.e.*, persistence and liveness, as discussed in [26].

[1] If $f(\lambda) \in \tilde{\mathsf{O}}(\lambda)$, there exists $k > 0$ such that $f(\lambda) \in \mathsf{O}(\lambda \cdot \log^k(\lambda)$.

- **Persistence.** For any two nodes, say, S_1 and S_2, and any two slots $t_1 \leq t_2$, if the list of settled transaction of S_1 in t_1 is equal to LOG_1, and the total transactions for S_2 is LOG_2, thus LOG_1 is a prefix of LOG_2;
- **Liveness.** If a transaction tx is available to all nodes at a point when the latest slot for the honest nodes is t, then any node whose clock advances τ slots to the point of t', we will have a ledger such that the state st, for which it holds that $\mathsf{st}_0 \overset{*}{\rightarrow} \mathsf{st}_1 \overset{\mathsf{tx}}{\rightarrow} \mathsf{st}_2 \overset{*}{\rightarrow} \mathsf{st}$.

Robust ledger properties. The definitions of robust ledger properties, *i.e.*, chain growth, chain quality, and common prefix, originate from [26]. We adopt the modified version from [23] for the longest-chain-based protocol. Moreover, we also include the weight-based variant from [33].

Definition 3 (Robust Ledger Properties) *The three aspects of the robust ledger properties are defined as follows.*

- *Chain growth: For any honest miner with chain* chain *at a round, the chain growth with parameter* $\tau \in (0, 1]$ *and* $s \in \mathbb{N}$ *states that for any portion of* chain *spanning* s *consecutive rounds, the number of blocks in this portion is at least* τs*;*

- *Existential chain quality: For any honest miner with chain* chain *at a round, the existential chain quality with parameter* $s \in \mathbb{N}$ *states that for any portion of* chain *spanning* s *consecutive rounds, at least one honestly-generated block appears in this portion;*

- *Common prefix: For any two honest miners with chains* $\mathsf{chain}_1, \mathsf{chain}_2$ *at round* r_1, r_2 *respectively, where* $r_1 \leq r_2$*, the common prefix with parameter* $s \in \mathbb{N}$ *indicates that* chain_1 *should be a prefix of* chain_2 *after removing the last* s *blocks.*

Definition 4 (Weight-Based Robust Ledger Properties) *The three aspects of the weight-based robust ledger properties are defined as follows.*

- *Chain growth: For any honest miner with chain* chain *at a round, the chain growth with parameter* $\tau \in (0, 1]$ *and* $s \in \mathbb{N}$ *states that for any portion of* chain *spanning* s *consecutive rounds, the accumulated weights appearing in this portion is at least* $\mathsf{W}(\mathsf{chain}_2) \geq \mathsf{W}(\mathsf{chain}_1) + \tau s$*;*

- *Existential chain quality: For any honest miner with chain* chain *at a round, the existential chain quality with parameter* $s \in \mathbb{N}$ *states that for any portion of* chain *spanning* s *consecutive rounds, the fraction of honest blocks' weights in this portion is at least* μ*;*

- *Common prefix: For any two honest miners with chains* $\textsf{chain}_1, \textsf{chain}_2$ *at round* r_1, r_2 *respectively, where* $r_1 \leq r_2$*, the common prefix with parameter* $s \in \mathbb{N}$ *indicates that* $\textsf{chain}_1$ *should be a prefix of* $\textsf{chain}_2$ *after removing the last* s *blocks.*

As explained in [45], persistence is equivalent to the common prefix property, and liveness can be derived from robust ledger properties.

Chapter 3

PoW from Computationally Hard Primitives

This chapter takes a detour to investigate PoW outside the blockchain scenario. In this case, it can be regarded as a resource-demanding computation mechanism with public and efficient verification. We show a generic construction that can be instantiated with primitives requiring different kinds of resources (*i.e.*, time and memory) [52].

3.1 Overview

Since our purpose is to construct PoW from computationally hard primitives, a naive idea is to invert one-way functions [29]. However, (1) a PoW should be moderately hard, and (2) the difficulty of PoW should be easy to adjust, we notice that: (1) inverting a general one-way function may require more computing power than being "moderate", and (2) adjusting difficulty usually ends in changing the security parameter, hence, the security guarantee of the whole PoW.

Another straightforward idea is to require provers to evaluate a hard function. Proposed in [8], evaluating such functions requires a significant but adjustable amount of resources. However, it is hard to verify the result of a general hard function[1], *i.e.*, verifiers may have to run the whole evaluation, hence, violating the efficiency requirement of PoW.

Now, we consider a trapdoored version of the one-way function. Inverting such a function with the corresponding trapdoor will be easy. Hence, our final idea is to provide provers with the trapdoor of the one-way function from

[1] There exists efficient proofs [46, 54] for concrete hard functions, *e.g.*, the RSW time-lock puzzle [48].

the evaluation of a hard function. However, this approach requires that the task generator must set the result of hard function evaluation to the trapdoor of the one-way function, which means: (1) the task generator must be trustworthy; (2) an efficient task generator must be able to bypass the hardness of the hard function. For the second aspect, we adopt the asymmetrically hard function [8], in which resource consumption for evaluation can be reduced with a corresponding bypassing key.

In the following section, we will first present the syntax of our trusted generator PoW (TG-PoW).

3.2 Our Trusted Generator PoW Scheme

As introduced above, our TG-PoW extends the conventional PoW by granting provers an auxiliary input corresponding to the task. This design enables provers to compute a piece of aid from the auxiliary that significantly reduces the computing time of solving the task. Hence, instead of relying on the difficulty of the computational task, TG-PoW requires its provers to contribute enough resources in computing the aid. With concrete constructions given in Section 3.3.3, we will show that the resource can be *time* like in conventional PoW schemes, or *memory*, without changing the TG-PoW framework.

3.2.1 Formal Syntax

Like the general PoX framework, the TG-PoW scheme consists of three algorithms. However, in order to reflect the characteristic of our task generation and solving process, we add modifications to these algorithms. Note that we only consider time and memory as the resource in the following.

Definition 5 (TG-PoW Scheme) *The tuple of algorithms* $(\mathsf{tdTaskGen}, \mathsf{EvalSolve}, \mathsf{Verify})$ *in a TG-PoW scheme performs as follows:*

- $\mathsf{tdTaskGen}(1^\lambda, \mathsf{kind}, u, \mathsf{td})$ *takes as input the security parameter* λ*, the kind of computational resource* $\mathsf{kind} \in \{\mathsf{time}, \mathsf{memory}\}$*, the designed amount of resource units* u*, and an additional trapdoor* td*. It outputs public parameters* pp *such that* $(\mathsf{kind}, u) \in \mathsf{pp}$*, and a computational task* task *with its corresponding auxiliary* aux*;*

- $\mathsf{EvalSolve}(\mathsf{pp}, \mathsf{task}, \mathsf{aux})$ *takes as input* pp*, a computational task* task*, and an auxiliary input* aux*. It is divided into two sub-procedures:*

  - $\mathsf{Eval}(\mathsf{pp}, \mathsf{aux})$ *outputs a piece of aid information* aid*;*

  - $\mathsf{Solve}(\mathsf{pp}, \mathsf{task}, \mathsf{aid})$ *outputs a solution* solution *of the given task;*

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{solution})$ *takes as input* pp*, a task* task*, and a solution* solution*. It outputs* 1 *if* solution *is a valid solution of* task*; Otherwise, it outputs* 0*.*

3.2.2 Security Properties

Similar to the general PoX framework, we define correctness.

Definition 6 (Correctness) *A TG-PoW scheme satisfies perfect correctness if the following property holds for any* $\lambda, \mathsf{kind}, u, \mathsf{td}$*, and* $(\mathsf{task}, \mathsf{aux}) \leftarrow \mathsf{tdTaskGen}(1^\lambda, \mathsf{kind}, u, \mathsf{td})$*:*

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{EvalSolve}(\mathsf{pp}, \mathsf{task}, \mathsf{aux}) = 1\right] = 1.$$

Next, we consider the primary security property, *i.e.*, difficulty, of our TG-PoW scheme. Following the same notation of the hash-based PoW, we denote the task difficulty with parameter $T(\lambda, u)$ where λ is the security parameter, and u is the resource demand. Difficulty requires that any prover can only produce a valid solution to the task by using significantly fewer resources with negligible probability of λ.

In order to formally define difficulty with respect to TG-PoW, we introduce two prior properties: soundness and hardness. Soundness requires that unless the adversary $\mathcal{A}$ honestly computes the piece of aid aid' from Eval with the given aux, the probability of the obtained solution $\mathsf{solution}' \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{task}, \mathsf{aid}')$ such that $\mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{solution}') = 1$ is negligible of λ. In contrast, hardness is defined over the resource, *i.e.*, time and memory. It indicates that any adversary on given aux can only obtain the valid aid using less than the designed amount of resource with negligible probability of λ.

Formally, we define soundness as follows.

Definition 7 (Soundness) *A TG-PoW scheme satisfies soundness if for any adversary* $\mathcal{A}$ *who runs in time* $t(\lambda, c) \in (T(\lambda, u), \mathsf{O}(2^{\lambda/c}))$ *where* c *is a significant large constant value, the following property holds for any large enough* λ*, any* $\mathsf{kind} \in \{\mathsf{time}, \mathsf{memory}\}, u, \mathsf{td}$*, and any* $(\mathsf{task}, \mathsf{aux}) \leftarrow \mathsf{tdTaskGen}(1^\lambda, \mathsf{kind}, u, \mathsf{td})$*:*

$$\Pr\left[\begin{array}{ll} \mathsf{aid}' \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{task}) : & \mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{solution}') = 1 \\ \mathsf{solution}' \leftarrow \mathsf{Solve}(\mathsf{pp}, \mathsf{task}, \mathsf{aid}') & \end{array}\right] \le \mathsf{negl}(\lambda).$$

Remark 1 (Achievable Soundness) *Note that soundness requires bounds on the adversary's run time, i.e.,* $t(\lambda, c) \in (T(\lambda, u), \mathsf{O}(2^{\lambda/c}))$*. The upper* *bound derives from the run time upper bound of solving the underlying* task *by brute force; whereas, the lower bound indicates that solving* task *without using* aux *should be more difficult than evaluating the hard function on* aux[2]*.*

Next, we show the formal definitions for hardness with respect to time and memory. First, we consider a generic game between a hardness adversary $\mathcal{A}$ and a challenger $\mathcal{CH}$. The adversary is given access to an evaluation oracle $\mathcal{O}_{\mathsf{Eval}}$ such that on query aux, and the oracle returns $\mathsf{aid} \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{aux}, \mathsf{td})$. In general, the definitions are parameterized by a function $\delta(\lambda, u)$ (in memory-hardness, we use $(\delta_{\mathsf{time}}, \delta_{\mathsf{memory}})$ for potential time-memory trade-offs [31]), which represents the designed resource demand when running Eval honestly. The time-hardness is defined as follows.

Definition 8 (Time-Hardness) *A TG-PoW scheme satisfies $\delta(u)$-time-hardness if for any $\epsilon > 0$ and any adversary $\mathcal{A}$ who runs in time $\delta(u)^{1-\epsilon}$ after oracle access ($\mathcal{O}_{\mathsf{Eval}}$), the following property holds for any large enough λ, u,* $\mathsf{kind}{=}\mathsf{time}, \mathsf{td}$*, and any* $(\mathsf{task}, \mathsf{aux}^*) \leftarrow \mathsf{tdTaskGen}(1^\lambda, \mathsf{kind}, u, \mathsf{td})$*:*

$$\Pr\left[\begin{array}{ll} \mathsf{st} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Eval}}}(\mathsf{pp}, \{\mathsf{aux}\}_{[q]}) & \\ \mathsf{aid}' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{aux}^*) : & \mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{solution}') = 1 \\ \mathsf{solution}' \leftarrow \mathsf{Solve}(\mathsf{pp}, \mathsf{task}, \mathsf{aid}') & \wedge \mathsf{aux}^* \notin \{\mathsf{aux}\}_{[q]} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Similarly, we have the following definition for memory-hardness.

Definition 9 (Memory-Hardness) *A TG-PoW scheme satisfies $(\delta_{\mathsf{time}}(\lambda), \delta_{\mathsf{memory}}(u))$-memory-hardness if for any $\epsilon > 0$ and any adversary $\mathcal{A}$ who runs with $(\delta_{\mathsf{time}}(\lambda), \delta_{\mathsf{memory}}(u)^{1-\epsilon})$ resource (considering potential time-memory trade-off) after oracle access ($\mathcal{O}_{\mathsf{Eval}}$), the following property holds for any large enough λ, u,* $\mathsf{kind}{=}\mathsf{time}, \mathsf{td}$*, and any* $(\mathsf{task}, \mathsf{aux}^*) \leftarrow \mathsf{tdTaskGen}(1^\lambda, \mathsf{kind}, u, \mathsf{td})$*:*

$$\Pr\left[\begin{array}{ll} \mathsf{st} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Eval}}}(\mathsf{pp}, \{\mathsf{aux}\}_{[q]}) & \\ \mathsf{aid}' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{aux}^*) : & \mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{solution}') = 1 \\ \mathsf{solution}' \leftarrow \mathsf{Solve}(\mathsf{pp}, \mathsf{task}, \mathsf{aid}') & \wedge \mathsf{aux}^* \notin \{\mathsf{aux}\}_{[q]} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Finally, in the difficulty game, we provide the adversary the full power with task and auxiliary. Recall that the soundness game only provides task, whereas the hardness games only provide auxiliary. We grant the adversary access to the evaluation oracle and an additional solving oracle $\mathcal{O}_{\mathsf{Solve}}$ such that on query $(\mathsf{task}, \mathsf{aux})$, $\mathcal{O}_{\mathsf{Solve}}$ returns $\mathsf{solution} \leftarrow \mathsf{EvalSolve}(\mathsf{pp}, \mathsf{task}, \mathsf{aux})$ if $(\mathsf{task}, \mathsf{aux}) \leftarrow \mathsf{tdTaskGen}(\cdot)$; otherwise, it returns $\perp$. The difficulty is defined as follows.

[2] Otherwise, provers will solve task directly.

Definition 10 (Difficulty) *A TG-PoW scheme satisfies $T(\lambda, u)$-difficulty if for any $\epsilon > 0$ and any adversary $\mathcal{A}$ who runs in time $T(\lambda, u)^{1-\epsilon}$ (or $T(\lambda)$ time with at most $u^{1-\epsilon}$ memory units after oracle access ($\mathcal{O} \triangleq (\mathcal{O}_{\mathsf{Eval}}, \mathcal{O}_{\mathsf{Solve}})$), the following property holds for any large enough λ, u,* $\mathsf{kind}{=}\mathsf{time}, \mathsf{td}$*, and any* $(\mathsf{task}^*, \mathsf{aux}^*) \leftarrow \mathsf{tdTaskGen}(1^\lambda, \mathsf{kind}, u, \mathsf{td})$*:*

$$\Pr\left[\begin{array}{ll} \mathsf{st} \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \{(\mathsf{task}, \mathsf{aux})\}_{[q]}) : & \mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{solution}') = 1 \wedge \\ \mathsf{solution}' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{task}^*, \mathsf{aux}^*) & \mathsf{task}^* \notin \{\mathsf{task}\}_{[q]} \wedge \mathsf{aux}^* \notin \{\mathsf{aux}\}_{[q]} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

3.3 Our Construction

In this section, we first introduce the formal treatments of our aforementioned building blocks, *i.e.*, one-way trapdoor functions [29] and asymmetrically hard functions [8]. Then, we show a generic construction of our TG-PoW scheme from these building blocks. Finally, we instantiate the generic construction with the RSA [49] problem representing one-way trapdoor functions, the RSW time-lock puzzle [48] for asymmetrically time-hard functions, and the DIODON function [8] for asymmetrically memory-hard functions.

3.3.1 Building Blocks

We first adopt the definition of one-way trapdoor function families.

Definition 11 (One-Way Trapdoor Function Families) *Let I be an index set. A collection of functions $\mathcal{F} \triangleq \{f_i : \mathcal{D}_i^{\mathcal{F}} \rightarrow \mathcal{R}_i^{\mathcal{F}}\}_{i \in I}$ is one-way trapdoor if the following properties hold.*

- *There exists a PPT algorithm* $\mathsf{Gen}_{\mathcal{F}}(1^\lambda)$ *that takes as input the security parameter λ, and outputs an index $i \in I$ with the corresponding trapdoor* td_i *of function f_i;*
- *There exists a PPT algorithm* $\mathsf{Sample}_{\mathcal{F}}(1^\lambda, i)$ *that takes as input λ and an index $i \in I$, and outputs $x \in \mathcal{D}_i^{\mathcal{F}}$;*
- *Given (i, x), the value $f_i(x)$ is polynomial-time computable;*
- One-wayness. *For any PPT adversary $\mathcal{A}$, for any* $(i, \mathsf{td}_i) \leftarrow \mathsf{Gen}_{\mathcal{F}}(1^\lambda)$ *and any* $x \leftarrow \mathsf{Sample}_{\mathcal{F}}(1^\lambda, i)$*, we have:*

$$\Pr[f_i(\mathcal{A}(1^\lambda, i, f_i(x))) = f_i(x)] \leq \mathsf{negl}(\lambda);$$

- Trapdoor one-wayness. *There exists a PPT algorithm* $\mathsf{Invert}(1^\lambda, i, \mathsf{td}_i, f_i(x))$ *that outputs x for all $x \in \mathcal{D}_i^{\mathcal{F}}$.*

Here, we formalize the implied definitions of asymmetrically hard functions from [8] under the similar syntax of function families. The definition goes as follows.

Definition 12 (Asymmetrically Hard Function Families) *Let I be an index set. A collection of functions $\mathcal{G} \triangleq \{g_i : \mathcal{D}_i^{\mathcal{G}} \to \mathcal{R}_i^{\mathcal{G}}\}_{i \in I}$ is asymmetrically hard if the following properties hold.*

- *There exists a PPT algorithm $\mathsf{Gen}_{\mathcal{G}}(1^\lambda, \mathsf{kind})$ that takes as input the security parameter λ and resource kind $\mathsf{kind} \in \{\mathsf{time}, \mathsf{memory}\}$, and outputs an index $i \in I$ with the corresponding bypassing key ak_i of function g_i;*
- *There exists a PPT algorithm $\mathsf{Sample}_{\mathcal{G}}(1^\lambda, i)$ that takes as input λ and an index $i \in I$, and outputs $x \in \mathcal{D}_i^{\mathcal{G}}$ and resource demand u;*
- Resource hardness. *For any $\epsilon > 0$ and any adversary $\mathcal{A}$ with at most $\delta(u)^{1-\epsilon}$ resource for time (or $(\delta_{\mathsf{time}}(\lambda), \delta_{\mathsf{memory}}(u)^{1-\epsilon})$ resource for memory), for any $(i, \mathsf{ak}_i) \leftarrow \mathsf{Gen}_{\mathcal{G}}(1^\lambda)$ and any $(x, u) \leftarrow \mathsf{Sample}_{\mathcal{G}}(1^\lambda, i)$, we have:*
$$\Pr[\mathcal{A}(1^\lambda, i, x, u) = g_i(x, u)] \leq \mathsf{negl}(\lambda);$$
- Asymmetrical hardness. *There exists a PPT algorithm $\mathsf{AsyEval}(1^\lambda, i, \mathsf{ak}_i, x, u)$ that outputs $g_i(x, u)$ for all $x \in \mathcal{D}_i^{\mathcal{G}}$.*

Asymmetrically hard function candidates. This chapter focuses on asymmetrically hard functions with two kinds of resources, *i.e.*, time and memory. Here, we show the candidate constructions: the RSW time-lock puzzle [48] for time-hardness and the DIODON function [8] for memory-hardness. We give descriptions of these constructions and prove them satisfying Definition 12.

Construction 1 (RSW Time-Lock Puzzle) *The algorithms in the RSW time-lock puzzle perform as follows. We omit $\mathcal{G}$ for simplicity.*

- $\mathsf{Gen}(1^\lambda, \mathsf{time})$ *samples a large integer $N = pq$ where p, q are prime numbers of length λ. It outputs $(N, \phi(N) = (p-1)(q-1))$;*
- $\mathsf{Sample}(1^\lambda, N)$ *samples $x \xleftarrow{\$} \mathbb{Z}_N^*$ and outputs (x, u);*
- *The function $g(x, u) = x^{2^u} \mod N$;*
- $\mathsf{AsyEval}(1^\lambda, N, \phi(N), x)$ *outputs $y = x^{2^u \mod \phi(N)} \mod N$.*

By Euler's theorem, it is easy to prove $y = g(x, u)$. Now, we formalize the following assumption, which was implicitly mentioned in [10]. Recently, this assumption has been proven to hold under the strong algebraic group model [34].

Assumption 1 (RSW Time-Lock Assumption) *For any $\epsilon > 0$ and any adversary $\mathcal{A}$ runs in at most $\delta(u)^{1-\epsilon}$ time, for any $(i, \mathsf{ak}_i) \leftarrow \mathsf{Gen}_{\mathcal{G}}(1^\lambda)$ and any $(x, u) \leftarrow \mathsf{Sample}_{\mathcal{G}}(1^\lambda, i)$, the following property holds:*

$$\Pr[\mathcal{A}(1^\lambda, N, x, u) = x^{2^u} \mod N] \leq \mathsf{negl}(\lambda).$$

Hence, we conclude that the RSW time-lock puzzle scheme satisfies Definition 12 with the following lemma.

Lemma 1 *RSW time-lock puzzles form an asymmetrically time-hard function family if the RSW time-lock assumption holds.*

Now, we introduce the construction of the DIODON function from [8]. It puts forward the idea of the RSW time-lock puzzle, generating a list that stores the results of RSW evaluations. Memory-hardness derives from chaining these results with a memory-hard hash function, *i.e.*, the Scrypt [2], so that provers cannot delete a large fraction of stored results to finish the evaluation of the DIODON function.

Construction 2 (DIODON Function) *The algorithms in the DIODON function perform as follows. We omit $\mathcal{G}$ for simplicity.*

- $\mathsf{Gen}(1^\lambda, \mathsf{memory})$ *samples a large integer $N = pq$ where p, q are prime numbers of length λ. It outputs $(N, \phi(N) = (p-1)(q-1))$;*

- $\mathsf{Sample}(1^\lambda, N)$ *samples $x \xleftarrow{\$} \mathbb{Z}_N^*$ and outputs $(x, (k, l, u))$ where k, l are additional parameters for the RSW time-lock puzzle and the Scrypt hash function;*

- *The function first computes u RSW time-lock puzzles and stores the results in a list $L = \{L_0\} \cup \{L_i\}_{i \in [u-1]}$ where $L_0 = x$ and $L_i = L_{i-1}^{2^k} \mod N$. Hence, for any $i \in [u-1]$, $L_i = x^{2^{k \cdot i}} \mod N$. On the list, the function performs the Scrypt hash function with parameter l, i.e., starting from L_{u-1}, it computes an index $j = L_{u-1} \mod u$, and hashes over L_{u-1} and L_j. It then iterates $l - 1$ times with the result from the previous hash as input. Finally, the function value $g(x, u)$ is set to the last result. Concrete algorithm can be found in Algorithm 1;*

- $\mathsf{AsyEval}(1^\lambda, N, \phi(N), x, (k, l, u))$ *evaluates the Scrypt hash function by computing each result block of index* $i \in [u-1]$ *with* $x^{2^{k \cdot i} \mod \phi(N)} \mod N$*. It sets the final result as* y*. The concrete algorithm can be found in Algorithm 2.*

Likewise, we have the following lemma for the DIODON function.

Lemma 2 *DIODON functions form an asymmetrically memory-hard function family if the RSW time-lock assumption holds.*

Proof 1 *By Euler's theorem, it is easy to prove* $y = g(x, u)$*. Moreover, as shown in [2], considering time-memory trade-off, the DIODON function is optimally linearly memory-hard, i.e., there exists* $\delta_{\mathsf{time}}(k, l, u) \times \delta_{\mathsf{memory}}(u)$ *is constant. Hence, to evaluate an instance of the DIODON function, a prover can save a fraction of memory but must pay the same factor in time.*

Algorithm 1: The DIODON Evaluation $g(x, u)$

```
Input $N$ and $(x, (k, l, u))$;
Let $L_0 = x$;
for $i \in [u-1]$ do
    $L_i = L_{i-1}^{2^k} \mod N$;
end
Let $\mathsf{temp} = L_{u-1}$;
for $i \in \{0\} \cup [l-1]$ do
    Let $j = \mathsf{temp} \mod u$;
    Compute $\mathsf{temp} = H(\mathsf{temp}, L_j)$;
end
Return $g(x, u) = \mathsf{temp}$
```

Algorithm 2: The DIODON $\mathsf{AsyEval}$

```
Input $(N, \phi(N))$ and $(x, (k, l, u))$;
Compute $e = 2^{k \cdot (u-1)} \mod \phi(N)$;
Let $\mathsf{temp} = x^e \mod N$;
for $i \in \{0\} \cup [l-1]$ do
    Let $j = \mathsf{temp} \mod u$;
    Compute $e_j = 2^{k \cdot j} \mod \phi(N)$;
    $\mathsf{temp} = H(\mathsf{temp}, (x^{e_j} \mod N))$;
end
Return $y = \mathsf{temp}$;
```

3.3.2 Generic Construction

Intuitively, the trusted generator first samples an asymmetrically hard function $g(\cdot) \xleftarrow{\$} \mathcal{G}$ and its instance as aux. The generator evaluates $g(\mathsf{aux})$ with the asymmetrical evaluation process of the hard function, *i.e.*, $\mathsf{AsyEval}(g, \mathsf{aux})$. By regarding the aid $\mathsf{aid} \triangleq g(\mathsf{aux})$ as the trapdoor of a one-way trapdoor function, the generator can then find such a function $f(\cdot) \in \mathcal{F}$. Finally, $\mathsf{tdTaskGen}$ outputs an instance $\mathsf{task} = f(\mathsf{solution})$ of inverting the one-way trapdoor function providing provers the auxiliary aux.

Construction 3 (Generic TG-PoW Construction) *The tuple of algorithms* $(\mathsf{tdTaskGen}, \mathsf{EvalSolve}, \mathsf{Verify})$ *of a TG-PoW scheme performs as follows.*

- *The trusted task generator performs* $\mathsf{tdTaskGen}(1^\lambda, \mathsf{kind}, u, \mathsf{td})$*:*
  - *Run* $(g, \mathsf{ak}) \leftarrow \mathsf{Gen}_\mathcal{G}(1^\lambda, \mathsf{kind})$ *such that* $g : \mathcal{D}^\mathcal{G} \to \mathcal{R}^\mathcal{G}$ *is an asymmetrical* kind*-hard function; Sample* $a \leftarrow \mathsf{Sample}_\mathcal{G}(1^\lambda, g)$ *and set* $\mathsf{aux} \triangleq a$*; Run* $y \leftarrow \mathsf{AsyEval}(1^\lambda, g, \mathsf{ak}, a, u)$ *and set* $\mathsf{aid} \triangleq y$*;*
  - *Run* $(f, \mathsf{td}) \leftarrow \mathsf{Gen}_\mathcal{F}(1^\lambda)$ *such that* $f : \mathcal{D}^\mathcal{F} \to \mathcal{R}^\mathcal{F}$ *is an one-way trapdoor function with trapdoor* aid*; Sample* $x \leftarrow \mathsf{Sample}_\mathcal{F}(1^\lambda, f)$ *and set* $\mathsf{task} \triangleq f(x)$*;*
  - *Output* $(\mathsf{task}, \mathsf{aux}) \triangleq (f(x), a)$*.*
- *Provers perform* $\mathsf{EvalSolve}(\mathsf{pp}, \mathsf{task}, \mathsf{aux})$*:*
  - *Parse* $(\mathsf{kind}, u) \in \mathsf{pp}$ *and* $(\mathsf{task}, \mathsf{aux}) = (f(x), a)$*;*
  - *Compute* $\mathsf{aid}' = g(a, u)$*;*
  - *Output* $\mathsf{solution} \triangleq x' \leftarrow \mathsf{Invert}(1^\lambda, f, \mathsf{aid}', f(x))$*.*
- *Any participant can perform* $\mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{solution})$*:*
  - *Parse* $(\mathsf{kind}, u) \in \mathsf{pp}$*,* $(\mathsf{task}, \mathsf{aux}) = (f(x), a)$*, and* $\mathsf{solution} = x'$*;*
  - *Output* 1*, if* $f(x') = f(x)$*; otherwise, output* 0*.*

3.3.3 Concrete Constructions

Now, we instantiate our generic construction with two concrete constructions. In both cases, we use the RSA problem [49] for the one-way trapdoor function. Whereas, in the time-hard TG-PoW scheme, we utilize the RSW time-lock puzzle [48] for the asymmetrically time-hard function; and in the memory-hard scheme, we use the DIODON function [8]. Concrete instantiations are as follows.

Construction 4 (Time-Hard TG-PoW) *Let p,q be two prime numbers of length λ.*

- $\mathsf{tdTaskGen}(1^\lambda, \mathsf{time}, u, \mathsf{td})$. *Parse* $\mathsf{td} = (p, q)$. *Compute* $N = pq$ *and* $\phi(N) = (p-1)(q-1)$. *Sample* $a \leftarrow \mathbb{Z}_N^*$ *and compute the u-RSW time-lock puzzle with* td *(or precisely,* $\phi(N)$*), i.e.,* $\mathsf{aid} = a^{2^u \mod \phi(N)} \mod N$. *If* $\gcd(\mathsf{aid}, \phi(N)) \neq 1$, *resample* a. *Otherwise, set* $\mathsf{aux} \overset{\Delta}{=} a$. *Compute* e*, such that* $e \cdot \mathsf{aid} \equiv 1 \mod \phi(N)$. *Sample* $y \leftarrow \mathbb{Z}_N^*$ *and output* $(\mathsf{task}, \mathsf{aux}) \overset{\Delta}{=} ((N, e, y), a)$*;*

- $\mathsf{EvalSolve}(\mathsf{pp}, \mathsf{task}, \mathsf{aux})$. *Parse* $(\mathsf{kind}, u) \in \mathsf{pp}$ *and* $(\mathsf{task}, \mathsf{aux}) = ((N, e, y), a)$. *Evaluate the u-RSW time-lock puzzle, i.e.,* $\mathsf{aid}' = a^{2^u} \mod N$. *Compute* $x' = y^{\mathsf{aid}'} \mod N$ *and output* $\mathsf{solution} \overset{\Delta}{=} x'$*;*

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{solution})$. *Parse* $(\mathsf{kind}, u) \in \mathsf{pp}$, $(\mathsf{task}, \mathsf{aux}) = ((N, e, y), a)$, *and* $\mathsf{solution} = x'$. *Output* 1 *if* $x'^e = y \mod N$*; otherwise, output 0.*

Construction 5 (Memory-Hard Construction) *Let p,q be two prime numbers of length λ.*

- $\mathsf{tdTaskGen}(1^\lambda, \mathsf{memory}, (k, l, u), \mathsf{td})$. *Parse* $\mathsf{td} = (p, q)$. *Compute* $N = pq$ *and* $\phi(N) = (p-1)(q-1)$. *Sample* $a \leftarrow \mathbb{Z}_N^*$ *and compute the* (k, l, u)*-DIODON function with* td *(or precisely,* $\phi(N)$*). If* $\gcd(\mathsf{aid}, \phi(N)) \neq 1$, *resample* a. *Otherwise, set* $\mathsf{aux} \overset{\Delta}{=} a$. *Compute* e*, such that* $e \cdot \mathsf{aid} \equiv 1 \mod \phi(N)$. *Sample* $y \leftarrow \mathbb{Z}_N^*$ *and output* $(\mathsf{task}, \mathsf{aux}) \overset{\Delta}{=} ((N, e, y), a)$*;*

- $\mathsf{EvalSolve}(\mathsf{pp}, \mathsf{task}, \mathsf{aux})$. *Parse* $(\mathsf{kind}, u) \in \mathsf{pp}$ *and* $(\mathsf{task}, \mathsf{aux}) = ((N, e, y), a)$. *Evaluate the* (k, l, u)*-DIODON function for* aid'. *Compute* $x' = y^{\mathsf{aid}'} \mod N$ *and output* $\mathsf{solution} \overset{\Delta}{=} x'$*;*

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{task}, \mathsf{solution})$. *Parse* $(\mathsf{kind}, u) \in \mathsf{pp}$, $(\mathsf{task}, \mathsf{aux}) = ((N, e, y), a)$, *and* $\mathsf{solution} = x'$. *Output* 1 *if* $x'^e = y \mod N$*; otherwise, output 0.*

3.4 Efficiency and Security

We first estimate the lower bound of solving the RSA problem so that the soundness parameter in the TG-PoW scheme, *i.e.*, $t(\lambda, c)$ in Definition 7, is meaningful (as discussed in Remark 1). Considering the general number field sieve algorithm [47], one of the fastest algorithms for factoring large integers[3], we have the following assumption.

[3] It is not clear if solving the RSA problem is as hard as factoring [11].

Assumption 2 *Let p,q be two prime numbers of length λ. Let $N = pq$ and $\phi(N) = (p-1)(q-1)$. Given a well-chosen instance of the RSA problem, i.e., (N, e, y) where $\gcd(e, \phi(N)) = 1$ and $y \xleftarrow{\$} \mathbb{Z}_N^*$, for any algorithm $\mathcal{A}$ runs in $\omega(2^{\lambda/c})$ time with c being a large enough coefficient, the following property holds:*

$$\Pr[x' \leftarrow \mathcal{A}(N, e, y) : x'^e = y \mod N] \leq \mathsf{negl}(\lambda).$$

Assumption 2 draws the line for soundness. Hence, we have a precise relation: $T(\lambda, u) \ll 2^{\lambda/c}$ so that the soundness of TG-PoW with such parameters is achievable. With this in mind, the efficiency of the concrete constructions (time-hard one as an example) is as follows.

- $\mathsf{tdTaskGen}$ samples two elements from $\mathbb{Z}_N^*$ and computes the RSW time-lock puzzle via asymmetrical evaluation. The cost is determined by two exponentiation operations in $\mathbb{Z}_N^*$, hence, in $\tilde{O}(\lambda)$;
- $\mathsf{EvalSolve}$ evaluates the RSW time-lock puzzle without the bypassing key and computes the inversion of the RSA problem with a corresponding trapdoor. The cost is determined by the RSW evaluation, which is $\delta(u)$, hence, matching the designed resource demand.
- Verify checks the validity of inverting the RSA problem with one exponentiation in $\mathbb{Z}_N^*$. Hence, the cost is $\tilde{O}(\lambda)$.

Security Analysis In this section, we showcase the security proofs for the time-hard TG-PoW construction. We first prove soundness and time-hardness under Assumption 1 and Assumption 2c, respectively. Then, we show difficulty as a derived property from soundness and hardness. The security proof for memory-hard construction follows the same process with only minor changes concerning the time-memory trade-off.

Theorem 1 *Our time-hard TG-PoW in Construction 4 satisfies the following properties:*

- *Correctness.*
- *Soundness under Assumption 2.*
- *Time-hardness under Assumption 1, following Lemma 1.*
- *Difficulty, by soundness and time-hardness.*

Proof 2 *Correctness is obvious due to the construction. We prove soundness, time-hardness, and difficulty as follows.*

Soundness: The uniform sampling $a \xleftarrow{\$} \mathbb{Z}_N^*$ *ensures the uniformity of* aid *and* e *that satisfy* $e \cdot \mathsf{aid} \equiv 1 \mod \phi(N)$. *In order to break soundness, given* (N, e, y), *the adversary should produce a valid* aid'. *However, by uniformity,* $\Pr[\mathsf{aid}' = \mathsf{aid}] \leq \mathsf{negl}(\lambda)$. *Moreover, as we bound the adversary's run time by* $2^{\lambda/c}$, *it cannot invert* (N, e, y) *directly.*

Time-hardness: Both hardness definitions enable the adversary to make at most q *queries to the hard function evaluation oracle. This approach captures the pre-processing procedure of the adversary learning polynomial many* $(\mathsf{aux}, \mathsf{aid})$ *pairs from previous evaluations. In order to break time-hardness, the adversary must produce a valid* aid' *given a freshly generated* aux^*. *As shown above, the uniformity of auxiliary ensures that* aid' *being valid is negligible of* λ. *Moreover, by Lemma 1, the time-hardness of the RSW time-lock puzzle ensures that the probability of the adversary obtaining a valid solution with less than* $\delta(u)^{1-\epsilon}$ *is negligible of* λ.

Difficulty: From soundness and time-hardness, we cannot directly yield difficulty. This is because aux *may leak information of* aid, *i.e., accelerating the evaluation of the hard function. Hence, following [10], we consider the unpredictability (min-entropy) of the RSW time-lock puzzle, which guarantees that no adversary can guess any bits in* aid *from* aux *without performing enough evaluation, i.e., less than* $\delta(u)^{1-\epsilon}$ *(captured by hardness). As soundness already shows that no adversary can break the TG-PoW task by brute force, we consider a reduction from difficulty to time-hardness with the following chain of games:*

1. *Game 0: The original difficulty game (Definition 10). The adversary is given* $(\mathsf{task}^*, \mathsf{aux}^*) \leftarrow \mathsf{tdTaskGen}(\cdot)$ *with the corresponding* pp *as the inputs for its* $\mathsf{EvalSolve}$ *algorithm;*

2. *Game 1: We replace the challenge task* task^* *with an arbitrary task* task*;*

3. *Game 2: The original hardness game (Definition 8). The adversary is only given* aux^*.

Since reduction loss between each game is negligible of λ, *difficulty holds as long as hardness holds.*

3.5 Discussion

This chapter presents a PoW scheme under the assumption that task generator is trustworthy. Such an assumption will significantly prevent our construction from being adopted in decentralized environments as the hash-based PoW. Interestingly, this problem also burdens the existing deep learning-based PoUW schemes, hence, being a part of our motivation for the next result presented in Chapter 4. However, unlike our next result successfully overcomes this drawback, we argue that until today, it is not easy to create a distributed task generation protocol for our TG-PoW constructions. One reason is that there is no simple distributed generation protocol for the RSA problem.

Recall that our initial purpose is to investigate the original use case of PoW schemes, *i.e.*, a resource-demanding computation mechanism with public and efficient verification. We argue that our TG-PoW constructions (generic and concrete) satisfy this goal. Moreover, like the timed-release symmetric key encryption scheme proposed in [48] using the RSW time-lock puzzle, we observe that our generic construction is easy to transform into a timed-release public key encryption scheme under slight modifications.

Chapter 4

Blockchain from Deep Learning Tasks

The starting point of this chapter [53] lies in an observation: Despite the potential applications, research of deep learning-based proof-of-useful-work (PoUW) severely lacks formality in syntax and security analysis. Our result overcomes these problems by proposing a generic and two concrete blockchain protocols based on a novel *distributed* proof-of-deep-learning (D-PoDL) scheme. Moreover, our protocols can be proven secure under robust ledger properties (Definition 3 and 4, which further yields persistence and liveness, *i.e.*, the standard notion of consensus protocols.

4.1 Overview

Recently, Chenli et al. [17] propose a PoUW scheme that utilizes the training process of deep learning tasks as useful work. To the best of our knowledge, there are only a handful of papers targeting the same problem [5,16,17,38–40]. We show a brief analysis to them in the following.

4.1.1 Review of Existing Works

These projects all involve interactions between task publishers who oversee the publication of deep learning tasks and miners who aim to solve these tasks. In most cases, task publishers are restricted from participating as miners due to limited computational power or security concern. An exception to this is Proof-of-Learning (PoLe) [38], which avoids this impractical limitation by introducing secure mapping layers during model training. However, this approach hinders collaboration among miners, which goes against our objective. A typical deep learning task comprises several components: a description of the task, a training dataset, a potential test dataset, and accuracy target thresholds. In Proof-of-Deep-Learning (PoDL) [17], Li et al.'s work [39], and PoLe [38], miners are required to train a model on the training dataset, and the model is verified according to the test dataset and test accuracy. This approach relies on a strong synchronous network assumption, where the task publisher needs to release the test dataset only after miners have finished training their models. Without this synchronicity, an adversary can train their model directly using the test dataset, undermining the process.

DLchain [16] overcomes the strong synchronous assumption by removing the test dataset-based verification. Instead, it focuses on improving training accuracy. In order to verify a trained model efficiently, DLchain utilizes a merkle-tree-based verification [18] to check training history. Moreover, DLchain considers a similar goal to distribute PoDL, *i.e.*, achieving better accuracy distributively. They *partially* fulfill the goal with priorly determined "short-term targets" which are accuracy targets below the threshold. Miners can generate blocks once their models surpass a short-term accuracy target. However, considering only training accuracy may result in overfitting, and determined short-term targets can affect blockchain growth rate, which may weaken the security of the protocol [26].

CoinAI [5] is a descriptive work that proposes an outline for designing a deep learning-based PoUW and proof-of-storage scheme. The authors propose a "hash-to-architecture" mapping based on format context-free grammar. It maps a hash value to an initial deep learning model concerning model architectures, including hyper-parameters and initial learnable-parameters. The hash-to-architecture technique is vital for security since it prevents miners from grinding initial parameters. However, the security impacts are not clarified due to the lack of formality in [5]. Instead of proposing a PoUW-based blockchain protocol, Lihu et al. [16] aim at taking blockchain's security to enhance artificial intelligence systems. However, the protocol requires a dedicated blockchain structure and suffers from complicated system design. For example, participants must select their role before execution, and a unique type of participant called the supervisor needs to monitor all message history during the execution. Thus, their work cannot be integrated into any current blockchain-based protocols.

To sum up, none of these works can serve as a fully distributive deep learning task solver, which is more desirable in distributed environments. Another crucial problem is the lack of proper security analysis of the blockchain protocol. For example, only DLchain [16] provides security proof against double-spending attacks. However, a secure blockchain protocol should satisfy robust ledger properties, *i.e.*, the chain growth, chain quality, and common prefix. Therefore, our motivation for this chapter is to overcome the problems in the deep learning-based PoUW schemes mentioned above, *i.e.*, (1) to remove strong or impractical assumptions; (2) to *distribute* the computation of deep learning-based PoUW; (3) to provide concrete and thorough *security analysis* for blockchain protocols based on our extended scheme. Next, we further detail our result's significance.

4.1.2 Our Approach and Contributions

This chapter proposes a distributed proof-of-deep-learning (D-PoDL) scheme by extending deep learning-based PoUW schemes so that provers can work collaboratively on given tasks. Note that the term "distributed" in D-PoDL differs from distributed deep learning, *i.e.*, we do not require provers to perform a single training course together but let them train atop published pre-trained models.

Intuitively, D-PoDL provers train a model from a given deep learning task as their useful work. We propose a *"hash-training-hash"* structure to achieve adjustable difficulty while preventing provers from cherry-picking initial parameters (grinding attack) and pre-computing task instances (pre-computation resilience). As a result, the provers output a trained model with the corresponding accuracy and step number for D-PoDL verifiers to check. Another novelty of our scheme lies in how we handle intermediate models. Throughout the chapter, an intermediate model, also called a pre-trained model, is a model "somewhat" trained yet failing to meet a given accuracy or security level. Instead of discarding such a model, we propose *"model-referencing"* that enables any prover to reference the pre-trained model. Hence, provers can start their training process atop the referenced model. Moreover, a referenced model will be rewarded so that even if the prover fails to meet the goals, it is incentivized to do more training iterations. We emphasize that this approach forms the distributed training process among provers, and such a design is never discussed in any previous work.

The second contribution is that we build a generic blockchain protocol based on our proposed D-PoDL scheme. We clarify the roles of participants: task publishers, miners, and external storage providers. Instead of assuming task publishers' inability to train models properly, we enable them to perform as miners while preventing them from pre-computing deep learning tasks with the hash-training-hash structure. Only Pole [38] shares the same property by embedding secure mapping layers into its training algorithm. Moreover, we make use of both training and test datasets. Concretely, miners (D-PoDL provers) extend the blockchain with models that have better *training* accuracy. In order to mitigate the overfitting problem while avoiding the strong synchronous network setting, we require miners to work on each deep learning task for multiple time slots. Hence, task publishers can evaluate the produced models with test dataset and select according to *test* accuracy. Since the training process is publicly verifiable, task publishers cannot take advantage by training directly on the test dataset. We will also discuss model verification and storage issues in Section 4.2.3 and Section 4.3.1.

Furthermore, the generic D-PoDL-based blockchain protocol is capable of two different chain selection rules: *i.e.*, the conventional "longest-chain rule" [26] and the "weight-based" framework [27, 33]. The former requires honest miners to choose the longest branch as their chain whenever a fork occurs. In contrast, the weight-based framework assigns blocks with weights according to their quality. Hence, honest miners choose the branch with higher accumulated weight as their chain. Although the longest-chain rule can be considered a special case of the weight-based framework, we separate them into two concrete protocols and prove the robust ledger properties for each. Finally, we implement our D-PoDL scheme and compare it to existing schemes.

Table 4.1 compares our result and related works. Note that we omit CoinAI [5] due to its informality and Lihu et al.'s work [40] due to their different research focus. We also include a recent result on stochastic local search-based PoUW [23]. The difference between our result and the PoUW [23] is that we leverage deep learning characteristics, *e.g.*, verifiable training steps and test datasets, and derive a simple yet versatile protocol (*i.e.*, proven secure under different chain selection rules).

Table 4.1: Comparison with Previous Works

Protocols	Work Evaluation	Network Synchronicity	Publisher As Miner	*Distributed* Task Solver	Formal Security
Chenli et al. [17]	Test accuracy	Strong	X	X	X
Lan et al. [38]	Test accuracy	Strong	✓[1]	X	X
Li et al. [39]	Training accuracy	Bounded	X	X	X
Chenli et al. [16]	Training accuracy	Bounded	X	△[2]	△[3]
Fitzi et al. [23]	—[4]	Bounded	✓	✓	✓[5]
Our result	Training and test	Bounded	✓	✓	✓[6]

Notes: (1) By secure mapping layers; (2) By pre-determined short-term targets; (3) Against double-spending attack; (4) Stochastic local search; (5) Under the longest-chain rule [26]; (6) Against robust ledger properties under the longest-chain rule [26] and the weight-based framework [27, 33].

4.1.3 Final Preparations

We finalize our overview by presenting our modification to the hash-to-architecture mapping mechanism [5] and the concrete execution setting of the blockchain protocol.

Hash-to-architecture mapping. The hash-to-architecture mapping mechanism from [5] is based on the formal context-free grammar and used to establish a surjective function between a hash value and a proper deep learning architecture setup. Denote the original hash-to-architecture mapping with HtoA^*, *i.e.*, given a hash value h, $\mathsf{HtoA}^*(h) = (\mathsf{A}(\mathsf{hpp}), \mathsf{initLP})$ where $\mathsf{A}(\mathsf{hpp})$ is the architecture A concerning hyper-parameters hpp, and initLP denotes the initial learnable parameters. Our modification, denoted by HtoA, is to generate an additional random value from the hash, *i.e.*, given a hash value $h = h_1||h_2$ and a hash function $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^\lambda$, we extract $r = \mathsf{Hash}(h_2)$ and run $\mathsf{HtoA}^*(h_1) = (\mathsf{A}(\mathsf{hpp}), \mathsf{initLP})$ so that the outputs of $\mathsf{HtoA}(h)$ is $(\mathsf{A}(\mathsf{hpp}), \mathsf{initLP}, r)$.

Concrete execution settings. We present our D-PoDL-based blockchain protocol settings as follows.

- Time and network: We assume the protocol execution proceeds in rounds, which corresponds to the smallest unit of time of interest. The network is synchronous with a *known* bounded delay δ time on the delivery time, *i.e.*, any message sent by an honest node in round r is guaranteed to arrive at all honest nodes until round $r + \delta$;
- Corruptions: We allow the adversary to corrupt up to $\beta < \frac{1}{2}$ fraction of nodes before each round, *i.e.*, a corrupted node is under the adversary's complete control from the round. We also assume the adversary is rushing, *i.e.*, it receives honest users' messages first and decides the order of message delivery or whether to inject messages for each recipient.

4.2 The D-PoDL Scheme

As an extension to deep learning-based PoUW schemes, our D-PoDL scheme provides an interface for its provers to solve a deep learning task together. Like PoUW, a D-PoDL scheme involves two types of participants: provers and verifiers. On a given deep learning task, a prover intends to output a trained model, and claims the corresponding training accuracy and step number. Whereas, a verifier checks if the model matches the prover's claims and responds accordingly. This section presents the D-PoDL scheme in terms of requirements and syntax. We focus on a setting where provers work on a priorly given deep learning task with a designed target threshold. We clarify that the scheme focuses on solving the task and verifying the model. Discussions about task selection, block generation, and blockchain dynamics can be found in the protocol description in later sections, *i.e.*, Section 4.3 and Section 4.4.

Additional requirements for D-PoDL. A D-PoDL scheme should satisfy the requirements given in Section 2.2.1 and follow the same security requirements [23] as the PoUW, *i.e.*, no-grinding, pre-computation resilience, and adjustable difficulty. Here, we list the informal **requirements** as follows.

- (1a) No-grinding: The adversary cannot cherry-pick hyper-parameters to gain training advantages, *i.e.*, less training steps with higher accuracy;
- (1b) Pre-computation resilience: The adversary cannot manufacture problem instances to train the model faster;
- (1c) Adjustable difficulty: The block difficulty (measured by training accuracy) can be adjusted to the computing power of the network;
- (2a) Efficient verification: The running time of the verification algorithm should be at most poly-logarithm of provers' training time;
- (2b) Measurable usefulness: The usefulness of a training process can be quantified and compared to each other.

4.2.1 Design Overview

Along with the two processes in a D-PoDL scheme, *i.e.*, solving a deep learning task and verifying the correctness of the solution, we propose a novel "hash-training-hash" structure for the solving process and utilize a widely used merkle-tree-based verification procedure [18] as a black-box for the verification process. Additionally, we propose a weighting algorithm to evaluate a weight function that quantifies a solution's usefulness. We describe the "hash-training-hash" structure briefly in this section. More details of our design choices can be found after the formal definition.

Intuitively, on a given deep learning task, we enable provers to initialize its solving algorithm with either a fresh or a pre-trained model from any prover, *i.e.*, for "model-referencing". The first hash requires provers to perform a proof-of-work (PoW) with threshold T_1, *i.e.*, a prover needs to find a nonce such that the hash value of the previous block, potentially a pre-trained model and the nonce is less than T_1. If the hash value passes the PoW check (less than T_1), the prover can map the hash value to an architecture with respect to hyper-parameters, (initial) learnable-parameters and a random seed with our modified hash-to-architecture algorithm. As introduced in Section 4.1.3, the architecture (with hyper-parameters) and learnable-parameters determine a deep learning model. The prover trains the model by updating learnable-parameters iteratively. The post-hash checks the output model against threshold T_2 to decide if the models are eligible for publishing. If the post-hash fails, the prover can return to the pre-hash or training process. The prover must perform more training iterations in both cases to generate a valid model.

4.2.2 Formal Syntax and Construction

A D-PoDL scheme involves a tuple of algorithms $(\mathsf{Setup}, \mathsf{Solve}, \mathsf{Verify}, \mathsf{Weight})$. Setup extracts a training dataset and a designed target threshold from a deep learning task. Solve consists of three sub-algorithms $\mathsf{PreHash}$, Train, and $\mathsf{PostHash}$. In general, $\mathsf{PreHash}$ determines the initial model, including its architecture, hyper-parameters, learnable-parameters, and a random seed. Train casts the training process and outputs a model with the corresponding accuracy and step number. Note that we do *NOT* restrict the training algorithm to provide generality for our design. Instead, as we will show in Section 4.4.1, we model it with an oracle due to its stochastic nature and model provers' computing power by their capability of oracle queries. Next, due to security concerns, $\mathsf{PostHash}$ returns a bit according to a hash proof. Verify verifies the trained model's validity concerning accuracy. Weight is available to both provers and verifiers, and it evaluates a weight function $w : \mathsf{acc} \times T_{\mathsf{acc}} \rightarrow \mathbb{R}$, which maps the model's accuracy and a priorly decided target threshold to a real value. We present the formal syntax and construction of the D-PoDL scheme as follows.

Construction 6 (D-PoDL Scheme) *Given the hash-to-architecture algorithm* $\mathsf{HtoA}(\cdot)$ *from Section 4.1.3 and the weight function* $w : \mathsf{acc} \times T_{\mathsf{acc}} \rightarrow \mathbb{R}$*, the tuple algorithm of a D-PoDL scheme* $(\mathsf{Setup}, \mathsf{Solve}, \mathsf{Verify}, \mathsf{Weight})$ *works as follows:*

- $\mathsf{Setup}(1^\lambda, \mathsf{task})$ *takes as input the security parameter* λ *and the description of a deep learning task* task *from the task publisher.* Setup extracts *the public parameter* pp *and a pair of threshold* (T_1, T_2) *for security concerns from the system. It* parses *the task with a training dataset* D *and* *a target threshold* T_{acc}. Setup *outputs* $(\mathsf{pp}, T_1, T_2, \mathsf{D}, T_{\mathsf{acc}})$. *We omit* pp *later for simplicity;*

- $\mathsf{Solve}((T_1, \mathsf{prevBK}, \mathsf{refM}), (\mathsf{D}, T_{\mathsf{acc}}), T_2)$. *We divide* Solve *into three algorithms:* $(\mathsf{PreHash}, \mathsf{Train}, \mathsf{PostHash})$.
  - $\mathsf{PreHash}(T_1, \mathsf{prevBK}, \mathsf{refM})$ *takes as input* T_1*, a previous block* prevBK *and potentially a pre-trained model* refM. *It samples* nonce *such that* $\mathsf{Hash}(\mathsf{prevBK}, \mathsf{refM}, \mathsf{nonce}) = h_1 \leq T_1$. *If* $\mathsf{refM} = \perp$, $\mathsf{PreHash}$ *runs* $\mathsf{HtoA}(h_1) = (\mathsf{A}(\mathsf{hpp}), \mathsf{lp}, r)$ *where* $\mathsf{A}(\mathsf{hpp})$ *denotes the architecture,* lp *is the learnable-parameters, and* r *is the random seed. It sets* $\mathsf{initM} = (\mathsf{A}(\mathsf{hpp}), \mathsf{lp})$*; Otherwise, It parses* $\mathsf{refM} = (\mathsf{A}(\mathsf{hpp}_{\mathsf{ref}}), \mathsf{lp}_{\mathsf{ref}})$ *and sets* $\mathsf{initM} \in \{\mathsf{refM}, (\mathsf{A}(\mathsf{hpp}), \mathsf{lp})\}$. *Then,* $\mathsf{PreHash}$ *returns* $(\mathsf{nonce}, \mathsf{initM}, r)$*;*
  - $\mathsf{Train}(\mathsf{D}, T_{\mathsf{acc}}, \mathsf{initM}, r)$ *takes as input the training dataset* D*, a target threshold* T_{acc}*, a initial model* initM *and a random seed* r. *It parses* $\mathsf{initM} = (\mathsf{A}(\mathsf{hpp}), \mathsf{lp})$ *and trains the model by updating learnable-parameters iteratively.* Train *returns* $\mathsf{M} = (\mathsf{A}(\mathsf{hpp}), \mathsf{lp}^*)$*, the corresponding training accuracy* $\mathsf{acc} \in [0, 1]$*, step number* S *and a list of checkpoints* $\mathsf{CPs} \triangleq \{(\mathsf{M}_i, \mathsf{acc}_i, S_i)\}$ *where each entry denotes an intermediate result of the training process;*
  - $\mathsf{PostHash}(T_2, \mathsf{M}, \mathsf{acc}, S)$ *takes as input* T_2 *and a model* M *with the corresponding accuracy* acc *and step number* S. *It computes* $\mathsf{Hash}(\mathsf{M}, \mathsf{acc}, S) = h_2$. *If* $h_2 \leq T_2$, $\mathsf{PostHash}$ *returns* 1*; Otherwise, it returns* 0.

  Finally, Solve *outputs* $((\mathsf{refM}, \mathsf{nonce}, \mathsf{initM}, r), (\mathsf{M}, \mathsf{acc}, S), b)$ *where* $b \in \{0, 1\}$*;*

- $\mathsf{Verify}((T_1, \mathsf{prevBK}, \mathsf{refM}, \mathsf{nonce}, \mathsf{initM}, r), (\mathsf{D}, T_{\mathsf{acc}}, \mathsf{M}, \mathsf{acc}, S, \mathsf{CPs}), (T_2, b))$ *checks:*
  - *If* $\mathsf{Hash}(\mathsf{prevBK}, \mathsf{refM}, \mathsf{nonce}) = h_1 \leq T_1$ *and if* initM *is derived correctly from* refM*;*
  - *If* M *is trained correctly from* initM *with* Train *according to* (S, CPs) *and if the corresponding accuracy* $\mathsf{acc}' = \mathsf{acc}$*;*
  - *Compute* $\mathsf{PostHash}(T_2, \mathsf{M}, \mathsf{acc}, S) = b'$ *and check if* $b' = b$.

  If the situations above are satisfied, Verify *outputs* 1*; Otherwise, it outputs* 0.

- $\mathsf{Weight}(\mathsf{acc}, T_{\mathsf{acc}})$ *evaluates the weight function* w *and outputs* $\mathsf{w} \in \mathbb{R}$.

4.2.3 Design Choices Explanation

Here, we explain our construction choices with respect to the **requirements**.

Setting up initial models with pre-hash. There are countless different architectures in deep learning, each with its characteristics and limitations. After selecting an appropriate architecture A, provers need to choose hyper-parameters and initial learnable-parameters for the model, which may affect the speed and quality of the training process. Usually, hyper-parameters are not learnable, so provers must go through random sampling before obtaining a good set of hyper-parameters. However, we may open a gate for grinding attacks (**Requirement 1a**) if we offer provers the ability to choose hyper-parameters and initial learnable-parameters. An adversary may outperform honest users' training speed and quality by cherry-picking.

In order to mitigate this problem, we adopt the same approach as in Ofelimos [23]. Concretely, we rely on a PoW scheme with threshold T_1, which requires provers to sample a nonce nonce randomly and compute the hash of the previous block (prevBK), potentially a pre-trained model refM with the nonce such that $\mathsf{Hash}(\mathsf{prevBK}, \mathsf{refM}, \mathsf{nonce}) = h_1 \leq T_1$. The hash function's uniformity prevents provers from grinding hyper-parameters and learnable-parameters. Note that our T_1 should not be as hard as a stand-alone PoW, *e.g.*, the one in the Bitcoin system, because we intend to encourage provers to train models instead of solving PoW. Finally, if refM is empty, the prover needs to generate an initial model with $\mathsf{HtoA}(h_1) = (\mathsf{A}(\mathsf{hpp}), \mathsf{lp}, r)$ such that $\mathsf{initM} = (\mathsf{A}(\mathsf{hpp}), \mathsf{lp})$; Otherwise, the prover can either refer to the pre-trained model $\mathsf{refM} = (\mathsf{A}(\mathsf{hpp}_{\mathsf{ref}}), \mathsf{lp}_{\mathsf{ref}})$ or use the freshly generated hyper-parameters and the pre-trained learnable-parameters $(\mathsf{A}(\mathsf{hpp}), \mathsf{lp}_{\mathsf{ref}})$ as its initial model. In this case, the pre-hash enforces provers to establish links from their model to previous blocks and the referenced models. Such links are crucial to the security of model-referencing.

Model-referencing and pre-computation resilience. An initial model can be sampled from HtoA or from a pre-trained model refM. The purpose of taking as input a pre-trained model is to enable provers to work atop any valid but not-good-enough model. Hence, we prevent their computing power from being wasted and form a distributed solver for given deep learning tasks. However, starting from a pre-trained model can shorten the prover's training iteration because these models may be only a few steps from reaching the accuracy target threshold. For example, an adversary may steal an honest prover's outputs $(\mathsf{M}_0, \mathsf{acc}_0, S_0)$ and produce a new model $\mathsf{M}_{\mathcal{A}}$ with accuracy $\mathsf{acc}_{\mathcal{A}} \geq T_{\mathsf{acc}} \geq \mathsf{acc}_0$ and a claimed step number $S_{\mathcal{A}}$. Such an attack violates pre-computation resilience (**Requirement 1b**) because the adversary achieves better accuracy while performing only $(S_{\mathcal{A}} - S_0)$ training steps.

In order to tackle this problem, we design a novel mechanism called "model-referencing". We require provers to make references if their models are trained based on another model. Otherwise, their models are regarded as invalid. The reference is $(\mathsf{prevBK}, \mathsf{refM}, \mathsf{nonce})$, which can be publicly verified with $\mathsf{Hash}(\mathsf{prevBK}, \mathsf{refM}, \mathsf{nonce}) \leq T_1$. Hence, model-referencing enables provers to train each others' models together for the same goal (surpassing the target threshold and post-hash check threshold) while preventing them from stealing others' models (by discarding those "use-without-reference" models). Furthermore, the provers should only reference the latest models, *i.e.*, if two pre-trained models share the same setup, provers should reference the model with higher accuracy and step number. With this setting, we also prevent provers from flooding the system with too many pre-trained models. Therefore, the mechanism inherently forms an additional "link" (like the hash link between blocks) that connects models, *i.e.*, a valid block must be linked to a previous block and a previous model. More details can be found in Section 4.3.1.

Adjusting computation with post-hash. One argument concerning the adjustable difficulty (**Requirement 1c**) is that training a model so that its accuracy surpasses the target threshold T_{acc} should be harder than finding a nonce to meet the PoW puzzle with threshold T_1. Otherwise, the computational difficulty is determined by the PoW rather than the training process, which violates the usefulness of our scheme. Hence, we propose a solution based on [9]'s approach, which requires the provers to perform *one* "post-hash" against a threshold T_2 to decide if their models are eligible for publishing. If a model fails the post-hash, the prover must revert to the pre-hash or training process. The threshold T_2 guarantees the overall security and usefulness level for our scheme, *e.g.*, to preserve the 10 minutes interval for block generation while enforcing provers' computation focus on model training instead of PoW. We will show the impact of the post-hash algorithm during our implementation of the D-PoDL scheme in Section 4.5.

Model verification. In order to verify the outputs of a prover, a verifier needs to check three conditions: (1) If the nonce satisfies the PoW check with threshold T_1; (2) If the model has the claimed accuracy; (3) If the post-hash outputs a correct bit. In this section, we focus on verifying the model and its accuracy. A naive approach is to check the prover's model with the given training dataset. However, it takes as many iterations as the training algorithm, which violates the efficiency requirement (**Requirement 2a**).

We solve this problem by adopting the widely used merkle-tree-based verification [18] as a black box. This approach is also mentioned in the previous work [16]. Namely, provers are required to include several intermediate results as checkpoints into their training outputs and build a merkle-tree accordingly. Hence, verifiers only need to check the validity of these checkpoints. Given n checkpoints, the time complexity for verifiers can be reduced to $\mathsf{O}(\mathsf{polylog}(n))$ at the cost of provers' space complexity being $\mathsf{O}(\mathsf{poly}(n))$. Moreover, there is a trivial trade-off between the interval of two checkpoints and the granularity of the check. As pointed out by [16], the interval setting can be left to users in real-life applications and adjusted according to accuracy thresholds. However, since each checkpoint has the size of a model, we explain this in Section 4.3.1 with respect to external storage providers.

Measuring usefulness. The D-PoDL scheme focuses on improving the models' training accuracy, whereas, the test accuracy is left for the protocol. Except for the conventional longest-chain-based blockchain protocols [26], we intend to build our D-PoDL scheme under a weight-based framework by [33]. In such a setting, blocks are assigned with weight, and the chain is selected based on the accumulated weight. We argue that the weight-based approach is natural for the D-PoDL scheme because accuracy can be regarded as a quantified measurement for usefulness (**Requirement 2b**). Moreover, we can generalize the weight-based approach to arbitrary PoUW schemes as long as their usefulness is measurable.

4.3 D-PoDL-Based Blockchain Protocols

This section describes the transformation from our D-PoDL scheme to D-PoDL-based blockchain protocols. As mentioned in the execution model from Section 4.1.3, our protocol proceeds in rounds. Honest users may share a slightly different view of the round number. We further divide our protocol execution into time slots. Each time slot is associated with a deep learning task, and the time slot ends when a validly generated block is added to the blockchain. Thus, a time slot may include multiple rounds. Considering the workflow within a time slot, we propose a generic D-PoDL blockchain protocol design as in Figure 4.1. Then, two concrete protocols are derived from the generic design by instantiating the chain selection rule with the longest-chain rule [26] and the weight-based framework [33]. Finally, we discuss the incentive model of our protocols.

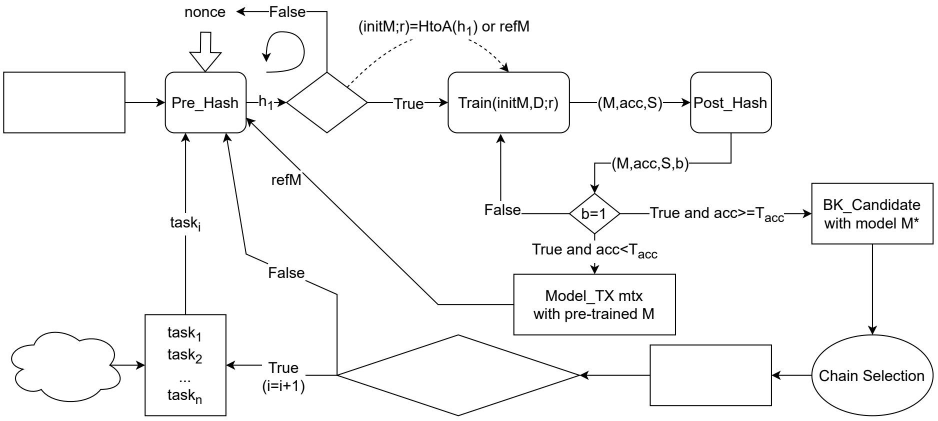


Figure 4.1: Design of our Generic D-PoDL Blockchain Protocol

4.3.1 Generic Protocol Workflow

The generic blockchain protocol involves three types of participants: task publishers, miners, and external storage providers. Task publishers handle deep learning tasks. Each task is associated with a dataset and desired accuracy thresholds. Task publishers first split the dataset into a training dataset and a test dataset. They publish the task description, the training dataset, and the corresponding desired training accuracy as the target threshold. Miners perform the protocol by generating and verifying blocks according to the D-PoDL scheme's instructions. Concerning the size of deep learning models and checkpoints (which are the same size as models), we employ the approach from [16] to prevent storage overhead, *i.e.*, embedding only a downloadable link within the block and relying on external storage to store the whole model and checkpoints.

Task publication. We start our generic D-PoDL blockchain protocol from the task publication mechanism. In order to keep the task publication as generic as possible, we consider a situation in which these publishers form a network to publish and decide the order of tasks. They aim to organize a distributed solver for deep learning tasks and can be benefited from receiving the solutions. The only requirement is that the outputs of the publisher network should be an ordered list of deep learning tasks. We denote the output as $\{\mathsf{task}_i\}_{i\in[n]}$ where task_i spans over a period of ℓ_i time slots in $\mathsf{T}_i = \{t_{i,j}\}_{j\in[\ell_i]}$.

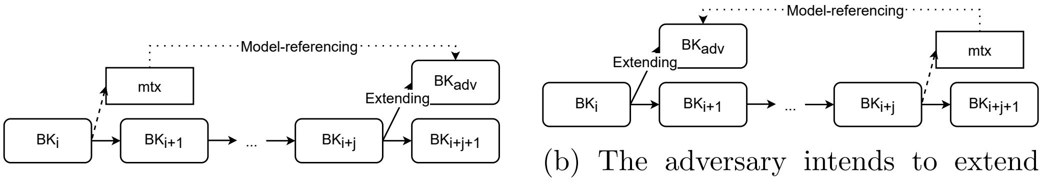
Note that we do *NOT* separate task publishers from miners, *i.e.*, a task publisher can participate in the protocol as a miner and gain mining rewards. We argue that the task publisher cannot pre-compute the task to gain advantages over regular miners due to the pre-hash algorithm and the model-referencing mechanism. Without loss generality, let the current time slot be $t_{p,q}$, we consider an adversarial publisher who intends to pre-compute deep learning task task_i where $i > p, q \in [\ell_p]$. Since $i > p$, without pre-trained models, the publisher has to train an initial model generated from HtoA, *i.e.*, to find nonce such that $\mathsf{Hash}(\mathsf{prevBK}, \mathsf{nonce}) = h_1 \leq T_1$ where prevBK associates with slot $t_{i-1,\ell_{i-1}-1} \in \mathsf{T}_{i-1}$ and compute initM from $\mathsf{HtoA}(h_1)$. To find such a nonce requires the publisher to either predict the block in the future or find a collision in the hash function. Since the probabilities of both cases are negligible, the publisher cannot produce a trained or pre-trained model to pass the D-PoDL scheme's verification by pre-computing task_k.

Execution of D-PoDL scheme. Now, we consider a deep learning task task_i given to miners in time slot $t_{i,j}$ where $j \in [\ell_i]$. Each miner runs as a prover of the D-PoDL scheme. The Setup algorithm first extracts public parameters, a training dataset, and thresholds $(T_1, T_2, T_{\mathsf{acc}})$. The miner then finds a nonce and initializes initM with $\mathsf{PreHash}$; It runs the training course on the initial model with the randomness r to obtain a model M, the corresponding accuracy acc and step number S; $\mathsf{PostHash}$ tests the model according to T_2 and outputs a bit b. The miner outputs a tuple, including potentially a pre-trained model as the reference, a nonce, an initial model, a random training seed, a trained model with the corresponding accuracy and step number, and a post-hash check bit.

According to the post-hash check, the miner decides if its model is eligible for publishing. Moreover, for generality, we introduce a relation between the model accuracy and the target threshold as $\mathcal{R}(\mathsf{acc}, T_{\mathsf{acc}})$, which will be instantiated in concrete protocols. Hence, when $b = 1 \wedge \mathcal{R}(\mathsf{acc}, T_{\mathsf{acc}}) = 1$, the miner collects transactions from the mempool as in conventional blockchain protocols and generates a block candidate embedding the obtained model M; Otherwise, the miner generates a special *model-transaction* mtx for model-referencing, which contains the outputs of the Solve algorithm, *i.e.*, $\mathsf{mtx} = (\mathsf{prevBK}, (\mathsf{refM}, \mathsf{nonce}, \mathsf{initM}, r), (\mathsf{D}, T_{\mathsf{acc}}, \mathsf{M}, \mathsf{acc}, S, \mathsf{CPs}), 1)$. mtx is published into the mempool as ordinary transactions. Any miner can reference the model M in the model-transaction mtx by including mtx in the miner's newly found block or model-transaction. That is, the miner takes as input $\mathsf{refM}' = \mathsf{M}$ for its Solve algorithm. Note that different miners can refer to the same model-transaction. We do not count this as "double-spending" since no ordinary (money-used) transaction is involved. Newly trained models still need to compete for acceptance. Moreover, miners can reference model-transactions recursively, *i.e.*, generating a model-transaction mtx' with higher accuracy from mtx is acceptable. The only restriction here is that miners must reference the latest model-transaction, which embeds a pre-trained model with the highest accuracy observed so far. This prevents the adversary from releasing a large amount of model-transaction to DoS attack [44] the network.

Cross time slot attacks and restrictions on step number. We leave block selection (with respect to forks) to concrete protocols in Section 4.3.2. Here, we consider the whole period (a span of time slots) associated with a deep learning task. Once the blockchain gets updated, miners proceed to the next time slot. A task can span over multiple time slots so that the publisher network can check each selected model with the corresponding *test* dataset. This approach is to mitigate the trend toward overfitting models since miners are given only training datasets to overcome the strong synchronous network assumption.

However, this approach allows adversaries to reference models generated in different time slots from the block they extend. Given a fragment of the blockchain that associates with a deep learning task, we illustrate two attack strategies in Figure 4.2a and 4.2b. Note that the adversary can also reference models embedded in blocks. We use model-transactions here for generality.



(a) The adversary intends to extend block bk_{i+j} by referencing an mtx that links to block bk_i.

(b) The adversary intends to extend block bk_i by referencing a model-transaction mtx that links to block bk_{i+j}.

Figure 4.2: Intuition of Cross Time Slot Attacks

The first attack enables the adversary to extend the blockchain with fewer training steps while not violating model-referencing requirements. In the second attack, the adversary can produce a model with higher accuracy or weight using new information, *e.g.*, the model in mtx of Figure 4.2b. This attack may subvert blockchain history if the adversary produces enough blocks to compete with the selected chain.

In order to tackle these problems, we restrict the training step number in published *blocks*. We introduce a lower bound of acceptable step number as smin to control the selected blocks' step number during the period of a given task task. Denote the period with $\mathsf{T} = \{t_i\}_{i\in[\ell]}$, and for each $i \in [\ell]$, we denote the selected model (in a block on the chain) with $\mathsf{M}_i \in \mathsf{bk}_i$ and the model's corresponding step number with S_i. Now, consider a block candidate bk embedding M_{bk} trying to extend bk_n with $n \in [\ell - 1]$. Let $M = (\mathsf{M}'_j)_{j\in[k]}$ be M_{bk}'s recursively referenced model list. Each of these models is either embedded in a block or a model transaction that extends some blocks on the blockchain. Without loss of generality, we assume the first block being extended by one of these models in the period to be bk_m where $m \in [\ell - 1], m \leq n$. The **restriction** on M_{bk}'s step number S_{bk} is:

$$\sum_{j=0}^{k-1} S'_j + S_{\mathsf{bk}} \geq \sum_{i=m}^{n-1} S_i + \mathsf{smin}. \tag{4.1}$$

Intuitively, the restriction requires that a newly generated block and its referred models have no less training steps than the steps on the main blockchain. It is reasonable in the sense that we require not only the accuracy of models/blocks, but also miners donating enough computing power (training steps). Moreover, our D-PoDL scheme enables us to leverage steps in model verification. The goal is to stabilize the block generation rate, and we will discuss this later in Section 4.4.2. Finally, miners repeat the above process when the publisher network proceeds to a new task.

4.3.2 Concrete Protocols

In this section, we instantiate the chain selection rule with the longest-chain rule from [26] and the weight-based framework from [33].

Longest-chain-based D-PoDL blockchain. First, in the longest-chain-based protocol, we clarify the relation introduced above as: $\mathcal{R}(\mathsf{acc}, T_{\mathsf{acc}}) = 1$ if $\mathsf{acc} \geq T_{\mathsf{acc}}$. Hence, a model is eligible to be published as a block if $b = 1 \wedge \mathsf{acc} \geq T_{\mathsf{acc}}$. Otherwise, i.e., $\mathsf{acc} < T_{\mathsf{acc}}$, the model can be embedded in a model-transaction mtx and published to the mempool; if $b = 0$, the miner must continue training the model or resample the nonce for another initial model to be trained. By the longest-chain rule, miners of each time slot add blocks to the end of the longest blockchain they have observed and broadcast the chain to the network. Later, we will show that forks of the same length as the main chain can exist only with negligible probability by proving the robust ledger properties [26] for our longest-chain-based protocol.

Weighted-based D-PoDL blockchain. In the weight-based protocol, $\mathcal{R}(\mathsf{acc}, T_{\mathsf{acc}}) = 1$ for any pair of $(\mathsf{acc}, T_{\mathsf{acc}})$. The situation indicates that a miner can generate a block even if its deep learning model fails to surpass the target threshold. However, the number of blocks produced in a time slot can be overwhelming without proper filtering. Therefore, Kamp et al. [33] introduce a weight function to quantify the quality of blocks so that miners only choose the blockchain with the highest (accumulated) weight. Our weight function evaluates the embedded model according to $(\mathsf{acc}, T_{\mathsf{acc}})$ with the Weight algorithm. Instead of showing specific constructions, we will introduce two crucial properties for proving the security of the weight-based D-PoDL blockchain in the next section. With these properties, we show that forks with comparable weights as the main chain can only exist with negligible probability by proving the weight-based variant of robust ledger properties [33] for our weight-based protocol.

Discussion: Incentive models. The incentive model is crucial to a practical protocol. We aim to reward miners according to their useful computation, *i.e.*, training iterations. Moreover, our protocol differs from previous works in that miners can reference model-transactions to generate another model-transaction, or a block, without training models from the sketch. Models in model-transactions may have inferior accuracy. However, they are crucial in forming the distributed deep learning task solver. Hence, in order to incentivize miners to produce models, we reward not only the miners who produce selected blocks but also the model-transactions referenced by selected blocks. The rewards are given according to the model accuracy and the step number. For example, let M be the selected model, which is trained for S steps. Furthermore, let its recursively referenced models set be $M = \{\mathsf{M}_j\}_{j\in[k]}$, and each $j \in [k]$, M_j is trained for S_j steps. Hence, M's miner receives $S/(\sum S_j + S)$ fraction of the total rewards, and each M_j's miner receives $S_j/(\sum S_j + S)$ of the total rewards. Finally, we want to clarify that incentive models may affect the assumptions on honest miners' fraction, which will further affect chain growth for robust ledgers [3]. However, we assume honest miners' fractions directly. Hence, the incentive model in this section will not change our security proofs.

4.4 Security Analysis

Our security analysis focuses on the period of one single deep learning task because switching to a new task can be regarded as a mining difficulty shift in the PoW-based protocols. However, extending the result to the whole blockchain is easy if we assume the difficulty, represented by the accuracy target threshold, is stable across different tasks. To clarify, the terms "model-transaction" and "block" refer to the model embedded in the model-transaction or block.

Robust ledger properties. In this section, we focus on proving the robust ledger properties, *i.e.*, the chain growth, chain quality, and common prefix, for our concrete protocols. Recall the definitions from Chapter 2. We unify them as follows.

Definition 13 ((Weight-Based) Robust Ledger Properties) *The three aspects are defined as follows.*

- *Chain growth: For any honest miner with chain* chain *at a round, the chain growth with parameter* $\tau \in (0, 1]$ *and* $s \in \mathbb{N}$ *states that for any portion of* chain *spanning* s *consecutive rounds, the number of blocks appearing in this portion is at least* $\tau \cdot s$ *(the accumulated weights* $\mathsf{W}(\mathsf{chain}_2) \geq \mathsf{W}(\mathsf{chain}_1) + \tau s$*);*

- *Existential chain quality: For any honest miner with chain* chain *at a round, the existential chain quality with parameter* $s \in \mathbb{N}$ *states that for any portion of* chain *spanning* s *consecutive rounds, at least one honestly-generated block appears in this portion (the fraction of honest blocks' weights is at least* μ*;);*

- *Common prefix: For any two honest miners with chains* $\mathsf{chain}_1, \mathsf{chain}_2$ *at round* r_1, r_2 *respectively, where* $r_1 \leq r_2$*, the common prefix with parameter* $s \in \mathbb{N}$ *indicates that* chain_1 *should be a prefix of* chain_2 *after removing the last* s *blocks.*

4.4.1 The Training Oracle

Our first step models the combination of training and post-hash process with a training oracle. Following [33]'s approach, we assume protocol participants can make at most one query to the training oracle in each round. This assumption is reasonable because a round is the smallest unit of time of interest in our protocol and corresponds to the time for evaluating the hash function over one training iteration on one miner's computing device. For a real-world miner with the computing power of more than one device, we model it as a collection of "one-query-per-round" participants.

In each round, a miner queries the oracle $\mathcal{O}_{\mathsf{Train}}$ with $(\mathsf{M}_{\mathrm{pre}}, \mathsf{acc}_{\mathrm{pre}}, S; r)$ where $\mathsf{M}_{\mathrm{pre}}$ denotes the pre-query model, $\mathsf{acc}_{\mathrm{pre}}$ and S denotes the corresponding training accuracy and step number, and r denotes the random seed for training. The oracle $\mathcal{O}_{\mathsf{Train}}$ first verifies the queried model and returns $\perp$ if the model is invalid. Otherwise, $\mathcal{O}_{\mathsf{Train}}$ performs one training iteration with r to obtain $(\mathsf{M}_{\text{after}}, \mathsf{acc}_{\text{after}}, S{+}1)$ where $\mathsf{M}_{\text{after}}$ and $\mathsf{acc}_{\text{after}}$ denote the model and training accuracy after query, respectively. It samples a random value $h_2 \leftarrow \{0,1\}^{\lambda}$ uniformly, where λ is the security parameter that indicates the length of the hash function output. $\mathcal{O}_{\mathsf{Train}}$ returns $(\mathsf{M}_{\text{after}}, \mathsf{acc}_{\text{after}}, S{+}1, h_2)$. Moreover, regarding queries with different random seeds (r) as different queries, $\mathcal{O}_{\mathsf{Train}}$ keeps a list of performed queries and replies to former queries according to the list.

The uniqueness of our model is that $\mathcal{O}_{\mathsf{Train}}$ performs one training iteration before sampling the random value. A query is said to be successful only if the output model satisfies: $h_2 \leq T_2 \wedge \mathcal{R}(\mathsf{acc}_{\text{after}}, T_{\mathsf{acc}}) = 1$. Since h_2's distribution is defined to be uniform, we now consider the distribution of the output accuracy $\mathsf{acc}_{\text{after}}$. Note that training accuracy usually grows faster before achieving a certain value. Like in [16], we name this value difficulty threshold, denoted by D_{acc}. Our model focuses on the training process after such a threshold. The reason is that, as explained in [7], increasing the training accuracy requires stochastic/random search after this threshold. Hence, we assume that if $\mathsf{acc}_{\text{pre}} \geq D_{\mathsf{acc}}$, $\mathsf{acc}_{\text{after}}$ follows an arbitrary distribution $\mathcal{D}$ over $\{\mathsf{acc} : \mathsf{acc} \geq D_{\mathsf{acc}}\}$ such that $f_1 \triangleq \Pr[\mathsf{acc}_{\text{after}} \geq T_{\mathsf{acc}} | \mathsf{acc}_{\text{pre}} \geq D_{\mathsf{acc}}]$, *e.g.*, when $\mathcal{D}$ is uniform, $f_1 = \frac{1-T_{\mathsf{acc}}}{1-D_{\mathsf{acc}}}$. Otherwise, *i.e.*, $\mathsf{acc}_{\text{pre}} < D_{\mathsf{acc}}$, we assume $\mathsf{acc}_{\text{after}}$ increase be monotonically but unlikely to surpass T_{acc}, *i.e.*, less than ϵ, negligible of the security parameter λ. Therefore, we argue that the overall probability is $\Pr[\mathsf{acc}_{\text{after}} \geq T_{\mathsf{acc}}] \geq \frac{1}{2} f_1$ because the number of training steps before reaching the difficulty threshold is much less than the step number afterward. The training oracle goes as follows.

Training Oracle $\mathcal{O}_{\mathsf{Train}}$

Let task be a deep learning task with dataset D and accuracy target threshold T_{acc}. The oracle $\mathcal{O}_{\mathsf{Train}}$ keeps a list $\mathbb{L}$ with performed queries. **On a query $(\mathsf{M}_{\text{pre}}, \mathsf{acc}_{\text{pre}}, S; r)$ from a miner in a round:**

- If $(\mathsf{M}_{\text{pre}}, \mathsf{acc}_{\text{pre}}, S; r)$ is invalid, *i.e.*, M_{pre} has unmatched accuracy or step number, return $\perp$;
- If $(\mathsf{M}_{\text{pre}}, \mathsf{acc}_{\text{pre}}, S; r) \in \mathbb{L}$, return the reply entry $(\mathsf{M}_{\text{after}}, \mathsf{acc}_{\text{after}}, S{+}1)$ according to the list $\mathbb{L}$;
- Otherwise, run one training step $\mathsf{Train}(\mathsf{D}, T_{\mathsf{acc}}, \mathsf{M}_{\text{pre}}, r) \rightarrow (\mathsf{M}_{\text{after}}, \mathsf{acc}_{\text{after}}, S{+}1)$ and sample $h_2 \leftarrow \{0,1\}^{\lambda}$ uniformly at random.

Add $((\mathsf{M}_{\text{pre}}, \mathsf{acc}_{\text{pre}}, S; r), (\mathsf{M}_{\text{after}}, \mathsf{acc}_{\text{after}}, S{+}1, h_2))$ to $\mathbb{L}$ and return $(\mathsf{M}_{\text{after}}, \mathsf{acc}_{\text{after}}, S{+}1, h_2)$ to the miner.

We assume the distribution of $\mathsf{acc}_{\text{after}}$ following the distribution $\mathcal{D}$ over $\{\mathsf{acc} : \mathsf{acc} \geq D_{\mathsf{acc}}\}$ such that $\Pr[\mathsf{acc}_{\text{after}} \geq T_{\mathsf{acc}} | \mathsf{acc}_{\text{pre}} \geq D_{\mathsf{acc}}] = f_1$, and $\Pr[\mathsf{acc}_{\text{after}} \geq T_{\mathsf{acc}} | \mathsf{acc}_{\text{pre}} < D_{\mathsf{acc}}] = \epsilon$ where ϵ is negligible of the security parameter λ.

4.4.2 Proving Ledger Properties

Consider the situation in which a deep learning task task_i spans over time slots $\mathsf{T}_i = \{t_{i,j}\}_{j\in[\ell]}$. We omit i in the following for simplicity. The hash function in the $\mathsf{PreHash}$ algorithm guarantees that a new block is never added between two existing blocks (insertions), the same block never occurs in two different positions (copies), and a block never extends a block that will be mined in later time slots (predictions).

The longest-chain-based protocol. A miner who outputs a block in slot t_j that meets the post-hash check, target accuracy, and step restriction has to perform at least S_j training steps, which is equivalent to S_j queries to $\mathcal{O}_{\mathsf{Train}}$. As miners, honest or adversarial, are bounded by the number of queries they can make in each round, they cannot generate too many blocks in any polynomial many consecutive rounds within the period T of task_i. Simultaneously, miners cannot generate too few blocks because the probability of at least one honest miner outputting a block is lower bounded by the success rate of the oracle.

Like [26], we define typical execution for the situation in which miners generate not too many nor too few blocks in polynomial many consecutive rounds of protocol execution. First, we consider three Boolean random variables X_r, Y_r, Z_{rpq}. If at round r an honest miner obtains an output from the oracle $\mathcal{O}_{\mathsf{Train}}$ that satisfies $h_2 \leq T_2 \wedge \mathcal{R}(\mathsf{acc}_{\text{after}}, T_{\mathsf{acc}}) = 1$, then $X_r = 1$, otherwise $X_r = 0$. If at round r exactly one honest miner obtains such an output, then $Y_r = 1$, otherwise $Y_r = 0$. For the adversary, if at round r, the p-th corrupted miner's q-th query to the oracle $\mathcal{O}_{\mathsf{Train}}$ obtains such an output, then $Z_{rpq} = 1$, otherwise $Z_{rpq} = 0$. Hence, we define a variable $Z_r = \sum_p \sum_q Z_{rpq}$. For a set $\mathcal{X}$ of k consecutive rounds, we define $X(\mathcal{X}) = \sum_{r\in\mathcal{X}} X_r, Y(\mathcal{X}) = \sum_{r\in\mathcal{X}} Y_r, Z(\mathcal{X}) = \sum_{r\in\mathcal{X}} Z_r$.

Definition 14 ((ϵ, k)-typical execution) *Let $\epsilon \in (0, 1)$ and $k \in \mathbb{N}$, an execution is (ϵ, k)-typical if for any set $\mathcal{X}$ of at least k consecutive rounds* *within the period of a deep learning task, the following holds:*

- $(1-\epsilon)\mathbb{E}[X(\mathcal{X})] < X(\mathcal{X}) < (1+\epsilon)\mathbb{E}[X(\mathcal{X})], (1-\epsilon)\mathbb{E}[Y(\mathcal{X})] < Y(\mathcal{X})$;
- $Z(\mathcal{X}) < \mathbb{E}[S(\mathcal{X})] + \epsilon\mathbb{E}[X(\mathcal{X})]$.

Theorem 2 *Assume the training oracle and at most $\beta < \frac{1}{2}$ corrupted miners each round, the longest-chain-based D-PoDL blockchain protocol satisfies the robust ledger properties (Definition 3).*

Proof 3 *We first prove the following lemma.*

Lemma 3 *An execution is (ϵ, k)-typical with probability $1-e^{-\Omega\left(\epsilon^2 kp\right)}$ where p is the probability of at least one honest miner obtaining a model that satisfies $h_2 \leq T_2 \wedge \mathsf{acc} \geq T_{\mathsf{acc}}$.*

Let acc_{after} be the input accuracy to $\mathcal{O}_{\mathsf{Train}}$ and D_{acc} be the difficulty threshold, according to our oracle description, the probability of the output accuracy acc_{after} surpassing the target threshold T_{acc} in a query reply is:

$$\begin{aligned}\Pr[\mathsf{acc}_{after} \geq T_{\mathsf{acc}}] &= \Pr[\mathsf{acc}_{after} \geq T_{\mathsf{acc}} \wedge \mathsf{acc}_{pre} \geq D_{\mathsf{acc}}] \\ &\qquad + \Pr[\mathsf{acc}_{after} \geq T_{\mathsf{acc}} \wedge \mathsf{acc}_{pre} < D_{\mathsf{acc}}] \\ &\geq \Pr[\mathsf{acc}_{after} \geq T_{\mathsf{acc}} | \mathsf{acc}_{pre} \geq D_{\mathsf{acc}}] \cdot \Pr[\mathsf{acc}_{pre} \geq D_{\mathsf{acc}}] \\ &= f_1 \cdot \Pr[\mathsf{acc}_{pre} \geq D_{\mathsf{acc}}],\end{aligned}$$

For a query, we have $\Pr[\mathsf{acc}_{pre} \geq D_{\mathsf{acc}}] \geq 1/2$, as we assumed training steps before reaching D_{acc}, which is less than the step number afterward. Thus, a miner who makes at least one query to $\mathcal{O}_{\mathsf{Train}}$ in a round obtains $\mathsf{acc}_{after} \geq T_{\mathsf{acc}}$ from $\mathcal{O}_{\mathsf{Train}}$ with probability at least $\frac{1}{2} \cdot f_1$.

Let f_2 be the probability of at least one honest miner obtaining an h_2 from the training oracle $\mathcal{O}_{\mathsf{Train}}$ that satisfies $h_2 \leq T_2$ in a round. Because the output of the training algorithm is independent to the hash function, the probability of at least one honest miner obtaining a tuple $(\mathsf{M}_{after}, \mathsf{acc}_{after}, S{+}1, h_2)$ that satisfies $h_2 \leq T_2 \wedge \mathsf{acc}_{after} \geq T_{\mathsf{acc}}$ should be at least $p \geq \frac{1}{2} \cdot f_1 \cdot f_2$ (and at most $p \leq f_2$).

Next, we analyze the probability of execution being typical. Note that the training oracle $\mathcal{O}_{\mathsf{Train}}$ takes queries with different training random seeds (r) as different queries. Moreover, a hash function, modeled as a random oracle, generates such random seeds in the pre-hash PreHash *algorithm. Hence, the probability of two honest parties querying $\mathcal{O}_{\mathsf{Train}}$ with the same input in polynomial many rounds of execution is negligible of the security parameter λ. Such property enables us to condition the probability space on the event that* *no two honest parties query $\mathcal{O}_{\mathsf{Train}}$ with the same input in a polynomial many rounds of execution. In this space, the random variables X_r (and similarly Y_r and Z_{rpq}) are independent Bernoulli trials where each trail is successful with probability $\Theta(p)$ (as analyzed above). Hence, by the Chernoff bound, we prove the probability of an (ϵ, k)-typical execution is $1 - e^{-\Omega\left(\epsilon^2 kp\right)}$.*

Directly from [26], we have the following lemma that parameterizes the chain growth, existential chain quality, and common prefix in a typical execution.

Lemma 4 *In an (ϵ, k)-typical execution, the chain growth property holds for parameter $\tau = (1-\epsilon)p$ and $s \geq k$, the existential chain quality property holds for parameter $s \geq 2kp$, and the common prefix property hold for parameter $s \geq 2kp$.*

Finally, by Lemma 3, we choose $k = \Omega(\log^2 \lambda)$ so that an execution fails to be typical with negligible probability of the security parameter λ. Therefore, we prove Theorem 2 with the parameters following Lemma 4.

The weight-based protocol. One concern is that selecting models with inferior accuracy may accelerate the block generation rate because training such models requires fewer steps in each time slot. The block may not be adequately propagated to all honest miners before the next block is generated. To prevent so, we require the weight function to be appropriately bounded (with isolated-lower-bounds and upper-bounds, definitions can be found in [33]) so that the low block weight indicates low model accuracy and the low accuracy models are hard to be selected according to the weight function. Like the typical execution, we adopt model isolation (Definition 15) from [33] for the situation in which the round gap between any two models with sufficient accuracy is longer than the unknown network delay. Under the properly bounded weight functions, we further argue that our model-referencing mechanism cannot break model isolation. A miner has to perform enough training steps so that the total step number of its model, including all the referenced models, is no less than the total steps of the selected models (**Restriction**). Hence, the model-referencing mechanism offers no advantage to the miner in generating a model *faster*. Finally, we conclude the following theorem for the weight-based protocol.

Definition 15 (acc-Isolation) *Let M be the model embedded in the block mined in round r within the period of a deep learning task. M is left-isolated if M is generated by an honest miner, $\mathsf{acc}_{\mathsf{M}} \geq \mathsf{acc}$, and there is no block on the left embedding a model with accuracy higher than acc in rounds $[r - \Delta, r]$*

where Δ is the unknown network delay. M *is isolated if* M *is generated by an honest miner,* $\mathsf{acc}_{\mathsf{M}} \geq \mathsf{acc}$*, and there is no block embedding a model with accuracy higher than* acc *in rounds* $[r - \Delta, r + \Delta]$.

Theorem 3 *Assume the training oracle and at most $\beta < \frac{1}{2}$ corrupted miners each round, the weight-based D-PoDL blockchain protocol satisfies the weight-based robust ledger properties (Definition 4) if the weight function is isolated-lower-bounded and upper-bounded.*

Proof 4 *It has been proven that a secure longest-chain-based blockchain protocol can be transformed into a secure weight-based protocol as long as the weight function is properly bounded [33]. We refer to their results and argue that our model-referencing mechanism will not break the proof.*

First, for chain growth*, the restrictions on training step number in Section 4.3.1 enable honest miners to have enough time for block propagation. Therefore, honest miners will have at least one chain that accumulates the weight from all left-isolated blocks. Assuming the weight function is left-isolated-lower-bounded, the probability of this accumulated weight being inferior to the lower bound is negligible to the security parameter. Next, for* chain quality*, the chain growth property guarantees that the chain will accumulate at least all left-isolated blocks' weights. Moreover, the adversary cannot generate left-isolated blocks fast enough because it has to perform enough training steps, and the total weight is upper bounded by the weight function. Finally, since model-referencing will not change block selection, i.e., honest miners will only extend chains with sufficient weights by each round,* common prefix *preserves regardless of model-referencing.*

4.5 Implementation of D-PoDL Scheme

The average time consumption of block generation and the variance in the time reflect the stability of the protocol, which may affect users' experience. Further, this time consumption can be analyzed with the solving time of the underlying schemes. Therefore, this section shows a toy example for our D-PoDL scheme implementation. We compare the PoW scheme and the plain deep learning to our D-PoDL scheme with different sets of threshold parameters $(T_1, T_2, T_{\mathsf{acc}})$. We utilize the MNIST dataset to implement deep learning-based schemes, *i.e.*, plain deep learning and our D-PoDL, and follow the original split of 60000 images for training and 10000 images for testing. Results can be found in Table 4.2.

In the PoW implementation, we use a 256-bit hash function, *e.g.*, SHA-256, and set the difficulty to be $T = 2^{228}$, *i.e.*, to find a nonce with $\mathsf{Hash}(\mathsf{prevBK},$

Table 4.2: Experiment Results

Scheme/Algorithm	Average	Maximum	Minimum	Variance
PoW (2^{224})	433.37	1242.53	0.63	122336.11
Deep learning (0.97)	81.88	126.61	75.24	109.19
D-PoDL ($2^{240}, 2^{256}, 0.97$)	94.26	122.38	75.23	320.99
D-PoDL ($2^{244}, 2^{255}, 0.97$)	140.60	195.68	115.36	648.97

On MacBook Pro with 2.3GHz quad-core Intel Core i5 and 8GB of 2133MHz LPDDR3 onboard memory; Time consumption is presented in seconds and recorded for 20 attempts.

$\mathsf{nonce}) < T$. Hence, the expected hash iteration is 2^{28}. Next, In the plain deep learning implementation, we set the batch size to 128, the learning rate to 0.001, and the target threshold to 0.97. Most models reach this threshold in 2 epochs, each including 387 training steps. Finally, in the two D-PoDL's Solve algorithm, we implement with $(T_1, T_2, T_{\mathsf{acc}}) = (2^{240}, 2^{256}, 0.97)$ and $(2^{244}, 2^{255}, 0.97)$. For $T_2 = 2^{256}$, since the post-hash accepts all models, we use it to distinguish the impact of the pre-hash algorithm PreHash. The change in time comes from two factors: (1) Computation overhead from the hash function; (2) Training speed due to the different hyper-parameters. For the second D-PoDL implementation, the post-hash check significantly prolongs the average solving time despite the fact that we lower the pre-hash threshold T_1 to 2^{244}. The result indicates that the post-hash plays an important role in controlling the solution generation speed, which is the overall difficulty of the scheme.

Concerning variance values, we observe a big gap between deep learning-based schemes and the PoW scheme. The reason is that the stochastic gradient descent algorithm that optimizes the neural network has a more consistent convergence speed. In contrast, a well-behaving hash function in the PoW scheme should follow the uniform distribution with a high variance. However, low variance is not always preferable because the algorithm should involve enough stochasticity to prevent domination, *i.e.*, the miner with the most computing power generates all blocks. By comparing the variance value of the deep learning-based schemes, we notice that both pre-hash and post-hash algorithms involve randomness in the solving time, which can benefit the fairness among miners.

4.6 Discussion

This chapter extends the existing deep learning-based PoUW schemes by removing impractical assumptions and providing a distributed approach for provers to collaborate together solving the PoUW. Based on the proposed D-PoDL scheme, we construct blockchain protocols and show the protocol can enable its participants to achieve consensus. However, we need to point out that our proofs are also based on two crucial assumptions: (1) The parameter in the protocol, *i.e.*, $(T_1, T_2, T_{\mathsf{acc}})$, are well-chosen so that the rate of block generation prevents too many forks from being added to the main blockchain; (2) We assume honest majority so that more than half of the participants follows our protocol instruction. The former highlights the needs of taking deep consideration in deploying the protocol into practice; whereas, the latter requires us to reconsider the incentive model (like in Section 4.3.2) and give better reasons why participants will follow instruction.

Chapter 5

Consensus from Competitive PoX

The initial motivation of this chapter [51] is a concrete problem: Can blockchain-aided protocols contribute to the development and deployment of "smart grids", a novel power grid structure that enables peer-to-peer energy trading (P2PET). We answer this question with a blockchain protocol based on a novel PoW variant from an assignment problem (bid-assignment problem (BAP)) in general double auction systems. Further, we generalize the idea of the BAP-based PoW alongside the weight-based blockchain framework into what we call "competitive PoX". Finally, we give a security analysis for our blockchain protocol based on the competitive PoX.

5.1 Overview

The shift to renewable energy sources, which are less reliable than traditional options, urges a significant change in the current power grid structure, *i.e.*, moving away from the centralized uni-directional system to decentralized bi-directional smart grids. In a smart grid, individuals known as prosumers are both producers and consumers of energy. The smart-grid infrastructure enables its users (prosumers) to trade energy and exchange data, thereby promoting regional self-sustainability. A control system, *i.e.*, the P2PET system, is employed to monitor and manage user operations within the smart grid. Its primary functionality is to securely record the history of energy trading, allowing users to perform accordingly. Usually, in a decentralized environment, a public ledger is used to ensure user consensus and to prevent any manipulation of data by a single entity. As introduced in Chapter 1, it is a natural choice to utilize a blockchain as the implementation of the distributed ledger in P2PET systems.

Related works. Numerous studies investigate the application of blockchain technology in P2PET [1,30,36]. However, these studies are built atop general-purpose blockchain protocols and rely on smart contract capabilities [41], *e.g.*, as provided in Ethereum [55] and Cardano [24]. Unfortunately, this approach has its limitations, as it restricts opportunities for optimization and often leads to high maintenance fees for users when submitting smart contract-based transactions to the network [41, 50]. Additionally, most protocols employ the hash-based PoW, which involves computationally intensive tasks, and hence, being energy intensive. This requirement contradicts the objectives of enhancing energy trading and reducing overall energy consumption. Needless to say, these repeated hash evaluations have no connection with the market that may exist on the top of the system. Therefore, we aim to address these gaps.

We explore the design of a dedicated blockchain protocol for P2PET, departing from the reliance on smart contracts and conventional PoW schemes. We thoroughly redesign, from the fundamental components, the blockchain data structure itself. This allows us to leverage the operations within the P2PET system and introduce a new approach called BAP-based PoW (to be detailed later). Unlike the PoW paradigm, in which much computing (and electrical) power is wasted in order to pace the block generation as a cornerstone for ensuring a robust ledger, the BAP-based PoW scheme capitalizes the existing architecture and structure of power distribution to provide a leaner and more effective solution. In summary, our novel design integrates the underlying network mechanism that ensures the robust ledger with the market dynamics of the system.

5.1.1 Our Approach and Contributions

Now, we show a brief image of our approach. The first step is to extract settings and unique operations from the P2PET system.

Abstracted P2PET. In a P2PET system, the control system operates within a smart grid and facilitates interactions between users and potentially regional power plants within a *tight-knit* community, *e.g.*, a town. The smart grid infrastructure provides users with equipment to produce, store, and transmit energy. Each user is equipped with certified smart meters that honestly measure energy production and consumption, ensuring tamper-proof measurements. Additionally, users have Home Energy Management

Systems (HEMS) associated with their smart meters, which oversee energy management processes. Unlike smart meters, HEMS are more sophisticated and capable of computations but are not assumed to be entirely trustworthy. Multiple local smart grids can be combined to form larger systems, and this hierarchical structure can be recursively implemented.

This chapter focuses on a typical standalone P2PET system within a small community, where users and a power plant are interconnected through bi-directional power lines with a fixed physical topology. Each user has the freedom to produce or consume energy as needed. However, due to the unpredictable nature of energy production, users may encounter situations where there is a shortage of energy when they are not producing enough or an excess of energy when they are producing more than required. In such cases, the trading capability of the P2PET system becomes crucial, as it allows users to buy or sell energy to meet their needs. Furthermore, the smart grid infrastructure enables users to store energy. Therefore, users can also buy energy at low prices for storage and sell it at higher prices for profit. While the specific purpose of users is not specified, we primarily focus on the buy/sell operations within the system. In order to support these operations, recall a double-sided auction market in which users buy some product at a price by issuing a buy bid, and sell some product at a price by issuing a sell bid. We follow the same process of bidding with energy as the product.

Next, we consider the formation of trading agreements from the bids. Each agreement, later referred to as a transaction, consists of a buy bid and a sell bid that satisfy specific constraints, *e.g.*, the buy bid's price is higher than the sell bid's price. The public ledger of P2PET should record these properly formed transactions, allowing users to transmit energy and make payments accordingly. It may have been noticed that two crucial questions are not mentioned above: (1) Given a set of bids, how to generate a set of transactions; (2) How to ensure a robust ledger in such a setting. The following sections thoroughly answer these questions.

It is important to note that our result primarily focuses on the design of the blockchain protocol level rather than the intricacies of the P2PET system or smart grids. Therefore, aspects such as the dynamics of the energy trading market or issues related to the physical infrastructure (e.g., energy transmission loss) are not thoroughly investigated. However, the protocol proposed is designed to be flexible enough to accommodate such challenges.

P2PET meets blockchain: A brief description. In order to integrate the bidding operations of P2PET with blockchain protocols, we begin by refining the blockchain data structure. Specifically, we introduce a bid layer containing information such as the type (buy or sell), quantity, and unit price. In this context, a transaction is defined as the combination of a buy and a sell bid. The block is then defined as a container of bids and transactions, and the blockchain is represented as an ordered linked list of blocks.

Next, we formalize the process of generating transactions using a many-to-many assignment problem, *i.e.*, the Bid Assignment Problem (BAP). The BAP can be viewed as a special case of the Generalized Multiple-Assignment Problem (GMAP) [43]. We establish a generic framework for the BAP in which a set of bids serves as the input, and an index set of assigned bid pairs is the output based on a scoring function. Then, by instantiating the BAP's outputs with blocks and defining the scoring function's domain over the block's space, we create the instantiation of the BAP referred to as the "BAP for block generation" (bk-BAP). Ultimately, the bk-BAP will serve as the foundation for our "proof-of" scheme, the BAP-based PoW scheme.

Before introducing the formal syntax of the BAP-based PoW scheme, we introduce the concept of a bidpool, which resembles the mempool of transactions in conventional blockchain protocols. Each BAP-based PoW user (formally called prover) maintains and verifies a bidpool based on their view of the blockchain. The BAP-based PoW scheme consists of algorithms for (1) sampling a bid set from the bidpool, (2) solving the bk-BAP using the sampled bid set, and (3) evaluating the bk-BAP solution according to the public scoring function. Here, the scoring function can incorporate specific market dynamics by assigning different scores to particular transactions. As we aim to showcase the flexibility of our design and not fully explore the underlying market, we keep the scoring function as general as possible. When analyzing the computation in BAP-based PoW with respect to the general scoring function, we employ the universal sampler [9, 32], which follows arbitrary distributions when sampling blocks and their corresponding scores. This modeling approach is similar to how hash computations in PoW blockchain protocols are represented by the query to a random oracle [26].

The earlier outlined bid-related block design is a key technical innovation in our proposal. To the best of our knowledge, this has not been introduced before. Additionally, our protocol's block selection and blockchain dynamics provide another unique feature. Instead of requiring optimality in each user's block generation, *i.e.*, finding the best possible bid match combination, we encourage users to compete with each other. This competition eventually leads to a chain of blocks with the highest overall score. To maintain the blocks visible to users in the protocol (referred to as the local view), we consider a tree structure where each branch represents a valid chain of blocks. Branches are assigned scores based on the score of each block in the branch, allowing users to select the branch with the highest score. The security proof of our system is intricate and constitutes a significant technical contribution to this result. In essence, the security is demonstrated by analyzing the local tree dynamics of honest users. We prove that, given any score distribution (using the universal sampler of [9, 32]), our protocol satisfies robust ledger properties with overwhelming probability.

To sum up, the contributions are threefold: (1) we abstract the process of generating transactions from bidding with a combinatorial optimization problem that extends the GMAP [43]; (2) we design a dedicated blockchain protocol for the P2PET system that can be further extended for general double-sided auction markets; (3) we prove that our protocol fulfills the ledger properties by modeling the computation in BAP-based PoW with the modified universal sampler [9, 32].

5.1.2 Final Preparations

We finalize our overview by presenting the concrete execution setting of the blockchain protocol.

Concrete execution settings. The settings are as follows.

- Time and slots: Time is divided into discrete units called time slots, indexed by an integer $\ell \in \{1, 2, \dots\}$. We assume a globally synchronized clock $\mathcal{T}$ is equipped with a key pair $(\mathsf{sk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}})$ from the signature scheme SIG. Users can submit queries $(\sigma_{\mathcal{T}}, \ell) \leftarrow \mathcal{T}(m)$ such that $(\sigma_{\mathcal{T}}, \ell) \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}_{\mathcal{T}}, m, \ell)$ where m is the query message and ℓ is the index of the current slot;

- Synchrony: We adapt the δ-synchronous setting from [12] to our slot-based execution where δ is the known network delay. Suppose an honest user sends a message in slot t, the message is guaranteed to be received by all honest users in any slot $\ell \geq t + \delta$. Moreover, we assume the diffusion functionality from [26];

- Rushing adversary: We consider a rushing network adversary who is able to: (1) receive any message from honest users first; (2) decide for each recipient whether to inject additional messages; (3) decide the order of message delivery; (4) diffuse its (the adversary's) messages after seeing all honest messages;

- Permissionless setting with static corruption: We follow the constrained permissionless setting from [45]. In each slot, there are exactly $n \in \mathbb{N}$ users executing the protocol, and at least one is honest. Whenever an honest user joins, the protocol informs her with the parameters (n, δ). Moreover, we assume a static corruption model so that the adversary cannot corrupt honest users after they are spawned.

Remark 2 (P2PET Rationale) *In the P2PET system, users must perform energy transmission and payment activities periodically. In order to handle the periodical activities, time is divided into discrete time slots like in [35]. Each user has a local clock implemented by the certified smart meter, which may not be fully synchronized in real-life. However, the length of each time slot can be adjusted to ensure that any differences in local time between users are negligible. For the sake of convenience, we assume the presence of a globally synchronized clock. Moreover, the certified hardware used in the system allows a permissioned setting, where a fixed number of users are initialized before the protocol starts, and everyone knows who the honest users are. However, the security proof does not depend on this specific setting. Therefore, we adopt a constrained permissionless setting, which is more general and inclusive.*

5.2 Model of Blockchain-Based Double Auction

This section presents the basic system model abstracted from the underlying blockchain-based P2PET, including our redesigned data structure and the considered problem.

The first step is to define the blockchain's data structure to support operations in the P2PET system. Recall that the system enables its users to buy or sell energy with energy storage and transmit equipment. Concretely, if a user produces more energy than she needs, the surplus can be stored. Then, the user can use the energy afterward or sell it through the system network. The buy/sell operation resembles a double-sided auction market with energy as the trading resource. Hence, we borrow the term "bid", which is further categorized as "buy bid" (from buyers) or "sell bid" (sometimes called "ask", from sellers). Therefore, in order to support the bid operation, we add a bid layer to the conventional "transaction-block" structure. Moreover, as in conventional blockchain protocols, a transaction is regarded as the agreement of trading, which, in our case, is the energy transmission and payment agreement between a buyer and a seller. Thus, we define the transaction as a combination of a buy and a sell bid with some restrictions which are specified in later sections.

Next, we abstract the process of generating a set of transactions from a given set of bids, as the early outlined BAP, in which we define a general scoring function to quantify the quality of the solutions, *i.e.*, each presented set of transactions. The BAP, with respect to the scoring function, is a combinatorial optimization problem, and can be regarded as a many-to-many assignment problem, a special case of the GMAP [43], as it was already described. Note that the BAP is the underlying problem of our PoW for our blockchain protocol. We formally introduce the BAP and review the GMAP in the next sections.

5.2.1 Definitions for the Blockchain Data Structure

The redefined definitions include bids, transactions, and, for completeness, block, and chain [51]. In the following, we denote each instance as bid, tx, bk, and chain, whereas we denote users as $\mathcal{U}$. For bids or blocks, when specifying issuer (or generator) $\mathcal{U}$ and time slot $\ell \geq 1$, we denote them explicitly as $\mathsf{bid}_{\mathcal{U}}^{\ell}$ or $\mathsf{bk}_{\mathcal{U}}^{\ell}$, respectively. Moreover, each instance is associated with an identifier, which is set to be the hash of the instance's contents, *e.g.*, $\mathsf{bidID} = \mathsf{Hash}(\mathsf{bid})$ is the identifier of a given bid bid. As we mentioned in the execution model, within the definitions, we assume that the global time server $\mathcal{T}$ is secured by the signature scheme SIG. The server $\mathcal{T}(m)$ returns queries for m, from users, with $(\sigma_{\mathcal{T}}, \ell) \leftarrow \mathcal{T}(m)$ such that $\sigma_{\mathcal{T}} = \mathsf{SIG.Sign}(\mathsf{sk}_{\mathcal{T}}, m, \ell)$ and time slot ℓ.

Starting with bids, users have two options: (1) to buy a quantity of energy units for an initial price (anything lower is acceptable); (2) to sell a quantity of energy units for an initial price (anything upper is acceptable). A bid should include its generation and expiration time slots. They must be signed by their issuer and the time server. The formal definition is as follows.

Definition 16 (Bid) *A bid* bid *issued by a user* $\mathcal{U}$ *who holds a key pair* $(\mathsf{sk}, \mathsf{pk})$ *from the signature scheme* SIG *is defined as* $\mathsf{bid} \triangleq (\mathsf{bid}_{\mathsf{raw}}, \mathsf{aux}_{\mathcal{U}}, \mathsf{aux}_{\mathcal{T}})$ *and is associated with an identifier* $\mathsf{bidID} = \mathsf{Hash}(\mathsf{bid})$ *where:*

- *The bid's raw content,* $\mathsf{bid}_{\mathsf{raw}} \triangleq (\mathsf{kind} \in \{\mathsf{buy}, \mathsf{sell}\}, q, p, t_{\mathsf{Gen}}, t_{\mathsf{Exp}})$*:*
  - $\mathsf{kind} \in \{\mathsf{buy}, \mathsf{sell}\}$ *indicates the bid's kind, i.e., a buy bid or a sell bid;*
  - $q, p > 0$ *denote the bid's quantity and unit price, respectively;*
  - $t_{\mathsf{Gen}}, t_{\mathsf{Exp}}$ *denote the bid's generation and expiration time slots, respectively.*

- *Information relates to the user's signature,* $\mathsf{aux}_{\mathcal{U}} \triangleq ((\mathsf{pk}, \sigma), \mathsf{misc})$*:*
  - (pk, σ) *is the user's public key and signature, i.e.,* $\sigma = \mathsf{SIG.Sign}(\mathsf{sk}, \mathsf{bid}_{\mathsf{raw}})$*;*
  - misc *contains additional information from the user, e.g., a certificate by a trusted authority attesting the user's public key.*

- $\mathsf{aux}_{\mathcal{T}} \triangleq (\mathsf{pk}_{\mathcal{T}}, \sigma_{\mathcal{T}})$ *consists of the public key and signature from the time server* $\mathcal{T}$*, i.e.,* $\sigma_{\mathcal{T}} = \mathsf{SIG.Sign}(\mathsf{sk}_{\mathcal{T}}, (\mathsf{bid}_{\mathsf{raw}}, \mathsf{aux}_{\mathcal{U}}))$.

Moreover, we denote the bid space by $\mathbb{BID}$*.*

Next, a transaction combines a buy and a sell bid at the selected quantity and price. Intuitively, the transaction's quantity should not exceed the original bids' quantity, and the price should be in the range of the original bids' prices. However, note that we enable our protocol to handle more complex bid assignments. Concrete restrictions will be presented in Section 5.3.

Definition 17 (Transaction) *A transaction* tx *is defined as* $\mathsf{tx} \triangleq (\mathsf{bidID}_1, \mathsf{bidID}_2, q_{\mathsf{tx}}, p_{\mathsf{tx}})$ *and is associated with an identifier* $\mathsf{txID} = \mathsf{Hash}(\mathsf{tx})$ *where:*

- $\mathsf{bidID}_1, \mathsf{bidID}_2$ *are the identifiers of two bids* $\mathsf{bid}_1, \mathsf{bid}_2$*. We may use* $\mathsf{bid}_1, \mathsf{bid}_2$ *directly for simplicity;*

- $q_{\mathsf{tx}} \geq 0, p_{\mathsf{tx}} > 0$ *denote the agreed quantity and unit price of the trade.*

Moreover, we denote the transaction space by $\mathbb{TX}$*.*

Here, for completeness, we also present the definition of a block. A block embeds both a bid set and a transaction set, in which the transaction set is derived from the bid set via the BAP. The block should also include a hash link pointing to its parent block. Similar to bids, the user who generates blocks need to secure the block with the signatures from herself and the time server.

Definition 18 (Block) *A block* bk *generated by a user* $\mathcal{U}$ *with* $(\mathsf{sk}, \mathsf{pk})$ *from* SIG *is defined as* $\mathsf{bk} \triangleq (\mathsf{prevHash}, \mathsf{bk}_{\mathsf{raw}}, \mathsf{aux}_{\mathcal{U}}, \mathsf{aux}_{\mathcal{T}})$*. It is associated with an identifier* $\mathsf{bkID} = \mathsf{Hash}(\mathsf{bk})$ *and a score* S_{bk}*. In a block* bk*:*

- $\mathsf{prevHash}$ *denotes the identifier of the block's parent block, i.e., the hash of the block that* bk *intends to extend;*

- *The block's raw content,* $\mathsf{bk}_{\mathsf{raw}} \triangleq (\mathsf{BIDs}, \mathsf{TXs}, t)$*:*

  - $\mathsf{BIDs} \neq \emptyset, \mathsf{TXs}$ *denote the set of bids and transactions embedded in the block;*
  - t *denotes the block's generation time slot.*

- *Information of signatures* $\mathsf{aux}_{\mathcal{U}} \triangleq ((\mathsf{pk}, \sigma), \mathsf{misc})$ *and* $\mathsf{aux}_{\mathcal{T}} \triangleq (\mathsf{pk}_{\mathcal{T}}, \sigma_{\mathcal{T}})$ *are similar as in bids (Definition 16) where* $\sigma = \mathsf{SIG.Sign}(\mathsf{sk}, (\mathsf{prevHash}, \mathsf{bk}_{\mathsf{raw}}))$ *and* $\sigma_{\mathcal{T}} = \mathsf{SIG.Sign}(\mathsf{sk}_{\mathcal{T}}, (\mathsf{prevHash}, \mathsf{bk}_{\mathsf{raw}}, \mathsf{aux}_{\mathcal{U}}))$.

Moreover, we denote the block space by $\mathbb{BK}$.

Finally, as in conventional blockchain protocols, the chain is defined as an ordered linked list of blocks. The first block is called *genesis block* and denoted by bk^G, which contains the public keys of users and is publicly known to all users. Then, all the users, when aware of new block candidates, will discard the ones which are signed with public keys that are not in the initial list.

Definition 19 (Chain) *Let* $||$ *denotes block concatenation, i.e., if* $\mathsf{bk}^t||\mathsf{bk}^{t+1}$, *then* bk^{t+1}*'s* $\mathsf{prevHash}$ *equals to* bk^t*'s identifier. Hence, a chain* chain *is an ordered linked list of blocks defined as* $\mathsf{chain} \triangleq \mathsf{bk}^G||\mathsf{bk}^1||\mathsf{bk}^2||\cdots$ *where* bk^G *is the genesis block that contains the public keys of the users of the system.*

Note that we consider a tree structure in the protocol description (Section 5.4) and security analysis (Section 5.5). Hence, we may use "branch", *i.e.*, branch, as an interchangeable term of "chain" (Definition 30).

5.2.2 The Bid Assignment Problem (BAP)

Based on the refined data structure, we propose BAP that abstracts the process of finding an "optimal" set of combinations of bids, *i.e.*, a set of transactions. The optimality is defined based on a scoring function which should be sophisticatedly designed according to a real-life situation, *e.g.*, market dynamics in the P2PET system. We emphasize again that this chapter focuses on the protocol-level design instead of fully investigating the underlying energy trading market. Hence, we leave the scoring functions general to showcase the flexibility of our design.

Concretely, we proceed with our definitions by first showing a generic framework of the BAP with an unspecified scoring function. Next, we consider the scoring function on the transaction level and then extend it to the blocks. These scoring functions are still general, but their domains are defined on the concrete data structure, *i.e.*, transaction sets and blocks. The scoring function for transaction sets is taken as an intermediate step because a transaction set must be output before generating a block. Finally, with the scoring function for blocks, we define the BAP for block generation (bk-BAP). The purpose of bk-BAP is to argue our highest-score-based block selection rules in the protocol design.

The generic BAP. Our explanation starts from the generic version of the BAP, *i.e.*, we consider an unspecified scoring function $s : \mathbb{X} \to \mathbb{R}$ where $\mathbb{X}$ is an arbitrary space. Later, we instantiate s. The BAP takes as input a non-empty set of bids, denoted by BIDs. Here, for simplicity, we assume all bids in the set are "alive", *i.e.*, the bid was issued before and is not expired, and all bids' signatures are valid (the signature scheme's verification algorithm outputs 1, *i.e.*, $1 \leftarrow \mathsf{SIG.Verify}$). Hence, we can focus on the quantity and price constraints in the BAP. That is, we reframe the bids as $\mathsf{bid} = (\mathsf{kind}, q, p)$ in the following description. Next, according to each bid's kind, we can further divide the input set BIDs into a buy bid set and a sell bid set. Without loss generality, we write $\mathsf{BIDs} = B \cup S$ where $B \triangleq \{\mathsf{bid}_i^B = (\mathsf{buy}, q_i^B, p_i^B)\}_{i\in[m]}$ is a buy bid set of size m, and $S \triangleq \{\mathsf{bid}_j^S = (\mathsf{sell}, q_j^S, p_j^S)\}_{j\in[n]}$ is a sell bid set of size n such that $B \cap S = \emptyset$. Then, all the combinations of the bids can be given by the index set $I \triangleq \{(i,j) : \mathsf{bid}_i^B \in B \wedge \mathsf{bid}_j^S \in S\} = \{(i,j)\}_{i\in[m],j\in[n]}$. An assignment between a buy bid $\mathsf{bid}_i^B \in B$ and a sell bid $\mathsf{bid}_j^S \in S$ is a transaction in the sense of Definition 17, and it can be denoted by a tuple with respect to indices: (i, j, q_{ij}, p_{ij}) where q_{ij} and p_{ij} are the assigned quantity and price, respectively.

Next, we consider the assignment constraints. Note that, unlike matching problems, we enable many-to-many assignments, *i.e.*, a buy bid can be assigned to multiple sell bids, and multiple buy bids can be assigned to a sell bid. Hence, the total assigned quantity of a bid should not exceed the bid's original quantity, and the assigned price of a buy and a sell bid should fall in the range of their original prices. Then, for all $(i,j) \in I$, we have the following constraints.

$$\begin{aligned} & q_{ij} \geq 0,\ \sum_{j=0}^{n-1} q_{ij} \leq q_i^B,\ \sum_{i=0}^{m-1} q_{ij} \leq q_j^S; \\ & p_j^S \leq p_{ij} \leq p_i^B,\ \text{if } q_{ij} \neq 0. \end{aligned} \tag{5.1}$$

We can now clarify the scoring function's domain by defining the assignment space $\mathbb{A}$. Concerning the input bid set $\mathsf{BIDs} \neq \emptyset$, denote an assignment set with $A \triangleq \{(i, j, q_{ij}, p_{ij}) : \text{Given } \mathsf{BIDs}, \forall (i,j) \in I,\ \text{Equation 5.1 holds}\}$. Then, the space of all assignment sets with respect to BIDs can be defined as $\mathbb{A}_{\mathsf{BIDs}}$. We define the space of all assignment sets as $\mathbb{A} \overset{\Delta}{=} \{A : A \in \mathbb{A}_{\mathsf{BIDs}}\}_{\mathsf{BIDs} \in \mathbb{BID}}$ and rewrite the scoring function as $s : \mathbb{A} \to \mathbb{R}$. Finally, the formal definition of our Generic BAP is given as follows.

Definition 20 (Generic BAP) *Let* $\mathsf{BIDs} \subseteq \mathbb{BID}$ *be a non-empty set of bids, which can be divided into* $B = \{\mathsf{bid}_i^B = (\mathsf{buy}, q_i^B, p_i^B)\}_{i \in [m]}$ *and* $S = \{\mathsf{bid}_j^S = (\mathsf{sell}, q_j^S, p_j^S)\}_{j \in [n]}$*, such that* $\mathsf{BIDs} = B \cup S$ *and* $B \cap S = \emptyset$*. Let* $I = \{(i, j)\}_{i \in [m], j \in [n]}$ *be the index set. Given the scoring function* $s : \mathbb{A} \to \mathbb{R}$ *where* $\mathbb{A}$ *denotes the assignment space, the Generic BAP is to find a set* $X \overset{\Delta}{=} \{(i, j, q_{ij}, p_{ij}) : (i, j) \in I\}$ *that maximizes* $s(X)$ *and satisfies the constraints given by Equation 5.1.*

We say a solution to the BAP is "valid" if the output set X is an assignment set, *i.e.*, $X = \{(i, j, q_{ij}, p_{ij}) : \text{Given } \mathsf{BIDs}, \forall (i, j) \in I, \text{ Equation 5.1 holds}\}$. Hence, the validity of BAP solutions does not require optimality, which is convenient to define the BAP-based PoX scheme later in Section 5.3 (Definition 25).

BAP for block generation. The Generic BAP provides an intuition of the problem's input and output. However, in order to integrate the BAP into our P2PET blockchain protocol, we need to adapt the problem so that it can output blocks, and the scoring function should be able to quantify the quality of blocks. As mentioned above, we first consider a general scoring function for transaction sets. Then, here, we extend it to the scoring function for blocks s_{bk}. The latter will instantiate the scoring function in the Generic BAP (Definition 20) and complete our BAP for block generation (bk-BAP) Definition next.

We start from a function that evaluates one transaction $s_{\text{tx}} : \mathbb{TX} \to \mathbb{R}$ and an aggregation function $\mathsf{Agg}_{\text{txs}} : \mathbb{R}^* \to \mathbb{R}$. The aggregation function takes as input all evaluated values within the given transaction set and outputs the aggregated value as the score of the set. Hence, the scoring function for an arbitrary k-size transaction set $\mathsf{TXs} = \{\mathsf{tx}_i\}_{i \in [k]}$ can be written as $s_{\text{txs}}(\mathsf{TXs}) \overset{\Delta}{=} \mathsf{Agg}_{\text{txs}}(s_{\text{tx}}(\mathsf{tx}_1), \ldots, s_{\text{tx}}(\mathsf{tx}_k))$. Here, we show a toy example. Let $s_{\text{tx}}(\mathsf{tx}) = q_{\mathsf{tx}} \cdot p_{\mathsf{tx}}$ for any transaction $\mathsf{tx} = (\mathsf{bid}_1, \mathsf{bid}_2, q_{\mathsf{tx}}, p_{\mathsf{tx}}) \in \mathbb{TX}$, and let $\mathsf{Agg}_{\text{txs}}$ be the sum function. Then, given any transaction set $\mathsf{TXs} \subseteq \mathbb{TX}$, $s_{\text{txs}}(\mathsf{TXs}) = \sum_{\mathsf{tx} \in \mathsf{TXs}} q_{\mathsf{tx}} \cdot p_{\mathsf{tx}}$.

Next, recall the structure of blocks given in Definition 18. For simplicity, we reframe the blocks as $\mathsf{bk} = (\mathsf{TXs}, \mathsf{AUX})$ where $\mathsf{AUX} = (\mathsf{prevHash}, \mathsf{BIDs}, t, \mathsf{aux}_{\mathcal{U}}, \mathsf{aux}_{\mathcal{T}})$. Following the same process of defining s_{txs}, let s_{aux} be the scoring function for auxiliary information, and $\mathsf{Agg}_{\text{bk}} : \mathbb{R}^2 \to \mathbb{R}$ be a general aggregation function. The function Agg_{bk} aggregates the score of a transaction set and the score of auxiliary information within the given block. Hence, given any block $\mathsf{bk} = (\mathsf{TXs}, \mathsf{AUX}) \in \mathbb{BK}$, the scoring function for blocks is defined as $s_{\text{bk}}(\mathsf{bk}) \triangleq \mathsf{Agg}_{\text{bk}}(s_{\text{txs}}(\mathsf{TXs}), s_{\text{aux}}(\mathsf{AUX}))$. We have the following definition.

Definition 21 (Scoring Function for Blocks) *Let $s_{tx} : \mathbb{TX} \to \mathbb{R}$ be a function that maps a transaction to a real value, and let $\mathsf{Agg}_{txs} : \mathbb{R}^* \to \mathbb{R}$ be a general aggregation function. The scoring function for transaction sets is $s_{txs} : \mathbb{TX}^* \to \mathbb{R}$ such that for any k-size transaction set $\mathsf{TXs} = \{\mathsf{tx}_i\}_{i\in[k]}$, then $s_{txs}(\mathsf{TXs}) \triangleq \mathsf{Agg}_{txs}(s_{tx}(\mathsf{tx}_1), \ldots, s_{tx}(\mathsf{tx}_k))$. Let $s_{aux} : \{0,1\}^* \to \mathbb{R}$ be a function that maps auxiliary information of blocks to a real value. We use $\{0,1\}^*$ to denote the unspecified input domain. Next, let $\mathsf{Agg}_{bk} : \mathbb{R}^2 \to \mathbb{R}$ be another aggregation function that takes as input two real values. The scoring function for blocks is given by $s_{bk} : \mathbb{BK} \to \mathbb{R}$ such that for any block $\mathsf{bk} = (\mathsf{TXs}, \mathsf{AUX})$:*

$$s_{bk}(\mathsf{bk}) \triangleq \mathsf{Agg}_{bk}(s_{txs}(\mathsf{TXs}), s_{aux}(\mathsf{AUX})).$$

Finally, we refine the Generic BAP with our newly defined scoring function for blocks s_{bk} to complete the bk-BAP.

Definition 22 (bk-BAP) *Let $\mathsf{BIDs} \subseteq \mathbb{BID}$ be a non-empty set of bids, which can be divided into $B = \{\mathsf{bid}_i^B{=}(\mathsf{buy}, q_i^B, p_i^B)\}_{i\in[m]}$ and $S = \{\mathsf{bid}_j^S{=}(\mathsf{sell}, q_j^S, p_j^S)\}_{j\in[n]}$, such that $\mathsf{BIDs} = B \cup S$ and $B \cap S = \emptyset$. Let $I = \{(i,j)\}_{i\in[m],j\in[n]}$ be the index set. Given the scoring function for blocks $s_{bk} : \mathbb{BK} \to \mathbb{R}$ as in Definition 21, the bk-BAP is to find a block $\mathsf{bk} = (\mathsf{TXs}, \mathsf{AUX})$ where $\mathsf{TXs} = \{(\mathsf{bid}_i^B, \mathsf{bid}_j^S, q_{ij}, p_{ij}) : (i,j) \in I\}$ and $\mathsf{BIDs} \in \mathsf{AUX}$. The block bk should maximize $s_{bk}(\cdot)$, and the transaction set TXs should satisfy constrains given by Equation 5.1.*

Remark 3 (Extensions) *The data structure and BAPs can be extended in multiple ways depending on the real-life requirements of the P2PET system. For example: When trading energy, users may have preference targets to sell to or buy from. Hence, we can embed a target list (potentially ordered according to priority) within the bid data structure, i.e., $(\mathcal{U}, \mathsf{targets}) \in \mathsf{bid}$ where $\mathcal{U}$ is the bid's issuer, and $\mathsf{targets} \subseteq \{\mathcal{U}_i\}_{i\in[N]}$ is a subset of the all N users with whom the user willing to trade. Then, the BAPs should: (1) take into consideration the target constraints when assigning bids, i.e., for any pair of buy and sell bids, $\mathsf{bid}_1 = (\mathsf{buy}, \mathcal{U}_1, \mathsf{targets}_1)$ and $\mathsf{bid}_2 = (\mathsf{sell}, \mathcal{U}_2, \mathsf{targets}_2)$, addition to constraints in Equation 5.1, assignment $(\mathsf{bid}_1, \mathsf{bid}_2)$ should also satisfy $\mathcal{U}_1 \in \mathsf{targets}_2 \wedge \mathcal{U}_2 \in \mathsf{targets}_1$; (2) adjust scoring function so that prioritized target grant higher scores. We hope this example can demonstrate our abstraction's capability of modeling the real-life system.*

In the following two sections, we show our design of a blockchain protocol that takes the competitive PoX as its core. We argue that extensions like the example given above can be easily integrated into our design with significant changes in the (yet to be presented) security analysis.

5.3 BAP-Based PoW and Competitive PoX

This section introduces a PoW scheme based on the bk-BAP. Like conventional PoW, the BAP-based PoW involves two types of participants: provers and verifiers. Note that both types are performed by users in our protocol. The separation here only aims to clarify the algorithms, *i.e.*, the prover runs a solving algorithm to solve the bk-BAP; Whereas the verifier evaluates the solution's validity and its score.

In order to present our scheme, we consider the setting where provers maintain a pool of bids alongside their views of the blockchain. For simplicity, we assume all bids are issued with valid signatures and correct identifiers, *i.e.*, computed honestly from the hash function. The "bidpool" is similar to the mempool in other blockchain protocols, with the difference that the mempool keeps transactions. A prover needs to update her bidpool according to the bidding history in the P2PET system and the transaction history recorded by the blockchain.

Our BAP-based PoW scheme starts with each prover holding a bidpool, and then the prover samples a bid set as the input for the bk-BAP. Hence, we first (1) show an algorithm for updating the bidpool, and then (2) present the formal syntax of the BAP-based PoW scheme. Finally, we model the scheme with a universal sampler [9,32], which has the interesting property of allowing random sampling from arbitrary distribution. In our case, we rely on this property to randomly sample BAP-based PoW solutions (blocks and corresponding scores). Moreover, we argue the reason and the limitation of this modeling.

We clarify that this section focuses on algorithms and the model of BAP-based PoW. Block selection and blockchain maintenance are explained in Section 5.4, which utilizes our modeling, is presented in Section 5.5.

Starting with the bidpool. Its purpose is to keep track of a continuous view of all available bids in each time slot so that provers can sample their input bid sets for generating blocks by solving the bk-BAP. Each prover maintains her bidpool concerning two aspects: (1) The bidding and transaction history embedded in the prover's blockchain; (2) The newly issued bids in the previous slot. Therefore, this section starts with the definitions of the history. Then, by introducing the "residual" of bids, *i.e.*, the unassigned (quantity) part of bids in the history, we show the concrete approach and specify the algorithm for updating the bidpool (the yet to be introduced Algorithm 3) later.

Definition 23 (Bid History and Transaction History) *For any prover, let* $\mathsf{chain} = \mathsf{bk}^G||\mathsf{bk}^1||\dots||\mathsf{bk}^\ell$ *be the prover's blockchain, and for* $t \in [\ell]$*, let* $\mathsf{bk}^t_{\mathsf{raw}} = (\mathsf{BIDs}^t, \mathsf{tx}^t, t) \in \mathsf{bk}^t$*. The bid and transaction history with respect to* chain *is defined as* $\boldsymbol{H}^\ell_{bid} \triangleq \{\mathsf{bidID} : \mathsf{bid} \in \mathsf{BIDs}^1 \cup \dots \cup \mathsf{BIDs}^\ell\}$ *and* $\boldsymbol{H}^\ell_{tx} \triangleq \{\mathsf{txID} : \mathsf{tx} \in \mathsf{TXs}^1 \cup \dots \cup \mathsf{TXs}^\ell\}$ *where* bidID *and* txID *are the identifiers of* bid *and* tx*, respectively. For convenience, we may also use the corresponding bid or transaction for the given identifier.*

Recall that the transaction sets are generated following the constraints given in Equation 5.1, *i.e.*, the sum of assigned quantity in all transactions that involve a given bid should not surpass the bid's original quantity. Thus, in order to reduce the waste in energy trading, we enable provers to include bids with unassigned quantities into their bidpool. We name such bids as residual bids. A residual bid is a bid that exists on the blockchain, *i.e.*, in the bid history, that has unassigned quantities larger than 0.

Definition 24 (Residual Bid) *Let* $\mathsf{chain} = \mathsf{bk}^G||\mathsf{bk}^1||\dots||\mathsf{bk}^\ell$ *be a blockchain. Denote its bid and transaction history with* $\boldsymbol{H}^\ell_{bid}$ *and* $\boldsymbol{H}^\ell_{tx}$*, respectively. Given a bid* $\mathsf{bid} = (\mathsf{kind}, q, p, \mathsf{aux})$ *with identifier* $\mathsf{bidID} \in \boldsymbol{H}^\ell_{bid}$*, the residual bid of* bid *is defined as* $\mathsf{rbid} \triangleq (\mathsf{kind}, q_{\mathsf{rbid}}, p, \mathsf{aux})$ *if* $q_{\mathsf{rbid}} > 0$*, and* $\mathsf{rbid} \triangleq \perp$ *if* $q_{\mathsf{rbid}} \leq 0$*. Here:*

$$q_{\mathsf{rbid}} = q - \sum_{\{\mathsf{tx}:\mathsf{txID}\in\boldsymbol{H}^\ell_{tx}\wedge\mathsf{bidID}\in\mathsf{tx}\}} q_{\mathsf{tx}}. \tag{5.2}$$

If $\mathsf{rbid} \neq \perp$*, we set its identifier to* bidID*, i.e., the original bid's identifier. We denote the set of all residual bids with respect to a given chain as* R^{chain}*.*

Provers can assign a residual bid into a new transaction without exhausting the bid's q_{rbid}, thereby deriving a new residual bid from the original residual bid. That is, we enable provers to assign bids recursively as long as the bid is not expired. Moreover, because the residual bid's identifier is identical to its original (residual) bid, it is possible to maintain the history of each bid by tracking unique identifiers. This is also the reason we define the history using identifiers in Definition 23. Therefore, in the following, we denote the bidpool with Pool and use mappings to represent entries in the bidpool, *i.e.*, $(\mathsf{bidID}{:}\mathsf{bid} \text{ or } \mathsf{rbid}) \in \mathsf{Pool}$ where bidID is the identifier of a (residual) bid.

The algorithm for updating bidpools. Considering the process of updating the bidpool for any prover $\mathcal{P}$, at the beginning of time slot $\ell \geq 1$, let the prover hold a bidpool from the previous slot, denoted by $\mathsf{Pool}^{\ell-1}$, and a blockchain $\mathsf{chain}^{\ell-1} = \mathsf{bk}^G||\mathsf{bk}^1||\ldots||\mathsf{bk}^{\ell-1}$. We denote the set of bids in which all bids are issued in slot $\ell{-}1$ with $P^{\ell-1} \triangleq \{\mathsf{bid} : t_{\mathsf{Gen}} = \ell{-}1\}$. Here, t_{Gen} is the bid's generation time slot. Then, the algorithm that outputs the updated bidpool for slot ℓ can be written as $\mathsf{Pool}^\ell \leftarrow \mathsf{UpdatePool}(\mathsf{Pool}^{\ell-1}, \mathsf{chain}^{\ell-1}, P^{\ell-1})$. For consistency of format, we may reframe the bid set P in the form of mapping, *i.e.*, $P^{\ell-1} = \{(\mathsf{bidID}{:}\mathsf{bid}) : t_{\mathsf{Gen}} = \ell{-}1\}$. Note that $\ell = 1$ is slightly different because in the previous slot, $\mathsf{Pool}^G = \emptyset$ and $\mathsf{chain} = \mathsf{bk}^G$. That is, it contains no history, *i.e.*, $\boldsymbol{H}^G_{\text{bid}} = \emptyset, \boldsymbol{H}^G_{\text{tx}} = \emptyset$.

Next, we clarify that for any $t \in [\ell{-}1]$, the block bk^{t-1} is generated based on the bidpool in the same slot Pool^{t-1}. In other words, we need to remove the duplicated identifiers from the bidpool referring to the block's bid set. Then, by adding the set of freshly issued bids from the previous slot to the bidpool, we obtain a pool that contains all viable unique bid identifiers. Thus, we can refer to the transaction history on the blockchain to derive residual bids for these identifiers. The last step of the $\mathsf{UpdatePool}$ algorithm is to remove the outdated bids, *i.e.*, bids with expiration slots t_{Exp} earlier than the current slot. Finally, the algorithm outputs the new bidpool from the identifiers and their corresponding bids or residual bids. The procedure is formally specified in Algorithm 3.

5.3.1 Formal Syntax of BAP-based PoW

Given any time slot $\ell \geq 1$, the BAP-based PoW scheme consists of the tuple of algorithms $\mathsf{PoBA} \triangleq (\mathsf{SampleBIDs}, \mathsf{Solve}, \mathsf{Eval})$. The $\mathsf{SampleBIDs}$ algorithm samples a bid set from the prover's updated bidpool as the input of the bk-BAP; Solve outputs the corresponding blockchain of a valid solution (block) to the bk-BAP. As mentioned before, by valid, we mean the solution satisfying the constraints and signatures being valid; Finally, the evaluation algorithm Eval verifies the validity of the whole blockchain and outputs the score of the latest block according to the public scoring function. We define the BAP-based PoW correctness (Definition 27) after presenting the formal syntax in the next definition.

Definition 25 (BAP-based PoW Scheme) *Let* $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^\lambda$ *be a collision-free hash function, and let* $s_{bk} : \mathbb{BK} \to \mathbb{R}$ *be a publicly known scoring function for blocks as given in Definition 21. In time slot* $\ell \geq 1$, *for any prover* $\mathcal{P}$, *let* $\mathsf{Pool}^\ell_{\mathcal{P}}$ *be her updated bidpool from Algorithm 3, and*

Algorithm 3: The $\mathsf{UpdatePool}$ algorithm. Let $\ell \geq 1$ be the current time slot. $\mathsf{UpdatePool}$ is parameterized by a bidpool $\mathsf{Pool}^{\ell-1}$, a blockchain $\mathsf{chain}^{\ell-1}$, and a set of bids $P^{\ell-1}$.

```
function UpdatePool($\mathsf{Pool}^{\ell-1}, \mathsf{chain}^{\ell-1}, P^{\ell-1}$);
Let $\mathsf{Pool}^{\ell} = \emptyset$;
if $\ell = 1$ then
    Parse $\mathsf{Pool}^{G} = \emptyset$, $\mathsf{chain} = \mathsf{bk}^{G}$, and $P^{G} = \{(\mathsf{bidID}{:}\mathsf{bid}) : t_{\mathsf{Gen}} = G\}$;
    Return $\mathsf{Pool}^{1} = P^{G}$
else
    Parse $\mathsf{chain}^{\ell-1} = \mathsf{bk}^{G}||\mathsf{bk}^{1}||\ldots||\mathsf{bk}^{\ell-1}$ and $\mathsf{bk}_{\mathsf{raw}}^{\ell-1} = (\mathsf{BIDs}^{\ell-1}, \mathsf{tx}^{\ell-1}, \ell{-}1) \in \mathsf{bk}^{\ell-1}$;
    Parse $P^{\ell-1} = \{(\mathsf{bidID}{:}\mathsf{bid}) : t_{\mathsf{Gen}} = \ell{-}1\}$;
    // Remove duplicated identifiers from $\mathsf{Pool}^{\ell-1}$.
    for $\mathsf{bid} \in \mathsf{BIDs}^{\ell-1}$ do
        if $\mathsf{bidID} \in \mathsf{Pool}^{\ell-1}$ then
            Delete the entry from $\mathsf{Pool}^{\ell-1}$
        end
    end
    // Collect all unique bid identifiers.
    Set $\mathsf{Pool}^{\ell} = \mathsf{Pool}^{\ell-1} \cup P^{\ell-1}$;
    // Derive (residual) bids for each identifier with Equation 5.2.
    Parse the transaction history of $\mathsf{chain}$ as $\boldsymbol{H}_{\mathrm{tx}}^{\ell-1}$;
    for $\mathsf{bidID} \in \mathsf{Pool}^{\ell}$ do
        // Let $q$ be the original bid's quantity with $\mathsf{bidID}$.
        Compute $q_{\mathsf{rbid}} = q - \sum_{\{\mathsf{tx}:\mathsf{txID} \in \boldsymbol{H}_{\mathrm{tx}}^{\ell-1} \wedge \mathsf{bidID} \in \mathsf{tx}\}} q_{\mathsf{tx}}$;
        if $q_{\mathsf{rbid}} > 0$ then
            Replace the entry in $\mathsf{Pool}^{\ell}$ with $(\mathsf{bidID}{:}\mathsf{rbid})$ such that $q_{\mathsf{rbid}} \in \mathsf{rbid}$;
        end
    end
    // Remove outdated bids from $\mathsf{Pool}^{\ell}$.
    for $\mathsf{bidID} \in \mathsf{Pool}^{\ell}$ do
        // Let $t_{\mathsf{Exp}}$ be the bid's expiration slot with $\mathsf{bidID}$.
        if $t_{\mathsf{Exp}} < \ell$ then
            Delete the entry from $\mathsf{Pool}^{\ell}$
        end
    end
    Return $\mathsf{Pool}^{\ell}$
end
```

let $\mathsf{chain}_{\mathcal{P}}^{\ell-1} = \mathsf{bk}^{G}||\mathsf{bk}^{1}||\ldots||\mathsf{bk}^{\ell-1}$ *be her current blockchain. The prover performs* ($\mathsf{SampleBIDs}, \mathsf{Solve}$)*, and any verifier performs* Eval.

- $\mathsf{SampleBIDs}(\mathsf{Pool}_{\mathcal{P}}^{\ell}, \mathsf{N}; r_{\mathcal{P}}^{\ell})$ *takes as input the prover's bidpool* $\mathsf{Pool}_{\mathcal{P}}^{\ell}$*, an upper bound* N *for the size of bid sets, and a random seed* $r_{\mathcal{P}}^{\ell}$*. The randomness can be omitted, we write it explicitly for later modeling the computation in BAP-based PoW.* $\mathsf{SampleBIDs}$ *outputs a set of bids* $\mathsf{BIDs}_{\mathcal{P}}^{\ell} \subseteq \mathsf{Pool}_{\mathcal{P}}^{\ell}$ *such that* $|\mathsf{BIDs}_{\mathcal{P}}^{\ell}| \leq \mathsf{N}$*;*

- $\mathsf{Solve}(\mathsf{chain}_{\mathcal{P}}^{\ell-1}, \mathsf{BIDs}_{\mathcal{P}}^{\ell})$ *takes as input a bid set* $\mathsf{BIDs}_{\mathcal{P}}^{\ell}$ *that satisfies* $|\mathsf{BIDs}_{\mathcal{P}}^{\ell}| \leq \mathsf{N}$. *The algorithm outputs the prover's solution to the bk-BAP, i.e., a block candidate* $\mathsf{bk}_{\mathcal{P}}^{\ell} = (\mathsf{TXs}, \mathsf{AUX})$*, with the corresponding new* *blockchain* $\mathsf{chain}_{\mathcal{P}}^{\ell} \triangleq \mathsf{chain}_{\mathcal{P}}^{\ell-1}||\mathsf{bk}_{\mathcal{P}}^{\ell}$. *Here,* $\mathsf{AUX} = (\mathsf{prevHash}, \mathsf{bid}_{\mathcal{P}}^{\ell}, \ell, \mathsf{aux}_{\mathcal{P}}, \mathsf{aux}_{\mathcal{T}})$*,* $\mathsf{prevHash} = \mathsf{Hash}(\mathsf{bk}^{\ell-1})$*, and all signatures in* $(\mathsf{aux}_{\mathcal{P}}, \mathsf{aux}_{\mathcal{T}})$ *are valid;*

- $\mathsf{Eval}(\mathsf{chain}_{\mathcal{P}*}^{t}, \mathsf{N})$ *takes as input a blockchain* $\mathsf{chain}_{\mathcal{P}*}^{t}$ *from prover* $\mathcal{P}^*$ *and the size bound* N *for bid sets. If* $t \neq \ell$*,* Eval *outputs* $(0, \perp)$*. Otherwise, parse* $\mathsf{chain}_{\mathcal{P}*}^{\ell} = \mathsf{chain}^{\ell-1}||\mathsf{bk}_{\mathcal{P}*}^{\ell}$ *where* $\mathsf{bk}_{\mathcal{P}*}^{\ell}$ *is generated by* $\mathcal{P}^*$*, and assume* $(1, \cdot) \leftarrow \mathsf{Eval}(\mathsf{chain}^{\ell-1}, \mathsf{N})$*. Let* $\boldsymbol{H}_{bid}^{\ell-1}$ *and* $\boldsymbol{H}_{tx}^{\ell-1}$ *denote the bid and transaction history of* $\mathsf{chain}^{\ell-1}$*, respectively. Parse* $\mathsf{bk}_{\mathcal{P}*}^{\ell} = (\mathsf{prevHash}, (\mathsf{BIDs}^*, \mathsf{TXs}^*, t^*), \mathsf{aux}_{\mathcal{P}*}, \mathsf{aux}_{\mathcal{T}})$*, if all the following conditions hold, the algorithm outputs* $(1, s_{bk}(\mathsf{bk}_{\mathcal{P}*}^{\ell}))$*, and we say the blockchain and the new block are valid:*

  1. *For previous hash* $\mathsf{prevHash}$*: Let* $\mathsf{bk}^{\ell-1}$ *be the latest block on the verified blockchain* $\mathsf{chain}^{\ell-1}$*:* $\mathsf{prevHash} = \mathsf{Hash}(\mathsf{bk}^{\ell-1})$*;*
  2. *For bid set* bid^**: (1)* $|\mathsf{BIDs}^*| \leq \mathsf{N}$*; (2) For each* $\mathsf{bid} \in \mathsf{BIDs}^*$ *with generation and expiration time slots* $(t_{\mathsf{Gen}}, t_{\mathsf{Exp}})$*, the current slot* $\ell \in [t_{\mathsf{Gen}}, t_{\mathsf{Exp}}]$*; (3) Let* bidID *be* $\mathsf{bid} \in \mathsf{BIDs}^*$*'s identifier, if* $\mathsf{bidID} \notin \boldsymbol{H}_{bid}^{\ell-1}$*, and the signatures in* bid *are valid;*
  3. *For transaction set* TXs^**: Let* $BI = \{\mathsf{bidID} : \mathsf{bid} \in \mathsf{BIDs}^*\}$ *and* $TI = \{\mathsf{txID} : \mathsf{tx} \in \mathsf{TXs}^*\}$ *be the identifier sets given by* BIDs^* *and* TXs^**, respectively. Then: (1) For each* $\mathsf{tx} = (\mathsf{bidID}_1, \mathsf{bidID}_2, q_{\mathsf{tx}}, p_{\mathsf{tx}}) \in \mathsf{TXs}^*$*,* $\mathsf{bidID}_1, \mathsf{bidID}_2 \in \boldsymbol{H}_{bid}^{\ell-1} \cup BI$*; (2) Let* $\boldsymbol{H}_{tx,\mathcal{P}*}^{\ell} = \boldsymbol{H}_{tx}^{\ell-1} \cup TI$*. For each* $\mathsf{tx} = (\mathsf{bidID}_1, \mathsf{bidID}_2, q_{\mathsf{tx}}, p_{\mathsf{tx}}) \in \boldsymbol{H}_{tx,\mathcal{P}*}^{\ell}$*, without loss of generality, let* $\mathsf{bid}_{1\mathsf{raw}} = (\mathsf{buy}, q_1, p_1)$ *has identifier* bidID_1*, and* $\mathsf{bid}_{2\mathsf{raw}} = (\mathsf{sell}, q_2, p_2)$ *has identifier* bidID_2*, and the assigned quantity and price of* tx *satisfies:*

  $$\sum_{\mathsf{txID} \in \boldsymbol{H}_{tx,\mathcal{P}*}^{\ell} \wedge \mathsf{bidID}_1 \in \mathsf{tx}} q_{\mathsf{tx}} \leq q_1 \bigwedge \sum_{\mathsf{txID} \in \boldsymbol{H}_{tx,\mathcal{P}*}^{\ell} \wedge \mathsf{bidID}_2 \in \mathsf{tx}} q_{\mathsf{tx}} \leq q_2 \bigwedge p_2 \leq p_{\mathsf{tx}} \leq p_1.$$

  4. *For block generation slot* t^**:* $t^* = \ell$*;*
  5. *For auxiliary information* $(\mathsf{aux}_{\mathcal{P}*}, \mathsf{aux}_{\mathcal{T}})$*: The signatures are valid.*

  Otherwise, the algorithm outputs $(0, \perp)$*.*

Correctness of the BAP-based PoW scheme requires that the Eval algorithm accepts any blockchain from honestly executed $\mathsf{SampleBIDs}$ and Solve algorithms. Because the $\mathsf{SampleBIDs}$ algorithm starts with a bidpool, we need to first define validity for bidpools. Here, we consider a $\mathsf{VerifyPool}$ algorithm to check "conflicts" between an updated bidpool and the blockchain.

Definition 26 (Validity of Bidpool) *For any prover in time slot $\ell \geq 1$, let* Pool^{ℓ} *be her bidpool, and let* $\mathsf{chain}^{\ell-1} = \mathsf{bk}^{G}||\mathsf{bk}^{1}||\ldots||\mathsf{bk}^{\ell-1}$ *be her blockchain. The prover performs* $\mathsf{VerifyPool}$*.*

- $\mathsf{VerifyPool}(\mathsf{Pool}^{\ell}, \mathsf{chain}^{\ell-1})$ *takes as input the bidpool* Pool^{ℓ} *and the blockchain* $\mathsf{chain}^{\ell-1}$*. Let* $\boldsymbol{H}_{bid}^{\ell-1}$ *and* $\boldsymbol{H}_{tx}^{\ell-1}$ *denote the history of* $\mathsf{chain}^{\ell-1}$*. If the following conditions hold,* $\mathsf{VerifyPool}$ *outputs* 1*, and we say the bidpool* Pool *is valid regarding the blockchain* chain*.*

  1. *The blockchain is valid, i.e.,* $(1, \cdot) \leftarrow \mathsf{PoBA.Eval}(\mathsf{chain}^{\ell-1}, \mathsf{N})$ *where* N *is the size bound of the bid set embedded in each block on the blockchain;*
  2. *For each* $\mathsf{bidID} \in \mathsf{Pool}^{\ell}$*, let its corresponding (residual) bid have quantity* q_{rbid}*, and have generation and expiration time slots* $(t_{\mathsf{Gen}}, t_{\mathsf{Exp}})$*: (1) The current time slot* $\ell \in [t_{\mathsf{Gen}}, t_{\mathsf{Exp}}]$*; (2) If* $\mathsf{bidID} \in \boldsymbol{H}_{bid}^{\ell-1}$*,* q_{rbid} *computed from Equation 5.2 is larger than* 0*; (3) If* $\mathsf{bidID} \notin \boldsymbol{H}_{bid}^{\ell-1}$*, and the signatures in the corresponding bid are valid.*

  Otherwise, the algorithm outputs 0*.*

The reason why $\mathsf{VerifyPool}$ considers only an updated bidpool regarding the blockchain is that provers would not keep tracking old bidpools and bid sets after updating them to the new bidpool. Hence, the proposed algorithm is to verify the validity of bidpools, instead of deciding the *correctness* of the $\mathsf{UpdatePool}$ algorithm. Next, we define the correctness of the BAP-based PoW scheme.

Definition 27 (Correctness of the BAP-based PoW Scheme) *Given any prover* $\mathcal{P}$ *in time slot* $\ell \geq 1$*, let* $(\mathsf{chain}^{\ell-1}, \mathsf{Pool}^{\ell}, \mathsf{N})$ *be the prover's input tuple such that* $(1, \cdot) \leftarrow \mathsf{Eval}(\mathsf{chain}^{\ell-1}, \mathsf{N})$ *and* $1 \leftarrow \mathsf{VerifyPool}(\mathsf{Pool}^{\ell}, \mathsf{chain}^{\ell-1})$*. The BAP-based PoW scheme is correct, if* $\mathsf{BIDs} \leftarrow \mathsf{SampleBIDs}(\mathsf{Pool}^{\ell}, \mathsf{N})$ *and* $\mathsf{chain}_{\mathcal{P}}^{\ell-1}||\mathsf{bk}_{\mathcal{P}}^{\ell} \leftarrow \mathsf{Solve}(\mathsf{chain}_{\mathcal{P}}^{\ell-1}, \mathsf{BIDs})$ *are honestly executed, then:*

$$\Pr\left[(1, s_{\mathsf{bk}}(\mathsf{bk}_{\mathcal{P}}^{\ell})) \leftarrow \mathsf{Eval}(\mathsf{chain}_{\mathcal{P}}^{\ell-1}||\mathsf{bk}_{\mathcal{P}}^{\ell}, \mathsf{N})\right] = 1.$$

Usually, in general PoX schemes (Definition 1) that involve computational tasks, *e.g.*, PoW [20] and proof-of-useful-work [6,23], the difficulty is another crucial property. Intuitively, it requires provers to contribute enough computing power to generate *valid* blocks. Otherwise, adversarial provers can generate massive blocks in a short period of time, and the network cannot be finalized on a chain of blocks but with many "forks".

However, the validity of blocks in our BAP-based PoW scheme does not require optimality. Hence, it is easy to generate valid blocks for any prover.

Instead, our "difficulty" lies in the competition, *i.e.*, honest provers only select the *highest-scored* blocks (or precisely, blockchains as it will be introduced in Section 5.4.2). Relying on block scores in BAP-based PoW is meaningful because the score relates to the underlying P2PET system. Then, a higher-scored block is preferable to the system regardless of who (honest or not) generates the block. Hence, despite that we follow conventional modeling approaches [26] that differentiate the computing power of the honest provers in contrast with adversarial provers, we show in Section 5.5, that this differentiation does not change the security of our protocol (a higher score block, even if adversarial, is still useful for the overall system).

5.3.2 Competitive PoX and Scheme Modeling

Here, we abstract away from concrete problems but return to our general PoX framework in Definition 1. This approach is appropriate based on the following evidence: (1) Given a well-chosen scoring function, the BAPs (generic and bk-BAP) can be reduced to the generalized multiple-assignment problem, which has been proven to be NP-complete [43]; (2) Stochasticity arises in assignment problems due to the uncertainty in problem inputs [21], *i.e.*, in our case, BAP-based PoW requires provers to sample input bid sets from bidpools for the bk-BAP, hence, some bids may not need to be assigned. Therefore, instead of concrete implementations of the general scoring function, we consider a competition among provers whose solutions are assigned scores according to some distribution determined by the function. This abstraction enables us to model the provers' operations with queries to a scoring oracle represented by a modified universal sampler [9].

In the following, we mimic the BAP-based PoW and show the syntax of our competitive PoX framework.

Definition 28 (Competitive PoX CPoX) *Let $s : \mathbb{P} \to \mathbb{R}$ be a publicly known scoring function where $\mathbb{P}$ denotes the solution space. The tuple of algorithms* $(\mathsf{TaskGen}, \mathsf{Solve}, \mathsf{Eval})$ *in a general PoX scheme performs as follows:*

- $\mathsf{TaskGen}(1^\lambda)$ *takes as input the security parameter λ. It outputs public parameters* pp *and a task* task*;*

- $\mathsf{Solve}(\mathsf{pp}, \mathsf{task})$ *takes as input* pp*, a task* task*. It outputs a proof π for the given task* task*;*

- $\mathsf{Eval}(\mathsf{pp}, \mathsf{task}, \pi)$ *takes as input* pp*, a task* task*, and a proof π. It outputs $(1, s(\pi))$ if π is a valid according to* task*; Otherwise, it outputs $(0, \perp)$.*

Like Definition 25, we also extend the evaluation algorithm Eval to the whole blockchain. It outputs $(1, s(\pi))$ if all solution on the chain is valid and the last solution being π. Otherwise, it outputs 0.

Next, we first show the definition of the modified universal sampler [9]. Then, we detail the interaction between BAP-based PoW provers and the universal sampler.

Definition 29 (Universal Sampler [9]) *A universal sampler scheme consists of algorithms* $\mathsf{US} \triangleq (\mathsf{Setup}, \mathsf{Sample})$ *that are performed as follows.*

- $\mathsf{Setup}(1^\lambda)$ *takes as input the security parameter* λ *and outputs sampler parameters* U*;*
- $\mathsf{Sample}(U, \mathsf{d}, \beta)$ *takes as input sampler parameters* U*, the description of a program* d *with a random seed for the program to generate samples. It outputs induced samples* p_{d}*.*

We instantiate the definition above by specifying the program description with $\mathsf{d} \triangleq (\mathsf{task}, s)$ where task is the underlying task of the competitive PoX scheme and s is the general scoring function associated with the task. The universal sampler is accessible by any prover performing the competitive PoX scheme via queries in which the prover sends its own bidpool and a random seed. That is, for any prover $\mathcal{P}$, her query to the universal sampler is $\beta \triangleq (\mathsf{task}_{\mathcal{P}}, r_{\mathcal{P}})$. Here, we follow the conventional model approach that in each time slot, each honest prover can make at most $q > 0$ queries, whereas, the adversarial prover can make at most $q_{\mathcal{A}} > q$ queries. We denote the upper bound of total query number in each slot by $Q \in \mathbb{N}$. The difference in query capabilities indicates the difference in computing power between honest and adversarial provers. Moreover, we clarify that the communication between provers and the universal sampler cannot be delayed by the *network* adversary, given it is *oracle access*. This is a natural setting since the universal sampler captures the capability of provers to internally and locally select bids and generate blocks relying only on her randomness and the solving algorithm.

Then, the Sample algorithm has the single property of randomly sampling a solution $\pi \in \mathbb{P}$ such that $s(\pi) \overset{\mathcal{D}}{\leftarrow} \mathcal{S}$ where $\mathcal{D}$ and $\mathcal{S}$ denotes the score distribution and score space determined by the scoring function. With a well-chosen scoring function, blocks can be strictly ordered by the score with high probability. Concretely, the universal sampler works as follows.

Universal Sampler

In any time slot $\ell \geq 1$, setup up the universal sampler with $U \leftarrow \mathsf{US.Setup}(1^\lambda)$. Let $\mathbb{L}_{\mathcal{P}}^{\ell} = \{(\cdot,\cdot,\cdot,\cdot)\} \in U$ be the list kept by the universal sampler for any prover $\mathcal{P}$. The total size of lists is upper bounded by $Q \in \mathbb{N}$, *i.e.*, let $\mathbf{P}^\ell$ denote the set of all provers in slot ℓ, $|\bigcup_{\mathcal{P}\in\mathbf{P}^\ell} \mathbb{L}_{\mathcal{P}}^{\ell}| \leq Q$.

On a query $(\mathsf{Pool}_{\mathcal{P}}^{\ell}, r_{\mathcal{P}}^{\ell})$ **from** $\mathcal{P}$**:**

- If: there exists a tuple $(\mathsf{task}_{\mathcal{P}}^{\ell}, r_{\mathcal{P}}^{\ell}, \pi_{\mathcal{P}}^{\ell}, s(\pi_{\mathcal{P}}^{\ell})) \in \mathbb{L}_{\mathcal{P}}^{\ell}$, then, return $(\pi_{\mathcal{P}}^{\ell}, s(\pi_{\mathcal{P}}^{\ell}))$;
- Else if: $|\mathbb{L}_{\mathcal{P}}^{\ell}| > q$ when $\mathcal{P}$ is honest or $|\mathbb{L}_{\mathcal{P}}^{\ell}| > q_{\mathcal{A}}$ when $\mathcal{P}$ is adversarial, then, return $\perp$;
- Else: run and return $(\pi_{\mathcal{P}}^{\ell}, s(\pi_{\mathcal{P}}^{\ell})) \leftarrow \mathsf{US.Sample}(U, (\mathsf{task}, s), (\mathsf{task}_{\mathcal{P}}^{\ell}, r_{\mathcal{P}}^{\ell}))$, and add $(\mathsf{task}_{\mathcal{P}}^{\ell}, r_{\mathcal{P}}^{\ell}, \pi_{\mathcal{P}}^{\ell}, s(\pi_{\mathcal{P}}^{\ell}))$ to $\mathbb{L}_{\mathcal{P}}^{\ell}$.

The output distribution. The output from the universal sampler can be modeled as a continuous random variable X following a score distribution $\mathcal{D}$ on a score space $\mathcal{S} \triangleq [\mathsf{smin}, \mathsf{smax}]$. Here, $\mathcal{D}, \mathsf{smin}, \mathsf{smax}$ are determined by the general scoring function $s_{\mathrm{bk}}(\cdot)$. We denote the probability density function and the distribution function of $\mathcal{D}$ with $f(\cdot)$ and $F(\cdot)$ such that $F(x) = \Pr[X \leq x] = \int_{\mathsf{smin}}^{x} f(t)dt$.

Discussion: Block re-using attack. An issue with the BAP-based PoW (or competitive PoX) and our modeling is that it cannot prevent adversaries from re-using other users' valid block candidates, *e.g.*, the adversaries can claim others' blocks as theirs or modify the block slightly to achieve higher scores without performing enough computation. Hence, in order to tackle the block re-using attack, we consider an *exact* time barrier for diffusing block candidates in each time slot so that no honest user will diffuse its block before this time barrier. This is achievable given the globally synchronized clock $\mathcal{T}$. We clarify that this is the only place in this result where we use the strong synchronicity of the global clock, and it is a natural setting for P2PET.

5.4 Competitive PoX-Based Protocol

In any given time slot, each user obtains a list of solutions (or block candidates when considering blocks) from the competitive PoX, which is modeled by the universal sampler. Each honest user diffuses her highest-scored candidates through the network; Whereas, a rushing adversary, as mentioned in Section 5.1.2, receives all honest blocks, manages the order, and diffuses its (the adversary's) candidates accordingly. Since there is more than one block being delivered to users, and each block can only extend one blockchain, we consider a directed tree (or precisely, a directed forest due to missing blocks) structure that stores blocks locally for each user. The root of the "block-tree" is the genesis block as given in Definition 19. Users extend their block-tree in each time slot with newly received blocks and select the "best" branch on the block-tree as their blockchain. Hence, we extend the notion score for branches to support this selecting operation. Users will output the confirmed part of their blockchain when asked to report the ledger.

5.4.1 Block-Tree and Score of Branches

Now, we start with the necessary definitions for the block-tree structure and the scoring function for branches. Given a time slot $\ell \geq 1$, we first consider a master-tree mtree^{ℓ} that contains all *valid* blocks (block candidates) generated (*NOT* diffused) by users (honest or not) from the genesis slot to slot ℓ. Then, we define the master-tree $\mathsf{mtree}^{\ell} = (V, E)$ as a directed tree such that its vertex set corresponds to blocks and the edge set corresponds to the hash link between blocks. Hence, the genesis block bk^G is the root of mtree^{ℓ}. Recall the height definition from graph theory: (1) The vertex height in a directed tree is defined as the number of edges between the vertex and the root; (2) The tree height is defined as the number of edges in the longest path between a leaf vertex and the root. Hence, mtree^{ℓ} is of height ℓ. Furthermore, for vertices in mtree^{ℓ} of the same height, the corresponding blocks are generated in the same time slot. A user may only see a part of the master-tree because we assume the rushing adversary controls block diffusion. Hence, denote the block-tree of a user $\mathcal{U}$ as $\mathsf{tree}^{\ell}_{\mathcal{U}} = (V_{\mathcal{U}}, E_{\mathcal{U}})$, we have $\mathsf{tree}^{\ell}_{\mathcal{U}} \subseteq \mathsf{mtree}^{\ell}$, *i.e.*, $V_{\mathcal{U}} \subseteq V$ and $E_{\mathcal{U}} \subseteq E$. The formal definition is as follows.

Definition 30 (Master-Tree, User's Block-Tree, Branch) *Let* bk^G *be the genesis block. For any* $\ell \geq 1$ *and all* $i \in [n]$ *where* n *is the number of users participating the protocol, let* $BK^t_i \triangleq \{\mathsf{bk}^t_i\} \neq \emptyset$ *denote the set of valid blocks generated by user* $\mathcal{U}_i$ *in slot* $t \in [\ell]$*. Then,* $BK^t \triangleq \bigcup_{i\in[n]} BK^t_i$ *denotes the set of valid blocks generated in slot* t*. The master-tree of slot* ℓ *is defined as* $\mathsf{mtree}^{\ell} = (V, E)$ *such that* $V = \{\mathsf{bk}^G\} \cup \bigcup_{t\in[\ell]} BK^t$*, and* $E = \{(\mathsf{bk}^G, \mathsf{bk}^1) : \forall \mathsf{bk}^1 \in BK^1\} \cup \bigcup_{t\in[\ell-1]}\{(\mathsf{bk}^t, \mathsf{bk}^{t+1}) : \forall \mathsf{bk}^t \in BK^t, \mathsf{bk}^{t+1} \in BK^{t+1}, \mathsf{prevHash} = \mathsf{Hash}(\mathsf{bk}^t)\}$ *where* $\mathsf{prevHash}$ *is the previous hash value* *entry in block* bk^{t+1}*, i.e., blocks* $(\mathsf{bk}^t, \mathsf{bk}^{t+1})$ *are linked by the hash function* Hash *as in the BAP-based PoW scheme (Definition 25). A block-tree of user* $\mathcal{U}$ *is denoted by* $\mathsf{tree}_{\mathcal{U}}^{\ell} = (V_{\mathcal{U}}, E_{\mathcal{U}})$*, and satisfies* $\mathsf{tree}_{\mathcal{U}}^{\ell} \subseteq \mathsf{mtree}^{\ell}$*. Moreover, given a block-tree (master or user's) of slot* ℓ *as* $G^{\ell} = (V^{\ell}, E^{\ell})$*, a branch is defined* $\mathsf{branch}^{\ell} \triangleq \mathsf{bk}^G || \mathsf{bk}_{i_1}^1 || \ldots || \mathsf{bk}_{i_\ell}^{\ell}$ *where* $I \triangleq \{i_1, \ldots, i_\ell\}$ *is the index set of block generators such that* $\mathsf{branch}^{\ell} \subseteq G^{\ell}$*, i.e., for any* $t \in [\ell]$*,* $\mathsf{bk}_{i_t}^t \in V$*, and for any* $t \in [\ell - 1]$ *and* $i_t, i_{t+1} \in I$*,* $(\mathsf{bk}_{i_t}^t, \mathsf{bk}_{i_{t+1}}^{t+1}) \in E$*.*

Recall that the branch definition resembles the definition of blockchain as mentioned after Definition 19. We may also distinguish them by using branch for arbitrary branches on a given block-tree, and chain for the highest-scored branch.

Moreover, we use $\mathsf{branch}^{\ell\lceil k}$ for $k \in \mathbb{N}$ to denote the chain of blocks resulting from the removal of the k rightmost blocks of the branch branch^{ℓ}. If $k \geq \ell$, we define $\mathsf{branch}^{\ell\lceil k} = \varepsilon$, *i.e.*, the empty chain. Then, by $\mathsf{tree}^{\ell\lceil k}$, we denote the sub-tree constructing from $\mathsf{branch}^{\ell\lceil k}$ for all $\mathsf{branch} \subseteq \mathsf{tree}^{\ell}$. Note that in the case of master-tree mtree^{ℓ}, since it contains all generated blocks in the protocol, we have for any $t \in [\ell]$ and $k = \ell - t$: $\mathsf{mtree}^t = \mathsf{mtree}^{\ell\lceil k}$.

Next, in order to define the best branch with respect to scores, we extend the scoring function by taking blocks' generation slots into consideration. That is, given a branch $\mathsf{branch}^{\ell} = \mathsf{bk}^G || \mathsf{bk}^1 || \ldots || \mathsf{bk}^{\ell}$ in time slot $\ell \geq 1$, we introduce an accumulating function $\mathsf{acc}(t) \in \mathbb{R}$ for all $t \in [\ell]$. Then, the score of each slot t is defined as $\mathsf{acc}(t) \cdot s_{\mathrm{bk}}(\mathsf{bk}^t)$. Finally, the score of the branch is defined as the sum of all slot scores. Formally, we write the score of the branch $\mathsf{branch}^{\ell} = \mathsf{bk}^G || \mathsf{bk}^1 || \ldots || \mathsf{bk}^{\ell}$ as follows.

$$S_{\mathsf{branch}^{\ell}} \triangleq \sum_{t=1}^{\ell} \mathsf{acc}(t) \cdot s_{\mathrm{bk}}(\mathsf{bk}^t). \tag{5.3}$$

Note that in our protocol, blocks generated in the same time slot share the same height in the block-tree. Then, given an arbitrary block-tree, users can compute the score of all branches on the tree with Equation 5.3. Hence, for a user $\mathcal{U}$ holding a block-tree $\mathsf{tree}_{\mathcal{U}}^{\ell}$ in time slot $\ell \geq 1$, the user selects her branch, denoted by $\mathsf{chain}_{\mathcal{U}}^{\ell} \subseteq \mathsf{tree}_{\mathcal{U}}^{\ell}$, such that:

$$S_{\mathsf{chain}_{\mathcal{U}}^{\ell}} = \max_{\mathsf{branch} \subseteq \mathsf{tree}_{\mathcal{U}}^{\ell}} S_{\mathsf{branch}}. \tag{5.4}$$

As we will show in Section 5.5, the accumulating function parameterizes the possibility of our protocol achieving consensus.

5.4.2 Protocol Description

Finally, we can present the full description of our protocol. In the following, we show the workflow of an honest user in an arbitrary time slot where she maintains her bidpool, generates a block, extends her block-tree, and reports her confirmed blockchain to the system.

Let $\mathcal{U}$ be an honest user among the $n \in \mathbb{N}$ users performing the protocol in time slot $\ell \geq 1$. Denote the user's view of the block-tree at the end of slot $\ell-1$ with $\mathsf{tree}^{\ell-1}$. Here, we do not specify them to $\mathcal{U}$ because our permissionless setting cannot guarantee the user to participate in slot $\ell-1$. However, we require that for any branch $\mathsf{branch} \subseteq \mathsf{tree}^{\ell-1}$, $(1, \cdot) \leftarrow \mathsf{CPoX.Eval}(\mathsf{branch})$ as given in Definition 28. Then, following Equation 5.4, the user selects her branch, which is denoted by $\mathsf{chain}_{\mathcal{U}}^{\ell-1}$.

Let the input for $\mathcal{U}$'s execution in slot ℓ be $(\mathsf{task}^{\ell-1}, \mathsf{chain}_{\mathcal{U}}^{\ell-1})$. By performing the competitive PoX scheme as a prover, the user obtains a set of valid solutions (embedded in block candidates), *i.e.*, $\mathcal{U}$ generates candidates with $\pi_{\mathcal{U}}^{\ell} \leftarrow \mathsf{Solve}(\mathsf{task}_{\mathcal{U}}^{\ell-1})$. Denote the set of candidates that embed solutions as $BK_{\mathcal{U}} \triangleq \{\mathsf{bk}_{i,\mathcal{U}}^{\ell}\}_{i \in \mathbb{N}}$. Our protocol requires honest users only diffuse their highest-scored block candidate and the corresponding blockchain, *i.e.*, $\mathcal{U}$ diffuses the block $\mathsf{bk}_{\mathcal{U}}^{\ell}$ that satisfies $s(\pi_{\mathcal{U}}^{\ell}) = \max_{\mathsf{bk} \in BK_{\mathcal{U}}} \mathsf{bk}$, and the corresponding blockchain $\mathsf{chain}_{\mathcal{U}}^{\ell-1}$.

In addition to chain diffusion, users also receive blockchains from others. Here, we consider the process of the honest $\mathcal{U}$ updating her local block-tree $\mathsf{tree}^{\ell-1} \triangleq (V^{\ell-1}, E^{\ell-1})$ when receiving a blockchain (including her own) $\mathsf{chain}_{\mathcal{U}^*}^{t'}$. We use $\mathcal{U}^*$ for unspecified users and t' for the slot index to be verified. The user $\mathcal{U}$ first performs as the verifier in the competitive PoX with $(b, \cdot) \leftarrow \mathsf{CPoX.Eval}(\mathsf{chain}_{\mathcal{U}^*}^{t'})$. If $b = 0$, $\mathcal{U}$ discards the blockchain. Otherwise, she compares the incoming blockchain with her local block-tree and adds missing blocks with hash links to the tree. Concretely, when $b = 1$, we have $t' = \ell$. Then, we can rewrite the incoming blockchain with $\mathsf{chain}_{\mathcal{U}^*}^{\ell} = \mathsf{bk}^G || \mathsf{bk}_{\mathcal{U}^*}^1 || \ldots || \mathsf{bk}_{\mathcal{U}^*}^{\ell-1} || \mathsf{bk}_{\mathcal{U}^*}^{\ell}$. Note that users accept *blockchains*. Hence, we only need to consider a sub-chain $\mathsf{bk}_{\mathcal{U}^*}^{t} || \ldots || \mathsf{bk}_{\mathcal{U}^*}^{\ell} \subseteq \mathsf{chain}_{\mathcal{U}^*}^{\ell}$ such that $\mathsf{bk}_{\mathcal{U}^*}^{t-1} \in \mathsf{tree}^{\ell-1}$ and $\mathsf{bk}_{\mathcal{U}^*}^{t} \notin \mathsf{tree}^{\ell-1}$. Therefore, denote the updated block-tree as $\mathsf{tree}_{\mathcal{U}}^{\ell} = (V_{\mathcal{U}}^{\ell}, E_{\mathcal{U}}^{\ell})$, we have $V_{\mathcal{U}}^{\ell} = V^{\ell-1} \cup \{\mathsf{bk}_{\mathcal{U}^*}^{t}, \ldots, \mathsf{bk}_{\mathcal{U}^*}^{\ell}\}$ and $E_{\mathcal{U}}^{\ell} = E^{\ell-1} \cup \{(\mathsf{bk}_{\mathcal{U}^*}^{t-1}, \mathsf{bk}_{\mathcal{U}^*}^{t}), \ldots (\mathsf{bk}_{\mathcal{U}^*}^{\ell-1}, \mathsf{bk}_{\mathcal{U}^*}^{\ell})\}$. We summarize the update process of block-trees with an algorithm: $\mathsf{tree}' \leftarrow \mathsf{UpdateTree}(\mathsf{tree}, \mathsf{chain}_{\mathcal{U}^*}^{t'})$. Note that we use tree and tree' instead of specifying slot indices ($\ell-1$ and ℓ) because users may receive more than one blockchain candidate within one slot. The algorithm is specified in Algorithm 4.

Finally, the user is responsible for reporting her confirmed blockchain to the protocol with respect to a parameter k (to be estimated in Section 5.5),

Algorithm 4: The $\mathsf{UpdateTree}$ algorithm. Let $\ell \geq 1$ be the current time slot. $\mathsf{UpdateTree}$ is parameterized by a block-tree tree and a blockchain candidate $\mathsf{chain}_{\mathcal{U}*}^{t'}$.

```
**function** $\mathsf{UpdateTree}(\mathsf{tree}, \mathsf{chain}_{\mathcal{U}*}^{t'})$;
Parse $\mathsf{tree} = \{V, E\}$;
// Verify each blockchain candidate.
Run $(b, \cdot) \leftarrow \mathsf{CPoX.Eval}(\mathsf{chain}_{\mathcal{U}*}^{t'})$;
**if** $b = 1$ **then**
    // $t' = \ell$.
    Parse $\mathsf{chain}_{\mathcal{U}*}^{t'} = \mathsf{bk}^G || \mathsf{bk}_{\mathcal{U}*}^1 || \ldots || \mathsf{bk}_{\mathcal{U}*}^{\ell-1} || \mathsf{bk}_{\mathcal{U}*}^{\ell}$;
    // Find the first block not in tree.
    **for** $\mathsf{bk}_{\mathcal{U}*}^t \in \mathsf{chain}_{\mathcal{U}*}^{t'}$ **do**
        **if** $\mathsf{bk}_{\mathcal{U}*}^{t-1} \in \mathsf{tree}$ *and* $\mathsf{bk}_{\mathcal{U}*}^t \notin \mathsf{tree}$ **then**
            Set $V' = V \cup \{\mathsf{bk}_{\mathcal{U}*}^t, \ldots, \mathsf{bk}_{\mathcal{U}*}^{\ell}\}$;
            Set $E' = E \cup \{(\mathsf{bk}_{\mathcal{U}*}^{t-1}, \mathsf{bk}_{\mathcal{U}*}^t), \ldots (\mathsf{bk}_{\mathcal{U}*}^{\ell-1}, \mathsf{bk}_{\mathcal{U}*}^{\ell})\}$;
            **Return** $\mathsf{tree}^{\ell} = (V', E')$
        **end**
    **end**
**end**
```

i.e., given the user's updated block-tree $\mathsf{tree}_{\mathcal{U}}^{\ell}$, the user outputs $\mathsf{chain}_{\mathcal{U}}^{\ell}$ according to Equation 5.4 such that $\mathsf{chain}_{\mathcal{U}}^{\ell\lceil k}$ is the confirmed part of the blockchain.

Our Competitive PoX-Based Blockchain Protocol $\Pi^{n,\delta,k,s_{\mathrm{bk}},\mathsf{acc}}$

Let $\mathcal{U}$ be an honest user among the n users executing the protocol $\Pi^{n,\delta,k,s_{\mathrm{bk}},\mathsf{acc}}$ in time slot $\ell \geq 1$. Here, δ is the known network delay, k is a parameter for blockchain confirmation, $s_{\mathrm{bk}}(\cdot)$ is the general scoring function for blocks, and $\mathsf{acc}(\cdot)$ is the accumulating parameter function for branch scores. Given the $\mathsf{UpdateTree}$ algorithm and the competitive PoX scheme CPoX, the user $\mathcal{U}$ takes as input $\mathsf{tree}^{\ell-1}$ from the previous slot $\ell{-}1$.

- **Block Generation:** $\mathcal{U}$ generates block candidates with the competitive PoX scheme, *i.e.*, $\mathsf{chain}_{\mathcal{U}}^{\ell-1} || \mathsf{bk}_{\mathcal{U}}^{\ell} \leftarrow \mathsf{CPoX.Solve}(\mathsf{chain}_{\mathcal{U}}^{\ell-1})$;

- **Block-Tree Update:** $\mathcal{U}$ initializes an intermediate variable $\mathsf{tree} \triangleq \mathsf{tree}^{\ell-1}$. Whenever $\mathcal{U}$ receives a blockchain candidate $\mathsf{chain}_{\mathcal{U}*}^{t'}$, she runs $\mathsf{tree}' \leftarrow \mathsf{UpdateTree}(\mathsf{tree}, \mathsf{chain}_{\mathcal{U}*}^{t'})$ and updates her temporary variable with $\mathsf{tree} \leftarrow \mathsf{tree}'$. Finally, the user obtains $\mathsf{tree}_{\mathcal{U}}^{\ell}$ after updating her local block-tree with all blockchain candidates received in slot ℓ;

- **Ledger Reporting:** Upon queried by the protocol Π, $\mathcal{U}$ outputs her blockchain $\mathsf{chain}_{\mathcal{U}}^{\ell} \subseteq \mathsf{tree}_{\mathcal{U}}^{\ell}$ that satisfies Equation 5.4, and regard $\mathsf{chain}_{\mathcal{U}}^{\ell\lceil k}$ as the confirmed part of the blockchain.

Discussion: Incentive model. The security proofs of this result are based on practical *assumptions* in the sense of implementation. However, we do *NOT* analyze the reason behind these assumptions based on rational analysis as shown in [3,4,25]. Given the richness of the area, we only discuss the intuition of the incentive model. Like conventional blockchain protocols, there are two layers of incentive: inherent (*e.g.*, transaction fee) and explicit (*e.g.*, block reward). The inherent incentive in our protocol derives from where users can tweak transactions to benefit themselves, *e.g.*, assigning higher buy price for their sell bids or lower sell price for their buy bids. However, prioritizing their own bids in the solving algorithm will potentially sacrifice the block scores. Hence, the blocks may fail to be selected in the confirmed blockchain. The trade-off between this inherent reward and the scarification of block scores requires a case-by-case analysis with respect to concrete scoring functions.

However, problems arise when considering explicit incentives, *i.e.*, rewards to block generators. We argue that a well-chosen scoring function in BAP-based PoW prevents adversarial users from attacking the underlying P2PET and from disturbing consensus in the network (as shown in Section 5.5). However, the computation in BAP-based PoW is unfair, *i.e.*, adversaries can dominate the block generation without dominating computing power. The explicit incentive intensifies unfairness.

5.5 Security Analysis

This section proves the security of our protocol $\Pi^{n,\delta,k}$ with respect to ledger properties, *i.e.*, persistence and liveness [26]. We first provide the definition of persistence by adopting the slightly refined version from [23]. For liveness, existing definitions require that if an honest user receives a *transaction*, then, the transaction will eventually be output by all honest users in their ledger, *i.e.*, blockchain. However, as shown in previous sections, transactions are not diffused solely in our protocol but are released within blocks. Moreover, each user maintains a local block-tree to keep tracking *blocks* of the protocol. Hence, we define block-liveness instead of the original liveness to fit our design.

Definition 31 (Persistence and Block-Liveness) *Denote blockchain as* chain *and block-tree as* tree.

- *Persistence: For any two honest users with blockchains* $\mathsf{chain}_1^{\ell_1}, \mathsf{chain}_2^{\ell_2}$ *at time slot* $\ell_1, \ell_2 \geq 1$*, respectively. Without loss of generality, let* $\ell_1 \leq \ell_2$*. Persistence with parameter* $k \in \mathbb{N}$ *indicates that* $\mathsf{chain}_1^{\ell_1}$*, should be a prefix of* $\mathsf{chain}_2^{\ell_2}$ *after removing the rightmost* k *blocks;*
- *Block-liveness: For any honest user with* chain^{ℓ} *in time slot* $\ell \geq 1$*, block-liveness states that for any* $t \in [\ell]$*,* chain^t *is extended by at exact one block.*

Remark 4 (Relaxation in Liveness) *As discussed above, we cannot define liveness for transactions in our protocol due to the change in the data structure. There are two options: one is to define liveness for blocks as in Definition 31, which can be considered as a relaxation of the original liveness property, and in fact, our protocol satisfies block-liveness by design (Theorem 5). Whereas, the other option is to define liveness for bids, which can be meaningless in real life. This is because our main purpose is to require honest users to find good assignments (higher-scored blocks) according to the scoring function derived from the underlying P2PET system instead of enforcing them to include every bid.*

5.5.1 Persistence

In order to prove persistence, we first make an additional assumption on the adversary. Recall the rushing adversary in our execution model who can learn all block (blockchain) candidates generated, *i.e.*, the adversary holds the master-tree of the protocol. We assume that this adversary sends the highest-scored *blockchain* of each time slot to at least one honest user.

Assumption 3 *Let* mtree^{ℓ} *be the master-tree in time slot* ℓ*, and let* $\mathcal{A}$ *be the rushing adversary who holds* mtree^{ℓ}*. Let* $\mathsf{chain}^{\ell} \subseteq \mathsf{mtree}^{\ell}$ *be the highest-scored branch that satisfies:*

$$S_{\mathsf{chain}^{\ell}} = \max_{\mathsf{branch} \subseteq \mathsf{mtree}^{\ell}} S_{\mathsf{branch}}. \tag{5.5}$$

We assume that at least one honest user among the n *users who participate in the protocol receives* chain^{ℓ} *by the end of slot* ℓ*.*

Next, we consider honest users' local block-tree dynamics. It takes two steps to achieve persistence: (1) The highest-scored branch of each time slot should be eventually known to all honest users; (2) Highest-scored branches of different time slots should have a long enough common prefix.

Disclosing highest-scored branches. If a block or branch is known to all honest users, we say it is disclosed, *i.e.*, given a block bk (or a branch branch), for any honest user with block-tree tree, it holds $\mathsf{bk} \in \mathsf{tree}$ (or $\mathsf{branch} \subseteq \mathsf{tree}$). Remark that given the δ-bounded communication network, a block candidate generated by an honest user is always disclosed after δ time slots. Generally, we define d-disclosure for blocks and branches.

Definition 32 (d-Disclosure) *Let bk^t be a valid block generated in time slot $t \geq 1$, the block is d-disclosed if for any honest user with block-tree tree^ℓ in slot ℓ such that $\ell \geq t + d$, then, $\mathsf{bk}^t \in \mathsf{tree}^\ell$. We say a branch is d-closed if the rightmost block on the branch is d-disclosed.*

The following lemma indicates that if a branch is selected as the highest-scored branch by any user, the branch will eventually be disclosed.

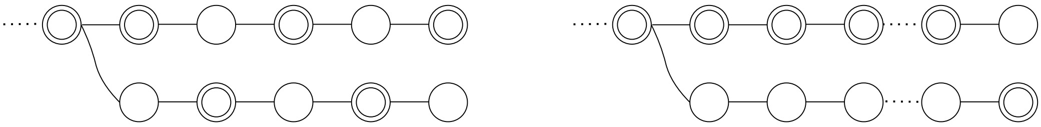
Lemma 5 *Let mtree^t be the master-tree in time slot $t \geq 1$. Assuming the network is δ-synchronous, if a branch $\mathsf{branch}^t \subseteq \mathsf{mtree}^t$ is selected as the highest-scored branch according to Equation 5.5, the branch is at most $(\delta+1)$-disclosed.*

Proof 5 *Denote the branch with $\mathsf{branch}^t = \mathsf{bk}^G || \ldots || \mathsf{bk}^t$, we first consider the situation where bk^t is generated by an honest user, then, it is δ-disclosed by δ-bounded network setting. Hence, branch is also δ-disclosed. Otherwise, the block is first received by the rushing adversary $\mathcal{A}$ in slot t. Because $\mathsf{branch}^t \subseteq \mathsf{mtree}^t$ is the highest-scored branch according to Equation 5.5, by Assumption 3, an honest user will receive the branch in slot t. Denote the user's local block-tree with tree^t. Since tree^t is a sub-graph of mtree^t, branch is the highest-scored branch in tree^t. Hence, the honest user will generate a block atop branch in the following slot $t+1$, which is also δ-disclosed. Therefore, the branch is at most $\delta + 1$-disclosed.*

Directly from Lemma 5, we have the following proposition for all disclosed highest-scored branches in the master-tree.

Proposition 1 *Assuming the network is δ-slot synchronous, let mtree^t be the master-tree in time slot t in which branch^t is selected as the highest-scored branch according to Equation 5.5. All blocks on branches in $\{\mathsf{branch}^t\}_{t \in [\ell-(\delta+1)]}$ is disclosed for any $\ell > \delta + 1$.*

Common prefix among selected branches. Proposition 1 only indicates that the highest-scored branch of each time slot will eventually be known to all honest users. However, even in the master-tree (*i.e.*, everything is known), given two conjunctive slots, the highest-scored branches may be different from each other, *e.g.*, a high-but-not-highest-scored branch gets extended by an extremely high-scored block so that the new branch is selected in the next time slot. Such a substitution causes the blockchain to be unstable and prevents honest users from agreeing on the same *chain.* We consider two situations: (In the illustration, the circle denotes the blocks on the branches, and the double circle denotes the branch being selected as the highest-scored one): (1) If the change of branch selection happens frequently, the block history cannot be settled (Figure 5.1a); (2) If the selected branches of different slots have too many distinct blocks, the block history can get reset (Figure 5.1b). In either case, invalid bids and transactions in the unconfirmed blocks can disturb the underlying P2PET market.



(a) The selected chain swings over conjunctive time slots and the blocks during these slots cannot settle.

(b) A branch substitutes the selected chain after it gets selected for multiple slots so that the history gets reset.

Figure 5.1: Intuition of Chain Instability: In the successful attack, the adversary forces the nodes to keep changing the chain between two cases.

In order to tackle this problem, we first define divergence between branches and branch viability. The definitions originate from [35], we refine them with respect to the score of branches. Moreover, since we focus on the highest-scored branches in each time slot, and they are disclosed as shown in Proposition 1, we will consider the master-tree in the following analysis for simplicity.

Definition 33 (Divergence and Viability) *Let* mtree^{ℓ} *be the master-tree of time slot* $\ell \geq 1$*. For any* $t \in [\ell]$*, given any two branches* $\mathsf{branch}_1^t, \mathsf{branch}_2^t \subseteq \mathsf{mtree}^t$*, the divergence of* branch_1^t *and* branch_2^t *is given by:*

$$\mathrm{div}(\mathsf{branch}_1^t, \mathsf{branch}_2^t) = |S_{\mathsf{branch}_1^t} - S_{\mathsf{branch}_2^t}|.$$

Moreover, let chain^t *be the highest-scored branch selected from* mtree^t *according to Equation 5.5. A branch* $\mathsf{branch}_*^t \neq \mathsf{chain}^t$ *is viable, if* $\mathrm{div}(\mathsf{branch}_*^t, \mathsf{chain}^t) \leq \mathsf{smax}{-}\mathsf{smin}$ *where* $[\mathsf{smin}, \mathsf{smax}]$ *is the range of block scores given by* $s_{bk}(\cdot)$*.*

A branch in slot t is viable if the divergence between the branch and the highest-scored branch can be covered by a single block. That is, the viable branch could be extended with a block with higher score and substitute the selected chain. We formally define this situation as chain substitution with a parameter $\tau \geq 1$. For convenience, we first introduce the injection operation for branches. Let $\mathsf{branch}_1 = \mathsf{bk}^G || \ldots || \mathsf{bk}^t || \mathsf{bk}_1^{t+1} \ldots || \mathsf{bk}_1^{\ell_1}$ be a branch of slot $\ell_1 \geq 1$, and let $\mathsf{branch}_2 = \mathsf{bk}^G || \ldots || \mathsf{bk}^t || \mathsf{bk}_2^{t+1} \ldots || \mathsf{bk}_2^{\ell_2}$ be a branch of slot $\ell_2 \geq 2$ where $t \in [\min(\ell_1, \ell_2)]$. Then, the intersection of branch_1 and branch_2 is denoted by $\mathsf{branch}_1 \cap \mathsf{branch}_2 = \mathsf{bk}^G || \ldots || \mathsf{bk}^t$.

Definition 34 (τ-Chain Substitution) *Let mtree^ℓ be the master-tree of time slot $\ell{>}\tau$. For any $t \in [\ell{-}\tau]$, let chain^i and chain^{i+1} be the highest-scored branches in slot i and $i{+}1$ where $i \in [t{-}1, t{-}2{+}\tau]$ such that $\mathsf{chain}^{i+1} = \mathsf{chain}^i || \mathsf{bk}_\mathsf{c}^{i+1}$. Let $\mathsf{bk}_\mathsf{c}^{t+\tau}$ be highest-scored block that extends $\mathsf{chain}^{t-1+\tau}$ in slot $t{+}\tau$. Denote the new branches $\mathsf{branch}_\mathsf{c} \triangleq \mathsf{chain}^{t-1+\tau} || \mathsf{bk}_\mathsf{c}^{t+\tau}$. A τ-chain substitution occurs if there exists a branch such that $\mathsf{branch}^{t-1+\tau} \cap \mathsf{chain}^{t-1+\tau} = \mathsf{chain}^{t-1}$, and is extended by block $\mathsf{bk}_\mathsf{b}^{t+\tau}$, denote the new branch $\mathsf{branch}_\mathsf{b} \triangleq \mathsf{branch}^{t-1+\tau} || \mathsf{bk}_\mathsf{b}^{t+\tau}$, such that:*

$$S_{\mathsf{branch}_\mathsf{b}} = \max_{\mathsf{branch}^{t+\tau} \subseteq \mathsf{mtree}^{t+\tau}} S_{\mathsf{branch}^{t+\tau}}.$$

A chain can only be substituted by a branch if the branch is viable, and the blocks extending them satisfies $s_{\mathrm{bk}}(\mathsf{bk}_\mathsf{b}^{t+\tau}) - s_{\mathrm{bk}}(\mathsf{bk}_\mathsf{c}^{t+\tau}) \geq S_{\mathsf{chain}^{t-1+\tau}} - S_{\mathsf{branch}^{t-1+\tau}}$, *i.e.*, $S_{\mathsf{branch}_\mathsf{b}} \geq S_{\mathsf{branch}_\mathsf{c}}$. The parameter $\tau \geq 1$ indicates that a chain is substituted after being selected for τ conjunctive slots. Assuming *arbitrary* distribution $\mathcal{D}$ for our scoring function $s_{\mathrm{bk}}(\cdot)$, the following lemma shows a loose upper bound for the probability of τ-chain substitution.

Lemma 6 *Let mtree^ℓ be the master-tree of time slot $\ell{>}\tau$. For any $t \in [\ell{-}\tau]$, given any score distribution $\mathcal{D}$, there exists an accumulating function $\mathsf{acc}(\cdot)$ (Equation 5.3) such that the probability of τ-chain substitution occurring is $\mathsf{O}(c^{-\tau})$ where c is a constant value given by $\mathsf{acc}(\cdot)$.*

Proof 6 *We start from the easy case where $\tau = 1$. By Definition 34, $\mathsf{chain}^t = \mathsf{chain}^{t-1} || \mathsf{bk}_\mathsf{c}^t$ and $\mathsf{branch}_\mathsf{c} = \mathsf{chain}^t || \mathsf{bk}_\mathsf{c}^{t+1}$. Consider a viable branch that satisfies $\mathsf{branch}^t \cap \mathsf{chain}^t = \mathsf{chain}^{t-1}$ and is extended by a block $\mathsf{bk}_\mathsf{b}^{t+1}$ such that $\mathsf{branch}_\mathsf{b} = \mathsf{branch}^t || \mathsf{bk}_\mathsf{b}^{t+1}$ substitutes $\mathsf{branch}_\mathsf{c}$. Then, by Equation 5.3, we have:*

$$\begin{aligned} s_{bk}(\mathsf{bk}_\mathsf{c}^t) &\geq s_{bk}(\mathsf{bk}_\mathsf{b}^t), \\ \frac{\mathsf{acc}(t)}{\mathsf{acc}(t{+}1)} s_{bk}(\mathsf{bk}_\mathsf{c}^t) + \cdot s_{bk}(\mathsf{bk}_\mathsf{c}^{t+1}) &\leq \frac{\mathsf{acc}(t)}{\mathsf{acc}(t{+}1)} s_{bk}(\mathsf{bk}_\mathsf{b}^t) + s_{bk}(\mathsf{bk}_\mathsf{b}^{t+1}). \end{aligned} \tag{5.6}$$

For simplicity, we omit the subscript in the scoring function with $s(\cdot)$ as we only consider block scores. We also rewrite $\frac{\mathsf{acc}(t)}{\mathsf{acc}(t+1)}$ with $c(0,1)$. Then, the *probability of* $\mathsf{branch_c}$ *substituted by* $\mathsf{branch_b}$*, denoted by* $\Pr[\tau = 1, \mathsf{branch_b}]$*, equals to joint probability of events in Equation 5.6. That is, we need to estimate the following.*

$$\Pr[s(\mathsf{bk}^t_\mathsf{c}) - s(\mathsf{bk}^t_\mathsf{b}) \geq 0 \wedge c(0,1) \cdot \big(s(\mathsf{bk}^t_\mathsf{c}) - s(\mathsf{bk}^t_\mathsf{b})\big) + \big(s(\mathsf{bk}^{t+1}_\mathsf{c}) - s(\mathsf{bk}^{t+1}_\mathsf{b})\big) \leq 0] \tag{5.7}$$

Now, we denote the random variables representing the scores of the tuple $(\mathsf{bk}^t_\mathsf{c}, \mathsf{bk}^t_\mathsf{b}, \mathsf{bk}^{t+1}_\mathsf{c}, \mathsf{bk}^{t+1}_\mathsf{b})$ *with* $(X^t_\mathsf{c}, X^t_\mathsf{b}, X^{t+1}_\mathsf{c}, X^{t+1}_\mathsf{b})$*. Furthermore, we use two random variables* Y^t *and* Y^{t+1} *to represent the subtraction of scores. That is,*

$$\begin{aligned} Y^t &= X^t_\mathsf{c} - X^t_\mathsf{b} = s(\mathsf{bk}^t_\mathsf{c}) - s(\mathsf{bk}^t_\mathsf{b}), \\ Y^{t+1} &= X^{t+1}_\mathsf{c} - X^{t+1}_\mathsf{b} = s(\mathsf{bk}^{t+1}_\mathsf{c}) - s(\mathsf{bk}^{t+1}_\mathsf{b}). \end{aligned}$$

Following in our universal sampler model, $(X^t_\mathsf{c}, X^t_\mathsf{b}, X^{t+1}_\mathsf{c}, X^{t+1}_\mathsf{b})$ *are independent and follow the same distribution* $\mathcal{D}_X = \mathcal{D}$ *on* $[\mathsf{smin}, \mathsf{smax}]$*. Hence,* Y^t *and* Y^{t+1} *are independent, and distributed identically and* symmetrically *[28] on* $[\mathsf{smin}{-}\mathsf{smax}, \mathsf{smax}{-}\mathsf{smin}]$*. We denote the distribution of* Y^t *and* Y^{t+1} *with* $\mathcal{D}_Y$*, and let* $f_Y(\cdot)$ *and* $F_Y(\cdot)$ *be the probability density function and the distribution function of* $\mathcal{D}_Y$*. We can rewrite Equation 5.7 in the form of random variables:*

$$\Pr[(Y^t \geq 0) \wedge c(0,1) \cdot Y^t + Y^{t+1} \leq 0)]. \tag{5.8}$$

Consider the event: $\{Y^t = y \wedge Y^{t+1} \leq -c(0,1)y\}$ *for all* $y \in [0, \mathsf{smax}{-}\mathsf{smin}]$*. For simplicity, we rewrite* $r = \mathsf{smax}{-}\mathsf{smin}$*. By* Y^t *is independent of* Y^{t+1}*, we have:*

$$\begin{aligned} \textit{Equation 5.8} &= \int_0^r \Pr[Y^t = y] \cdot \Pr[Y^{t+1} \leq -c(0,1)y] \mathrm{d}y \\ &= \int_0^r \int_{-r}^{-c(0,1)y} f_Y(y) f_Y(x) \mathrm{d}x \mathrm{d}y. \end{aligned} \tag{5.9}$$

Here, we consider the upper-bound of Equation 5.9 by scaling up $f_Y(\cdot)$ *with two coefficients* $c_1, c_2 > 0$ *as follows.*

$$f_Y(y) \begin{cases} \leq c_1 \cdot e^{-c_2 \cdot y^2}, & \textit{if } y \in [-r, r], \\ = 0, & \textit{otherwise.} \end{cases} \tag{5.10}$$

Note that $f_Y(y)$ *is a probability density function, hence, it satisfies* $\int_{-r}^r f_Y(y)\mathrm{d}y = 1$*. Then, for* c_1, c_2*, we have the estimation:* $\int_{-\infty}^{\infty} c_1 \cdot e^{-c_2 \cdot y^2} \mathrm{d}y \geq 1$*, which is* $c_1^2/c_2 \geq \pi^{-1}$*. The manipulation of inequality gives us:*

$$\textit{Equation 5.9} \leq \int_0^{\infty} \int_{-\infty}^{-c(0,1)y} e^{-y^2} \cdot e^{-x^2} \mathrm{d}x \mathrm{d}y = \frac{{c_1}^2}{2c_2} \cdot \tan^{-1}\left(\frac{1}{c(0,1)}\right) \leq \frac{{c_1}^2}{2c_2 \cdot c(0,1)}. \tag{5.11}$$

That is, $\Pr[\tau = 1, \mathsf{branch_b}] \leq {c_1}^2/(2c_2 \cdot c(0,1))$ *for any* $c_1, c_2 \geq 0$ *and* $c_1^2/c_2 \geq \pi^{-1}$ *where* $c(0,1) = \frac{\mathsf{acc}(t)}{\mathsf{acc}(t+1)}$.

Now, we consider the probability of 1*-chain substitution, denoted by* $\Pr[\tau = 1]$*. Let* q *be the number of viable branches of slot* t*. Recall our universal sampler,* Q *is the upper bounded of the total number of queries (regardless of honest or not) made in each time slot, i.e.,* $q \leq Q$ *as shown in Section 5.3.2 (universal sampler functionality). Then, we have:*

$$\Pr[\tau = 1] \leq 1 - \left(1 - \frac{c_1^2}{2c_2 \cdot c(0,1)}\right)^Q.$$

Next, ***chain substitution with*** $\tau > 1$ *follows the same methodology of* $\tau = 1$*. Consider a branch that satisfies* $\mathsf{branch}^{t-1+\tau} \cap \mathsf{chain}^{t-1+\tau} = \mathsf{chain}^{t-1}$*. Denote the distinct blocks on* $\mathsf{branch}^{t-1+\tau}$ *with* $\mathsf{bk}_\mathsf{b}^t, \ldots, \mathsf{bk}_\mathsf{b}^{t-1}+\tau$*. Comparing them with the blocks on* $\mathsf{chain}^{t-1+\tau}$*, and comparing the new blocks* $\mathsf{bk}_\mathsf{b}^{t+\tau}$ *in* $\mathsf{branch_b}$ *and* $\mathsf{bk}_\mathsf{c}^{t+\tau}$ *in* $\mathsf{branch_c}$*, by rewriting Equation 5.8, the probability of* $\mathsf{branch_c}$ *substituted by* $\mathsf{branch_b}$*, denoted by* $\Pr[\tau > 1, \mathsf{branch_b}]$*, equals to:*

$$\Pr\left[\left(\bigwedge_{i\in[\tau-1]}\left(\sum_{j=0}^{i}\frac{\mathsf{acc}(t+j)}{\mathsf{acc}(t+i)} \cdot Y^{t+j} \geq 0\right)\right) \wedge \left(\sum_{i=0}^{\tau}\frac{\mathsf{acc}(t+i)}{\mathsf{acc}(t+\tau)} \cdot Y^{i} \leq 0\right)\right], \tag{5.12}$$

where $Y^{t+i} = X_\mathsf{c}^{t+i} - X_\mathsf{b}^{t+i}$ *for any* $i \in \{0, \ldots, \tau\}$ *are independent and distributed identically with* $\mathcal{D}_Y$ *on* $[\mathsf{smin}-\mathsf{smax}, \mathsf{smax}-\mathsf{smin}]$*. For simplicity, we rewrite* $\frac{\mathsf{acc}(j)}{\mathsf{acc}(i)}$ *with* $c(j,i)$*, and let* $c(i,i) \triangleq 1$*. Hence, Equation 5.12 equals to:*

$$\begin{aligned}
&= \int_0^r \cdots \int_{r_{\tau-1}}^r \Pi_{i=0}^{\tau-1} \Pr[Y^{t+i} = y_i] \cdot \Pr\left[Y^{t+\tau} \leq -\sum_{i=0}^{\tau-1} c(i,\tau) \cdot y_i\right] \mathrm{d}y_0 \cdots \mathrm{d}y_{\tau-1} \\
&= \int_0^r \cdots \int_{r_{\tau-1}}^r \int_{-r}^{-\sum_{i=0}^{\tau-1} c(i,\tau)\cdot y_i} \Pi_{i=0}^{\tau-1} f_Y(y_i) \cdot f_Y(x) \mathrm{d}x \mathrm{d}y_0 \cdots \mathrm{d}y_{\tau-1}. \qquad (5.13)
\end{aligned}$$

We denote the lower bound of random variable Y^{t+i} *for any* $i \in [\tau-1]$ *as* r_i*. Hence* $r_i = -\sum_{j=0}^{i-1} c(j,i) \cdot y_j$*. Here, we use a small trick to scale Equation 5.13. Note that* r_i *is not necessarily larger than* 0*. We scale* y_i *down to* $-r$ *for all* $i \in [\tau-1]$*. Then, the upper bound of* $Y^{t+\tau}$ *satisfies* $Y^{t+\tau} \leq -c(0,\tau)y_0 + r \cdot \sum_{i=1}^{\tau-1} c(i,\tau)$*. By* $\int_{-r}^{r} f_Y(y_i)\mathrm{d}y_i \leq 1$ *for any* $i \in [\tau-1]$*, we have:*

$$\textit{Equation 5.13} \leq \int_0^r \int_{-r}^{-c(0,\tau)y_0 + r\cdot\sum_{i=1}^{\tau-1} c(i,\tau)} f_Y(y_0) f_Y(x) \mathrm{d}x \mathrm{d}y_0. \tag{5.14}$$

Finally, by setting $y' \triangleq y_0 - \frac{r}{c(0,\tau)} \cdot \sum_{i=1}^{\tau-1} c(i,\tau)$ *and the bound of* $f_Y(\cdot)$ *given in Equation 5.10:*

$$\textit{Equation 5.14} \leq \int_{-\infty}^{\infty} \int_{-\infty}^{-c(0,\tau)y'} e^{-y'^2} \cdot e^{-x^2} \mathrm{d}x\mathrm{d}y' = \frac{{c_1}^2}{c_2} \cdot \tan^{-1}\left(\frac{1}{c(0,\tau)}\right) \leq \frac{c_1^2}{c_2} \cdot c(0,\tau). \tag{5.15}$$

That is, $\Pr[\tau > 1, \mathsf{branch}_\mathsf{b}] \leq {c_1}^2/c_2 \cdot c(0,\tau)\}$*, for any* $c_1, c_2 \geq 0$ *and* $c_1^2/c_2 \geq \pi^{-1}$ *where* $c(0,\tau) = \frac{\mathsf{acc}(t)}{\mathsf{acc}(t+\tau)}$*. Then, similar to* $\tau = 1$*, we compute the probability of* $\tau \geq 1$*-chain substitution, denoted by* $\Pr[\tau > 1]$*, as follows. Let* q *be the number of viable branches in slot* t*. Then, by our universal sampler,* $q \leq Q$*, where* Q *is the upper bound of the total number of queries (regardless of honest or not) made in each time slot. Hence,*

$$\Pr[\tau > 1] \leq 1 - \left(1 - \frac{c_1^2}{c_2 \cdot c(0,\tau)}\right)^Q.$$

Combining the discussion above, we consider a constant value $c > \max\{1, \frac{c_1^2}{c_2}\}$*, and let* $\mathsf{acc}(t) = c^t$*. Then,* $\Pr[\tau \geq 1] \leq 1 - \left(1 - \frac{c_1^2}{c_2 \cdot c^{-\tau}}\right)^Q$ *is of the same order as* $Q \cdot c^{-\tau}$*. Note that we only use the upper bound of total queries to the universal sampler instead of honest queries. Finally, we can conclude that* τ*-chain substitution occurs with probability* $\mathsf{O}(c^{-\tau})$*.*

Therefore, we have the following theorem on persistence.

Theorem 4 (Persistence) *Assuming at least one honest user, the protocol* $\Pi^{n,\delta,k,s_{bk},\mathsf{acc}}$ *among the* n *users, it holds that the protocol parameterized with* $k \geq \delta{+}1$ *satisfies persistence (Definition 31) with probability at least* $1 - \Omega(c^{-k+\delta})$*.*

Proof 7 *Suppose persistence with parameter* $k \geq \delta + 1$ *is violated. It follows that, for honest users in two different time slots* $\ell_1 \leq \ell_2$ *holding* $\mathsf{chain}_1^{\ell_1}$ *and* $\mathsf{chain}_2^{\ell_2}$*,* $\mathsf{chain}_1^{\ell_1 \lceil k}$ *is not the prefix of* $\mathsf{chain}_2^{\ell_2}$*. Hence, there exists blocks on* $\mathsf{chain}_1^{\ell_1 \lceil k}$ *not on* $\mathsf{chain}_2 \neq \emptyset$*. Denote the first distinct block with* $\mathsf{bk}_1^{\ell_1 - k'}$*, then* $k' \geq k$*.*

By Proposition 1, in slot ℓ_2*, highest-scored branches of any slot before* $\ell_2 - (\delta+1)$ *are disclosed to all honest users. Hence, the block-tree* tree_2^{ℓ} *of the user who holds* $\mathsf{chain}_2^{\ell_2}$ *contains all highest-scored branches before* $\ell_2 - (\delta+1)$*. Then,* $\mathsf{chain}_1^{\ell_1 \lceil \delta+1} \in \mathsf{tree}_2^{\ell}$*. The chain substitution must happen in the slot after* $\ell{-}\delta{+}1$*. Without loss of generality, we consider the situation that for all* $i \in \{\ell_1 - k', \ldots, \ell_1 - (\delta+1)\}$*,* $\mathsf{chain}_1^i = \mathsf{chain}_1^{i-1} || \mathsf{bk}^i$ *for all* $i \in \{\ell_1 - k', \ldots, \ell_1 - (\delta+1)\}$*, i.e., blocks on* chain^{ℓ_1} *are selected conjunctively for* $k'{-}\delta$

slots from $\ell{-}k'$ to $\ell{-}(\delta{+}1)$. Therefore, by Lemma 6, the probability of this $(k'{-}\delta)$-chain substitution occurs with probability of $\mathsf{O}(c^{-k'+\delta})$ which is less then $\mathsf{O}(c^{-k+\delta})$. Finally, we conclude that the probability of persistence with $k \geq \delta + 1$ is at least $1 - \Omega(c^{-k+\delta})$.

5.5.2 Block-Liveness

For completeness, we show the following theorem for block-liveness.

Theorem 5 (Block-liveness) *Assuming at least one honest user executes the protocol $\Pi^{n,\delta,k,s_{bk},\mathsf{acc}}$ among the n users, it holds that the protocol satisfies block-liveness (Definition 31) unconditionally.*

The proof is straightforward. Assuming the one honest user is unaware of any other blocks, she can trivially extend her block-tree by generating blocks locally with the bidpool and selecting the blockchain accordingly.

5.6 Discussion

The starting point of our protocol is to implement the ledger of the P2PET, where the underlying problem of matching buy and sell bids, can be used in conjunction with a blockchain data structure. Despite the fact that users need to be certificated to participate in the system, we adopt a more general execution: permissionless with static corruptions and δ-synchronous communication network. Moreover, we assume the existence of a globally synchronized clock, which, at first glance, is a constrained setting. However, it can easily be implemented by the secure hardware provided by the P2PET system.

One component of our construction is the scoring function for blocks, *i.e.*, s_{bk}, and the BAP-based PoW scheme (BAP-based PoW), implementing a novel consensus protocol. The purpose of the scoring function is to provide a "notion of optimal" to the system. Among all the blocks available in each time slot, it chooses the "most optimal" choice to extend a (redefined) blockchain data structure. At the same time, it is used as accounting for the auction market underpinning the system. Instantiated with a concrete function, the adversary, once given the description for the scoring function, could adapt and get an advantage in constructing the next block. Thus, its adaptability, in fact, helps the optimality of the overall protocol. We advocate that the study of more concrete constructions of s_{bk} is of independent interest and out of the scope of this chapter.

In our analysis, the competitive PoX scheme is replaced by a *universal sampler* which samples blocks and corresponding scores from score space in each time slot. This, in fact, simplifies our analysis without the loss of the whole motivation of our construction. We remark that, although it is an advanced random oracle, it is practical [32], since it can be easily used in practice with a random function.

Considering the universal sampler in our protocol execution setting, we showed that our protocol has persistence and block-liveness when at least one user is honest. We recall that *block-liveness* is a variant of the standard security liveness property. This variant is necessary and meaningful in our setting, given that in every time slot *all* honest participants issue candidate blocks. Thus, the property is that one block among them is always chosen from all candidates. Needless to say, by fulfilling the block-liveness property, we also obtain the regular liveness property, thereby constructing a fully secure system.

Finally, we remark that we did not thoroughly investigate potential incentive frameworks for our proposed system in the presence of a rational adversary. This topic seem to be out of the scope of the current chapter despite its importance. In particular, the study of the overall behavior of the system in the presence of such adversary. We leave this topic for future works.

Bibliography

[1] Akhras, R., El-Hajj, W., Majdalani, M., Hajj, H.M., Jabr, R.A., Shaban, K.B.: Securing smart grid communication using ethereum smart contracts. In: 16th International Wireless Communications and Mobile Computing Conference, IWCMC 2020, Limassol, Cyprus, June 15-19, 2020. pp. 1672–1678. IEEE (2020). https://doi.org/10.1109/IWCMC48107.2020.9148345, https://doi.org/10.1109/IWCMC48107.2020.9148345

[2] Alwen, J., Chen, B., Pietrzak, K., Reyzin, L., Tessaro, S.: Scrypt is maximally memory-hard. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10212, pp. 33–62 (2017). https://doi.org/10.1007/978-3-319-56617-7_2, https://doi.org/10.1007/978-3-319-56617-7_2

[3] Badertscher, C., Garay, J.A., Maurer, U., Tschudi, D., Zikas, V.: But why does it work? A rational protocol design treatment of bitcoin. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 34–65. Springer (2018). https://doi.org/10.1007/978-3-319-78375-8_2, https://doi.org/10.1007/978-3-319-78375-8_2

[4] Badertscher, C., Lu, Y., Zikas, V.: A rational protocol treatment of 51% attacks. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12827, pp. 3–32. Springer (2021). https://doi.org/10.1007/978-3-030-84252-9_1, https://doi.org/10.1007/978-3-030-84252-9_1

[5] Baldominos, A., Saez, Y.: Coin.ai: A proof-of-useful-work scheme for blockchain-based distributed deep learning. Entropy **21**(8), 723 (2019). https://doi.org/10.3390/e21080723, https://doi.org/10.3390/e21080723

[6] Ball, M., Rosen, A., Sabin, M., Vasudevan, P.N.: Proofs of work from worst-case assumptions. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 789–819. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_26, https://doi.org/10.1007/978-3-319-96884-1_26

[7] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012), http://dl.acm.org/citation.cfm?id=2188395

[8] Biryukov, A., Perrin, L.: Symmetrically and asymmetrically hard cryptography. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10626, pp. 417–445. Springer (2017). https://doi.org/10.1007/978-3-319-70700-6_15, https://doi.org/10.1007/978-3-319-70700-6_15

[9] Blocki, J., Zhou, H.: Designing proof of human-work puzzles for cryptocurrency and beyond. In: Hirt, M., Smith, A.D. (eds.) Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9986, pp. 517–546 (2016). https://doi.org/10.1007/978-3-662-53644-5_20, https://doi.org/10.1007/978-3-662-53644-5_20

[10] Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 757–788. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_25, https://doi.org/10.1007/978-3-319-96884-1_25

[11] Brown, D.R.L.: Breaking RSA may be as difficult as factoring. J. Cryptol. **29**(1), 220–241 (2016). https://doi.org/10.1007/s00145-014-9192-y, https://doi.org/10.1007/s00145-014-9192-y

[12] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA. pp. 136–145. IEEE Computer Society (2001). https://doi.org/10.1109/SFCS.2001.959888, https://doi.org/10.1109/SFCS.2001.959888

[13] Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Seltzer, M.I., Leach, P.J. (eds.) Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999. pp. 173–186. USENIX Association (1999), https://dl.acm.org/citation.cfm?id=296824

[14] Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982. pp. 199–203. Plenum Press, New York (1982). https://doi.org/10.1007/978-1-4757-0602-4_18, https://doi.org/10.1007/978-1-4757-0602-4_18

[15] Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings. Lecture Notes in Computer Science, vol. 403, pp. 319–327. Springer (1988). https://doi.org/10.1007/0-387-34799-2_25, https://doi.org/10.1007/0-387-34799-2_25

[16] Chenli, C., Li, B., Jung, T.: Dlchain: Blockchain with deep learning as proof-of-useful-work. In: Ferreira, J.E., Palanisamy, B., Ye, K., Kantamneni, S., Zhang, L. (eds.) Services - SERVICES 2020 - 16th World Congress, Held as Part of the Services Conference Federation, SCF 2020, Honolulu, HI, USA, September 18-20, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12411, pp. 43–60. Springer (2020). https://doi.org/10.1007/978-3-030-59595-1_4, https://doi.org/10.1007/978-3-030-59595-1_4

[17] Chenli, C., Li, B., Shi, Y., Jung, T.: Energy-recycling blockchain with proof-of-deep-learning. In: IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2019, Seoul,

Korea (South), May 14-17, 2019. pp. 19–23. IEEE (2019). https://doi.org/10.1109/BLOC.2019.8751419, https://doi.org/10.1109/BLOC.2019.8751419

[18] Coelho, F.: An (almost) constant-effort solution-verification proof-of-work protocol based on merkle trees. In: Vaudenay, S. (ed.) Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5023, pp. 80–93. Springer (2008). https://doi.org/10.1007/978-3-540-68164-9_6, https://doi.org/10.1007/978-3-540-68164-9_6

[19] David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 66–98. Springer (2018). https://doi.org/10.1007/978-3-319-78375-8_3, https://doi.org/10.1007/978-3-319-78375-8_3

[20] Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings. Lecture Notes in Computer Science, vol. 740, pp. 139–147. Springer (1992). https://doi.org/10.1007/3-540-48071-4_10, https://doi.org/10.1007/3-540-48071-4_10

[21] Dyer, M.E., Frieze, A.M.: Probabilistic analysis of the generalised assignment problem. Math. Program. **55**, 169–181 (1992). https://doi.org/10.1007/BF01581197, https://doi.org/10.1007/BF01581197

[22] Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9216, pp. 585–605. Springer (2015). https://doi.org/10.1007/978-3-662-48000-7_29, https://doi.org/10.1007/978-3-662-48000-7_29

[23] Fitzi, M., Kiayias, A., Panagiotakos, G., Russell, A.: Ofelimos: Combinatorial optimization via proof-of-useful-work - A provably secure

blockchain protocol. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13508, pp. 339–369. Springer (2022). https://doi.org/10.1007/978-3-031-15979-4_12, https://doi.org/10.1007/978-3-031-15979-4_12

[24] Foundation, C.: Cardano Hub. https://www.cardano.org/ (2023), [Online; accessed 26-May-2023]

[25] Garay, J.A., Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Rational protocol design: Cryptography against incentive-driven adversaries. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA. pp. 648–657. IEEE Computer Society (2013). https://doi.org/10.1109/FOCS.2013.75, https://doi.org/10.1109/FOCS.2013.75

[26] Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9057, pp. 281–310. Springer (2015). https://doi.org/10.1007/978-3-662-46803-6_10, https://doi.org/10.1007/978-3-662-46803-6_10

[27] Garay, J.A., Kiayias, A., Leonardos, N., Panagiotakos, G.: Bootstrapping the blockchain, with applications to consensus and fast PKI setup. In: Abdalla, M., Dahab, R. (eds.) Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10770, pp. 465–495. Springer (2018). https://doi.org/10.1007/978-3-319-76581-5_16, https://doi.org/10.1007/978-3-319-76581-5_16

[28] George E.P. Box, G.C.T.: Bayesian Assessment of Assumptions 1. Effect of Non-Normality on Inferences about a Population Mean with Generalizations, chap. 3, pp. 149–202. John Wiley and Sons, Ltd (1992). https://doi.org/https://doi.org/10.1002/9781118033197.ch3, https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118033197.ch3

[29] Goldreich, O.: The Foundations of Cryptography - Volume 1: Basic Techniques. Cambridge University Press (2001). https://doi.org/10.1017/CBO9780511546891, http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol1.html

[30] Górski, T., Bednarski, J.: Modeling of smart contracts in blockchain solution for renewable energy grid. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) Computer Aided Systems Theory - EUROCAST 2019 - 17th International Conference, Las Palmas de Gran Canaria, Spain, February 17-22, 2019, Revised Selected Papers, Part I. Lecture Notes in Computer Science, vol. 12013, pp. 507–514. Springer (2019). https://doi.org/10.1007/978-3-030-45093-9_61, https://doi.org/10.1007/978-3-030-45093-9_61

[31] Hellman, M.E.: A cryptanalytic time-memory trade-off. IEEE Trans. Information Theory **26**(4), 401–406 (1980). https://doi.org/10.1109/TIT.1980.1056220, https://doi.org/10.1109/TIT.1980.1056220

[32] Hofheinz, D., Jager, T., Khurana, D., Sahai, A., Waters, B., Zhandry, M.: How to generate and use universal samplers. In: Cheon, J.H., Takagi, T. (eds.) 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. Lecture Notes in Computer Science, vol. 10032, pp. 715–744 (2016). https://doi.org/10.1007/978-3-662-53890-6_24, https://doi.org/10.1007/978-3-662-53890-6_24

[33] Kamp, S.H., Magri, B., Matt, C., Nielsen, J.B., Thomsen, S.E., Tschudi, D.: Weight-based nakamoto-style blockchains. In: Longa, P., Ràfols, C. (eds.) Progress in Cryptology - LATINCRYPT 2021 - 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6-8, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12912, pp. 299–319. Springer (2021). https://doi.org/10.1007/978-3-030-88238-9_15, https://doi.org/10.1007/978-3-030-88238-9_15

[34] Katz, J., Loss, J., Xu, J.: On the security of time-lock puzzles and timed commitments. In: Pass, R., Pietrzak, K. (eds.) Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12552, pp. 390–413. Springer

(2020). https://doi.org/10.1007/978-3-030-64381-2_14, https://doi.org/10.1007/978-3-030-64381-2_14

[35] Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10401, pp. 357–388. Springer (2017). https://doi.org/10.1007/978-3-319-63688-7_12, https://doi.org/10.1007/978-3-319-63688-7_12

[36] Kirli, D., Couraud, B., Robu, V., Salgado-Bravo, M., Norbu, S., Andoni, M., Antonopoulos, I., Negrete-Pincetic, M., Flynn, D., Kiprakis, A.: Smart contracts in energy systems: A systematic review of fundamental approaches and implementations. Renewable and Sustainable Energy Reviews **158**, 112013 (2022)

[37] Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176, https://doi.org/10.1145/357172.357176

[38] Lan, Y., Liu, Y., Li, B., Miao, C.: Proof of learning (pole): Empowering machine learning with consensus building on blockchains (demo). In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. pp. 16063–16066. AAAI Press (2021), https://ojs.aaai.org/index.php/AAAI/article/view/18013

[39] Li, B., Chenli, C., Xu, X., Jung, T., Shi, Y.: Exploiting computation power of blockchain for biomedical image segmentation. In: IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019. pp. 2802–2811. Computer Vision Foundation / IEEE (2019). https://doi.org/10.1109/CVPRW.2019.00339, http://openaccess.thecvf.com/content_CVPRW_2019/html/BCMCVAI/Li_Exploiting_Computation_Power_of_Blockchain_for_Biomedical_Image_Segmentation_CVPRW_2019_paper.html

[40] Lihu, A., Du, J., Barjaktarevic, I., Gerzanics, P., Harvilla, M.: A proof of useful work for artificial intelligence on the blockchain. CoRR **abs/2001.09244** (2020), https://arxiv.org/abs/2001.09244

[41] Luu, L., Chu, D., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 254–269. ACM (2016). https://doi.org/10.1145/2976749.2978309, https://doi.org/10.1145/2976749.2978309

[42] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), http://bitcoin.org/bitcoin.pdf

[43] Park, J.S., Lim, B.H., Lee, Y.: A lagrangian dual-based branch-and-bound algorithm for the generalized multi-assignment problem. Manage. Sci. **44**(12), 271–275 (dec 1998)

[44] Pass, R., Shi, E.: Fruitchains: A fair blockchain. In: Schiller, E.M., Schwarzmann, A.A. (eds.) Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017. pp. 315–324. ACM (2017). https://doi.org/10.1145/3087801.3087809, https://doi.org/10.1145/3087801.3087809

[45] Pass, R., Shi, E.: Thunderella: Blockchains with optimistic instant confirmation. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 3–33. Springer (2018). https://doi.org/10.1007/978-3-319-78375-8_1, https://doi.org/10.1007/978-3-319-78375-8_1

[46] Pietrzak, K.: Simple verifiable delay functions. In: Blum, A. (ed.) 10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA. LIPIcs, vol. 124, pp. 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.ITCS.2019.60, https://doi.org/10.4230/LIPIcs.ITCS.2019.60

[47] Pomerance, C., Erdös, P.: A tale of two sieves (1998), https://api.semanticscholar.org/CorpusID:1695797

[48] Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., USA (1996)

[49] Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978). https://doi.org/10.1145/359340.359342, https://doi.org/10.1145/359340.359342

[50] Robertson, H.: Ethereum transaction fees are running sky-high. (2021), "https://markets.businessinsider.com/news/currencies/ethereum-transaction-gas-fees-high-solana-avalanche-cardano-crypto-blockchain-2021-12"

[51] Su, X., Défago, X., Larangeira, M., Mori, K., Oda, T., Okumura, Y., Tamura, Y., Tanaka, K.: Peer-to-peer energy trading meets blockchain: Consensus via score-based bid assignment. Cryptology ePrint Archive, Paper 2022/1471 (2022), https://eprint.iacr.org/2022/1471, https://eprint.iacr.org/2022/1471

[52] Su, X., Larangeira, M., Tanaka, K.: How to prove work: With time or memory. In: IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021, Sydney, Australia, May 3-6, 2021. pp. 1–3. IEEE (2021). https://doi.org/10.1109/ICBC51069.2021.9461131, https://doi.org/10.1109/ICBC51069.2021.9461131

[53] Su, X., Larangeira, M., Tanaka, K.: Provably secure blockchain protocols from distributed proof-of-deep-learning. In: Li, S., Manulis, M., Miyaji, A. (eds.) Network and System Security - 17th International Conference, NSS 2023, Canterbury, UK, August 14-16, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13983, pp. 114–136. Springer (2023). https://doi.org/10.1007/978-3-031-39828-5_7, https://doi.org/10.1007/978-3-031-39828-5_7

[54] Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11478, pp. 379–407. Springer (2019). https://doi.org/10.1007/978-3-030-17659-4_13, https://doi.org/10.1007/978-3-030-17659-4_13

[55] Wood, G.: Ethereum yellow paper (2014)