Incremental Proofs of Sequential Work

Nico Döttling[1], Russell W. F. Lai[2], and Giulio Malavolta[3]*

[1] CISPA Helmholtz Center for Information Security
[2] Friedrich-Alexander-Universität Erlangen-Nürnberg
[3] Carnegie Mellon University

Abstract. A proof of sequential work allows a prover to convince a verifier that a certain amount of sequential steps have been computed. In this work we introduce the notion of *incremental proofs of sequential work* where a prover can carry on the computation done by the previous prover incrementally, without affecting the resources of the individual provers or the size of the proofs.

To date, the most efficient instance of proofs of sequential work [Cohen and Pietrzak, Eurocrypt 2018] for N steps require the prover to have $\sqrt{N}$ memory and to run for $N + \sqrt{N}$ steps. Using incremental proofs of sequential work we can bring down the prover's storage complexity to $\log N$ and its running time to N.

We propose two different constructions of incremental proofs of sequential work: Our first scheme requires a single processor and introduces a poly-logarithmic factor in the proof size when compared with the proposals of Cohen and Pietrzak. Our second scheme assumes $\log N$ parallel processors but brings down the overhead of the proof size to a factor of 9. Both schemes are simple to implement and only rely on hash functions (modelled as random oracles).

1 Introduction

Imagine that you discover a candidate solution to a famous open problem (*e.g.*, the Riemann Hypothesis), and are *fairly* convinced that your solution is correct but not entirely. Before publishing your solution you want to scrutinize it further. However, fearing that someone else might make the same discovery, you need a way to timestamp yours. While there are many online timestamping services available[4], authenticity of such a timestamp depends on how much one trusts the service provider. Clearly, a solution independent of trust and resting only on a cryptographic assumption is more desirable.

Proofs of Sequential Work (PoSW) [10] is an emerging paradigm which offers a conceptually simple solution to the timestamping problem. Roughly speaking, proofs of sequential work allow a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that *almost* time T elapsed since a certain event happened. A little more concretely, a PoSW system consists of a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The prover takes as input a statement χ and a time parameter N. The statement χ can be something like a hash of the file which one wants to timestamp. After terminating, the prover interacts with the verifier $\mathcal{V}$ to convince him that at least time N has elapsed since χ was sampled.

* Work done while at Friedrich-Alexander-Universität Erlangen-Nürnberg

[4] *e.g.*, https://www.freetsa.org

We require a PoSW to be complete, sound and efficient. Here completeness means that an honest prover will succeed in convincing the verifier that time N has elapsed since the sampling of χ. Soundness means that a cheating prover will not succeed in convincing the verifier that time N has elapsed if, in fact significantly less time has passed. Finally, efficiency means that time N is also sufficient for the prover to generate such a prove. Another practically important aspect is memory complexity of the prover, *i.e.*, how much memory is required to compute a proof for time parameter N. Regardless of the requirements on prover efficiency, the verifier's runtime should be essentially independent of N. Finally, for practical reasons such a proof should be non-interactive. That is, after a proof π is computed by the prover $\mathcal{P}$ and published, no further interaction with $\mathcal{P}$ is necessary to verify the proof.

1.1 Incremental Proofs of Sequential Work

An aspect not considered in the original formulation of proofs of sequential work is whether a *still running* proof of sequential work can be migrated from one prover to another, or forked to two provers. This aspect becomes relevant when considering that real computers are not immune to hardware failure, so one may want to spawn clones of important proofs that have been running for a long time.

In this work, we introduce the notion of *incremental proofs of sequential work (iPoSW)*. Essentially, an iPoSW is a non-interactive PoSW with the additional feature that anyone who obtains a proof π for a statement χ and time parameter N can *resume* the computation of π, thereby generating a proof π' for χ with time parameter $N + N'$. More formally, we require that there exists an algorith Inc which takes as input a proof π for time N and a parameter N' and outputs a proof π'. We require that π' has the same distribution as a proof for χ for time $N + N'$.

One could imagine a direct construction of iPoSW from PoSW as follows. To increment a proof π for a statement χ and time N, first derive a new statement χ' from χ and π, *e.g.*, by computing a hash $\chi' \leftarrow \mathsf{H}(\chi, \pi)$. Now compute a proof π' for statement χ' and time N' and then append π' to π, *i.e.*, output (π, π'). To verify (π, π') that (π, π') is a proof for χ and time $N + N'$, compute $\chi' \leftarrow \mathsf{H}(\chi, \pi)$ and check whether π is a proof for χ and time N and π' is a proof for χ' and time N'.

This simple solution has, however, an obvious drawback: The size of the proof grows linearly in the number of increments, which very is undesirable if the proof is frequently passed on to new provers.

Moreover, if we look at existing constructions of PoSW [10,5], a prover $\mathcal{P}$ computing a proof π for a statement χ and time N needs to commit memory proportional to N. Cohen and Pietrzak [5] propose a tradeoff which reduces the memory requirement of p_i to a sublinear but still polynomial amount, however this comes at the expense of additional *sequential* computation time, *i.e.*, prover efficiency is affected by this tradeoff.

1.2 Our Results

In this work we provide constructions of incremental proofs of sequential work where the sequential runtime of an honest prover is N, while its memory complexity is $\mathsf{poly}(\log N)$.

We provide two instantiations, both based on the construction of Cohen and Pietrzak [5], which differ in terms of prover resources and the proof size.

- The first construction is single-threaded, *i.e.*, the prover needs a single processor. Compared to the construction of [5], the proof size grows by a factor of $(\log N)^2$.
- The second construction is multi-threaded, where the prover needs $\log N$ parallel processors. Compared to [5], the proof size grows by a factor of 9.

In particular, our results close the soundness gap between a prover with a large memory and a prover with a poly-logarithmic memory present in previous constructions.

We remark that from a technological point of view the assumption of prover parallelism is justified. For actual applications, the expression $\log N$ can be upper-bounded by 100, which corresponds to a processor capable of computing 100 hashes in parallel, a number well in the reach of modern GPUs.

1.3 Technical Overview

The starting point of our construction is the recent elegant PoSW construction of Cohen and Pietrzak [5]. We will henceforth refer to this scheme as the CP scheme, which is briefly reviewed below. The CP construction relies on properties of a special directed acyclic graph, which we will refer to as CP_n. This graph is constructed as follows: Let B_n be a complete binary tree of depth n, *i.e.*, the longest leaf-to-root path consists of n edges, with edges pointing from the leaves towards the root. Each node in B_n is indexed by a bit string of length at most n, while the root node is indexed by the empty string ϵ. The graph CP_n is constructed by adding edges from all nodes v to all leaves u such that v is a left-sibling of the path from u to the root.

The CP Approach. For a time parameter N, choose n such that CP_n contains (at least) N nodes. The prover is given a statement χ which is used to seed random oracles $\mathsf{H}_\chi(\cdot) := \mathsf{H}(\chi, \cdot)$ and $\mathsf{H}'_\chi(\cdot) := \mathsf{H}'(\chi, \cdot)$ given the random oracles H and H' respectively. Using H_χ, the prover computes a label for each node v in CP_n by hashing the labels of all nodes with incoming edges to v. Starting from the leftmost leaf 0^n, which is assigned a label 0^λ, the prover iteratively computes the labels of all nodes in CP_n, completing each subtree before starting a new leaf. Eventually the prover obtains a label ℓ_ϵ for the root node.

Next, the prover computes $\mathsf{H}'_\chi(\ell_\epsilon)$ which outputs a randomness for sampling t challenge leaves, where t is a statistical security parameter. The proof then consists of the labels of all t challenge leaves, as well as the labels of all siblings of the paths from the challenge leaves to the root. To verify a proof, the verifier recomputes $\mathsf{H}'_\chi(\ell_\epsilon)$ to verify if the prover provided the correct paths, and if so checks that the t paths provided by the prover are consistent.

Note that in order to compute a proof, the prover has to either remember the N labels for the entire CP_n graph, or recompute the labels required in the proof once the challenge leaves are chosen, which requires N sequential hash computations. This introduces a soundness slack of $\frac{1}{2}$ between these two strategies, *i.e.*, the memory efficient prover has to compute for time $2N$ to prove a statement for time N. This factor becomes particularly significant when large values of N are considered, *e.g.*, a PoSW that 10 years of sequential operations have been performed may take between 10 and 20 years to be computed. To attenuate this problem, Cohen and Pietrzak propose a hybrid approach where the prover stores $\sqrt{N}$ nodes and can then recompute the challenge root-to-leaf paths in time $\sqrt{N}$.

At the Heart of the Problem. This soundness slack is clearly undesirable as the value of N grows: A prover with access to a large amount of memory can achieve a non-trivial speed up in the computation of the proof over a prover with polylogarithmic memory. As it turns out, this issue is tightly connected with the fact that the CP proofs cannot be extended incrementally: On a very high level, the crux of the problem is that the challenge leaves are determined solely by the root of the CP_n tree. Extending the tree causes the root to change and renders the previous challenge set obsolete.

The main idea in our first construction is to choose challenge leaves "on-the-fly" at *each node* of the tree and then gradually discard some of them as the tree grows. This will allow us to compute a proof π *in a single pass*.

More precisely, our selection mechanism works as follows: For any node v in CP_n which has at most t leaves, we assign all these leaves to be the challenge leaves for the node v. Let l and r be the children of a node v which has more than t leaves, and let S_l and S_r be the challenge leaves for l and r respectively. To determine the set S_v of challenge leaves for v, we first compute the label ℓ_v of v as in the CP scheme, and then hash the label ℓ_v with H'_χ to obtain random coins[5]. Using these random coins, we can sample S_v as a random subset of size t from the set $S_l \cup S_r$. This operation is visualized in Figure 1 and Figure 2.

In a bit more detail, due to the way the graphs are traversed, we only need to store challenge-paths at what we call *unfinished nodes*. A node is unfinished if it has already been traversed/processed, but its right sibling has not yet been traversed. Consequently, only left siblings can be unfinished. Moreover, due to the structure of the graph CP_n and the way it is traversed, at each step the unfinished nodes are exactly the left siblings on the path from the root to the node which is currently processed. Consequently, at each step there are at most $\log N$ unfinished nodes. Essentially, when a node l becomes unfinished, it *waits* until its right sibling r is processed. By the way we traverse CP_n, the next node to be traversed is the parent v of l and r. Once the label of v has been computed, we can compute a set of challenge paths for v as described above and remove l from the list of unfinished nodes.

Observe that if a leaf previously chosen as a challenge leaf is dropped due to the above subset sampling, this leaf will not be chosen as a challenge leaf again in the rest of the computation. Therefore the prover can safely erase the labels of some of the nodes lying on the paths from these dropped challenge nodes to the root, which surely will not appear in the eventual proof. On the other hand the final challenge set is still unpredictable to the eyes of the prover since the decision which paths are discarded is uniquely determined by the complete labelling of the tree.

[5] As we are working in the random oracle model, these coins can be taken directly from ℓ_v if we make the hashes sufficiently longer. However, for presentation purposes we use a separate hash function which hashes ℓ_v.

It is not immediately clear that the strategy we just described lead to a sound protocol. Infact, a malicious prover can already see a large fraction of the challenge path before the label of the root node is even computed and *adaptively* recompute parts of the proof. The main observation on which our analysis is based is that, once a node v becomes unfinished, its label commits to all the leafs under v, thus the challenge paths at v provide a good statistical sample of the overall fraction of invalid leafs in the subtree of v.

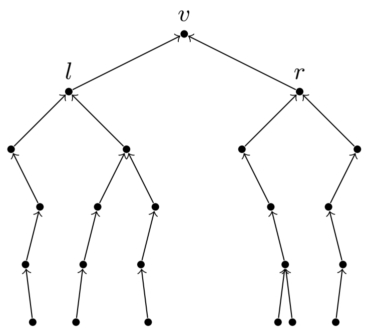

Fig. 1: Before choosing challenge subset.

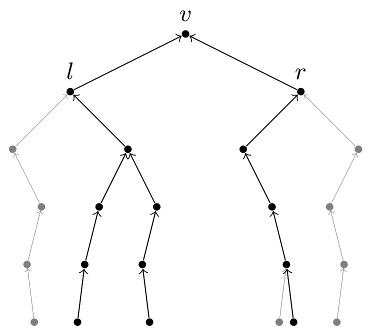

Fig. 2: After choosing challenge subset.

Recomputation to the Rescue. The above strategy seems to solve all problems at once:

1. The prover algorithm can traverse the tree and remember the local challenge paths using poly-logarithmic memory in N and in sequential time N. Once the root is reached, the set of challenge paths is already in the memory of the prover! Therefore no recomputation is needed and the source of the slack is obliterated.
2. The proof is naturally incremental: Further iterations of the tree only shave off root-to-leaf paths in the challenge set, as opposed to determining a completely new set of challenges.

However there is still a challenge to be addressed: Due to the adaptivity of the adversary, our strategy introduces a factor of $\log N$ in the soundness loss. That is, if the CP scheme with a set of parameters achieves soundness α, *i.e.*, the prover cannot cheat by computing less than $(1-\alpha)N$ steps, our scheme only achieves soundness $\log N \cdot \alpha$. This in turn means that in order to achieve the same soundness parameter as the CP scheme, we need to increase the number of challenge paths by a factor of $(\log N)^2$, which also results in an increase of the proof size by a factor of $(\log N)^2$. Although this does not affect the asymptotic performance of our scheme, it has an impact on the concrete proof sizes. For $N = 2^{40}$, our proofs are bigger than those obtained with the CP scheme by a factor of ~ 1600. To bring down the proof sizes to a practical regime, we reconcile CP scheme with our "on-the-fly" selection strategy. Our second construction assumes that the prover is a parallel machine, but we can show that the number of parallel processors required will never exceed $\log N$.

Our second scheme is based on the following observation. Let v be a node in CP_n, and assume that l is its left child and r is its right child. Further assume that the prover

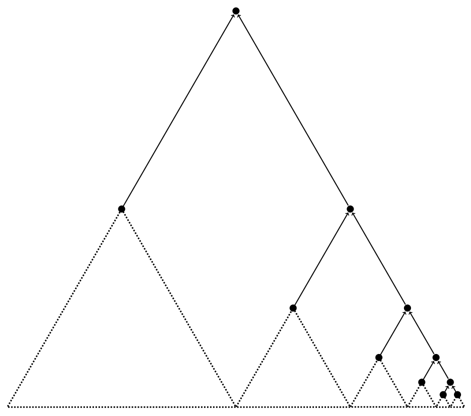

Fig. 3: Recomputation of Sub-Trees.

just finished traversing the tree under l, that is l becomes processed but unfinished. By the structure of CP_n, the prover next traverses the tree underneath r. In our first scheme the node l would just be on a *waiting list* of unfinished nodes and has to wait and remember its challenge paths until r is processed. However, due to symmetry it will take the prover the same amount of sequential steps to traverse the tree underneath r as it took to traverse the tree under l. This suggests a strategy (depicted in Figure 3): While l is unfinished and waiting for the r to be processed, we can *recompute* the subtree underneath l in order to fetch fresh challenge paths using an additional parallel processor. By the time r is finished, this process will have terminated and we do not need to bear the above soundness loss for l.

Notice further that, to recompute the tree underneath l, all the prover needs is the labels of the currently unfinished nodes on the path from the root to v, which the prover needs to keep in memory regardless. This modification of the prover strategy must also be reflected by the verifier. When we verify a root-to-leaf path, the verification strategy will change once the path takes a left turn.

Note that the memory complexity of the main thread is unchanged and that at any point in time there are at most $\log N$ parallel processes. The parallel threads are identical to the recomputation step. Therefore, the complexity of each parallel thread is essentially the same as that of the CP scheme. This hybrid construction brings down the loss in soundness to a factor of 3, which corresponds to an increase of the proof size by a factor of 9. We consider this to be a modest price to pay in exchange for getting the additional feature of incremental proofs and an essentially optimal prover complexity.

1.4 Perspectives

Merkle trees are ubiquitous in cryptographic protocol design, allowing to compress large amounts of data into a succinct digest. Membership proofs are particularly efficient as they usually consist of root-to-leaf paths and can be encoded with logarithmic-size strings. The de-facto methodology to non-interactively probe Merkle trees at random locations is to apply the Fiat-Shamir [7] transform, on input the root of the tree. This means that the challenge locations are determined only when the full tree is computed. Thus, the prover must either recompute the paths or store the full tree in its memory.

Using our techniques one can compress data and generate challenges in a single pass, without any memory blowup. This becomes particularly advantageous when computing over very large databases or data streams. Here we exemplify the applications of our methods to scenarios of interest.

Verifiable Probing. Consider a stream of data where some statistical measure is computed by an untrusted party. Using our approach we can increase the confidence in the validity of the statistics by probing the stream on random locations. The prover iteratively computes a Merkle commitment of the stream and selects random probes using our "on-the-fly" selection strategy. The verifier can then non-interactively check whether the distribution of the probes resembles the reported statistics.

Streaming Arguments. In Micali's CS proofs paradigm [11,8], witnesses for NP relations are encoded into probabilistically checkable proofs (PCP) [1] and then committed using a Merkle tree. The testing locations for the PCP are selected using Fiat-Shamir [7] and the corresponding root-to-leaf paths form the CS proof. Our techniques can be useful for memory-constrained provers that cannot store the complete PCP encoding in their memory. Our challenge-selection algorithm allows the provers to compute the CS proof using only one stream of the encoding.

1.5 Related Work

Proofs of work, a concept introduced by Dwork and Naor [6] and having become wildly popular in the context of cryptocurrencies, allow a prover to convince a verifier that a certain amount of computational effort has been invested in a certain task. However, the computation can be parallelized, thus it generates a mismatch among players which have different resource constraints. Mahmoody, Moran, and Vadhan [10] introduced the concept of PoSW and provided a construction based on depth-robust graphs. Very recently, Cohen and Pietrzak [5] provided an elegant construction based on a binary tree with some useful combinatorial properties. Their scheme improves over [10] in terms of conceptual simplicity, concrete efficiency, and can reduce the memory complexity of the prover up to $\log N$. A shortcoming of their approach is that, in order to achieve such a memory bound, one has to perform the same amount of computation twice.

Incrementally verifiable computation (IVC) was introduced by Valiant [14] and allows a machine to output short proofs that arbitrary parts of the computation have been done correctly without significantly affecting the resources of such a machine. As observed by Boneh et al. [4], IVC is a more general primitive than PoSW. The main construction paradigm for IVC consists of a recursive composition of succinct arguments of knowledge [11], which means that existing constructions of IVC either

- make non-black-box use of random oracles [14], or
- require a trusted setup [2].

In general, incremental PoSW appears to be an easier problem than IVC which justifies the existence of more efficient solutions based on weaker assumptions.

Verifiable delay functions (VDF) have been introduced by Boneh et al. [4] and can be seen as PoSW with unique proofs: The prover can only convince the verifier with a single value, which is uniquely determined by the time parameter N and by the statement. Thus VDF constitutes a stronger primitive than PoSW and as to our current understanding requires stronger cryptographic material: Known constructions [12,15] rely either on IVC or on specific number-theoretic assumptions related to factoring large integers.

Time-lock puzzles [13] encapsulate a secret information for a pre-determined amount of time. This primitive is tightly related to sequential computation as it needs to withstand attacks from highly parallel processors. Time lock-puzzles can be constructed assuming the hardness of a variant of the RSA assumption [13] or succinct randomized encodings and the existence of a worst case non-parallelizable language [3]. Unlike PoSW, no construction based on symmetric-key primitives is known and [9] gave a black-box separation for these two objects.

2 Preliminaries

2.1 Notations

Let $G = (V, E)$ be a graph where V is the set of nodes and E is the set of edges. If $v \in V$, we write also $v \in G$ for convenience. Let $\mathfrak{T}$ be a tree and $i \in \mathfrak{T}$ be a node. $\mathfrak{T}_i$ denotes the set of nodes in the subtree rooted at node i. $\mathsf{leaf}(i)$ denotes the set of all leaf nodes that are descendants of i. $\mathsf{parent}(i)$ and $\mathsf{child}(i)$ denote the parent of and the set of children of i, repectively. $\mathsf{path}(i)$ returns the set of nodes located at the (unique) path from the root (inclusive) to node i (inclusive). The notations are extended naturally to sets of nodes. Let $S \subseteq \mathfrak{T}$ be a set of nodes, then $\mathfrak{T}_S := \bigcup_{i \in S} \mathfrak{T}_i$, $\mathsf{leaf}(S) := \{\mathsf{leaf}(i) : i \in S\}$ and $\mathsf{path}(S) := \{\mathsf{path}(i) : i \in S\}$.

For a complete binary tree $B_n = (V, E')$ of $N = 2^{n+1} - 1$ nodes, we say that B_n is of depth n (counting the number of edges in the longest leaf-to-root path). The nodes $V = \{0,1\}^{\leq n}$ are identified by binary strings of length at most n and the empty string ϵ represents the root. The edges $E' = \{(x||b, x) : b \in \{0,1\}, x \in \{0,1\}^i, i < n\}$ are directed from the leaves towards the root. Let $v \in \{0,1\}^{n_v} \subseteq B_n$ be a node n_v edges away from the root. We say that v is of depth n_v or height $h_v := n - n_v$.

2.2 Statistical Distance

In the following we recall the definition of statistical distance.

Definition 1 (Statistical Distance). *Let X and Y be two random variables over a finite set $\mathcal{U}$. The statistical distance between X and Y is defined as*

$$\mathbb{SD}[X, Y] = \frac{1}{2} \sum_{u \in \mathcal{U}} |\Pr[X = u] - \Pr[Y = u]|$$

2.3 Tail Bound for the Hypergeometric Distributions

Here we introduce a useful inequality by Hoeffding.

Theorem 1 (Hoeffding Inequality). *Let X be distributed hypergeometrically with t draws. Then it holds that*

$$\mathsf{Pr}\left[X < \mathsf{E}[X] - \zeta\right] < e^{-2\zeta^2 t}.$$

3 Incremental Proofs of Sequential Work

Below we define incremental proof of sequential work in the same spirit as Cohen and Pietrzak [5], except that we state directly the non-interactive variant.

Definition 2. *A (non-interactive) incremental proof of sequential work (iPoSW) scheme consists of a tuple of* PPT *oracle-aided algorithms* $(\mathsf{Prove}, \mathsf{Inc}, \mathsf{Vf})$*, executed by a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ in the following fashion:*

Common Inputs. $\mathcal{P}$ and $\mathcal{V}$ get as common input a computation security parameter $\lambda \in \mathbb{N}$, a statistical security parameter $t \in \mathbb{N}$, and a time parameter $N \in \mathbb{N}$. All parties have access to a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$.

Statement. $\mathcal{V}$ samples a random statement $\chi \leftarrow_{\$} \{0,1\}^\lambda$ and sends it to $\mathcal{P}$.

Prove. $\mathcal{P}$ computes $\pi \leftarrow \mathsf{Prove}^{\mathsf{H}}(\chi, N)$ and sends π to $\mathcal{V}$.

Increment. $\mathcal{P}$ computes $\pi' \leftarrow \mathsf{Inc}^{\mathsf{H}}(\chi, N, N', \pi)$ and sends π' to $\mathcal{V}$.

Verify. $\mathcal{V}$ computes and outputs $\mathsf{Vf}^{\mathsf{H}}(\chi, N, \pi)$.

We require a PoSW scheme to be complete in the following sense.

Definition 3 (Completeness). *For all $\lambda \in \mathbb{N}$, all $N \in \mathbb{N}$, all random oracles* H*, and all statements $\chi \in \{0,1\}^\lambda$ we say that a tuple (χ, N, π) is honest if*

$$\pi \in \mathsf{Prove}^{\mathsf{H}}(\chi, N) \qquad \textit{or} \qquad \pi \in \mathsf{Inc}^{\mathsf{H}}(\chi, N', N'', \pi'),$$

where $N' + N'' = N$ and the tuple (χ, N', π') is also honest. A (non-interactive) incremental proof of sequential work is complete if for all honest tuples (χ, N, π) it holds that

$$\mathsf{Vf}^{\mathsf{H}}(\chi, N, \pi) = 1.$$

In the following we define soundness for incremental proofs of sequential work.

Definition 4 (Soundness). *A (non-interactive) incremental proof of sequential work PoSW is sound if for all $\lambda, N \in \mathbb{N}$, for all $\alpha > 0$, for all adversaries $\mathcal{A}$ that make at most $(1-\alpha)N$ sequential queries to* H*, it holds that*

$$\mu := \mathsf{Pr}\left[\chi \leftarrow \{0,1\}^\lambda; \pi \leftarrow \mathcal{A}^{\mathsf{H}}(\chi, N) : \mathsf{Vf}^{\mathsf{H}}(\chi, N, \pi) = 1\right] \in \mathsf{negl}(\lambda)$$

where μ is called the soundness error.

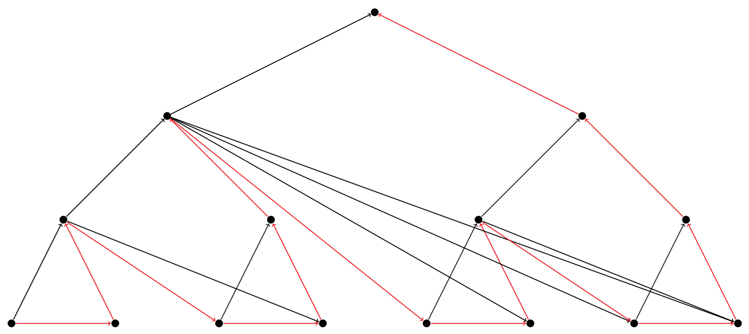
Fig. 4: CP_3 with traversal order highlighted in red.

For our construction we recall the following directed acyclic graph constructed by Cohen and Pietrzak [5] which has some nice combinatorial properties.

Definition 5 (CP Graphs). *For $n \in \mathbb{N}$, let $N = 2^{n+1} - 1$ and $B_n = (V, E')$ be a complete binary tree of depth n with edges pointing from the leaves to the root. The graph $CP_n = (V, E)$ is a directed acyclic graph constructed from $B_n = (V, E')$ as follows. For any leaf $u \in \{0,1\}^n$, for any node v which is a left-sibling of a node on the path from u to the root ϵ, an edge (v, u) is appended to E'. Formally, $E := E' \cup E''$ where*

$$E'' := \{(v, u) : u \in \{0,1\}^n, u = a||1||a', v = a||0, \textit{ for some } a, a' \in \{0,1\}^{\leq n}\}.$$

An illustration of CP_3 is in Figure 4, with its traversal order (c.f. Lemma 2) highlighted in red. Here we recall some technical lemmas from [5].

Lemma 1 ([5]). *The labels of a CP_n graph can be computed in topological order using $\lambda(n+1)$ bits of memory.*

Let $\mathfrak{T}$ be a tree and $S \subseteq \mathfrak{T}$ be a subset of nodes. We denote by S^* the minimal set of nodes with exactly the same set of leaves as S, in other words, S^* is the smallest set such that $\mathsf{leaf}(S^*) = \mathsf{leaf}(S)$.

Lemma 2 ([5]). *For all $S \subseteq V$, the subgraph of $CP_n = (V, E)$ on vertex set $V \setminus \mathfrak{T}_{S^*}$ has a directed path going through all the $|V| - |\mathfrak{T}_{S^*}|$ nodes.*

Lemma 3 ([5]). *For all $S \subseteq V$, $\mathfrak{T}_{S^*}$ contains $\frac{|\mathfrak{T}_{S^*}|+|S|}{2}$ many leaves.*

4 Main Construction

For any $n \in \mathbb{N}$, we construct an incremental PoSW scheme based on the graph $CP_n = (V, E)$ as follows. We assume without loss of generality that, given a random oracle H, one can sample a fresh random oracle indexed by a string x, which we denote by H_x. This can *e.g.*, be done by prepending x and a special separator symbol to any query to H, *i.e.*, $\mathsf{H}_x(y) := \mathsf{H}(x\#y)$ for a separator symbol $\#$.

4.1 Parameters

Our incremental Proof-of-Sequential-Work system depends on the following parameters and objects.

– A time parameter N of the form $N = 2^{n+1} - 1$, for some integer $n \in \mathbb{N}$.
– A computational security parameter λ
– A statistical security parameter t
– A full-domain hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ modelled as a random oracle.
– A full-domain hash function $\mathsf{H}' : \{0,1\}^* \to \{0,1\}^{3t}$ modelled as a random oracle.
– A sampler $\mathsf{RandomSubset}(M, m; \mathsf{r})$ which takes a universe size M, a sample size m and uniform random coins r and outputs a uniformly random subset $X \subseteq [M]$ such that $|X| = m$. In our application, we will always set $M = 2t$ and $m = t$. Since $\binom{2t}{t} < \left(\frac{2t \cdot e}{t}\right)^t = (2e)^t$, where $\log 2e \approx 2.44$, random coins of size $3t$ are sufficient to sample statistically close to a uniform subset.

Notation. Let ϵ be the root-node of CP_n and 0^n the left-most leaf in the tree or starting-node. We will call a left node $v \in V$ unfinished, if v has been traversed by the prover algorithm but $\mathsf{parent}(v)$ has not yet been. At any time, the prover will keep a list of the currently unfinished nodes U. At each unfinished node v, the prover will store $\mathcal{L}_v$, a set of extended labeled paths from v to $\mathsf{leaf}(v)$. An extended labeled path consists of a list of tuples of the form $(v_i, \ell_{l_i}, \ell_{r_i}, \mathsf{ind}_i)$, where v_i is the index/address of a node on the path, l_i is the left child of v_i, r_i is the right child of v_i and consequently ℓ_{l_i} is the label of l_i and ℓ_{r_i} is the label of r_i. Finally, ind_i is a local path index, the meaning of which will be explained later.

For simplicity of exposition, we assume that t is a power of 2. Our construction can be easily adapted to the more general case where t is arbitrarily chosen. For convenience, we denote by $n^* = n^*(n,t)$ the depth at which every node has exactly t leafs underneath, *i.e.*, it holds for every node v which is n^* edges from ϵ that $|\mathsf{leaf}(v)| = t$.

4.2 Scheme Description

$\underline{\mathsf{Prove}^{\mathsf{H},\mathsf{H}'}(\chi, N)}$:

1. Initialize $U \leftarrow \emptyset$, the set of unfinished nodes.
2. Assign $\ell_{0^n} \leftarrow 0^\lambda$ as the label of the starting node.
3. Traverse the graph CP_n starting from 0^n. At every node $v \in V$ which is traversed, do the following:
   (a) Compute the label ℓ_v by
   $$\ell_v \leftarrow \mathsf{H}_{(\chi,v)}(\ell_{v_1}, \ldots, \ell_{v_d})$$
   where $v_1, \ldots, v_d \in V$ are all nodes with edges pointing to v, *i.e.*, $(v_i, v) \in E$.
   (b) Let l and r be the children of v.
   (c) If $|\mathsf{leafs}(v)| \leq t$, set $\mathcal{L}_v \leftarrow \{[(v, \ell_l, \ell_r, \bot)\|L] \text{ where } L \in \mathcal{L}_l \cup \mathcal{L}_r\}$.
   (d) Otherwise (*i.e.*, if $|\mathsf{leafs}(v)| \geq 2t$), do the following:

i. Compute

$$\mathsf{r}_v \leftarrow \mathsf{H}'_{(\chi,v)}(\ell_v).$$

ii. Choose a random t-subset S_v of $[2t]$ via $S_v \leftarrow \mathsf{RandomSubset}(2t, t; \mathsf{r}_v)$.

iii. For $j \in \{0, \ldots, t-1\}$, write $S_v[j] = at + b$ with $a \in \{0, 1\}$ and $0 \leq b < t$.

$$\mathcal{L}_v[j] \leftarrow \begin{cases} [(v, \ell_l, \ell_r, j) \| \mathcal{L}_l[b]], & \text{if } a = 0 \\ [(v, \ell_l, \ell_r, j) \| \mathcal{L}_r[b]], & \text{if } a = 1 \end{cases}$$

(e) Mark l as finished, *i.e.*, remove l from U and, if v is a left child, mark v as unfinished, *i.e.*, add v to U.

4. Once the set of unfinished nodes consists only of the root-node (*i.e.*, $U = \{\epsilon\}$), terminate and output $\pi \leftarrow (\ell_\epsilon, \mathcal{L}_\epsilon)$.

$\underline{\mathsf{Inc}^{\mathsf{H},\mathsf{H}'}(\chi, N, N', \pi):}$

1. Initialize $U \leftarrow \emptyset$.
2. Parse π as $(\ell_\epsilon, \mathcal{L}_\epsilon)$
3. Assign $\ell_{0^{n'-n}} := \ell_\epsilon$ and $\mathcal{L}_{0^{n'-n}} := \mathcal{L}_\epsilon$.
4. Execute the algorithm $\mathsf{Prove}^{\mathsf{H},\mathsf{H}'}(\chi, N')$ starting from step 3 with a slight change: Traverse the graph $CP_{n'}$ starting from $0^{n'-n-1}\|1\|0^n$ (instead of from $0^{n'}$).

$\underline{\mathsf{Vf}^{\mathsf{H},\mathsf{H}'}(\chi, N, \pi):}$

1. Parse π as $(\ell_\epsilon, \mathcal{L}_\epsilon)$.
2. For all paths $\mathsf{path} \in \mathcal{L}_\epsilon$ do the following:
   (a) Parse path as $[(v_0, \ell_{l_0}, \ell_{r_0}, \mathsf{ind}_0)\| \ldots \|(v_n, \ell_{l_n}, \ell_{r_n}, \mathsf{ind}_n)]$.
   (b) For every node $v \in \{v_0, \ldots, v_n\}$ on the path, check if the label ℓ_v was computed correctly. That is, for $v = 0^n$ check whether $\ell_v = 0^\lambda$, and for any other node $v \in V \backslash \{0^n\}$ check whether $\ell_v = \mathsf{H}_{(\chi,v)}(\ell_{v_1}, \ldots, \ell_{v_d})$, where $v_1, \ldots, v_d$ are all the nodes with edges pointing to v. The value ℓ_v can either be retrieved from the parent node of v, or is directly available for the case of the root-node ϵ. For the special case of leaf-nodes, the values $\ell_{v_1}, \ldots, \ell_{v_d}$ are not stored locally with the node v, but are stored at some other (a-priori known) nodes along the path path (refer to the structure of the graph CP_n).
   (c) For all $j \in \{0, \ldots, n^*\}$, compute $\mathsf{r}_{v_j} \leftarrow \mathsf{H}'_{(\chi,v_j)}(\ell_{v_j})$ and $S_{v_j} \leftarrow \mathsf{RandomSubset}(2t, t; \mathsf{r}_{v_j})$. If v_{j+1} is the left child of v_j, check if

$$S_v[\mathsf{ind}_j] = \mathsf{ind}_{j+1}.$$

   Otherwise, if v_{j+1} is the right child of v_j, check if

$$S_v[\mathsf{ind}_j] = t + \mathsf{ind}_{j+1}.$$

3. If all checks pass output 1, otherwise 0.

Incomplete Trees. We briefly outline how to handle incomplete binary trees. If N does not define a complete tree, then at the end of the prover's iteration the list of unfinished nodes consists of several elements: $U = \{v_1, \ldots, v_n\}$. The new proof π consists of the tuples $(\ell_{v_1}, \mathcal{L}_{v_1}), \ldots, (\ell_{v_n}, \mathcal{L}_{v_n})$. The proof can be easily verified by running the standard verification algorithm on each pair $(\ell_{v_i}, \mathcal{L}_{v_i})$ separately and outputting 1 if all the verifications succeeds. In a similar way, one can increment the proof by recovering the trees computed so far, setting the labels of the unfinished nodes to $(\ell_{v_1}, \ldots, \ell_{v_n})$ and the corresponding sets to $(\mathcal{L}_{v_1}, \ldots, \mathcal{L}_{v_n})$. Given such a snapshot of the execution, one can continue the standard iteration and complete the proof for the new (larger) tree.

4.3 Efficiency Analysis

We now discuss the efficiency of our scheme in terms of proof size, computation and communication.

Proof Size. The proof consists of the root-label ℓ_ϵ and t challenge paths $\mathsf{path}_0, \ldots, \mathsf{path}_{t-1}$. Each $\mathsf{path} \in \{\mathsf{path}_0, \ldots, \mathsf{path}_{t-1}\}$ consists of n tuples of the form $(v, \ell_l, \ell_r, \mathsf{ind})$, where v is the index of a node, ℓ_l and ℓ_r are the labels of the left and right children of v, and $\mathsf{ind} \in [t]$ is the index of path in the challenge set S_v at v. The node index v can be stored using a single bit per node, indicating whether it is the left or right child of its parent. Each of ℓ_l and ℓ_r can be stored using λ bits, and ind can be represented using $\log t$ bits. Consequently, the entire proof has size at most $t \cdot n \cdot (1 + 2\lambda + \log t) = O(t \cdot \lambda \cdot n)$ (assuming $t \in \mathsf{poly}(\lambda)$). Later, in the soundness analysis, we will show that our construction is sound if $t \in O(\lambda \cdot n^2)$. With such choice of t, the proof size is bounded by $O(\lambda^2 \cdot n^3)$.

Prover Efficiency. The prover traverses the N nodes of the graph CP_n in the same manner as the prover algorithm of the CP scheme. Additionally, at each node the prover computes a challenge using the random oracle $\mathsf{H}'_{(\chi,v)}$.

The challenges $\mathsf{H}'_{(\chi,v)}$ can be computed in a way that does not increase the parallel time complexity of the prover. Specifically, instead of computing the randomness for the challenges via $\mathsf{r}_v \leftarrow \mathsf{H}'_{(\chi,v)}(\ell_v)$, we can equivalently compute the r_v similar to ℓ_v via $\mathsf{r}_v \leftarrow \mathsf{H}'_{(\chi,v)}(\ell_{v_1}, \ldots, \ell_{v_d})$. This is possible as both H and H' are random oracles. The proof changes only slightly, but we kept the naive version for presentation purposes. In the modified scheme H and H' can be evaluated in parallel. thus the parallel complexity is not increased by the evaluation of H'. To conclude, the parallel complexity of the prover is bounded by the time needed for $O(N)$ sequential calls to the random oracles.

For the memory complexity of the prover, Cohen and Pietrzak [5] show using a standard pebbling argument (c.f. Lemma 1) that the labels of CP_n can be computed in topological order storing at most $n+1$ labels at any time, i.e., having at most $n+1$ pebbles in the graphs at any time. This corresponds to the number of unfinished nodes, i.e. at every time-step there are at most $n+1$ unfinished nodes. At each unfinished node $v \in U$, the prover keeps a list $\mathcal{L}_v$ consisting of t labeled paths. By the analysis above these t paths can be stored using $O(\lambda^2 \cdot n^3)$ bits. Consequently, the space complexity of the prover is bounded by $O(\lambda^2 \cdot n^4)$.

Verifier Efficiency. The verifier needs to check the consistency of t paths, each consisting n nodes. Checking a node incurs the computation of a hash using $\mathsf{H}_{(\chi,v)}$ and one using $\mathsf{H}'_{(\chi,v)}$. All nodes can be checked in parallel with by computing a constant number of hashes. After that, the verifier has to check whether all $t \cdot n$ checks are passed, which can be performed in parallel time $O(\log(t \cdot n)) = O(\log(\lambda \cdot n^3))$.

4.4 Soundness

We now establish soundness of our construction. Before proving the main theorem, we prove some useful lemmas. Throughout the following analysis, we always assume that N and t are powers of two, but the arguments naturally extend to the more generic case. We denote by $\mathsf{L} := (\mathfrak{T}_v, \{\ell_u\}_{u \in \mathfrak{T}_v})$ the labelling for a sub-tree $\mathfrak{T}_v$. We slightly abuse the notation and we say that $u \in \mathsf{L}$ if $u \in \mathfrak{T}_v$.

Lemma 4. *Let $\mathcal{A}$ be an algorithm with access to a random oracle* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ *which outputs a root-hash of a Merkle tree of depth n and a (valid) root-to-leaf path* path *with siblings. Then there exists an efficient online extractor* $\mathsf{Extract}$*, which on input a node $v \in \mathfrak{T}$, a label ℓ^* and a list Q (of size q) of all* H*-queries of $\mathcal{A}$ so far, outputs a labelling* L *of the sub-tree $\mathfrak{T}_v$ rooted at v such that the following holds. Let* path^* *be the leaf-to-root path p truncated at v and let* path_L *be the same path in* L*, then* $\mathsf{path}^* = \mathsf{path}_\mathsf{L}$*, except with probability $\frac{1+q(q-1)}{2^\lambda}$, over the choice of* H.

Proof. We assume without loss of generality that the list Q is of the form $\{(\mathsf{in}, \mathsf{out})\}$, and that the depth n_v of a node is efficiently computable from its identifier. We define the algorithm $\mathsf{Extract}$ in the following.

$\mathsf{Extract}(v, \ell^*, Q)$: The root of the tree L set to be ℓ^* and the rest of the tree is recursively constructed applying $(n - n_v)$ times the following function $f(\mathsf{node})$: Parse Q for an entry of the form $(\mathsf{in}, \mathsf{node})$, if such an entry does not exist then return $\perp$. Else parse in as $\ell_0 \| \ell_1$, set ℓ_0 as the left child of node and ℓ_0 as the right child of node in L. Then run $f(\ell_0)$, $f(\ell_1)$ and return L.

The algorithm runs with a logarithmic factor in the size of $\mathfrak{T}_v$ (assuming an ordered list Q) and therefore it is efficient. Let BAD be the event such that there exists a node $v \in \mathsf{path}^*$ labelled ℓ'_v such that $\ell_v \neq \ell'_v$ and $\ell_{\mathsf{parent}(v)} = \ell'_{\mathsf{parent}(v)}$, where ℓ'_v and ℓ_v are the labelling output by $\mathcal{A}$ and by the extractor, respectively. By the law of total probability we have that

$$\begin{aligned}\Pr[\mathsf{BAD}] &= \Pr[\mathsf{BAD} \mid \ell_v = \perp] \Pr[\ell_v = \perp] + \Pr[\mathsf{BAD} \mid \ell_v \neq \perp] \Pr[\ell_v \neq \perp] \\ &\leq \Pr[\mathsf{BAD} \mid \ell_v = \perp] + \Pr[\mathsf{BAD} \mid \ell_v \neq \perp].\end{aligned}$$

To bound the first summand observe that $\mathsf{H}(\ell'_v \| \ell'_{v'}) = \ell'_{\mathsf{parent}(v)}$, where v' is the sibling of v, since the path path needs to be valid. Further note that there exists no entry of the form $(\cdot, \ell'_{\mathsf{parent}(v)}) \in Q$, since ℓ_v is set to $\perp$ and $\ell'_{\mathsf{parent}(v)} = \ell_{\mathsf{parent}(v)}$. This implies that the adversary has correctly guessed a pre-image of $\ell'_{\mathsf{parent}(v)}$ without querying H, which happens with probability $2^{-\lambda}$. Thus we can bound from above

$$\Pr[\mathsf{BAD} \mid \ell_v = \perp] \leq 2^{-\lambda}.$$

For the second summand consider again that $\mathsf{H}({\ell'}_v \| {\ell'}_{v'}) = {\ell'}_{\mathsf{parent}(v)}$ and that $\mathsf{H}(\ell_v \| \ell_{v'}) = \ell_{\mathsf{parent}(v)}$. Since ${\ell'}_{\mathsf{parent}(v)} = \ell_{\mathsf{parent}(v)}$ we have that $\mathsf{H}({\ell'}_v \| {\ell'}_{v'}) = \mathsf{H}(\ell_v \| \ell_{v'})$, which is a valid collision for H since, by assumption, ${\ell'}_v \neq \ell_v$. Therefore we have that

$$\begin{aligned} \mathsf{Pr}\left[\mathsf{BAD} \mid \ell_v \neq \bot\right] &\leq 1 - \prod_{k=0}^{q-1}\left(1 - \frac{k}{2^\lambda}\right) = 1 - \frac{2^\lambda}{2^\lambda} \cdot \frac{2^\lambda - 1}{2^\lambda} \cdots \frac{2^\lambda - (q-1)}{2^\lambda} \\ &\leq 1 - \left(\frac{2^\lambda - (q-1)}{2^\lambda}\right)^q = 1 - \left(1 - \frac{q-1}{2^\lambda}\right)^q \leq \frac{q(q-1)}{2^\lambda} \end{aligned}$$

where the last inequality is due to Bernoulli. Thus by triangle inequality we have that

$$\mathsf{Pr}\left[\mathsf{BAD}\right] \leq \frac{1}{2^\lambda} + \frac{q(q-1)}{2^\lambda} = \frac{1 + q(q-1)}{2^\lambda},$$

which implies that the complementary event happens with all but negligible probability. That is, for all nodes in $v \in \mathsf{path}^*$ labelled ℓ_v such that and $\ell_{\mathsf{parent}(v)} = \ell_{\mathsf{parent}(v)}$ it holds that ${\ell'}_v = \ell_v$. Since L is rooted at ℓ^* and path^* and L have the same depth, it follows by induction that path^* must be identical to $\mathsf{path}_{\mathsf{L}}$, with the same probability. □

Given a labeled tree L, we say that a node $v \in \mathsf{L}$ is inconsistent if it holds that $\ell_v \neq \mathsf{H}(\ell_{v_1}, \dots, \ell_{v_d})$, where $(v_1, \dots, v_d)$ are the nodes with an incoming edge to v. Let $n(\mathsf{L})$ be the depth of L, then L has $2^{n(\mathsf{L})}$-many paths (or, equivalently, leaves) and we define C as the set of paths which contain at least one inconsistent node. Note that L uniquely defines a set of t challenge paths (as specified in the description of the prover algorithm) which we denote by Z. For convenience we define the functions $\gamma(\mathsf{L}) := \frac{|C|}{2^{n(\mathsf{L})}}$ and $\delta(\mathsf{L}) := \frac{|Z \cap C|}{|Z|}$.

Lemma 5. *Let v be a node and let l and r be the left and right child of v, respectively. If*

$$\delta(\mathsf{L}_l) \geq \gamma(\mathsf{L}_l) - \eta_l \text{ and } \delta(\mathsf{L}_r) \geq \gamma(\mathsf{L}_r) - \eta_r$$

then it holds that

$$\mathsf{Pr}\left[\gamma(\mathsf{L}_v) \leq \delta(\mathsf{L}_v) + \eta_v\right] \geq \left(1 - e^{-2\left(\eta_v - \frac{(\eta_l + \eta_r)}{2}\right)^2 t}\right).$$

Proof. Recall that $\gamma(\mathsf{L}_v)$ counts the fraction of inconsistent paths of v. Since l and r are the children of v it holds that

$$\gamma(\mathsf{L}_v) = \frac{(\gamma(\mathsf{L}_l) + \gamma(\mathsf{L}_r))}{2}. \tag{1}$$

Rearranging the terms we have that

$$\gamma(\mathsf{L}_l) \leq \delta(\mathsf{L}_l) + \eta_l \tag{2}$$
$$\gamma(\mathsf{L}_r) \leq \delta(\mathsf{L}_r) + \eta_r, \tag{3}$$

thus combining (1), (2), and (3) we obtain

$$\gamma(\mathsf{L}_v) \leq \frac{(\delta(\mathsf{L}_l) + \eta_l + \delta(\mathsf{L}_r) + \eta_r)}{2} = \frac{(\delta(\mathsf{L}_l) + \delta(\mathsf{L}_r))}{2} + \frac{(\eta_l + \eta_r)}{2}. \tag{4}$$

Let Z'_v be the set of all paths in $Z_l \cup Z_r$ extended to v, i.e. $Z'_v = \{(v,p) \mid p \in Z_l \cup Z_r\}$. By construction, the set Z_v is a random t-subset of Z'_v (where the randomness for this choice is taken from $\mathsf{H}'_{(\chi,v)}(\ell_v)$). Assume now that there are s_l rejecting paths in Z_l and s_r rejecting paths in Z_r, i.e. it holds that $\delta(\mathsf{L}_l) = \frac{s_l}{t}$ and $\delta(\mathsf{L}_r) = \frac{s_r}{t}$. That is, there are $s_l + s_r$ rejecting paths in Z'_v. Consequently, the expected number of rejecting paths in Z_v is $\frac{s_l+s_r}{2t} \cdot t = \frac{1}{2}(\delta(\mathsf{L}_l) + \delta(\mathsf{L}_r)) \cdot t$, that is

$$\mathsf{E}[\delta(\mathsf{L}_v)] = \frac{(\delta(\mathsf{L}_l) + \delta(\mathsf{L}_r))}{2}, \tag{5}$$

where the expectation is taken over the random choice $\mathsf{H}'_{(\chi,v)}(\ell_v)$. Thus we can rewrite

$$\begin{aligned}
\Pr\left[\gamma(\mathsf{L}_v) > \delta(\mathsf{L}_v) + \eta_v\right] &= \Pr\left[\delta(\mathsf{L}_v) < \gamma(\mathsf{L}_v) - \eta_v\right] \\
&< \Pr\left[\delta(\mathsf{L}_v) < \frac{(\delta(\mathsf{L}_l) + \delta(\mathsf{L}_r))}{2} + \frac{(\eta_l + \eta_r)}{2} - \eta_v\right] \\
&= \Pr\left[\delta(\mathsf{L}_v) < \mathsf{E}[\delta(\mathsf{L}_v)] + \frac{(\eta_l + \eta_r)}{2} - \eta_v\right] \\
&< e^{-2\left(\eta_v - \frac{(\eta_l+\eta_r)}{2}\right)^2 t}
\end{aligned}$$

where the first inequality holds by (4), the second equality holds by (5), and the last inequality is a direct application of the Hoeffding inequality for hypergeometric distributions (Theorem 1). □

We are now ready to state and prove the main theorem.

Theorem 2. *The construction given in Section 4.2 is sound for any $t \in O(\lambda \cdot n^2)$, and the soundness error is given by $\frac{1+q(q-1)}{2^\lambda} + q \cdot e^{-2(\frac{\alpha}{n})^2 t}$.*

Proof. Let χ be the challenge statement and let q_v be the number of calls of $\mathcal{A}$ to the random oracle $\mathsf{H}'_{(\chi,v)}$, i.e., the adversary makes at most $q = \sum_{v \in \mathfrak{T}} q_v$ calls to H' in total. By Lemma 4, there exists an (efficient) algorithm Extract which on input a node $v \in \mathfrak{T}$, a label ℓ_v and a list Q of all query to H by $\mathcal{A}$ with their responses, outputs a a labelling L_v of the sub-tree $\mathfrak{T}_v$ rooted at v. For $i = \{0, \dots, n\}$ and for $j = \{1, \dots, 2^i\}$, let $v_{i,j}$ be the j-th node at layer i of the tree (counting from the root towards the leaves).

Consider the following sequence of hybrids.

- Hybrid $\mathcal{H}_0$: This is identical to the real experiment.
- Hybrid $\mathcal{H}_1$: The same as $\mathcal{H}_0$, except for the following modifications.
  - The experiment records a list Q of all H queries made by $\mathcal{A}$ with their responses.
  - Every time $\mathcal{A}$ queries $\mathsf{H}'_{(\chi,v)}$ for a $v \in V$ with a label ℓ_v, a labelling L_v for the sub-tree under v is computed via $\mathsf{L}_v \leftarrow \mathsf{Extract}(v, \ell_v, Q)$.

- If it holds for any path opened by $\mathcal{A}$ that the labels on the path are different from the labels in L_ϵ (where ϵ is the root), then $\mathcal{H}_1$ aborts and outputs 0.

Let BAD_v be the following event: $\mathcal{A}$ queries $\mathsf{H}'_{(\chi,v)}$ with a query $\hat{\ell}_v$ corresponding to a labeled sub-tree $\mathsf{L}_v \leftarrow \mathsf{Extract}(v, \hat{\ell}_v, Q)$ for which it holds that $\delta(\mathsf{L}_v) < \gamma(\mathsf{L}_v) - \frac{n^* - n_v}{n^*} \cdot \alpha$, where n_v is the depth of v (i.e. the distance between the root-node ϵ and v) and n^* is the depth at which every node has exactly t leafs underneath.

For $i = n^*, \ldots, 0$ and $j = 1, \ldots, 2^i$ define the following hybrids.

– Hybrid $\mathcal{H}_{i,j}$: The same as the previous hybrid, except that the experiment outputs 0 if the event $\mathsf{BAD}_{v_{i,j}}$ happens (Recall that $v_{i,j}$ is the j-th node at layer i of the tree, counting from the root towards the leaves).

We will now show indistinguishability between the hybrids. By Lemma 4 it holds that $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable. We now turn to the indistinguishability of hybrids $\mathcal{H}_{i,j}$. For notational convenience, let $\mathcal{H}^{\downarrow}_{i,j}$ be the hybrid before $\mathcal{H}_{i,j}$.

First consider $i = n^*$. It holds for each node v at level i that the set Z_v of challenge paths consists of all paths from v to the leaves under v. Consequently, it holds for all v at level i that $\delta(\mathsf{L}_v) = \gamma(\mathsf{L}_v)$ and therefore BAD_v happens with probability 0.

Now consider the case of $i < n^*$ and let $v = v_{i,j}$. Moreover, let l and r be the the left and right children of v.

First notice that, conditioned on that the event BAD_v does not happen, hybrid $\mathcal{H}_{i,j}$ is distributed identically to the previous hybrid, i.e. $\Pr[\mathcal{H}_{i,j}(\mathcal{A}) = 1 | \neg\mathsf{BAD}_v] = \Pr\left[\mathcal{H}^{\downarrow}_{i,j}(\mathcal{A}) = 1 | \neg\mathsf{BAD}_v\right]$. Therefore

$$\begin{aligned}\mathbb{SD}[\mathcal{H}_{i,j}, \mathcal{H}^{\downarrow}_{i,j}] &= \Pr[\mathsf{BAD}_v] \cdot \underbrace{\left|\Pr[\mathcal{H}_{i,j}(\mathcal{A}) = 1 | \mathsf{BAD}_v] - \Pr\left[\mathcal{H}^{\downarrow}_{i,j}(\mathcal{A}) = 1 | \mathsf{BAD}_v\right]\right|}_{\leq 1} \\ &\leq \Pr[\mathsf{BAD}_v]\end{aligned}$$

It is thus sufficient to bound the probability for the event BAD_v. $\mathcal{A}$ queries the random oracle $\mathsf{H}'_{(\chi,v)}$ with at most q_v distinct queries. Fix a query $\hat{\ell}_v$, and let $\hat{\ell}_l$ and $\hat{\ell}_r$ be the corresponding labels of the children l and r of v. It holds that

$$\begin{aligned}\delta(\mathsf{L}_l) &\geq \gamma(\mathsf{L}_l) - \frac{n^* - (i+1)}{n^*} \cdot \alpha \\ \delta(\mathsf{L}_r) &\geq \gamma(\mathsf{L}_r) - \frac{n^* - (i+1)}{n^*} \cdot \alpha,\end{aligned}$$

as otherwise one of the events BAD_l or BAD_r would have happened and the experiment would have aborted. We can now rewrite

$$\begin{aligned}\Pr\left[\mathsf{BAD}_v\right] &= \Pr\left[\delta(\mathsf{L}_v) < \gamma(\mathsf{L}_v) - \frac{n^* - i}{n^*}\cdot\alpha\right]\\ &= 1 - \Pr\left[\gamma(\mathsf{L}_v) \leq \delta(\mathsf{L}_v) + \frac{n^* - i}{n^*}\cdot\alpha\right]\\ &< e^{-2\left(\frac{n^*-i}{n^*}\cdot\alpha - \frac{n^*-(i+1)}{n^*}\cdot\alpha\right)^2 t}\\ &= e^{-2\left(\frac{\alpha}{n^*}\right)^2 t}\end{aligned}$$

by Lemma 5. A union-bound over all queries to $\mathsf{H}_{(\chi,v)}$ yields

$$\Pr\left[\mathsf{BAD}_v\right] < q_v \cdot e^{-2(\frac{\alpha}{n^*})^2 t}.$$

Thus we conclude that the statistical distance between $\mathcal{H}_{i,j}$ and $\mathcal{H}_{i,j}^{\downarrow}$ is at most $q_v \cdot e^{-(\frac{\alpha}{n^*})^2 t}$. Consequently, we can bound the statistical distance between the first hybrid $\mathcal{H}_0$ and the last hybrid $\mathcal{H}_{0,1}$ by

$$\mathbb{SD}[\mathcal{H}_0, \mathcal{H}_{0,1}] = \frac{1 + q(q-1)}{2^\lambda} + \sum_{v\in\mathfrak{T}} q_v \cdot e^{-2(\frac{\alpha}{n^*})^2 t} = \frac{1 + q(q-1)}{2^\lambda} + q\cdot e^{-2(\frac{\alpha}{n^*})^2 t}.$$

We will finally bound the success probability of $\mathcal{A}$ in the last hybrid $\mathcal{H}_{0,1}$. This is in fact identical to the analysis of [5]. Let S denote the set of all inconsistent nodes in the tree output by $\mathcal{A}$ in $\mathcal{H}_{0,1}$. Then by Lemma 2 there exists a path going though all the nodes in $V \setminus \mathfrak{T}_{S^*}$. We distinguish two cases

1. $|\mathfrak{T}_{S^*}| \leq \alpha N$
2. $|\mathfrak{T}_{S^*}| > \alpha N$

For the first case $\mathcal{A}$ must have done at least $(1-\alpha)N$ sequential queries, so we are left with a bound on the second case. By Lemma 3 $\mathfrak{T}_{S^*}$ (and therefore S^*) contains at least $\frac{|S^*|+|\mathfrak{T}_{S^*}|}{2} > \alpha 2^n$ leaves. However, note that in the experiment $\mathcal{H}_{0,1}$ the challenger aborts whenever the adversary satisfies the winning conditions, since

$$\gamma(\mathsf{L}_\epsilon) > \frac{\alpha 2^n}{2^n} = \alpha$$

and therefore

$$\delta(\mathsf{L}_\epsilon) \geq \gamma(\mathsf{L}_\epsilon) - \alpha > 0.$$

Consequently, as $\delta(\mathsf{L}_\epsilon) = \frac{|Z\cap C|}{|Z|}$, this implies that $|Z \cap C| > 0$ and therefore at least one of the paths in Z is also in C and therefore we detect an inconsistent node. This however implies that the proof is always rejected by the verifier. So in the final experiment $\mathcal{H}_{0,1}$ the success-probability of the adversary is exactly 0. This concludes our proof. □

5 Multi-Thread Construction

In this section we show how to improve the concrete efficiency of incremental proofs of sequential work by assuming some parallel capability of the prover. More specifically, we assume that the prover can spawn n parallel threads, where n denotes the depth of the graph CP_n. Note that we can upper bound n by $\lambda = 100$, since we require the prover to be polynomial time.

5.1 Parameters

Throughout the following section we use the same parameters and notation of Section 4.1 and we define the following additional subroutines.

– A full-domain hash function $\mathsf{H}'' : \{0,1\}^* \to \{0,1\}^{t(n+2)}$ modelled as a random oracle.
– A sampler $\mathsf{RandomPath}(v; \mathsf{r})$ which takes as input a node v and uniform random coins r, and outputs a set of t uniformly random paths with common prefix v. Since $\log \binom{2^{h_v}}{t} < \log \left(\frac{2^{h_v} \cdot e}{t}\right)^t < t(h_v + 2) \leq t(n+2)$, random tapes of size $t(n+2)$ always suffice to sample a uniform set.
– A function $\mathsf{FetchPath}(S_v, U, \{\ell_v : v \in U\})$ which takes as input a set S_v of t paths with common prefix v, a set of $U = \{u : \exists v' \in \mathfrak{T}_v \; s.t. \; (u, v') \in E\}$ where with edges pointing to $\mathfrak{T}_v$, and the set $\{\ell_v : v \in U\}$ of labels of all nodes in U. The function recomputes the labelling of $\mathfrak{T}_v$ using the labels of nodes in U. The output of the function is the labelling of all paths in S_v. Note that such a function can be computed in time $O(2^{h_v})$ and with memory $O(t \cdot h_v)$.

5.2 Scheme Description

$\underline{\mathsf{Prove}^{\mathsf{H},\mathsf{H}',\mathsf{H}''}(\chi, N)}$:

1. Initialize $U \leftarrow \emptyset$ to be the set of unfinished nodes.
2. Assign $\ell_{0^n} \leftarrow 0^\lambda$.
3. Traverse the graph CP_n starting from 0^n. At every node $v \in V$ which is traversed, do the following:
   (a) Compute the label ℓ_v by
$$\ell_v \leftarrow \mathsf{H}_{(\chi,v)}(\ell_{v_1}, \ldots, \ell_{v_d})$$
   where $v_1, \ldots, v_d \in V$ are all nodes v is adjacent with, *i.e.*, $(v_i, v) \in E$.
   (b) Let l and r be the children of v.
   (c) If $|\mathsf{leafs}(v)| \leq t$, set $\mathcal{L}_v \leftarrow \{[(v, \ell_l, \ell_r, \bot) \| L] \text{ where } L \in \mathcal{L}_l \cup \mathcal{L}_r\}$.
   (d) Otherwise (*i.e.*, if $|\mathsf{leafs}(v)| \geq 2t$), do the following:
      i. Compute
$$\mathsf{r}_u \leftarrow \mathsf{H}'_{(\chi,u)}(\ell_u).$$
      ii. Choose a random t-subset S_v of $[2t]$ via $S_v \leftarrow \mathsf{RandomSubset}(2t, t; \mathsf{r}_v)$.

iii. For $j \in \{0, \ldots, t-1\}$, write $S_v[j] = at + b$ where $a \in \{0,1\}$ and $0 \leq b < t$. Set

$$\mathcal{L}_u[j] := \begin{cases} [(u, \ell_l, \ell_r, j) \| \mathcal{L}_l[b]], & \text{if } a = 0 \\ [(u, \ell_l, \ell_r, j) \| \mathcal{L}_r[b]], & \text{if } a = 1 \end{cases}$$

(e) If v is a left node (*i.e.*, it is the left child of its parent):

i. Compute

$$\mathsf{r}_u \leftarrow \mathsf{H}''_{(\chi,u)}(\ell_u).$$

ii. Choose a random t-set of paths with prefix v via $S_v \leftarrow \mathsf{RandomPath}(v; \mathsf{r}_v)$.

iii. Execute in a parallel thread $\mathcal{L} \leftarrow \mathsf{FetchPath}(S_v, U, \{\ell_v : v \in U\})$ and set $\mathcal{L}_v := \{[(v, \ell_l, \ell_r, \perp) \| L] \text{ where } L \in \mathcal{L}\}$.

iv. Mark l as finished, *i.e.*, remove l from U and mark v as unfinished, *i.e.*, add v to U.

4. Once the set of unfinished nodes consists only of the root-node (*i.e.*, $U = \{\epsilon\}$), terminate and output $\pi \leftarrow (\ell_\epsilon, \mathcal{L}_\epsilon)$.

$\underline{\mathsf{Inc}^{\mathsf{H},\mathsf{H}',\mathsf{H}''}(\chi, N, N', \pi)}$: Defined as in Section 4.2.

$\underline{\mathsf{Vf}^{\mathsf{H},\mathsf{H}',\mathsf{H}''}(\chi, N, \pi)}$:

1. Parse π as $(\ell_\epsilon, \mathcal{L}_\epsilon)$.
2. For all paths path $\in \mathcal{L}_\epsilon$ do the following:

(a) Parse path as $[(v_0, \ell_{l_0}, \ell_{r_0}, \mathsf{ind}_0) \| \ldots \| (v_n, \ell_{l_n}, \ell_{r_n}, \mathsf{ind}_n)]$.

(b) For every node $v \in \{v_0, \ldots, v_n\}$ on the path, check if the label ℓ_v was computed correctly. That is, for $v = 0^n$ check whether $\ell_v = 0^\lambda$, and for any other node $v \in V \backslash \{0^n\}$ check whether $\ell_v = \mathsf{H}_{(\chi,v)}(\ell_{v_1}, \ldots, \ell_{v_d})$, where $v_1, \ldots, v_d$ are the nodes with edges pointing to v. The value ℓ_v can either be retrieved from the parent node of v, or is directly available for the case of the root-node ϵ. For the special case of leaf-nodes, the values $\ell_{v_1}, \ldots, \ell_{v_d}$ are not stored locally with the node v, but are stored at some other (a-priori known) nodes along path (refer to the structure of the graph CP_n).

(c) For all $j \in \{0, \ldots, n^*\}$:

i. If v_j is a right node or $j = 0$: Compute $\mathsf{r}_{v_j} \leftarrow \mathsf{H}'_{(\chi,v_j)}(\ell_{v_j})$ and $S_{v_j} \leftarrow \mathsf{RandomSubset}(2t, t; \mathsf{r}_{v_j})$. If v_{j+1} is the left child of v_j, check if

$$S_u[\mathsf{ind}_j] = \mathsf{ind}_{j+1}.$$

Otherwise, if v_{j+1} is the right child of v_j, check if

$$S_u[\mathsf{ind}_j] = t + \mathsf{ind}_{j+1}.$$

ii. If v_j is a left node: Compute $\mathsf{r}_{v_j} \leftarrow \mathsf{H}''_{(\chi,v_j)}(\ell_{v_j})$ and $S_{v_j} \leftarrow \mathsf{RandomPath}(v_j; \mathsf{r}_{v_j})$. Check if all paths in S_{v_j} are present in $\mathcal{L}_\epsilon$.

3. If all checks pass output 1, otherwise 0.

5.3 Efficiency Analysis

The verifier efficiency is essentially unchanged from the construction in Section 4.2.

Prover Efficiency. For the main thread the prover complexity is identical to our construction in Section 4.2. For the parallel threads the prover has to recompute a CP_n graph of size at most n, so we can again upper bound their memory complexity to $\lambda(n+1)$ by Lemma 1.

In the following we argue that the number of parallel threads of our protocol is upper-bounded by n. Recall that a new thread is spawned each time the main thread traverses a left node v (*i.e.*, a node which is the left child of its parent). The complexity of each parallel thread is dominated by the factor $O(2^{h_v})$ of the function $\mathsf{FetchPath}$, where h_v is the height at which the thread was spawned. However, note that the main thread must perform at least $O(2^{h_v})$ steps before spawning a new sub-thread at height h_v. This implies that for each $h_v = 1, \ldots, n$ there can be at most one parallel thread running. It follows that n parallel processors are sufficient to run the prover algorithm.

Proof Size. As for our construction in Section 4.2, the proof size is $O(t \cdot \lambda \cdot n)$. Theorem 3 shows that our construction is sound if $t = O(\lambda)$, which gives proofs of size $O(\lambda^2 \cdot n)$. Concretely, our proofs are larger than those of the CP scheme by a factor of roughly 9.

5.4 Soundness

Theorem 3. *The construction given in Section 5.2 is sound for any $t \in O(\lambda)$, and the soundness error is given by $\frac{1+q(q-1)}{2^\lambda} + q \cdot e^{-\frac{2\alpha^2 t}{9}}$.*

Proof. Let χ be the challenge statement and let q_v be the number of calls of $\mathcal{A}$ to the random oracle $\mathsf{H}'_{(\chi,v)}$, *i.e.*, the adversary makes at most $q = \sum_{v \in \mathfrak{T}} q_v$ calls to H' in total. Let η be a free (positive) variable to be fixed later. Consider the following sequence of hybrids.

- Hybrid $\mathcal{H}_0$: This is identical to the real experiment.
- Hybrid $\mathcal{H}_1$: The same as $\mathcal{H}_0$, except for the following modifications.
  - The experiment records a list Q of all H queries made by $\mathcal{A}$ with their responses.
  - Every time $\mathcal{A}$ queries $\mathsf{H}'_{(\chi,v)}$ for a $v \in V$ with a label ℓ_v, a labelling L_v for the sub-tree under v is computed via $\mathsf{L}_v \leftarrow \mathsf{Extract}(v, \ell_v, Q)$.
  - If it holds for any path opened by $\mathcal{A}$ that the labels on the path are different from the labels in L_ϵ (where ϵ is the root), then $\mathcal{H}_1$ aborts and outputs 0.

Let BAD_v be the following event: $\mathcal{A}$ queries $\mathsf{H}'_{(\chi,v)}$ with a query $\hat{\ell}_v$ corresponding to a labeled sub-tree $\mathsf{L}_v \leftarrow \mathsf{Extract}(v, \hat{\ell}_v, Q)$ for which it holds that $\delta(\mathsf{L}_v) < \gamma(\mathsf{L}_v) - \eta$.

For $v \in \{1^{n^*-1} \| 0, \ldots, 10, 0\}$ define the following hybrids.

- Hybrid $\mathcal{H}_1^v$: The same as the previous hybrid, except that the experiment outputs 0 if the event BAD_v happens.

Let $\hat{\mathsf{BAD}}_v$ be the following event: $\mathcal{A}$ queries $\mathsf{H}'_{(\chi,v)}$ with a query $\hat{\ell}_v$ corresponding to a labeled sub-tree $\mathsf{L}_v \leftarrow \mathsf{Extract}(v, \hat{\ell}_v, Q)$ for which it holds that $\delta(\mathsf{L}_v) < \gamma(\mathsf{L}_v) - \left(3\eta - 2^{n^*-n_v}\eta\right)$, where n_v is the depth of v (*i.e.*, the distance between the root-node ϵ and v).

For $v \in \{1^{n^*}, \ldots, 1, \epsilon\}$ define the following hybrids.

- Hybrid $\mathcal{H}_2^v$: The same as the previous hybrid, except that the experiment outputs 0 if the event $\hat{\mathsf{BAD}}_v$ happens.

We will now show indistinguishability between the hybrids. By Lemma 4 it holds that $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable. We now turn to the indistinguishability of hybrids $\mathcal{H}_1^v$. For notational convenience, let $\mathcal{H}_1^{v\downarrow}$ be the hybrid before $\mathcal{H}_1^v$.

First consider $v = 1^{n^*-1}\|0$. For each node v at level n it holds that the set Z_v of challenge paths consists of all paths from v to the leaves under v. Consequently, it holds that $\delta(\mathsf{L}_v) = \gamma(\mathsf{L}_v)$ and therefore BAD_v happens with probability 0.

First notice that, conditioned on that the event BAD_v does not happen, hybrid $\mathcal{H}_1^v$ is distributed identically to the previous hybrid, *i.e.*, $\Pr\left[\mathcal{H}_1^v(\mathcal{A}) = 1 | \neg\mathsf{BAD}_v\right] = \Pr\left[\mathcal{H}_1^{v\downarrow}(\mathcal{A}) = 1 | \neg\mathsf{BAD}_v\right]$. Therefore

$$\begin{aligned}\mathbb{SD}[\mathcal{H}_1^v, \mathcal{H}_1^{v\downarrow}] &= \Pr\left[\mathsf{BAD}_v\right] \cdot \underbrace{\left|\Pr\left[\mathcal{H}_1^v(\mathcal{A}) = 1 | \mathsf{BAD}_v\right] - \Pr\left[\mathcal{H}_1^{v\downarrow}(\mathcal{A}) = 1 | \mathsf{BAD}_v\right]\right|}_{\leq 1} \\ &\leq \Pr\left[\mathsf{BAD}_v\right]\end{aligned}$$

It is thus sufficient to bound the probability for the event BAD_v. $\mathcal{A}$ queries the random oracle $\mathsf{H}'_{(\chi,v)}$ with at most q_v distinct queries. Note that v is always a left node and therefore the challenge set Z is chosen uniformly at random for each label. Hence we have that $\mathsf{E}[\delta(\mathsf{L}_v)] = \gamma(\mathsf{L}_v)$, *i.e.*, the fraction of inconsistent paths is preserved in expectation, over the random coins of $\mathsf{H}'_{(\chi,v)}$. We can then rewrite

$$\begin{aligned}\Pr\left[\mathsf{BAD}_v\right] &= \Pr\left[\delta(\mathsf{L}_v) < \gamma(\mathsf{L}_v) - \eta\right] \\ &= \Pr\left[\delta(\mathsf{L}_v) < \mathsf{E}[\delta(\mathsf{L}_v)] - \eta\right] \\ &< e^{-2\eta^2 t}\end{aligned}$$

by Theorem 1. A union-bound over all queries to $\mathsf{H}_{(\chi,v)}$ yields

$$\Pr\left[\mathsf{BAD}_v\right] < q_v \cdot e^{-2\eta^2 t}.$$

Thus we conclude that the statistical distance between $\mathcal{H}_1^v$ and $\mathcal{H}_1^{v\downarrow}$ is at most $q_v \cdot e^{-\eta^2 t}$. We now turn to the indistinguishability of hybrids $\mathcal{H}_2^v$. Again we use the convention that $\mathcal{H}_2^{v\downarrow}$ denotes the hybrid before $\mathcal{H}_2^v$.

First consider $v = 1^{n^*}$. As argued above, for each node at depth n it holds that $\delta(\mathsf{L}_v) = \gamma(\mathsf{L}_v)$ and therefore $\hat{\mathsf{BAD}}_v$ happens with probability 0. For the rest of the cases, bounding the probability that $\hat{\mathsf{BAD}}_v$ happens suffice, since, if $\hat{\mathsf{BAD}}_v$ does not happen, the hybrids are identical. We bound the probability that $\hat{\mathsf{BAD}}_v$ happens with an inductive argument over $v \in \{1^{n^*}, \dots, 1, \epsilon\}$. The base case $v = 1^{n^*}$ is settled above.

For any node $v \in \{1^{n^*-1}, \dots, 1, \epsilon\}$, fix a query $\hat{\ell}_v$ and let l and r be the left and right child of v. Since l is a left node, we have that

$$\delta(\mathsf{L}_l) \geq \gamma(\mathsf{L}_l) - \eta \tag{6}$$

as otherwise BAD_l would be triggered. For the right node r we have that

$$\delta(\mathsf{L}_r) \geq \gamma(\mathsf{L}_r) - \left(3\eta - 2^{n^*-(n_v+1)}\eta\right) \tag{7}$$

by induction hypothesis, as otherwise $\hat{\mathsf{BAD}}_r$ would be triggered. We can now rewrite

$$\begin{aligned}
\mathsf{Pr}\left[\hat{\mathsf{BAD}}_v\right] &= \mathsf{Pr}\left[\delta(\mathsf{L}_v) < \gamma(\mathsf{L}_v) - \left(3\eta - 2^{n^*-n_v}\eta\right)\right] \\
&= 1 - \mathsf{Pr}\left[\gamma(\mathsf{L}_v) \leq \delta(\mathsf{L}_v) + \left(3\eta - 2^{n^*-n_v}\eta\right)\right] \\
&< e^{-2\left(\left(3\eta - 2^{n^*-n_v}\eta\right) - \frac{\eta + \left(3\eta - 2^{n^*-(n_v+1)}\eta\right)}{2}\right)^2 t} \\
&= e^{-2\left(3\eta - \frac{(\eta+3\eta)}{2}\right)^2 t} \\
&= e^{-2\eta^2 t}
\end{aligned}$$

by (6), (7), and Lemma 5. A union-bound over all queries to $\mathsf{H}_{(\chi,v)}$ yields

$$\mathsf{Pr}\left[\hat{\mathsf{BAD}}_v\right] \leq q_v \cdot e^{-2\eta^2 t}.$$

This bounds the statistical distance between $\mathcal{H}_2^v$ and $\mathcal{H}_2^{v\downarrow}$ by $q_v \cdot e^{-2\eta^2 t}$.

We are now in the position to bound the statistical distance between the first hybrid $\mathcal{H}_0$ and the last hybrid $\mathcal{H}_2^\epsilon$. Let $\mathfrak{T}_l$ be the set $\{1^{n^*-1}\|0, \dots, 10, 0\}$ and let $\mathfrak{T}_r$ be the set $\{1^{n^*}, \dots, 1, \epsilon\}$

$$\begin{aligned}
\mathbb{SD}[\mathcal{H}_0, \mathcal{H}_2^\epsilon] &= \frac{1 + q(q-1)}{2^\lambda} + \sum_{v \in \{\mathfrak{T}_l \cup \mathfrak{T}_r\}} q_v \cdot e^{-2\eta^2 t} \\
&\leq \frac{1 + q(q-1)}{2^\lambda} + q \cdot e^{-2\eta^2 t}.
\end{aligned}$$

Setting $\eta := \frac{\alpha}{3}$ we obtain

$$\mathbb{SD}[\mathcal{H}_0, \mathcal{H}_2^\epsilon] \leq \frac{1 + q(q-1)}{2^\lambda} + q \cdot e^{-2\frac{\alpha^2 t}{9}}.$$

What is left to be shown is that $\mathcal{A}$ cannot win in $\mathcal{H}_2^\epsilon$. Note that in the latter experiment we have that for all L_ϵ computed via $\mathsf{Extract}$ we have that

$$\delta(\mathsf{L}_\epsilon) \geq \gamma(\mathsf{L}_\epsilon) - \left(3\eta - 2^{n^*}\eta\right) \geq \gamma(\mathsf{L}_\epsilon) - 3\eta = \gamma(\mathsf{L}_\epsilon) - \alpha.$$

The same argument as in the proof of Theorem 2 can be used to show that the success probability of $\mathcal{A}$ is exactly 0. □

General Arity Trees. Both schemes presented in this work can be generalized to work over p-ary trees, for any $p \geq 2$. By adjusting the value p, we can achieve slightly better concrete proof sizes and prover efficiency. We refer the reader to Section A for an extensive treatment on the matter.

References

1. Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.
2. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.
3. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016*, pages 345–356, Cambridge, MA, USA, January 14–16, 2016. ACM.
4. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.
5. Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 451–467, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.
6. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany.
7. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.
8. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732, Victoria, British Columbia, Canada, May 4–6, 1992. ACM Press.
9. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 39–50, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.
10. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 373–388, Berkeley, CA, USA, January 9–12, 2013. ACM.
11. Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453, Santa Fe, New Mexico, November 20–22, 1994. IEEE Computer Society Press.
12. Krzysztof Pietrzak. Simple verifiable delay functions. Cryptology ePrint Archive, Report 2018/627, 2018. https://eprint.iacr.org/2018/627.
13. Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.
14. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany.
15. Benjamin Wesolowski. Efficient verifiable delay functions. Cryptology ePrint Archive, Report 2018/623, 2018. https://eprint.iacr.org/2018/623.

A General Arity Constructions

The schemes described in Section 4.2 and Section 5.2 can be generalized rather easily to work with p-ary trees for any $p \geq 2$.

A.1 Generalized CP Graphs

We begin by describing the generalized CP graph CP_n^p, and generalizing Lemma 1, Lemma 2, and Lemma 3.

Definition 6 (Generalized CP Graphs). *For $n \in \mathbb{N}$, let $N = p^{n+1} - 1$ and $T_{p,n} = (V, E')$ be a complete p-ary tree of depth n. Let $\Sigma := \{0, \ldots, p-1\}$ be an alphabet set of size p. The nodes $V = \Sigma^{\leq n}$ are identified by p-ary strings of length at most n and the empty string ϵ represents the root. The edges $E' = \{(x||s, x) : s \in \Sigma, x \in \Sigma^i, i < n\}$ are directed from the leaves towards the root.*

The graph $CP_n^p = (V, E)$ is a DAG constructed from $T_{p,n} = (V, E')$ as follows. For any leaf $u \in \Sigma^n$, for any node v which is a left-sibling of a node on the path from u to the root ϵ, an edge (v, u) is appended to E'. Formally, $E := E' \cup E''$ where

$$E'' := \{(v, u) : u \in \Sigma^n, u = a||r||a', v = a||s, r > s \text{ for some } a, a' \in \Sigma^{\leq n}\}.$$

We state and prove the generalizations of Lemma 1, Lemma 2, and Lemma 3.

Lemma 6. *The labels of a CP_n^p graph can be computed in topological order using $\lambda((p-1)n+1)$ bits of memory.*

Proof. We prove by induction on n. Let $0, \ldots, p-1$ be the children of ϵ. For $i \in \Sigma = \{0, \ldots, p-1\}$, let $\mathfrak{T}_i$ be the subtree rooted at the i. Note that $\mathfrak{T}_i$ is isomorphic to CP_{n-1}^p. To compute the labels of CP_n^p, we first compute the labels of $\mathfrak{T}_0$. Upon completion, we store only the label of 0, denoted ℓ_0. Next, we compute the labels of $\mathfrak{T}_1$ using ℓ_0. This is possible since all edges start from the node 0. Upon completion, we store the label ℓ_1. Now suppose that for some $i \in \{1, \ldots, p\}$ the labels of $\mathfrak{T}_0, \ldots, \mathfrak{T}_{i-1}$ are computed, and we have stored $\ell_0, \ldots, \ell_{i-1}$. The labels of $\mathfrak{T}_i$ can be computed since all edges start from the nodes $0, \ldots, i-1$. Eventually, we obtain the last label ℓ_{p-1}. Using this with $\ell_0, \ldots, \ell_{p-2}$ stored in the memory, we can compute the label of ϵ.

Since for each $i \in \Sigma$, storing ℓ_i requires λ bits of memory, the memory required for computing the label of CP_n^p equals to that of CP_{n-1}^p plus $\lambda(p-1)$ extra bits. Furthermore, CP_0^p has exactly 1 node and its label can be computed using λ bits of memory. Solving the recursion gives the claimed bound.

Lemma 7. *For all $S \subseteq V$, the subgraph of $CP_n^p = (V, E)$ on vertex set $V \setminus \mathfrak{T}_{S^*}$, has a directed path going through all the $|V| - |\mathfrak{T}_{S^*}|$ nodes.*

Proof. We prove by induction on n. The lemma is trivial for CP_0^p as it contains only 1 node. Now, suppose the lemma is true for CP_{n-1}^p. Consider CP_n^p, and let $0, \ldots, p-1$ be the children of ϵ. For $i \in \Sigma = \{0, \ldots, p-1\}$, let $\mathfrak{T}_i$ be the subtree rooted at the i. Note that $\mathfrak{T}_i$ is isomorphic to CP_{n-1}^p. CP_n^p consists of the root ϵ, the subtrees $\mathfrak{T}_0, \ldots, \mathfrak{T}_{p-1}$, and edges going from i to the leaves of $\mathfrak{T}_j$ for all $i < j$ and $i, j \in \Sigma$.

The lemma is true if $\epsilon \in S^*$, as $|V| - |\mathfrak{T}_{S^*}| = 0$. Otherwise, let $I := S^* \cap \Sigma$ be the subset of children of ϵ which are in S^*. For concreteness, we write $I = \{i_1, \ldots, i_k\}$ for some $k \in \{1, \ldots, p\}$. We apply the lemma to $\mathfrak{T}_i$ for all $i \in \Sigma \setminus I$, so that for each $\mathfrak{T}_i$ there exists a directed path going from the left-most leaf of $\mathfrak{T}_i$, *i.e.*, $i0\ldots0$, to i. Since for all $i, j \in \Sigma$ where $i < j$, there exists an edge from i to $j0\ldots0$, it means that for each $i' \in I$, there exists a edge $(i'-1, (i'+1)0\ldots0)$ which "skips" $\mathfrak{T}_{i'}$. Formally, the following edges exist:

$$\begin{gathered}(0, 10\ldots0), \ldots, (i_1 - 2, (i_1-1)0\ldots0),\\ (i_1 - 1, (i_1+1)0\ldots0), \ldots, (i_k - 1, (i_k+1)0\ldots0),\\ (i_k + 1, (i_k+2)0\ldots0), \ldots, (p-1, p0\ldots0).\end{gathered}$$

Finally, we note that there also exists an edge (i^*, ϵ) where $i^* := \max_{i \notin I}(i \in \Sigma)$, which completes the path from $0\ldots0$ to ϵ, passing through all $|V| - |\mathfrak{T}_{S^*}|$ nodes.

Lemma 8. *For all $S \subset V$, $\mathfrak{T}_{S^*}$ contains $\frac{|\mathfrak{T}_{S^*}|+|S|}{p}$ many leaves.*

Proof. Let $S^* = \{v_1, \ldots, v_k\}$. Since S^* is minimal, it holds that $\mathfrak{T}_{v_i} \cap \mathfrak{T}_{v_j} = \emptyset$ for all $i, j \in \{1, \ldots, k\}$ with $i \neq j$. Therefore we can write

$$|\Sigma^n \cap \mathfrak{T}_{S^*}| = \sum_{i=1}^{k} |\Sigma^n \cap \mathfrak{T}_{v_i}|.$$

As for all $i \in \{1, \ldots, k\}$, $\mathfrak{T}_{v_i}$ is a complete p-ary tree, it has $(|\mathfrak{T}_{v_i}| + 1)/p$ many leaves. Thus,

$$\sum_{i=1}^{k} |\Sigma^n \cap \mathfrak{T}_{v_i}| = \sum_{i=1}^{k} \frac{|\mathfrak{T}_{v_i}| + 1}{p} = \frac{|\mathfrak{T}_{S^*}| + |S|}{p}.$$

A.2 Generalized Single-Thread Construction

The generalized construction is almost identical to the basic one presented in Section 4.2, except the graph CP_n is replaced with CP_n^p, and the computation of the labels is changed accordingly.

$\underline{\mathsf{Prove}^{\mathsf{H},\mathsf{H}'}(\chi, N)}$:

```
1. Initialize $U \leftarrow \emptyset$.
2. Assign $\ell_{0^n} \leftarrow 0^\lambda$.
3. Traverse the graph $CP_n^p = (V, E)$ starting from $0^n$. At every node $v \in V$ which is
   traversed, do the following:
   (a) Compute the label $\ell_v$ by $\ell_v \leftarrow \mathsf{H}_{(\chi,v)}(\ell_{v_1}, \ldots, \ell_{v_d})$, where $v_1, \ldots, v_d \in V$ are
       all nodes with edges pointing to $v$, i.e., $(v_i, v) \in E$.
   (b) Let $c_0, \ldots, c_{p-1}$ be the children of $v$.
   (c) If $|\mathsf{leafs}(v)| \leq t$, set
           $\mathcal{L}_v \leftarrow \{[(v, \ell_{c_0}, \ldots, \ell_{c_{p-1}}, \perp) \| L] \text{ where } L \in \mathcal{L}_{c_0} \cup \ldots \cup \mathcal{L}_{c_{p-1}}\}.$
```

    (d) Otherwise (*i.e.*, if $|\mathsf{leafs}(v)| \geq pt$), do the following:
        i. Compute $\mathsf{r}_v \leftarrow \mathsf{H}'_{(\chi,v)}(\ell_v)$.
        ii. Choose a random t-subset S_v of $[pt]$ via $S_v \leftarrow \mathsf{RandomSubset}(pt, t; \mathsf{r}_v)$.
        iii. For $j \in \{0, \dots, t-1\}$, write $S_v[j] = at + b$ where $0 \leq a < p$ and $0 \leq b < t$ and set $\mathcal{L}_v[j] \leftarrow (v, \ell_{c_0}, \dots, \ell_{c_{p-1}}, j) \| \mathcal{L}_{c_a}[b]$.
    (e) Mark $c_0, \dots, c_{p-2}$ as finished, *i.e.*, remove $c_0, \dots, c_{p-2}$ from U and, if v is not the right-most child of its parent, mark v as unfinished, *i.e.*, add v to U.
4. Once the set of unfinished nodes consists only of the root-node (*i.e.*, $U = \{\epsilon\}$), terminate and output $\pi \leftarrow (\ell_\epsilon, \mathcal{L}_\epsilon)$.

$\underline{\mathsf{Inc}^{\mathsf{H},\mathsf{H}'}(\chi, N, N', \pi)}$:

1. Initialize $U := \emptyset$.
2. Parse π as $(\ell_\epsilon, \mathcal{L}_\epsilon)$
3. Assign $\ell_{0^{n'-n}} := \ell_\epsilon$ and $\mathcal{L}_{0^{n'-n}} := \mathcal{L}_\epsilon$.
4. Execute the algorithm $\mathsf{Prove}^{\mathsf{H},\mathsf{H}'}(\chi, N')$ starting from step 3 with a slight change: Traverse the graph $CP^p_{n'}$ starting from $0^{n'-n-1}\|1\|0^n$ (instead of from $0^{n'}$).

$\underline{\mathsf{Vf}^{\mathsf{H},\mathsf{H}'}(\chi, N, \pi)}$:

1. Parse $\pi = (\ell_\epsilon, \mathcal{L}_\epsilon)$.
2. For all paths $\mathsf{path} \in \mathcal{L}_\epsilon$ do the following:
    (a) Parse path as $[(v_0, \ell_{c_{0,0}}, \dots, \ell_{c_{0,p-1}}, \mathsf{ind}_0), \dots, (v_n, \ell_{c_{n,0}}, \dots, \ell_{c_{n,p-1}}, \mathsf{ind}_n)]$.
    (b) For every node $v \in \{v_0, \dots, v_n\}$ on the path, check if the label ℓ_v was computed correctly. That is, for $v = 0^n$ check whether $\ell_v = 0^\lambda$, and for any other node $v \in V \backslash \{0^n\}$ check whether $\ell_v = \mathsf{H}_{(\chi,v)}(\ell_{v_1}, \dots, \ell_{v_d})$, where $\ell_{v_1}, \dots, \ell_{v_d}$ are the nodes with edges pointing to v. The value ℓ_v can either be retrieved from the parent node of v, or is directly available for the case of the root-node ϵ. For the special case of leaf-nodes, the values $\ell_{v_1}, \dots, \ell_{v_d}$ are not stored locally with the node v, but are stored at some other (a-priori known) nodes along the path path (refer to the structure of the graph CP^p_n).
    (c) For all $j \in \{0, \dots, n^*\}$, compute $\mathsf{r}_{v_j} \leftarrow \mathsf{H}'_{(\chi,v_j)}(\ell_{v_j})$ and $S_{v_j} \leftarrow \mathsf{RandomSubset}(pt, t; \mathsf{r}_{v_j})$. Let $i \in \{0, \dots, p-1\}$ so that v_{j+1} is the i-th child of v_j. Check if $S_v[\mathsf{ind}_j] = i \cdot t + \mathsf{ind}_{j+1}$.
3. If all checks pass then output 1. Otherwise output 0.

We state the soundness error and the efficiency of the generalized construction. The analysis is essentially identical to that in Section 4.3 and is therefore omitted.

Soundness. Here we state a generalized version of Lemma 5 for p-ary trees.

Lemma 9. *Let v be a node and let $(v_1, \dots, v_p)$ the set of children of v. If for all $i \in \{1, \dots, p\}$ we have*

$$\delta(\mathsf{L}_{v_i}) \geq \gamma(\mathsf{L}_{v_i}) - \eta_{v_i}$$

then it holds that

$$\Pr\left[\gamma(\mathsf{L}_v) \leq \delta(\mathsf{L}_v) + \eta_v\right] \geq 1 - e^{-2\left(\eta_v - \frac{\sum_{i \in p} \eta_{v_i}}{p}\right)^2 t}.$$

The bound for the soundness error has the same form as that in the basic construction, except that $n = \log_p(N+1)-1$. Previously, $n = \log(N+1)-1$. The proof is identical to that of Theorem 2, except that we apply Lemma 9 instead of Lemma 5.

Theorem 4. *The construction given in Section A.2 is sound for any $t \in O(\lambda \cdot n^2)$, and the soundness error is given by $\frac{1+q(q-1)}{2^\lambda} + q \cdot e^{-2(\frac{\alpha}{n})^2 t}$.*

Efficiency. In the following, we set $t = O(\lambda \cdot n^2)$ and $n = \log_p N$. The parallel time complexity of the prover remains unchanged at $O(N)$. The parallel time complexity of the verifier is $O(\log(\frac{1}{\log^3 p} \cdot \lambda \cdot \log^3 N))$, which decreases at p increases. The proof size and the space complexity of the prover are $O(\frac{p}{\log^3 p} \cdot \lambda^2 \cdot \log^3 N)$ and $O(\frac{p^2}{\log^4 p} \cdot \lambda^2 \cdot \log^4 N)$ respectively. The fractions $\phi_p := \frac{p}{\log^3 p}$ and $\theta_p := \frac{p^2}{\log^4 p}$ are minimized at $p = 20$ and $p = 7$ respectively. Compared to $p = 2$, we have $\phi_{20}/\phi_2 \approx 0.124$ and $\theta_7/\theta_2 \approx 0.197$.

A.3 Generalized Multi-Thread Construction

Similar to the above, we present a generalization of the construction in Section 5.2.

$\underline{\mathsf{Prove}^{\mathsf{H},\mathsf{H}',\mathsf{H}''}(\chi, N)}$:

1. Initialize $U \leftarrow \emptyset$ to be the set of unfinished nodes.
2. Assign $\ell_{0^n} \leftarrow 0^\lambda$.
3. Traverse the graph CP_n^p starting from 0^n. At every node $v \in V$ which is traversed, do the following:
   (a) Compute the label ℓ_v by $\ell_v \leftarrow \mathsf{H}_{(\chi,v)}(\ell_{v_1}, \ldots, \ell_{v_d})$, where $v_1, \ldots, v_d \in V$ are all nodes nodes v is adjacent with, *i.e.*, $(v_i, v) \in E$.
   (b) Let $c_0, \ldots, c_{p-1}$ be the children of v.
   (c) If $|\mathsf{leafs}(v)| \le t$, set
   $$\mathcal{L}_v \leftarrow \{[(v, \ell_{c_0}, \ldots, \ell_{c_{p-1}}, \perp) \| L] \text{ where } L \in \mathcal{L}_{c_0} \cup \ldots \cup \mathcal{L}_{c_{p-1}}\}.$$
   (d) Otherwise (*i.e.*, if $|\mathsf{leafs}(v)| \ge pt$), do the following:
      i. Compute $\mathsf{r}_v \leftarrow \mathsf{H}'_{(\chi,v)}(\ell_v)$.
      ii. Choose a random t-subset S_v of $[pt]$ via $S_v \leftarrow \mathsf{RandomSubset}(pt, t; \mathsf{r}_v)$.
      iii. For $j \in \{0, \ldots, t-1\}$, write $S_v[j] = at + b$ where $0 \le a < p$ and $0 \le b < t$. Set $\mathcal{L}_v[j] := [(v, \ell_l, \ell_r, j) \| \mathcal{L}_{c_a}[b]]$.
   (e) If v is not a right node (*i.e.*, it is not the right-most child of its parent):
      i. Compute $\mathsf{r}_v \leftarrow \mathsf{H}''_{(\chi,v)}(\ell_v)$.
      ii. Choose a random t-set of paths with prefix v via $S_v \leftarrow \mathsf{RandomPath}(v; \mathsf{r}_v)$.
      iii. Execute in a parallel thread $\mathcal{L} \leftarrow \mathsf{FetchPath}(S_v, U, \{\ell_v : v \in U\})$ and set $\mathcal{L}_v := \{[(v, \ell_l, \ell_r, \perp) \| L] \text{ where } L \in \mathcal{L}\}$.
      iv. Mark $c_0, \ldots, c_{p-2}$ as finished, *i.e.*, remove $c_0, \ldots, c_{p-2}$ from U and mark v as unfinished, *i.e.*, add v to U.
4. Once the set of unfinished nodes consists only of the root-node (*i.e.*, $U = \{\epsilon\}$), terminate and output $\pi \leftarrow (\ell_\epsilon, \mathcal{L}_\epsilon)$.

$\underline{\mathsf{Inc}^{\mathsf{H},\mathsf{H}',\mathsf{H}''}(\chi, N, N', \pi)}$: Defined as in Section A.2.

$\underline{\mathsf{Vf}^{\mathsf{H},\mathsf{H}',\mathsf{H}''}(\chi, N, \pi)}$:

1. Parse π as $(\ell_\epsilon, \mathcal{L}_\epsilon)$.
2. For all paths path $\in \mathcal{L}_\epsilon$ do the following:
   (a) Parse path as $[(v_0, \ell_{c_{0,0}}, \ldots, \ell_{c_{0,p-1}}, \mathsf{ind}_0) \| \ldots \| (v_n, \ell_{c_{n,0}}, \ldots, \ell_{c_{n,p-1}}, \mathsf{ind}_n)]$.
   (b) For every node $v \in \{v_0, \ldots, v_n\}$ on the path, check if the label ℓ_v was computed correctly. That is, for $v = 0^n$ check whether $\ell_v = 0^\lambda$, and for any other node $v \in V \backslash \{0^n\}$ check whether $\ell_v = \mathsf{H}_{(\chi,v)}(\ell_{v_1}, \ldots, \ell_{v_d})$, where $v_1, \ldots, v_d$ are the nodes with edges pointing to v. The value ℓ_v can either be retrieved from the parent node of v, or is directly available for the case of the root-node ϵ. For the special case of leaf-nodes, the values $\ell_{v_1}, \ldots, \ell_{v_d}$ are not stored locally with the node v, but are stored at some other (a-priori known) nodes along path (refer to the structure of the graph CP_n^p).
   (c) For all $j \in \{0, \ldots, n^*\}$:
      i. If v_j is the right-most child of its parent or $j = 0$: Compute $\mathsf{r}_{v_j} \leftarrow \mathsf{H}'_{(\chi,v_j)}(\ell_{v_j})$ and $S_{v_j} \leftarrow \mathsf{RandomSubset}(pt, t; \mathsf{r}_{v_j})$. Let v_{j+1} be the i-th child of v_j, check if $S_v[\mathsf{ind}_j] = i \cdot t + \mathsf{ind}_{j+1}$.
      ii. If v_j is not the right-most child of its parent: Compute $\mathsf{r}_{v_j} \leftarrow \mathsf{H}''_{(\chi,v_j)}(\ell_{v_j})$ and $S_{v_j} \leftarrow \mathsf{RandomPath}(v_j; \mathsf{r}_{v_j})$. Check if all paths in S_{v_j} are present in $\mathcal{L}_\epsilon$.
3. If all checks pass output 1, otherwise 0.

Next we state the soundness error and the efficiency.

Soundness. The soundness analysis requires some tweaking of the argument.

Theorem 5. *The construction given in Section A.3 is sound for any $t \in O((1+\frac{p}{p-1})^2 \cdot \lambda)$, and the soundness error is given by $\frac{1+q(q-1)}{2^\lambda} + q \cdot e^{-\left(\frac{\alpha}{1+\frac{p}{p-1}}\right)^2 t}$.*

Proof. The proof follows the blueprint of the proof of Theorem 3, except for the following changes. First we add a hybrid $\mathcal{H}_1^v$ for each sibling of the nodes $\{1^{n^*}, \ldots, 1, \epsilon\}$. The indistinguishability arguments are identical.

Then we define the event $\widehat{\mathsf{BAD}}_v$ as follows: $\mathcal{A}$ queries $\mathsf{H}'_{(\chi,v)}$ with a query $\hat{\ell}_v$ corresponding to a labeled sub-tree $\mathsf{L}_v \leftarrow \mathsf{Extract}(v, \hat{\ell}_v, Q)$ for which it holds that $\delta(\mathsf{L}_v) < \gamma(\mathsf{L}_v) - \left(2\eta + \eta \sum_{i=1}^{n^*-n_v} \frac{1}{p^i}\right)$, where n_v is the depth of v.

We bound the probability that $\widehat{\mathsf{BAD}}_v$ happens with an inductive argument over $v \in \{1^{n^*}, \ldots, 1, \epsilon\}$. For the base case $v = 1^{n^*}$ is enough to observe that $\delta(\mathsf{L}_v) = \gamma(\mathsf{L}_v)$ and therefore $\widehat{\mathsf{BAD}}_v$ happens with probability 0.

For any node $v \in \{1^{n^*-1}, \ldots, 1, \epsilon\}$, fix a query $\hat{\ell}_v$ and let $(v_1, \ldots, v_p)$ be the children of v. For all $i \in \{1, \ldots, p-1\}$ we have that

$$\delta(\mathsf{L}_{v_i}) \geq \gamma(\mathsf{L}_{v_i}) - \eta \tag{8}$$

as otherwise BAD_{v_i} would be triggered. For the node v_p we have that

$$\delta(\mathsf{L}_{v_p}) \geq \gamma(\mathsf{L}_{v_p}) - \left(2\eta + \eta \sum_{i=1}^{n^*-n_v-1} \frac{1}{p^i}\right) \tag{9}$$

by induction hypothesis, as otherwise $\hat{\mathsf{BAD}}_{v_p}$ would be triggered. We can now rewrite

$$\begin{aligned}
\mathsf{Pr}\left[\hat{\mathsf{BAD}}_v\right] &= \mathsf{Pr}\left[\delta(\mathsf{L}_v) < \gamma(\mathsf{L}_v) - \left(2\eta + \eta \sum_{i=1}^{n^*-n_v} \frac{1}{p^i}\right)\right] \\
&= 1 - \mathsf{Pr}\left[\gamma(\mathsf{L}_v) \leq \delta(\mathsf{L}_v) + \left(2\eta + \eta \sum_{i=1}^{n^*-n_v} \frac{1}{p^i}\right)\right] \\
&< e^{-2\left(\left(2\eta+\eta\sum_{i=1}^{n^*-n_v}\frac{1}{p^i}\right) - \frac{\eta(p-1)+\left(2\eta+\eta\sum_{i=1}^{n^*-n_v-1}\frac{1}{p^i}\right)}{p}\right)^2 t} \\
&= e^{-2\eta^2 t}
\end{aligned}$$

by (8), (9), and Lemma 9. For $p > 1$ we can bound

$$2\eta + \eta \sum_{i=1}^{n^*} \frac{1}{p^i} = \eta + \eta \sum_{i=0}^{n^*} \frac{1}{p^i} \leq \left(1 + \frac{p}{p-1}\right)\eta.$$

since it is a geometric series. Thus we can set $\eta := \frac{\alpha}{(1+\frac{p}{p-1})}$ and derive

$$\mathbb{SD}[\mathcal{H}_0, \mathcal{H}_2^\epsilon] \leq \frac{1 + q(q-1)}{2^\lambda} + q \cdot e^{-\frac{2\alpha^2 t}{(1+\frac{p}{p-1})^2}}.$$

The remainder of the analysis is unchanged. □

Efficiency. In the following, we set $t = O\left(\left(1 + \frac{p}{p-1}\right)^2 \cdot \lambda\right)$ and $n = \log_p N$. The parallel time complexity of the prover remains unchanged at $O(N)$. The number of parallel threads is bounded by $O(p \log_p N)$, which is minimized at $p = 3$. The parallel time complexity of the verifier is $O(\log(\frac{(1+\frac{p}{p-1})^2}{\log p} \cdot \lambda \cdot \log N))$, which decreases at p increases. The proof size and the space complexity of the prover are $O(\frac{p(1+\frac{p}{p-1})^2}{\log p} \cdot \lambda^2 \cdot \log N)$ and $O(\frac{p^2(1+\frac{p}{p-1})^2}{\log^2 p} \cdot \lambda^2 \cdot \log^2 N)$ respectively. The fractions $\phi'_p := \frac{p(1+\frac{p}{p-1})^2}{\log p}$ and $\theta'_p := \frac{p^2(1+\frac{p}{p-1})^2}{\log^2 p}$ are both minimized at $p = 4$. Compared to $p = 2$, we have $\phi'_4/\phi'_2 \approx 0.605$ and $\theta'_7/\theta'_2 \approx 0.605$.