SNACKs: Leveraging Proofs of Sequential Work for Blockchain Light Clients

Hamza Abusalah[1], Georg Fuchsbauer[2], Peter Gaži[3], and Karen Klein[4]

[1] IMDEA Software Institute, hamza.abusalah@imdea.org
[2] TU Wien, georg.fuchsbauer@tuwien.ac.at
[3] IOG, peter.gazi@iohk.io
[4] ETH Zurich, karen.klein@inf.ethz.ch

Abstract. The success of blockchains has led to ever-growing ledgers that are stored by all participating *full nodes*. In contrast, *light clients* only store small amounts of blockchain-related data and rely on the mediation of full nodes when interacting with the ledger. A broader adoption of blockchains calls for protocols that make this interaction trustless.

We revisit the design of light-client blockchain protocols from the perspective of classical proof-system theory, and explain the role that proofs of sequential work (PoSWs) can play in it. To this end, we define a new primitive called *succinct non-interactive argument of chain knowledge (SNACK)*, a non-interactive proof system that provides clear security guarantees to a verifier (a light client) even when interacting only with a single dishonest prover (a full node). We show how augmenting any blockchain with any *graph-labeling PoSW* (GL-PoSW) enables SNACK proofs for this blockchain. We also provide a unified and extended definition of GL-PoSWs covering all existing constructions, and describe two new variants. We then show how SNACKs can be used to construct light-client protocols, and highlight some deficiencies of existing designs, along with mitigations. Finally, we introduce *incremental* SNACKs which could potentially provide a new approach to *light mining*.

Keywords: Blockchains · Light clients · Proofs of sequential work

1 Introduction

Since the appearance of the seminal Bitcoin whitepaper [Nak08] and the subsequent launch of its implementation maintaining the Bitcoin ledger, blockchain technology has witnessed enormous growth in adoption.

However, this remarkable success also uncovered some of the deficiencies of the original Bitcoin protocol and its derivatives. Their objective is to maintain an append-only ledger of transactions that records the full financial (or computational) history of the system, and the size of this ledger therefore grows with speed proportional to the use of the system. For example, the Bitcoin and Ethereum blockchains both consist of hundreds of gigabytes. This makes maintain the full blockchain unattractive for ordinary users, and the requirement to do so would be prohibitive to a wider adoption of these systems.

Light-client blockchain protocols. The above development results in an urgent need for solutions that enable interaction with the blockchain for so-called *light clients*[5] that do not store the entire blockchain. This interaction is typically mediated by so-called *full nodes* that store the full blockchain state. This mediation should be *trustless* in that the light client is provided security guarantees without having to assume the honesty of the full node(s) it interacts with.

[5] The term *light node* or *light client* is sometimes used solely to refer to nodes adopting SPV (see below); we mean by it any node that does not store the full blockchain.

Trustless light-client protocols can power a variety of applications within the blockchain ecosystem. The basic one is *bootstrapping*, where a light client, holding only an authentic copy of the genesis block, tries to obtain a reliable picture of the current ledger state (or a commitment to it), that would then enable further interaction with the ledger such as verifying *transaction inclusion* or even contributing to extending the chain (called *light mining* [KLZ21]). Interestingly, light-client techniques also find applications in *cross-chain communication protocols* [BCD+14, GKZ19, KZ19], where the goal is to communicate the occurrence of an event on a source chain to an independent target chain: here the (validator of the) target chain plays the role of a light client for the source chain, seeking to verify the occurrence of that event without validating the entire source chain.

The need for light-client protocols was already predicted in the Bitcoin whitepaper, where so-called *simplified payment verification (SPV)* is proposed: the light client downloads only the block headers (which contain the proof-of-work solutions) from a full node; these suffice to trustlessly verify the amount of work invested to produce that chain; inclusion of individual transactions can then be verified by specifically asking for the openings of the respective Merkle-tree commitments contained in the block headers. Alas, while practically helpful, SPV still requires storage and communication linear in the length of the blockchain and hence provides no asymptotic improvement. On the other end of the spectrum are solutions based on *succinct non-interactive arguments of knowledge (SNARKs)* [GGPR13] that provide impressive asymptotic improvements, but often suffer from unfavorable concrete efficiency, reliance on a trusted setup, or on novel hardness assumptions.

NIPoPoWs. Given the initial success of proof-of-work (PoW) blockchains, there has been significant effort towards developing practical light-client protocols for PoW, aiming at *sublinear* (in the length of the blockchain) communication, while relying only on basic and efficient cryptographic building blocks, and requiring no additional trust assumptions. This has led to two main constructions: superblock-based non-interactive proofs of proof-of-work (NIPoPoWs) [KMZ20] and FlyClient [BKLZ20]. In greater detail, [KMZ20] provides a definition of the NIPoPoW primitive and an instantiation based on so-called superblocks: blocks that contain a PoW that is "stronger than needed", as it would remain valid also against a more restrictive difficulty threshold. Their technique is leveraged to enable light mining [KLZ21]; unfortunately, the approach only guarantees succinctness of the provided proofs if the adversary is limited to 1/3 of the total hash rate in the system and is only analyzed in the static-difficulty setting. On the other hand, FlyClient also instantiates the NIPoPoW primitive, is proven secure for any sub-1/2 adversary, provides significantly better efficiency, and is analyzed also in the variable-difficulty setting.

It appears natural that underlying any of these light-client protocols must be a classical two-party proof system which allows a *prover*, representing a full node, to convince a *verifier*, the light client, of its knowledge of a blockchain that it commits to. However, the protocols [KMZ20, BKLZ20] are not interpreted in this way: they were designed in a model where a light client is assumed to be connected to *multiple* provers, and the soundness guarantees are formulated only under the assumption that at least one of them is honest. This might appear unsatisfying, given that an inspection of the actual protocols would suggest that a modular interpretation with the above-discussed two-party building block playing the central role would be possible.

Proofs of sequential work. The incompleteness of the provided picture is further underscored by the structure of the FlyClient protocol, which is strongly reminiscent (as the authors themselves observe) to a seemingly unrelated primitive, a *proof of sequential work (PoSW)* [MMV13]. A PoSW is a proof system in which a prover, given a statement χ and a parameter n, computes a proof that convinces the verifier that n sequential computational steps have been performed since χ was received. The authors of [BKLZ20] indeed remark that their construction resembles the PoSW of Cohen and Pietrzak [CP18], but the exact relationship, as well as potential opportunities for further generalizations and alternative constructions, remain—to the best of our knowledge—unstudied.

Our contributions. In this paper, we set out to fill the above-described gaps in the theory underlying light-client protocols. Our main goals are to allow basing the development of these protocols on the classical theory of proof systems, and to explain the role that proofs of sequential work can play in their design. Our contributions can be summarized as follows:

1. We define a new general primitive called *succinct non-interactive argument of chain knowledge (SNACK)*, which is a non-interactive proof system for a specific NP language, formally capturing the above intuition.
2. To construct SNACKs from so-called *graph-labeling* proofs of sequential work (GL-PoSW), we unify existing definitions, add *knowledge-soundness* as a new property, and give two constructions rooted in previous work achieving it.
3. We show how to augment any blockchain with any knowledge-sound GL-PoSW and construct a SNACK system for the augmented chain.
4. We show how SNACKs can be used to construct light-client blockchain protocols, and compare them to existing solutions.
5. We present a novel *void-commitment attack* against a naive class of designs of bootstrapping protocols, and show how to mitigate this attack.
6. We define *incremental* SNACKs, which could allow for constructions of light miners with better resilience than existing proposals.

1. Defining SNACKs. Consider a family $\mathit{\Gamma} = (\mathit{\Gamma}_n)_{n\in\mathbb{N}}$ of weighted directed acyclic graphs (DAGs), that is, the vertices of each DAG are attributed non-negative weights summing to 1. Consider *labels* associated to the vertices and let R be a relation defined on the labels of the vertices, which formalizes some "validity" requirement. (The DAG will represent a blockchain, with edges capturing validation dependencies.) Let Com be a commitment scheme. We define a *chain commitment language* $\mathcal{L}_{\mathit{\Gamma},R,\mathsf{Com}}$ that consists of pairs (ϕ, n) where ϕ is a Com-commitment to the labels of $\mathit{\Gamma}_n$ that are valid with respect to R.

We then consider an interactive proof system for $\mathcal{L}_{\mathit{\Gamma},R,\mathsf{Com}}$ called an *argument of chain knowledge (ACK)*, which satisfies a relaxation of knowledge soundness called *α-knowledge soundness*: any prover that convinces a verifier of a statement (ϕ, n) must know a path in $\mathit{\Gamma}_n$ of weight at least α and being valid w.r.t. R. We focus on *succinct* and *non-interactive* arguments, i.e., SNACKs.

2. Knowledge-sound PoSW schemes. All known PoSW constructions [MMV13, CP18, AKK$^+$19, DLM19] (except for the subclass of the much stronger and less efficient *verifiable delay functions* [BBBF18]) are based on a random-oracle-induced labeling of a particular DAG G. We provide a unifying definition capturing all existing such *graph-labeling* PoSWs, for which we consider an arbitrary weight distribution on the vertices of G, which will define the distribution for challenge sampling (while prior constructions always sample uniformly). Furthermore, we define a notion of knowledge soundness for GL-PoSW, which will prove useful for constructing SNACKs from GL-PoSWs: knowledge soundness of the SNACK will follow from that of the underlying GL-PoSW.

We propose two knowledge-sound GL-PoSW schemes: The first is based on the PoSW scheme from [AKK$^+$19], the second is a slight modification of the PoSW scheme from [CP18] which is better suited to our SNACK applications (by allowing weight on *all* vertices as opposed to only the leaves as in [CP18]).

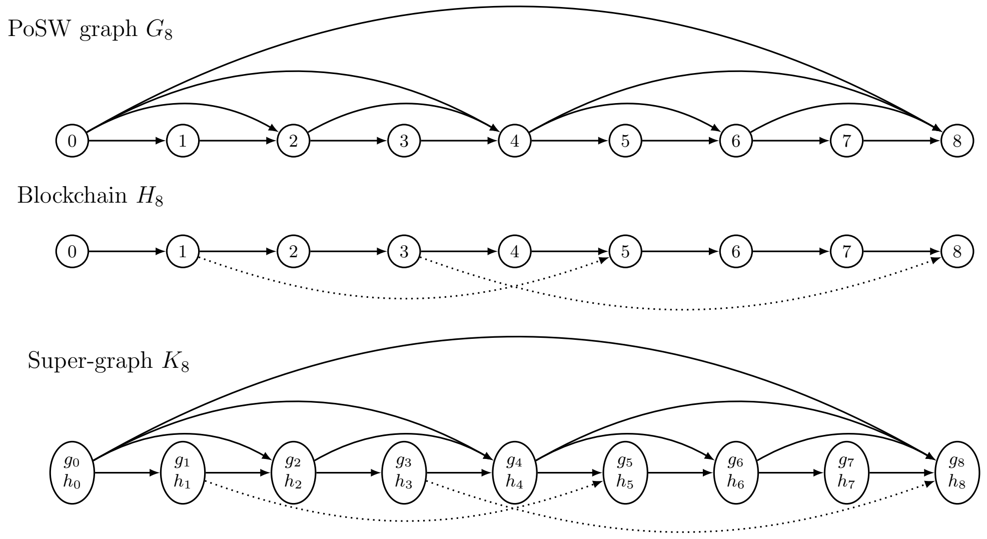

Fig. 1: Graph H_8 represents the dependency relations in the blockchain, G_8 is the chain graph underlying our PoSW scheme based on [AKK$^+$19]; K_8 represents the graph structure underlying the augmented blockchain. A label $k_i = (g_i, h_i)$ of K_8 consists of a blockchain block h_i and a PoSW-related label g_i, which is typically small, e.g. 256 bits, and consists of a hash the labels of the parents of vertex i in K_8. The "proof" in h_i, e.g. PoW in Bitcoin, must depend on all parent labels in K_8 as well as g_i.

3. Constructing SNACKs for blockchains. A blockchain can be viewed as a path with potential extra edges representing additional validation dependencies between blocks, so that block validity can be determined by a relation R applied to the block and its parents in this DAG. By adding extra (short) fixed-size data to each block, we show how to bind this blockchain DAG to the DAG of a GL-PoSW whose sequential computation can be efficiently verified (see Fig. 1).

The augmented blockchain then gives rise to a SNACK system for a validity relation $\tilde{R}$, which beyond checking the original block validity via R, also verifies the consistency of the added PoSW data. A proof in this SNACK scheme convinces the verifier that the prover *knows* blocks of a certain weight in a blockchain that are (i) valid and (ii) have been mined sequentially (since they lie on a path)—a crucial guarantee in light-client protocols, as discussed next.

4. SNACKS in the real world. Our treatment so far has been fully independent of the actual Sybil-protection mechanism (e.g., proofs of work/stake/space) of the underlying blockchain. However, the implications and usefulness of the sequentiality guaranteed by a SNACK highly depend on this mechanism. For example, in a proof-of-work (PoW) blockchain, it is costly to generate blocks and thus the guarantee of a sequentially-generated set of blocks is valuable, because an adversary controlling only a minority of the computational power cannot generate the longest such sequence. In contrast, in the proof-of-stake setting, sequentiality is a weak guarantee, as generating a block requires a mere digital signature and long sequences can be readily created. Hence, our main focus is on applying SNACKs in the PoW setting, which was the only setting considered in [KMZ20, BKLZ20]. Nonetheless, we believe that our approach has much wider applicability, for instance, to blockchains combining proofs of space [DFKP15, AAC$^+$17] with verifiable delay functions [BBBF18] such as Chia [CP19]. We leave this question to follow-up work.

SNACKs can be employed for bootstrapping in the presence of at least one honest prover: the light client simply obtains proofs from all provers and picks the heaviest successful one. However, SNACKs also allow for applications where there is only a single prover. For instance, if a verifier V knows, for application-specific reasons, (an estimate of) the current length of a blockchain, then V only has to check a single SNACK proof evidencing that the prover knows a path of roughly the correct length satisfying the SNACK guarantees of validity and sequentiality. This proof is then enough to convince the verifier that the prover holds the right blockchain (maybe up to a short suffix).

5. The Void-Commitment Attack and preventing it. Surprisingly, however, we observe that the above approach turns out insufficient for a typical bootstrapping scenario where the obtained chain commitment is meant to serve as a self-sufficient, universal anchor of trust in future interactions with other full nodes. We describe a simple attack, called the *void-commitment attack*, that allows the attacker to trick the light client into accepting a chain commitment that will turn out completely useless in future interactions with any honest full node. To the best of our knowledge, this attack has not been observed before.

We show how to remedy the attack by instead letting the prover establish a commitment to some *stable common prefix* of all honestly held chains, one that is then universally understood by all honest full nodes, which appears significantly more desirable in practice. Towards formalizing this, we introduce a security definition of *secure common-prefix bootstrapping* and prove that our final protocol achieves this notion. Our proof is based on an adversary-limiting assumption in the spirit of (c, L)-adversaries assumed in [BKLZ20]. However, we observe that their original (somewhat informal) assumption is insufficient for formal reasoning in any convincingly general model, and we hence present its formalization that addresses several of the original deficiencies (e.g., assuming that all competing chains—"forks"—must be of the same length).

6. Incremental SNACKs. A powerful extension of PoSWs, called *incremental* PoSWs [DLM19], allows to extend an existing PoSW by additional sequential computation into a new PoSW covering the full computation. We add this property to our formalism of GL-PoSWs and SNACKs. We observe that using an incremental GL-PoSW to construct a SNACK system on top of a blockchain whose underlying chain graph is simple (i.e., corresponds to a path) leads to an incremental SNACK. To date, the only existing construction of an incremental GL-PoSW [DLM19] is defined for a uniform challenge distribution, while our applications of SNACKs require different distributions. We leave it as an exciting open problem to construct incremental GL-PoSW for arbitrary weight functions.

This approach could allow for constructing *light miners* in the sense of [KLZ21] without having to assume a sub-1/3-adversary, hence providing a clear improvement over [KLZ21]. Pursuing this idea is outside of our current scope and we leave it to future work.

Further notes on related work. The superblock-based approach to light clients [KMZ20, KLZ21] draws inspiration from the interactive protocol of [KLS16], and has led to an exciting line of follow-up work [KKZ19, KPZ20, DKKZ20].

FlyClient [BKLZ20] is closely related to our generic SNACK construction when instantiated with the PoSW from [CP18]; we conjecture that FlyClient satisfies the guarantees of a SNACK, and discuss the relationship further in Sect. 6. It employs an elegant technique (variable-difficulty Merkle mountain ranges) to cope with the variable-difficulty setting. This is a strong indication that our generic design can also be tweaked to handle variable difficulty. We leave this as an interesting open problem, remarking that the possibility of using general weight functions could prove useful here. However, FlyClient does not support incremental proofs (and hence cannot be used for light mining).

While our concrete constructions do not outperform FlyClient efficiency-wise, we put some of the intuitions of FlyClient on more solid formal footing, generalize their construction, and propose extensions (such as incrementality). In particular, we formalize the concept and the security of light-client protocols, which we believe is lacking in FlyClient, as well as the concept of a commitment to the chain serving as anchor of trust for verifiers. While FlyClient leaves some room for interpretation (particularly relevant for the void commitment attack), we clearly specify our protocol and give a rigorous security analysis.

Regarding SNARK-based constructions, the light-client protocol Plumo [VGS+21] is designed for the BFT-based consensus of the Celo blockchain. While it achieves impressive concrete succinctness among SNARK-based proposals, it is still best suited for incremental proofs and requires heavier cryptographic tools, most importantly, a trusted setup. Mina (formerly Coda) [BMRS20] employs SNARKs for a significantly more ambitious goal of providing a constant-size blockchain.

2 Notation and Basic Definitions

General. We let $\mathbb{N}$ denote the set $\{1, 2, \ldots\}$ and set $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For integers i, j such that $i \leq j$, we let $[i : j] := \{i, i + 1, \ldots, j\}$, $[n] := [1 : n]$, and $[n]_0 := [0 : n]$. We let ε denote the empty string; for strings a and b we denote their concatenation by $a \| b$. For a distribution $\mathcal{D}$, we denote by $d \xleftarrow{\$} \mathcal{D}$ sampling d according to $\mathcal{D}$ (for a set $\mathcal{D}$, the uniform distribution is implied).

DAGs and chain graphs. For a directed acyclic graph (DAG) $G = (V, E)$ on $n + 1$ vertices, we always number its vertices $V = [n]_0$ in topological order and often write G_n to make this explicit. For $v \in [n]_0$, we denote the parent vertices of v in G by $\text{parents}_G(v)$, and their number (i.e., the indegree of v) by $\deg_G(v)$; thus, $\text{parents}_G(v) = (v_1, \ldots, v_{\deg_G(v)})$. We also let $\deg(G) := \max_{v \in V}\{\deg_G(v)\}$. We drop the subscript G when G is clear from the context. For convenience, we assume that the tuple $\text{parents}(v)$ is given in reverse topological order.

Let $(G_n = ([n]_0, E_n))_{n \geq 0}$ be a family of DAGs. We make the assumption that for each $n \geq 0$, G_n is obtained from G_{n+1} by removing the vertex $n + 1$ and its adjacent edges. We also assume that $\deg(G_n) \in \text{polylog}(n)$.

Definition 1 (Chain graph). *Let $G_n = ([n]_0, E_n)$ be a DAG. We call G_n a* chain graph *if $E_n \supseteq \{(i-1, i) : i \in [n]\}$. A chain graph $G_n = ([n]_0, E_n)$ is called* simple *if $E_n = \{(i-1, i) : i \in [n]\}$, i.e., it forms a path.*

Graph labeling and weighted DAGs. The following notion of weighted DAGs will be convenient when we define proof of sequential work (PoSW) schemes with arbitrary, not just uniform, sampling distributions.

Definition 2 (Weighted DAGs). *We call $\varGamma_n = (G_n, \varOmega_n)$ a* weighted DAG *if $G_n = ([n]_0, E_n)$ is a DAG and $\varOmega_n \colon [n]_0 \to [0, 1]$ is a function such that $\varOmega_n([n]_0) = 1$, where for $S \subseteq [n]_0$ we let $\varOmega_n(S) = \sum_{s \in S} \varOmega_n(s)$.*

In this work, we leverage the fact that the validity of (the headers of) a blockchain can be checked "locally", e.g., in Bitcoin, assuming fixed difficulty, (the header of) the i-th block h_i is valid if it contains the hash of the previous block h_{i-1} and a valid proof of work. We represent these dependencies by a chain graph which contains an edge $i \to j$ if block h_i is required to check the validity of h_j (in fixed-difficulty Bitcoin the graph is thus a simple chain graph).

Validity is captured by a relation R_ψ, where ψ is the genesis block of the blockchain, and h_i is valid if $R_\psi(i, h_i, (h_{\iota_1}, \dots, h_{\iota_q})) = 1$, where $h_{\iota_1}, \dots, h_{\iota_q}$ are h_i's parent blocks. We emphasize that we consider "SPV" or "header" validity, which is the standard desideratum for blockchains adopted by light clients, and in particular does not verify the validity of contained transactions.

Definition 3 (Labeled DAGs and blockchain validity). *Let $G_n = ([n]_0, E_n)$ be a DAG. A* (graph) labeling *of G_n is a mapping $L\colon [n]_0 \to \{0,1\}^*$. (This naturally extends to labelings of subgraphs of G_n.) A* (block-)chain *is a labeled chain graph.*[6] *For a polynomial-time (PT) relation R_ψ, a blockchain (G_n, L) with genesis block $L(0) = \psi$ is* R_ψ-valid *if for all $i \in [n]$: $R_\psi\big(i, L(i), (L(j))_{j \in \mathrm{parents}_G(i)}\big) = 1$.*

We define the notion of *oracle-based* graph labelings, which will later be used by graph-labeling proof of sequential work (GL-PoSW) schemes, where the prover computes the labels of a given graph, sends a commitment to them to the verifier and is then challenged to open some of them. An oracle-based labeling of a graph G defines the labels of the sources of G as the oracle evaluation on the empty string; the label of any other vertex is defined as the evaluation on the labels of the parents of the vertex. In our applications to blockchains the vertices are the blocks and their labels represent blockchain data. We therefore consider an "augmented" definition allowing to include arbitrary data in the labels.

Definition 4 (Oracle-based graph labeling). *Let $G_n = ([n]_0, E_n)$ be a DAG and $\tau = (\tau_i)_{i \in [n]_0}$ be a tuple of oracles, with each $\tau_i\colon \{0,1\}^* \to \{0,1\}^\lambda$. For any $X = (x_0, \dots, x_n) \in (\{0,1\}^*)^{n+1}$ the* X-augmented τ-labeling *$L^\tau\colon [n]_0 \to \{0,1\}^*$ of G_n is recursively defined as*

$$L^\tau(i) := \begin{cases} \tau_i(\varepsilon) \| x_i & \textit{if } \mathrm{parents}(i) = \emptyset, \\ \tau_i\big(L^\tau(\mathrm{parents}(i))\big) \| x_i & \textit{otherwise}, \end{cases} \tag{1}$$

where $L^\tau(\mathrm{parents}(i)) := L^\tau(i_1) \| \cdots \| L^\tau(i_k)$ for $(i_1, \dots, i_k) := \mathrm{parents}(i)$. If $X = (\varepsilon, \dots, \varepsilon)$, we call L^τ the τ-labeling *of G_n.*

Vector commitment (VC) schemes. A VC scheme Com lets one commitment to message vectors m and give short openings for (subsets of) components of m. It has four algorithms: The parameters cp are computed via setup; a vector is committed to via $(\phi, aux) \leftarrow \mathsf{commit}(cp, (m_1, \dots, m_n))$, which returns a commitment ϕ and auxiliary information aux. To give an opening ρ of ϕ to m_i at position i, run $\mathsf{open}(cp, \phi, aux, m_i, i)$. The opening is verified by running $\mathsf{ver}(cp, \phi, m_i, i, \rho)$, which returns a bit. (We generalize this to opening a set I of indices via $\mathsf{ver}(cp, \phi, (m_i)_{i \in I}, I, \rho)$.) The scheme must be *position-binding*, meaning no adversary can compute a commitment ϕ and two openings ρ, ρ' for $m_i \neq m'_i$ that both verify at position i. (See Appendix A for a formal definition.)

3 Defining SNACKs

In this section we introduce our main primitive of a *succinct non-interactive argument of chain knowledge (SNACK)*. Intuitively, it is an argument system for an NP language $\mathcal{L}_{\Gamma, R, \mathsf{Com}}$ that is

[6] We use the words *chain* and *blockchain* as synonyms throughout the paper.

parameterized by a family of weighted DAGs $\mathit{\Gamma} = (\mathit{\Gamma}_n)_{n\geq 0}$ with $\mathit{\Gamma}_n = (G_n = ([n]_0, E_n), \mathit{\Omega}_n)$, a polynomial-time relation $R \subseteq \mathbb{N}_0 \times (\{0,1\}^*)^2$, and a vector commitment scheme Com.

An element $(\phi, n) \in \mathcal{L}_{\mathit{\Gamma},R,\mathsf{Com}}$ consists of a Com commitment ϕ to a labeling of G_n that is *valid* as defined by R, which checks every label w.r.t. the labels of its parents (Def. 3). Looking ahead, R will serve two purposes: in GL-PoSW schemes, R checks the validity of an oracle-based graph labeling, and in our SNACK systems for *augmented* blockchains, whose vertex labels include a GL-PoSW label, R additionally verifies blockchain validity.

A SNACK proof for a statement (ϕ, n) proves knowledge of an R-valid labeling of $\mathit{\Gamma}_n$ as well as an opening of ϕ to this labeling. This is formalized by requiring that from a prover computing a valid proof such a labeling and an opening can be *extracted*. To enable more efficient schemes, we only require extraction of labels that lie on a path P that has a certain weight, as measured by $\mathit{\Omega}_n$. We call SNACK system *α-knowledge-sound*,[7] if it guarantees that from a valid proof for (ϕ, n) a labeled path of weight at least $\alpha \in (0,1]$ can be extracted, Setting $\alpha = 1$ recovers standard knowledge soundness.

Analogously to SNARKs, we require succinctness of SNACKs, that is, proofs are of size polylogarithmic in n and efficient to generate and verify.

Valid paths in a weighted DAG. To make the above formal, we need to adapt the definition of a valid labeling of a graph G to paths P in G. In a path, a vertex might have a parent in G that is not part of P, still the relation R expects a label for it. We therefore define a valid path as containing, for every $v \in P$, a "witness" p_v, which is the set of purported parent labels, which must be accepted by R. We require p_v to be in accord with the labeling of P, that is, if a parent u of v lies on P then p_v must contain the label of u.

Definition 5 (Valid paths). *Let $G_n = ([n]_0, E_n)$ be a DAG, and $R \subseteq \mathbb{N}_0 \times (\{0,1\}^*)^2$ a relation. Furthermore, let P be a path in G_n, L_P a labeling of P, and $(p_v)_{v\in P} \in (\{0,1\}^*)^{|P|}$ a $|P|$-tuple of bitstrings with $p_v = (p_v[1], \ldots, p_v[\deg(v)])$. We say that $(P, L_P, (p_v)_{v\in P})$ is an R-*valid *path in G_n if for all $v \in P$ with $(v_1, \ldots, v_{\deg(v)}) := \mathrm{parents}(v)$, we have*

$$R\big(v, L_P(v), p_v\big) = 1 \ \ \text{and} \ \ \forall i \in [\deg(v)] \ \text{if} \ v_i \in P \ \text{then} \ p_v[i] = L_P(v_i) \ . \tag{2}$$

*For a weighted DAG $\mathit{\Gamma}_n = (G_n = ([n]_0, E_n), \mathit{\Omega}_n)$, we say that $(P, L_P, (p_v)_{v\in P})$ is (α, R)-*valid *in $\mathit{\Gamma}_n$ if in addition $\mathit{\Omega}_n(P) \geq \alpha$.*

For blockchains we typically require $0 \in P$, so R_ψ verifies the genesis block.

Chain commitment languages. We formally define the language $\mathcal{L}_{\mathit{\Gamma},R,\mathsf{Com}}$ for a SNACK proof system, where a statement $\eta = (\phi, n)$ consists of a Com commitment ϕ to an R-valid labeling of the graph G_n. The labeling together with an opening of ϕ constitutes a witness w for η. We parameterize the language (akin to languages parameterized by a group [GS08]), where parameters *prm* are generated (formally by an algorithm G) during a setup phase. This allows us to capture SNACKs with instantiations of Com for which (position-)binding only holds under honestly generated parameters *cp*; it also allows us to include the salt χ defining a random oracle τ for τ-labelings in PoSW schemes, and to include the genesis block ψ of a blockchain, both of which are assumed to

[7] This is akin to f-extractability [BCKL08] of proof systems, which relaxes knowledge soundness by only requiring extraction of a partial witness (or a function thereof).

have been generated independently of the adversary. (Below these are subsumed into parameters σ on which the relation R can depend.)

Formally, $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ is defined via a parameter-dependent ternary polynomial-time (PT) relation $\mathcal{R}$ over tuples (prm, η, w) as the statements η for which there exists a witness such that $\mathcal{R}(prm, \eta, w) = 1$. For standard NP relations, $prm = \varepsilon$.

Definition 6 (Chain commitment language). *Let $\Gamma = (\Gamma_n)_{n\geq 0}$ be a family of weighted DAGs and* Com *a vector commitment scheme. We define*

$$\mathcal{R}^{(\alpha)}_{\Gamma,R,\mathsf{Com}} := \left\{ \begin{matrix} \big(prm = (\sigma, cp), \eta = (\phi, n), \\ w = (P, L_P, (p_v)_{v\in P}, \rho)\big) \end{matrix} : \begin{matrix} (P, L_P, (p_v)_{v\in P}) \text{ is } (\alpha, R)\text{-valid} \\ \wedge\ \mathsf{Com.ver}(cp, \phi, L_P, P, \rho) = 1 \end{matrix} \right\} \tag{3}$$

where $R \subseteq \mathbb{N}_0 \times (\{0,1\}^*)^2$ is a PT relation that depends on σ. We let $\mathcal{R}_{\Gamma,R,\mathsf{Com}} := \mathcal{R}^{(1)}_{\Gamma,R,\mathsf{Com}}$ and $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ denote the language defined by $\mathcal{R}_{\Gamma,R,\mathsf{Com}}$.

We now define the SNACK system which is the central primitive we are interested in. The generality of the SNACK definition stems from leaving the specification of both the underlying relation R and Com open.

Definition 7 (SNACK). *A tuple of PPT algorithms* (P, V) *is a* succinct non-interactive argument of chain knowledge (SNACK) *for the language $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ with parameter generator* G *from Def. 6 if the following properties hold:*

Completeness: *For all $\lambda \in \mathbb{N}$, $prm \leftarrow \mathsf{G}(1^\lambda)$, $\eta, w \in \{0,1\}^*$ with $(prm, \eta, w) \in \mathcal{R}_{\Gamma,R,\mathsf{Com}}$:*

$$\Pr\big[\pi \leftarrow \mathsf{P}(prm, \eta, w) :\ \mathsf{V}(prm, \eta, \pi) = 1\big] = 1\ .$$

(α, ϵ)**-Knowledge soundness:** *For every PPT prover $\tilde{\mathsf{P}}$ there exists a PPT extractor* E *such that*

$$\Pr\left[\begin{matrix} prm \leftarrow \mathsf{G}(1^\lambda); r \xleftarrow{\$} \{0,1\}^{\mathrm{poly}(\lambda)} \\ (\eta, \pi) := \tilde{\mathsf{P}}(prm; r) \\ w' \leftarrow \mathsf{E}(prm, r) \end{matrix} : \begin{matrix} \mathsf{V}(prm, \eta, \pi) = 1\ \wedge \\ \mathcal{R}^{(\alpha)}_{\Gamma,R,\mathsf{Com}}(prm, \eta, w') = 0 \end{matrix}\right] \leq \epsilon(\lambda)\ , \tag{4}$$

with $\mathcal{R}^{(\alpha)}_{\Gamma,R,\mathsf{Com}}$ from (3). We say that (P, V) *has* universal (α, ϵ)*-knowledge soundness if there exists a single extractor $\mathsf{E}^{\tilde{\mathsf{P}}}$ with oracle access to $\tilde{\mathsf{P}}$ that satisfies (4) for all PPT provers $\tilde{\mathsf{P}}$.*

Succinctness: *For all $prm \leftarrow \mathsf{G}(1^\lambda)$, $(prm, \eta, w) \in \mathcal{R}_{\Gamma,R,\mathsf{Com}}$ and $\pi \leftarrow \mathsf{P}(n, t\eta, w)$, we have $|\pi| \leq \mathrm{poly}(\lambda, \log n)$,* P *runs in time* $\mathrm{poly}(\lambda, n)$*, and* V *runs in time* $\mathrm{poly}(\lambda, \log n)$.

We remark that the SNACKs we construct in Sect. 5 have universal extractors in the random oracle model. Note that we could have first defined the notion of an*argument of chain knowledge (ACK)* as an interactive proof system in Def. 7without requiring succinctness. A SNACK would then be any succinct non-interactive ACK, as required by the main applications of interest.

4 Graph-Labeling Proofs of Sequential Work

Intuitively, proofs of sequential work (PoSW) are proof systems in which a prover, upon receiving a statement χ and a parameter n, convinces a verifier that n sequential computational steps have passed since χ was received. We define *augmented graph-labeling* PoSWs, a special class of PoSWs that covers all recent and efficient constructions [MMV13, CP18, AKK⁺19, DLM19]. Our definition does however not cover the number-theoretic constructions of *verifiable delay functions* [BBBF18, Pie19, Wes19], which are PoSWs in which statements have unique proofs. This is not required for our applications; moreover, known constructions are far less efficient than graph-labeling (GL) PoSWs.

4.1 Defining Graph-Labeling PoSW

All random-oracle-aided PoSW constructions [MMV13, CP18, AKK⁺19, DLM19] follow the same blueprintwhich we now describe. These PoSWs are defined and constructed interactively and then turned non-interactive using the Fiat-Shamir transformation [FS87].[8] A graph-labeling PoSW scheme is parameterized by a family of weighted DAGs $\mathit{\Gamma} = (\mathit{\Gamma}_n)_{n\in\mathbb{N}}$. Let $\mathit{\Gamma}_n = (G_n = ([n]_0, E_n), \mathit{\Omega}_n)$ be a DAG from this family with weight distribution $\mathit{\Omega}_n \colon [n]_0 \to [0,1]$ with $\mathit{\Omega}([n]_0) = 1$. The prover P, upon receiving a statement χ from the verifier V, uses χ to instantiate a sequence of oracles $\tau = (\tau_i)_{i\in[n]_0}$. In all constructions except [AKK⁺19], τ_i is defined by salting a random oracle $\mathcal{O}$ as $\tau_i(\cdot) := \mathcal{O}(\chi, i, \cdot)$; the construction from [AKK⁺19] uses χ to sample random permutations. Next, P computes a τ-labeling L (Def. 4) of the vertices $[n]_0$ of $\mathit{\Gamma}_n$ and produces a commitment ϕ_L to L using a vector commitment scheme. Finally, P and V run an interactive protocol in which V essentially checks that the responses of P to a challenge set $S \subset [n]_0$ sampled according to $\mathit{\Omega}_n$ are in accord with ϕ. See Fig. 2 for the syntax to formally define PoSW in Def. 9.

Our definition of graph-labeling PoSW generalizes existing definitions in several ways, which are particularly useful for constructing SNACKs from PoSWs: First, while previous work only considered challenges sampled uniformly at random, we allow for arbitrary sampling distributions $\mathit{\Omega}_n$. Second, we extend the security guarantees of PoSW by requiring *knowledge* soundness, which will be necessary when constructing SNACKs from PoSW. Existing PoSW schemes implicitly satisfy a form of knowledge soundness, and we make this explicit. Finally, we allow the prover to embed *arbitrary* additional data, as augmentation, into the computation. While this doesn't seem useful for classical applications of PoSWin the literature, it will be a crucial property for our later application of PoSW as a building block for to SNACKs.

In defining graph-labeling PoSW, we require proofs to be short and verification to be fast[9], as for SNARK systems. Unlike general-purpose SNARKs however, we require practically efficient provers and no setup assumptions.

Towards generalizing PoSW to arbitrary weight functions, we define the weight of a sequence of parallel oracle queries to $\tau = (\tau_i)_{i\in[n]_0}$. A parallel query is a set of simultaneous queries to oracles τ_i, i.e. a tuple $((x_1, i_1), \ldots, (x_m, i_m))$ which is answered by $(\tau_{i_1}(x_1), \ldots, \tau_{i_m}(x_m))$. The weight of a sequence of parallel queries is the sum of the respective "heaviest" nodes in each parallel query.

Definition 8 (Sequential weight). *Let $Q = (Q_1, \ldots, Q_\ell)$ be a sequence of parallel queries to an oracle $\tau = (\tau_i)_{i\in[n]_0}$. We define the* sequential weight *of Q with respect to a weight function $\mathit{\Omega}_n \colon [n]_0 \to [0,1]$ as*

$$\mathit{\Omega}_{\text{seq}}(Q) := \textstyle\sum_{i=1}^{\ell} \max\left\{\mathit{\Omega}_n(j) \,:\, Q_i \text{ contains a query to } \tau_j\right\} .$$

[8] The PoSW of [DLM19] is defined and constructed non-interactively by employing an on-the-fly sampling technique..

[9] E.g., a *hash-chain* construction, in which the prover computes and sends the verifier $\pi := \mathcal{H}^n(\chi)$ (the n times sequentially repeated evaluation of a hash function $\mathcal{H}$), does not satisfy our definition below as the verifier needs to recompute π to verify it.

Note that if Ω_n is uniform, i.e., $\forall i \in [n]_0 : \Omega_n(i) = \frac{1}{n+1}$, then $\Omega_{\text{seq}}(Q) = \frac{\ell}{n+1}$.

We write $(out_{\mathsf{A}}, out_{\mathsf{B}}) \leftarrow \langle \mathsf{A}(in_{\mathsf{A}}) \leftrightarrow \mathsf{B}(in_{\mathsf{B}})\rangle$ to denote an execution of interactive algorithms A, taking input in_{A} and outputting out_{A}, and B, taking input in_{B} and outputting out_{B}. We write $\mathsf{A}(in_{\mathsf{A}}; r)$ to make A's randomness explicit. We now define augmented graph-labeling PoSW.

Definition 9 (Augmented GL-PoSW). *Let $\Gamma = (\Gamma_n = (G_n, \Omega_n))_{n\in\mathbb{N}}$ be a family of weighted DAGs such that for all n, G_n has a unique sink n. A pair of PPT algorithms* $(\mathsf{P} := (\mathsf{P}_0, \mathsf{P}_1), \mathsf{V} := (\mathsf{V}_0, \mathsf{V}_1, \mathsf{V}_2))$*, with access to an oracle $\tau = (\tau_i)_{i\in\mathbb{N}_0}$ is an augmented (oracle-based)* graph-labeling proof of sequential work (GL-PoSW) *if it instantiates the template described in Fig. 2 by specifying a vector commitment scheme* $\mathsf{Com} = (\mathsf{setup}, \mathsf{commit}, \mathsf{open}, \mathsf{ver})$ *and the subroutines* PoSW.label, PoSW.open *and* PoSW.ver*; and it satisfies the following properties:*

Completeness: *For all $n, \lambda \in \mathbb{N}$ it holds that*

$$\Pr\left[(out_{\mathsf{P}}, out_{\mathsf{V}}) \leftarrow \langle \mathsf{P}(1^n) \leftrightarrow \mathsf{V}(1^\lambda, n)\rangle : out_V = 1\right] = 1 \ .$$

(α, ϵ)**-Soundness:** *For all $\lambda \in \mathbb{N}$ and every PPT adversary $(\tilde{\mathsf{P}}', \tilde{\mathsf{P}} = (\tilde{\mathsf{P}}_0, \tilde{\mathsf{P}}_1))$ s.t. $\tilde{\mathsf{P}}$ makes a sequence Q of parallel queries to $\tau = (\tau_j(\cdot))_{j\in[n]_0}$ of sequential weight $\Omega_{\text{seq}}(Q) < \alpha$, it holds that*

$$\Pr\left[\begin{matrix}(n, st) \leftarrow \tilde{\mathsf{P}}'(1^\lambda) \\ (out_{\tilde{\mathsf{P}}}, out_{\mathsf{V}}) \leftarrow \langle \tilde{\mathsf{P}}(st) \leftrightarrow \mathsf{V}(1^\lambda, n)\rangle\end{matrix} \ : \ out_{\mathsf{V}} = 1\right] \leq \epsilon(\lambda) \ .$$

Succinctness: *The size of the transcript $|\langle \mathsf{P}(1^n) \leftrightarrow \mathsf{V}(1^\lambda, n)\rangle|$ as a function of λ and n is upper-bounded by* $\text{poly}(\lambda, \log n)$*. The running time of* P *is* $\text{poly}(\lambda, n)$ *and that of* V *is* $\text{poly}(\lambda, \log n)$*.*

We say that (P, V) is (α, ϵ)-knowledge-sound we additionally have:

(α, ϵ)**-Knowledge soundness:** *There exists a PPT extractor E such that for every PPT adversary $(\tilde{\mathsf{P}}', \tilde{\mathsf{P}} = (\tilde{\mathsf{P}}_0, \tilde{\mathsf{P}}_1))$ we have*

$$\Pr\left[\begin{matrix}r \xleftarrow{\$} \{0,1\}^{\text{poly}(\lambda)};\ (n, st) := \tilde{\mathsf{P}}'(1^\lambda; r) \\ (out_{\tilde{\mathsf{P}}}, out_{\mathsf{V}}) \leftarrow \left\langle \tilde{\mathsf{P}}(st; r) \leftrightarrow \mathsf{V}(1^\lambda, n)\right\rangle \\ w' \leftarrow \mathsf{E}^{\tilde{\mathsf{P}}}(1^\lambda, r)\end{matrix} \ : \ \begin{matrix}out_{\mathsf{V}} = 1\ \wedge \\ \mathcal{R}^{(\alpha)}_{\Gamma,R,\mathsf{Com}}\big(prm, (out_{\tilde{\mathsf{P}}_0}, n), w'\big) = 0\end{matrix}\right] \leq \epsilon(\lambda)\,,$$

where prm is as sampled by V_0, $out_{\tilde{\mathsf{P}}_0}$ is the output ϕ_L of $\tilde{\mathsf{P}}_0$ and relation $\mathcal{R}^{(\alpha)}_{\Gamma,R,\mathsf{Com}}$ is as in (3) with $R := R_\chi$ defined as

$$R\big(i, L(i), p_i\big) = 1 \quad \textit{iff} \quad L(i) = \tau_i(p_i)\|x_i \textit{ for some } x_i \in \{0,1\}^* \ . \tag{5}$$

In Appendix C.1, we prove Theorem 1 below, which establishes that every knowledge-sound GL-PoSW is indeed a sound PoSW.

Theorem 1. *Every (α, ϵ)-knowledge-sound graph-labeling PoSW is (α, ϵ')-sound (cf. Def. 9) with $\epsilon' := \epsilon + (q^2 + 1)/2^\lambda$, where q is an upper bound on the number of the adversary's oracle queries.*

The following lemma now directly follows from the respective definitions.

```
Verifier $\mathsf{V} = (\mathsf{V}_0, \mathsf{V}_1, \mathsf{V}_2)$:

Stage $\mathsf{V}_0$: On input $(1^\lambda, n)$:
  1. $\chi \xleftarrow{\$} \{0,1\}^\lambda$
  2. $cp \leftarrow \mathsf{Com.setup}(1^\lambda)$
  3. send $prm := (\chi, cp)$ to $\mathsf{P}_0$

Stage $\mathsf{V}_1$: On input $\phi_L$:
  1. $\forall i \in [t]$ do $\iota_i \xleftarrow{\$} \Omega_n$
  2. send $\iota = (\iota_i)_{i=1}^t$ to $\mathsf{P}_1$

Stage $\mathsf{V}_2$: On input $(\gamma_i = (o_i, \rho_i))_{i=1}^t$:
  1. $\forall i \in [t]$ do
     (a) $b_i^{(1)} := \mathsf{PoSW.ver}(\chi, \iota_i, o_i)$
     (b) $b_i^{(2)} := \mathsf{Com.ver}(cp, \phi_L, L(\iota_i), \iota_i, \rho_i)$
  2. output $\bigwedge_{i=1}^t (b_i^{(1)} \wedge b_i^{(2)})$
```

```
Prover $\mathsf{P} = (\mathsf{P}_0, \mathsf{P}_1)$ :

Stage $\mathsf{P}_0$: On input $1^n$ and $prm := (\chi, cp)$:
  1. $L := \mathsf{PoSW.label}(\chi, 1^n)$
     Use $\chi$ to sample oracles $\tau := (\tau_i(\cdot))_{i \in [n]_0}$ and compute a (possibly augmented) $\tau$-labeling $L$ of $G_n$ satisfying Def. 4.
  2. $(\phi_L, \mathsf{aux}) \leftarrow \mathsf{Com.commit}(cp, L)$
  3. send $\phi_L$ to $\mathsf{V}_1$

Stage $\mathsf{P}_1$: On input $\iota = (\iota_i)_{i=1}^t$:
  1. $\forall i \in [t]$ do
     (a) $o_i \leftarrow \mathsf{PoSW.open}(\chi, cp, \phi_L, \mathsf{aux}, L, \iota_i)$
         We assume that $o_i$ contains $L(\iota_i)$
     (b) $\rho_i \leftarrow \mathsf{Com.open}(cp, \phi_L, \mathsf{aux}, L(\iota_i), \iota_i)$
     (c) $\gamma_i := (o_i, \rho_i)$
  2. send $(\gamma_1, \ldots, \gamma_t)$ to $\mathsf{V}_2$
```

Fig. 2: The template of a GL-PoSW, parametrized by a family of weighted DAGs $(\Gamma_n)_{n \in \mathbb{N}}$, a vector commitment scheme Com and the number of challenges t. Note that explicitly requiring P_1 to compute ρ_i does not exclude the possibility of P_1 opening the commitment also at other indices as part of $\mathsf{PoSW.open}$.

Lemma 1. *Let* (P, V) *be an (interactive)* (α, ϵ)*-knowledge-sound graph-labeling PoSW based on a family of weighted DAGs* Γ *and a commitment scheme* Com*. Then applying the Fiat-Shamir transformation [FS87] to* (P, V) *results in a SNACK system for the language* $\mathcal{L}_{\Gamma, R, \mathsf{Com}}$*, when* R *is defined as in* (5)*.*

All the constructions of graph-labeling PoSWs we consider here use the following very simple commitment scheme for the graph labeling L: The prover commits to the labels of a graph which were derived through a graph-labeling computation[10] (cf. Def. 4). While the verifier could simply recompute the labels to check consistency of a label $L(i)$, for certain graph structures the following scheme will allow for much more efficient verification. Intuitively, the idea is to check consistency of the labels of a randomly sampled subgraph.

To formalize consistency of labels in this context, we need the following definition of consistent strings, which is stronger than prior definitions in the literature [MMV13, CP18, AKK$^+$19]. We associate to each vertex i a value $y_i := p_i \| x_i$, where p_i represents the (augmented) labels of the *parents* of i and x_i some potential augmentation of i. In order to reason about the label of the last node as well, we introduce a dummy child for it, that is, we add vertex $n+1$ and an edge $(n, n+1)$.

Definition 10 (Consistent strings). *Let* $\tau = (\tau_i)_{i \in [n]_0}$ *be a tuple of oracles, with* $\tau_i : \{0,1\}^* \to \{0,1\}^\lambda$*. For a DAG* $G_n = ([n]_0, E_n)$*, let* $G_n^+ = ([n+1]_0, E_n^+)$ *with* $E_n^+ = E_n \cup \{(n, n+1)\}$*.*

[10] Note that the construction from [MMV13] can also be cast as an instantiation of our construction by defining G_n as the union graph of the underlying depth-robust graph and the Merkle tree used to commit to the labels of the base graph.

Furthermore, $\forall i \in [n+1]_0$ *let* $\deg(i)$ *be the number of parents of* i *in* G_n^+ *and*

$$y_i := p_i \| x_i \in \{0,1\}^* \text{ where } p_i := p_i[1] \| \ldots \| p_i[\deg(i)].$$

We say y_i *is* consistent with $y_{i'}$ w.r.t. G_n*, and denote it by* $y_i \prec y_{i'}$ *if* $(i, i') \in E_n^+$ *and if* i *is the* j*-th parent of* i' *in* G_n^+ *(in reverse topological order), then the* j*-th block in the decomposition of* $y_{i'}$ *is equal to* $\tau_i(p_i) \| x_i$*, i.e.,*

$$p_{i'}[j] = \tau_i(p_i) \| x_i.$$

We formalize vector commitment schemes in Def. 15 in Appendix A. Below we give a specific vector commitment scheme called SPC, which will also be used in our constructions of graph-labeling PoSW.

Construction 1 (Shortest Path Commitment). *Let* $G = (G_n = ([n]_0, E_n))_{n \in \mathbb{N}}$ *be a DAG family such that for all* n*,* G_n *has a unique sink* n*, and let* $\tau := (\tau_i)_{i \in \mathbb{N}_0}$ *be a tuple of oracles with* $\tau_i : \{0,1\}^* \to \{0,1\}^\lambda$*. We construct a* τ*-based vector commitment* SPC = (setup, commit, open, ver) *for universe* $\mathcal{U} = \{0,1\}^*$ *and message space* $\mathcal{M} = (\mathcal{M}_n)_{n \in \mathbb{N}}$ *where* $\mathcal{M}_n \subseteq \mathcal{U}^n$ *consists of the labels of nodes* $[n-1]_0$ *of all valid* X*-augmented* τ*-labelings of* G_n *using* $L := L^\tau$ *as per Def. 4.*

- $cp \leftarrow \mathsf{setup}(1^\lambda)$*: On input* 1^λ*, output empty public parameters* $cp := \varepsilon$*.*
- $(\phi_L, \mathrm{aux}) \leftarrow \mathsf{commit}(cp, L)$*: On input* $L \in \mathcal{M}^n$*, output the commitment* $\phi_L := \tau_n(L(\mathrm{parents}(n)))$ *(i.e. the first part of the label* $L(n)$*) and auxiliary information* $\mathrm{aux} := L$*.*
- $\rho \leftarrow \mathsf{open}(cp, \phi_L, \mathrm{aux}, L(i), i)$*: Let* $\mathrm{path}(i) \subseteq G_n$ *be the first shortest path from* i *to* n *in* G_n *with respect to the lexicographical ordering.*[11] *For all nodes* j *in* $\mathrm{path}(i)$ *output the labels of all parents of* j*, i.e.,* $\rho := \big(L(\mathrm{parents}(j)), x_j\big)_{j \in \mathrm{path}(i)}$*.*
- $\mathsf{ver}(cp, \phi_L, L(i), i, \rho) \in \{0,1\}$*: For* $\mathrm{path}(i) = (i_0, \ldots, i_l = n)$ *parse* $\rho = (\rho_{i_0}, \ldots, \rho_{i_l})$ *and* ρ_{i_j} *as* (p_{i_j}, x_{i_j})*, and check for all* $j \in [l]$ *whether* $\rho_{i_{j-1}} \prec \rho_{i_j}$ *according to Def. 10; output 1 iff all these checks pass and* $\tau_n(p_{i_l}) = \phi_L$*.*

4.2 Constructing Graph-Labeling PoSWs

In this section we describe two graph-labeling PoSW schemes. Constr. 2 is a new variant of the skiplist-based PoSW construction [AKK+19], where for efficiency reasons we replace random permutations by a hash function modeled as a random oracle. Constr. 3 in Appendix D is a slight adaptation of [CP18]. Both constructions are knowledge-sound and work with arbitrary weight distributions.

For simplicity of exposition, we consider these PoSW constructions with empty augmentation. Their augmented counterparts appear in the SNACK construction in Sect. 5, where the blockchain data is the augmentation data.

A graph-labeling PoSW based on skiplists. To define the PoSW construction we specify the unspecified parts in the blueprint in Fig. 2, namely a weighted DAG family $(G_n, \Omega_n)_{n \in \mathbb{N}}$ and algorithms PoSW.label, PoSW.open, PoSW.ver, Com.

[11] Note that such a path exists since G_n has a unique sink.

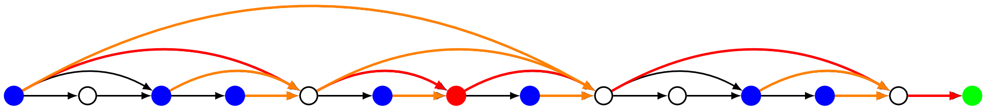

Fig. 3: Illustration of Constr. 2. The label of the last node (green) serves as the commitment. On input a challenge (red node), P opens all the nodes (blue) that are required to verify the shortest path (red edges) from source to sink which passes through the challenge node. To verify, V evaulates the opening (red and orange edges).

Construction 2. *Let $G_n = ([n]_0, E_n)$ be a DAG with edge set*

$$E_n = \left\{(i,j) \in [n]_0^2 : \exists\, k \geq 0 \ s.t.\ (j-i) = 2^k \wedge 2^k | i\right\}$$

(cf. Fig. 3). Let $\Omega_n : [n]_0 \to [0,1]$ be an arbitrary weight function, and let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function which we model as a random oracle.

- *$L :=$ PoSW.label(χ): Sample oracles $\tau_i(\cdot) := \mathcal{H}(\chi, i, \cdot)$ and output an augmented τ-based labeling $L := L^\tau$ of G_n as per Def. 4.*
- *Com is defined by SPC (Constr. 1). Hence $(\phi_L, \mathsf{aux}) \leftarrow \mathsf{SPC.commit}(cp, L)$ where $cp := \varepsilon$ is empty.*
- *$o_i \leftarrow \mathsf{PoSW.open}(\chi, cp, \phi_L, \mathsf{aux}, L, \iota_i)$: For each challenge ι_i, send the labels of all parents of the (unique) shortest path from 0 to ι_i in G_n: Let $\mathrm{path}'(\iota_i)$ denote the shortest path in G which starts at 0, ends at n and goes through ι_i. For each node in $\mathrm{path}'(\iota_i)$ output the labels of all its parents, i.e. $o_i := \big(L(\mathrm{parents}(j))\big)_{j \in \mathrm{path}'(\iota_i)}$.*
- *$b_i^{(1)} \leftarrow \mathsf{PoSW.ver}(\chi, \iota_i, o_i)$: Check the consistency of $\mathrm{path}'(\iota_i)$: For $\mathrm{path}'(\iota_i) = (i_0 = 0, \ldots, i_l = n)$ parse $o_i = (\nu_{i_0}, \ldots, \nu_{i_l})$ and check for all $i,j \in [l]$ with $(i,j) \in E_n$ whether $\nu_i \prec \nu_j$ according to Def. 10; output 1 iff all checks pass.*

We remark that we could optimize PoSW.open and PoSW.ver in Constr. 2 by removing the redundant output/checks that are already done by Com.open and Com.ver, but for readability's sake, we keep the current exposition.

In Appendix C.2 we prove that Constr. 2 is a knowledge-sound GL-PoSW as per Def. 9 for arbitrary weight function Ω_n. The proof closely resembles that of [AKK+19] by replacing the random permutations by random oracles and additionally taking into account non-uniform weights.

Theorem 2. *Let $\alpha \in (0,1]$. The scheme from Constr. 2 with parameter t and arbitrary weight function Ω_n is an (α, ϵ)-knowledge-sound augmented GL-PoSW with $\epsilon := \alpha^t + 3 \cdot q^2/2^\lambda$, where q is an upper bound on the number of the adversary's oracle queries.*

5 Constructing SNACKs from GL-PoSWs

We now show how to augment any blockchain with the computation of any (knowledge-sound) GL-PoSW scheme. This will then allow us to build a SNACK system for an augmented chain commitment language $\mathcal{L}_{\Gamma, R_\sigma, \mathsf{Com}}$, where R_σ is a PT relation that checks the validity of blocks in the (augmented) blockchain, whose genesis block is σ, as well as the consistency of the infused PoSW-related data. An accepting proof for a statement (ϕ, n) convinces a verifier that the prover *knows* a certain number of blocks (committed to by ϕ), in an augmented blockchain of length n with genesis block σ, and furthermore these blocks are (1) valid and (2) mined sequentially.

Augmented blockchains. We augment an existing blockchain by intertwining the computation of the PoSW and the mining of the blockchain. More concretely, let $\mathit{\Gamma}_n = (G_n = ([n]_0, E_G), \mathit{\Omega}_n)$ be the underlying weighted DAG of a graph-labeling PoSW scheme $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ adhering to Fig. 2 with $\mathit{\Omega}_n : [n]_0 \to [0,1]$ s.t. $\mathit{\Omega}_n([n]_0) = 1$. We furthermore assume that G_n is a chain graph as per Def. 1. Recall that the PoSW computation mainly involves computing labels of vertices by using an oracle $\tau := (\tau_i(\cdot))_{i\in[n]_0}$.

Now consider a blockchain with underlying chain graph $H_n = ([n]_0, E_H)$ and associated validity relation R_ψ. Recall (Def. 3) that a blockchain with genesis block $h_0 := \psi$ and a PT relation R_ψ is valid if and only if for every vertex $i \in [n]$, its label h_i, and its parents' labels $h_{i_1}, \ldots, h_{i_p}$, it holds

$$R_\psi(i, h_i, (h_{i_1}, \ldots, h_{i_p})) = 1 \ . \tag{6}$$

For example, in fixed-difficulty Bitcoin, the i-th block h_i has a single parent h_{i-1} and R_ψ checks whether h_i contains a valid proof of work w.r.t. h_i and h_{i-1}.

We combine the respective chain graphs G_n and H_n underlying the PoSW scheme and the blockchain to an *augmented* chain graph K_n (see Fig. 1):

$$K_n := ([n]_0, E_K) \quad \text{with} \quad E_K := E_G \cup E_H \ . \tag{7}$$

We obtain an *augmented blockchain* by labeling the chain graph K_n using algorithms Init and Mine, as formalized in Fig. 4. In particular, from an initial genesis block ψ, we define in Init, an augmented genesis block $\sigma := L_K(0)$ which contains, in addition to ψ, PoSW-related data such as χ and cp. For a vertex $i \in [n]$, algorithm Mine computes $L_K(i)$ by alternating in computing PoSW labels ℓ_i and blockchain-specific labels h_i. The computation of ℓ_i is defined as for the underlying PoSW scheme, except that the extra incoming edges inherited from the graph H_n are considered in the computation of ℓ_i. Additionally, the label ℓ_i is extended to $g_i := (\ell_i, \phi_i)$ by a commitment ϕ_i to all labels $((g_j, h_j))_{j\in[i-1]_0} || \ell_i$. The label h_i is computed the way miners in the original blockchain protocol generate blocks. Finally, the augmented label of the ith vertex is defined as $L_K(i) := k_i = (g_i, h_i)$.

In Line 4, Mine computes a proof π_i for the block, which is inherited from the underlying blockchain, for which a block $h_i := (i, d_i, \pi_i)$ was valid if $R_\psi\big(i, h_i, L_H(\text{parents}_H(i))\big) = 1$. For example, in Bitcoin, π_i is a PoW, computed based on (i, d_i) as well as the L_H-label (i.e., the corresponding block) of the single H-parent vertex in Bitcoin's chain graph H_n. In the augmented blockchain, we augment the validity relation R_ψ and define $\tilde{R}_\psi$ which still considers the same graph structure H but takes *augmented* labels as input; hence (6) becomes:

$$\tilde{R}_\psi\big(i, L_K(i), L_K(\text{parents}_H(i))\big) = 1 \ . \tag{8}$$

For example, for the augmented Bitcoin, the PoW π_i is now computed based on (i, d_i, g_i) as well as the single parent block $L_K(\text{parents}_H(i))$.

This overriding allows us to include PoSW labels g_i into the relation and make blockchain-specific labels h_i, or in particular the proofs π_i they contain, depend on the PoSW labels g_i's in a way that allows us to translate the PoSW sequentiality guarantees on g_i's to h_i's, or in particular π_i's. We elaborate more on the sequentiality guarantees towards the end of this section.

As the ith augmented block contains both blockchain-specific data h_i and PoSW-specific data g_i, we define an augmented validity relation that checks the validity of (a) the blockchain-specific data using $\tilde{R}_\psi$ and (b) the PoSW data as defined in (5), more concretely we define

$$\begin{aligned}
&R_\sigma(i, L_K(i), L_K(\text{parents}_K(i))) = 1 \Leftrightarrow \\
&\tilde{R}_\psi(i, L_K(i), L_K(\text{parents}_H(i))) = 1 \ \wedge \ \exists\, x_i \text{ s.t. } L_K(i) = \tau_i(L_K(\text{parents}_K(i))) \| x_i \ .
\end{aligned} \tag{9}$$

```
Algorithm Init: On input $1^\lambda$ and $\psi$:[12]
  1. $\chi \xleftarrow{\$} \{0,1\}^\lambda$
  2. $\ell_0 := \tau_0(\varepsilon)$
  3. $cp \leftarrow \mathsf{Com.setup}(1^\lambda)$
  4. $(\phi_0, \mathsf{aux}_0) \leftarrow \mathsf{Com.commit}(cp, \ell_0)$
  5. $g_0 := (\ell_0, \phi_0)$
  6. $h_0 := (0, d_0 := \psi \| \chi \| cp, \pi_0 := \varepsilon)$
  7. return $(\sigma := L_K(0) := k_0 := (g_0, h_0), \mathsf{aux}_0)$

Algorithm Mine: On input $\big((k_j := (g_j, h_j))_{j \in [i-1]_0}, d_i\big)$:
  1. $\ell_i := \tau_i(L_K(\mathrm{parents}_K(i)))$
  2. $(\phi_i, \mathsf{aux}_i) \leftarrow \mathsf{Com.commit}(cp, (k_j)_{j \in [i-1]_0} \| \ell_i)$
  3. $g_i := (\ell_i, \phi_i)$
  4. Compute $\pi_i$ s.t. $\tilde{R}_\psi\big(i, (g_i, h_i := (i, d_i, \pi_i)),$
                                    $L_K(\mathrm{parents}_H(i))\big) = 1$
  5. return $(L_K(i) := k_i := (g_i, h_i), \mathsf{aux}_i)$
```

Fig. 4: The mining algorithm Mine for augmented blockchains.

In order to verify that $L_K(0)$ indeed contains the blockchain genesis block ψ on which R_σ depends, we include 0 in the sequence of challenges ι.

Arguments of knowledge for augmented blockchains. We construct a SNACK system $\mathit{\Pi} = (\mathsf{SNACK.P}, \mathsf{SNACK.V})$ for the language $\mathcal{L}_{\Gamma, R_\sigma, \mathsf{Com}}$ as in Def. 6 with R_σ being as in (9). In our construction, the parameter generator G would simply output $\mathit{prm} := \sigma$ that defines R_σ.

As a first step in constructing a SNACK for $\mathcal{L}_{\Gamma, R_\sigma, \mathsf{Com}}$, we construct a succinct *interactive* argument system of chain knowledge (ACK) (P, V) for the language $\mathcal{L}_{\Gamma, R_\sigma, \mathsf{Com}}$, which is formally depicted in Fig. 6. The idea is to use the challenge/response phase of the underlying PoSW scheme, still with respect to the family $(G_n, \Omega_n)_{n \in \mathbb{N}}$, but with respect to the labeling L_K. Recall that in a PoSW scheme $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$, the verifier $\mathsf{PoSW.V}$ runs $\mathsf{Com.ver}$ and $\mathsf{PoSW.ver}$, where the latter operates w.r.t. the graph structure of G. However, in our augmentation, we added edges from H to this graph (resulting in K), which is why we extend $\mathsf{PoSW.ver}$ to $\mathsf{PoSW.ver}_K$ to take this into account. Analogously, we define $\mathsf{PoSW.open}_K$. Both algorithms are given in Fig. 5 and are used as subroutines in Fig. 6. Similarly, if $\mathsf{Com.ver}$ is defined w.r.t. some graph structure (e.g. shortest path in case of SPC or Merkle commitment), then this graph structure stays unchanged, but we require augmented labels and potentially also additional labels of H-parents (of the path).

To illustrate the necessity of these K-extended algorithms, i.e., $\mathsf{PoSW.ver}_K$ and $\mathsf{PoSW.open}_K$, consider the toy example of Fig. 1 and consider that $\mathsf{ACK.V}$ in Fig. 6 queried $\mathsf{ACK.P}$ on some $\iota_i = 5$, then the response of $\mathsf{ACK.P}$ must contain $L_K(\mathrm{parents}_H(5)) = (k_4, k_1)$ in order to verify $\tilde{R}_\psi(5, k_5, (k_4, k_1))$. However, $\mathsf{PoSW.open}$, which operates w.r.t. E_G, considers $\mathrm{parents}_G(5) = 4$ and therefore would not consider 1 to be a parent of 5, and hence k_1 would not be output by $\mathsf{PoSW.open}$. To solve this problem, we augment $\mathsf{PoSW.open}$ to $\mathsf{PoSW.open}_K$, which provides all necessary information that is needed to verify $\tilde{R}_\psi$ – for this example it provides k_1 in addition to what $\mathsf{PoSW.open}$ provides. $\mathsf{PoSW.ver}_K$ is modified accordingly.[13]

Remark 1 (On cp, ϕ_i and aux_i). *All known GL-PoSW (see Sect. 4), including those of Constr. 2 and 3, use the* SPC *commitment from Constr. 1 with* $\mathsf{cp} = \varepsilon$*. Moreover, if* $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ *is instantiated with Constr. 2 or 3, then* $\mathsf{cp} = \varepsilon, \phi_i = \ell_i$ *and* $\mathsf{aux}_i = \varepsilon$*. This means that the only*

[12] ψ is the initial genesis block, and $L_K(0)$ is the augmented genesis block.

[13] One could argue that an alternative and natural solution to this problem is to let $\mathsf{PoSW.open}$ right away work w.r.t. E_K, rather than E_G. This intuition is false, as the PoSW guarantees depend crucially on the underlying graph structure and changing the graph structure of G based on H might not maintain the PoSW guarantees.

```
Algorithm PoSW.ver_K :
On input $(\chi, \iota_i, o_i)$:
  1. Run $b_i$ := PoSW.ver$(\chi, \iota_i, o_i)$ modified as follows: whenever it queries
     $\tau_j(L_K(\text{parents}_G(j)))$ for some $j$, issue query $\tau_j(L_K(\text{parents}_K(j)))$ instead.
     (Missing labels are provided in $o_i^{(2)}$.)
  2. return $b_i$
```

```
Algorithm PoSW.open_K:
On input $(\chi, cp, \phi_n, \mathsf{aux}_n, L_K, \iota_i)$:
  1. $o_i^{(1)} \leftarrow$ PoSW.open$(\chi, cp, \phi_n, \mathsf{aux}_n, L_K, \iota_i)$
     (PoSW.open acts based on edges $E_G$.)
  2. $\mathcal{J} := \{j \in [n]_0 \colon L_K(\text{parents}_G(j))$ appear in $o_i^{(1)}\}$
  3. $o_i^{(2)} := \{(j, L_K(\text{parents}_H(j)))\}_{j \in \mathcal{J}}$
  4. return $o_i := (o_i^{(1)}, o_i^{(2)})$
```

Fig. 5: PoSW.open$_K$ and PoSW.ver$_K$ defined based on PoSW.open and PoSW.ver.

additional data stored in each block of the blockchain due to our augmentation is a label ℓ_i (no extra ϕ_i) and such a label could be a 256-bit string for a reasonable security level.

Remark 2 (Making (ACK.P, ACK.V) non-interactive). (ACK.P, ACK.V) *from Fig. 6 can be made non-interactive in the random oracle model by using the Fiat-Shamir transformation [FS87].*

In the following theorem we show that the Fiat-Shamir transform of this argument system is a SNACK system. As the underlying PoSW schemes support arbitrary weight functions Ω_n, so does our SNACK system.

Theorem 3. *Let* (SNACK.P, SNACK.V) *be the non-interactive version of* (ACK.P, ACK.V) *from Fig. 6, then in the random oracle model,* (SNACK.P, SNACK.V) *is an (α, ϵ)-knowledge-sound SNACK for $\mathcal{L}_{\Gamma, R_\sigma, \mathsf{Com}}$, as in Def. 6 and R_σ as in (9), if* (PoSW.P, PoSW.V) *is an (α, ϵ)-knowledge-sound τ-based graph-labeling PoSW as in Def. 9 with* Com *being its underlying commitment scheme, $(G_n = ([n]_0, E_G), \Omega_n)_{n \in \mathbb{N}}$ its weighted graph family, and τ modeled as a random oracle.*

To prove the theorem, we use $\Pi :=$ (ACK.P, ACK.V) and Alg. Mine (Fig. 4) to build an augmented PoSW whose knowledge-soundness implies that of the SNACK. We obtain a (non-

```
Verifier ACK.V = (V_1, V_2)
Stage V_1: On input $\eta$:
  1. $\forall i \in [t]$ do $\iota_i \xleftarrow{\$} \Omega_n$
  2. $\iota_0 := 0$
  3. send $\iota := (\iota_i)_{i=0}^t$ to P
Stage V_2: On input $\gamma = (o_i, \rho_i)_{i=0}^t$:
  1. $\forall i \in [t]_0$ do:
     (a) $b_i^{(1)} :=$ PoSW.ver$_K(\chi, \iota_i, o_i)$
     (b) $b_i^{(2)} := R_\sigma(\iota_i, L_K(\iota_i), p_i)$
     (c) $b_i^{(3)} :=$ Com.ver$(cp, \phi, L_K(\iota_i), \iota_i, \rho_i)$
  2. output $\bigwedge_{i=0}^t b_i^{(1)} \wedge b_i^{(2)} \wedge b_i^{(3)}$
```

```
Prover ACK.P:
On input $(\eta, (L_K(j))_{j \in [n]_0}, \mathsf{aux}_n)$ and $\iota$:
  1. parse $\eta$ as $\eta = (cp, \phi, n)$
  2. $\forall i \in [t]_0$ do:
     (a) $o_i \leftarrow$ PoSW.open$_K(\chi, cp, \phi, \mathsf{aux}_n, (L_K(j))_{j \in [n]_0}, \iota_i)$
         We assume $o_i$ contains $L_K(\iota_i)$ and $p_i := L_K(\text{parents}_K(\iota_i))$
     (b) $\rho_i \leftarrow$ Com.open$(cp, \phi, \mathsf{aux}_n, L_K(\iota_i), \iota_i)$
     (c) $\gamma_i := (o_i, \rho_i)$
  3. send $\gamma := (\gamma_i)_{i=0}^t$ to V_2
```

Fig. 6: The interactive proof system ACK which underlies our SNACK construction.

interactive) ($\mathsf{SNACK.P}, \mathsf{SNACK.V}$) by applying the Fiat-Shamir transform [FS87] to Π. We defer the proof to Appendix C.3.

Sequentiality of π_i's. By (α, ϵ)-knowledge soundness of ($\mathsf{SNACK.P}, \mathsf{SNACK.V}$) from Theorem 3, with probability at least $1 - \epsilon$, we can extract from any prover $\tilde{\mathsf{P}}$ that convinces $\mathsf{SNACK.V}$ of the validity of (ϕ, n), an (α, R)-valid path $(P, L_P, (p_v)_{v \in P})$ in Γ_n and an opening ρ of L_P w.r.t. ϕ. For concreteness, let $L_P = (k_{i_j} = (g_{i_j}, h_{i_j}))_{j \in [m]}$. By sequentiality of the underlying ($\mathsf{PoSW.P}, \mathsf{PoSW.V}$), it follows that $(g_{i_j})_{j \in [m]}$ was computed sequentially. However, for the corresponding proofs $(\pi_{i_j})_{j \in [m]}$ in $(h_{i_j})_{j \in [m]}$, it was not explicitly required by Mine (see Fig. 4) that π_{i_j} is computed *after* its corresponding, and sequentially computed, g_{i_j}. Therefore, in principle, all of these $(\pi_{i_j})_{j \in [m]}$ could have been computed in parallel by $\tilde{\mathsf{P}}$ *before* $\tilde{\mathsf{P}}$ started the sequential computation of $(g_{i_j})_{j \in [m]}$. This shows that while a SNACK system guarantees sequentiality on the augmented graph K_n via the sequentiality on G_n, this sequentiality does not necessarily translate to sequentiality on H_n, and guaranteeing sequentiality on H_n is what most applications of a SNACK system rely on (see Sect. 6).

This issue is however easy to resolve in any blockchain: To ensure that the proofs $(\pi_{i_j})_{j \in [m]}$ in $(h_{i_j})_{j \in [m]}$ were computed sequentially, it suffices for the blockchain mining protocol Mine to ensure the following:

Assumption: For all $j \in [n]$ it holds that π_j must have been computed after g_j.

Then the sequentiality of $(\pi_{i_j})_{j \in [m]}$ directly follows from the sequentiality of $(g_{i_j})_{j \in [m]}$: now that π_{i_j} is computed after g_{i_j}, one could think of this as an edge from g_{i_j} to π_{i_j}, and as g_{i_j} is computed after $k_{i_{j-1}} = (g_{i_{j-1}}, h_{i_{j-1}})$, a path that goes through $(g_{i_1}, \ldots, g_{i_m})$ translates to a path that goes through $(g_{i_1}, \pi_{i_1}, \ldots, g_{i_m}, \pi_{i_m})$, hence ensuring sequentiality of the proofs.

The above assumption is trivially satisfied by any blockchain which has immutability as one of its defining properties: a block in the chain cannot be modified without modifying all subsequent blocks. For example, in Bitcoin, where $h_j = (j, d_j, \pi_j)$, as assumed in Mine, contains a valid PoW π_j with respect to d_j where d_j is implicitly assumed to contain the hash of the previous block. Simply including g_j in d_j ensures that π_j is computed after g_j.

See Appendix E for instantiations and optimizations of the SNACK protocol.

6 Applications to Blockchain Light Clients

We now describe how SNACKs can be applied in the design of light-client blockchain protocols. Informally speaking, the goal of such protocols is to allow a light client, who only knows the genesis block, to *bootstrap* by obtaining a commitment to a chain that is, except for some unreliable suffix, matching the chains being held by honest full nodes. Others can then prove statements about the honest chain w.r.t. this commitment, and the light client does not need to trust the provers. The commitment thus serves as an anchor of trust.

We start by defining some vocabulary that allows us to make this intuition precise. We consider a long-term execution of a blockchain ledger protocol Π by a set of parties called *full nodes*. At a particular time t, we call a party *honest* if it is uncorrupted by the adversary (and hence follows Π), it is online and fully synchronized with the state of the protocol Π.[14] A chain is called *honest*

[14] Such honest parties are called *alert* in [PS17, BGK+18]; we will not maintain this distinction and will always assume honest parties to be alert.

at time t if it is held by some honest party executing the full protocol, i.e., a full node. We call an honest chain *maximal* if it has maximal length among all honest chains. Note that several different maximal chains might exist for any t, but they share the same length which we call the *honest length* at time t. We call an honest party *synchronized* at time t if it is holding a maximal chain.

General assumptions. The κ-*common-prefix* property [GKL15, PSs17] mandates that any two chains C_1, C_2 that are honest at times $t_1 \leq t_2$ satisfy $C_1^{\lceil\kappa} \preceq C_2$, where $(\cdot)^{\lceil\kappa}$ denotes removing the last κ blocks of a chain and $\preceq$ is the prefix relation. (This property has become a standard requirement for Nakamoto-style blockchain protocols: together with *chain growth* and *chain quality*, it implies consistency and liveness of the produced ledger [GKL15, PSs17].) It has been shown to be achieved, except with an error negligible in κ, by Nakamoto-style protocols across various Sybil-protection mechanisms: proof of work [GKL15, PSs17, BMTZ17], proof of stake [KRDO17, DPS19, DGKR18, BGK$^+$18], and proof of space [CP19]. We will assume that the considered blockchain protocol Π satisfies κ-common prefix for some κ so that the probability of this assumption being violated is acceptably small. We refrain from mentioning this error explicitly in our statements to maintain readability.

We assume that the execution of the protocol Π results in a family of chain graphs $\{K_n = ([n]_0, E_n)\}_{n\geq 0}$, meaning that at any point in the execution of Π, any blockchain with n blocks produced by Π has its chain-graph structure determined by K_n (cf. Def. 1). (Thus the chain-graph structure is independent of the data contained in the blocks.) Recall that for every $n \geq 0$, K_n is assumed to be obtained from K_{n+1} by removing vertex $n+1$ and adjacent edges.

Forking adversaries. In our analysis we need to assume some limitation on an adversary's ability to create an alternative chain that "forks away" from any currently honest chain at some significant depth and achieve (at least) the honest length, even if it contains some fraction of invalid blocks. This type of assumption was first described in [BKLZ20], who explicitly allow the adversary to include (a limited fraction of) *invalid* blocks in its fork. Below we discuss how we formalize the spirit of their assumption for our setting in Def. 11, while overcoming some shortcomings of the original formulation.

First, in their context of a PoW chain, an invalid block may contain an incorrect proof of work, but every block must still contain a correct hash of its predecessor. (If this was not required, the assumption would be false, as the adversary could simply glue parts of the honest blockchain together.) We capture validity of blocks by an abstract relation R, without assuming anything about "invalid" blocks. As we consider blockchains whose underlying graph structure is an arbitrary (rather than simple) chain graph, we employ the notion of R-valid paths as per Def. 5 to formally define forks. Our assumption requires that an adversary that creates a (valid) path that forks away from the honest blockchain sufficiently deep (as specified by a parameter ℓ) can only include in its path a c-fraction of blocks after the forking point. Note that in a typical PoW blockchain, any such adversary could also create the blocks not lying on its path by ignoring the PoW but including a correct hash; this would then also violate the assumption in [BKLZ20]. From this perspective, our assumption is *weaker* than that of [BKLZ20], as the adversary has to achieve a c-fraction of valid blocks *along a path*, but it is *stronger* in that the adversary need not produce blocks (with valid hashes) outside of its path.

Furthermore, the original assumption [BKLZ20] does not consider adversaries that create a fork (potentially containing invalid blocks) whose length exceeds the honest length n_h. As we show, such adversaries can be used to break light-client protocols in the sense of the precise security definition (Def. 12) we give. Our assumption will thus also contain a limit on the adversary's power of *extending* chains. Intuitively, this does not make the assumption stronger, since if the last block of the adversary's fork is at position $n^* > n_h$, then the adversary's task is harder, as it needs to include more blocks in its path so a c-fraction of its (now longer) fork is valid. A definitional subtlety arises when $n^* \geq n_h + \ell - \kappa$, meaning that a fork of length ℓ could start beyond the stable prefix of the honest chains, which ends at $s_h := n_h - \kappa$. If the adversary's chain agrees with the honest prefix, then "forking from the honest chains" is not well-defined, as honest chains can differ after s_h. In this case we consider the forking point f to be the adversary's last block before s_h. (This is captured by Case (2) in Def. 11 below.)

One might wonder if forking from an honest chain at some point $f' > s_h$ could give the adversary an advantage, as the c-fraction is now measured between $f < f'$ and n^*. However, a similar situation could also arise before s_h, in which case it is subject to Case (1) in Def. 11 (which corresponds to the original assumption [BKLZ20]): there might be an (honest) orphaned fork of length $\kappa' \leq \kappa$ (these are not excluded by κ-common-prefix) forking at $n_h - \ell$ and the adversary can try to extend it. To achieve a c-fraction in $[n_h - \ell + 1 : n_h]$, the adversary thus needs to mine $c\ell - \kappa'$ blocks while the honest miners only mine $\ell - \kappa'$ blocks. Arguably, a Case (2) fork is harder to compute: consider an adversary that extends an instable honest chain after $s_h + \kappa'$. Then to achieve a c-fraction of blocks in $[s_h + 1, n_h + \ell - \kappa]$ (the optimal choices of forking point f in Def. 11 and n^*), the adversary has to mine $c\ell - \kappa'$ while the honest miners only mine $\kappa - \kappa'$ blocks (thus in less time than in the Case (1) example before).

Definition 11 ((c, ℓ)-forks and (c, ℓ, ϵ_F)-adversaries). *Let Π be a blockchain protocol with validity relation R and chain graph $(K_n)_{n \geq 0}$, satisfying κ-common prefix. Let $c \in (0, 1]$, $\ell \in \mathbb{N}$ with $\ell > \kappa$. Fix some time t in the execution of Π and let n_h be the honest length at time t and $L_h \colon [s_h]_0 \to \{0,1\}^*$ be the labeling of the honest stable prefix, with $s_h := n_h - \kappa$.*

- *An ℓ-*fork *is an R-valid (Def. 5) path $\big(P = (i_0 = 0, \ldots, i_q = n^*), L, (p_v)_{v \in P}\big)$ in K_{n^*}, such that $n^* \geq n_h$ and either*
  - *(1) for some $j \in [q]$: $i_j \leq s_h$ and $n^* \geq i_j + \ell - 1$, and we have:*
    $$L(i_{j-1}) = L_h(i_{j-1}) \text{ and } L(i_j) \neq L_h(i_j).$$
  - *(2) or $n^* \geq s_h + \ell$ and for all $i_j \leq s_h$: $L(i_j) = L_h(i_j)$.*
- *A (c, ℓ)-*fork *is an ℓ-fork $(P, L_P, (p_v)_{v \in P})$ for which P contains at least a c-fraction of the blocks after the forking point f, i.e.,*
  $$\big|P \cap [f + 1 : n^*]\big| \geq c \cdot (n^* - f) \ ,$$
  where $f := i_{j-1}$ in Case (1) and $f := \max\{i_j : i_j \leq s_h\}$ in Case (2).
- *A (c, ℓ, ϵ_F)-*adversary *against Π is an adversary whose probability of producing a (c, ℓ)-fork at any point throughout the execution of Π is at most ϵ_F.*

Observe that for $c = 1$, the adversary's goal collapses to an ℓ-common-prefix violation. For PoW, one can thereforerely on existing bounds such as [GRR21]. For general c, the connection between the assumption of (c, ℓ, ϵ_F)-adversaries (or the original assumption in [BKLZ20]) and more basic blockchain assumptions remains open.

Our results. The high-level idea of our protocols is that any blockchain commitment suggested by a prover for adoption by a light client needs to be accompanied by a SNACK proof parametrized in such a way that no (c, ℓ, ϵ_F)-adversary would be able to produce this SNACK for a chain that does not share the necessary common prefix with honest chains, as this would require the prover (by SNACK knowledge soundness) to construct a valid path that is beyond the capabilities of any such restricted adversary.

As a warm-up, in Sect. 6.1 we present a naive SNACK-based protocol for light-client bootstrapping in the multi-prover setting. The light client obtains (concise) information from several full nodes, and, informally speaking, if at least one of them is (honest and) synchronized then the light client will end up holding a commitment to a chain of honest length. (If *all* provers are malicious, the client might adopt a commitment to a chain arbitrarily violating the common-prefix property or the maximal-length requirement.) This is analogous to the guarantees provided by the FlyClient protocol [BKLZ20].

We also present a simple variant of our first protocol for settings where the light client can be assumed to know an approximation of the current honest length (e.g. derived from the time passed since the client's previous bootstrapping). This protocol provides meaningful guarantees even when run with a single (potentially malicious) prover.

We don't formally analyze these two protocols, as their security guarantees described informally above turn out to be insufficient for the most common practical setting. To illustrate this, in Sect. 6.2 we describe an attack against the bootstrapping approach taken by both our proposals as well as a naive use of previous work, the *void-commitment attack*. It consists of an adversary producing a private chain almost identical to some maximal honest chain, and luring the light client into accepting a commitment to his chain. The obtained commitment is then useless in future interactions if the adversary keeps the opening secret.

Motivated by this attack, in Sect. 6.3 we first give a meaningful definition of *secure bootstrapping* (Def. 12): a light client is guaranteed to obtain a commitment to a *stable common prefix* of all honest chains, which can then serve as an anchor of trust. Anyone holding an honest chain can then prove properties about its stable prefix to the light client. We then propose our final SNACK-based protocol (Fig. 9) and prove that it satisfies our definition of secure bootstrapping. We do so via a technical result (Theorem 4) that allows us to reason about the limitations of (c, ℓ, ϵ_F)-adversaries in creating forks with (α, R)-valid paths and hence also producing valid SNACKs.

SNACK-compatible blockchains and commitments. As our protocols rely on a SNACK system, we assume the blockchain in question admits such a system. GL-PoSW-augmented blockchains as presented in Sect. 5 are one example. We let K_n for $n \in \mathbb{N}$ denote the DAG (technically a *chain graph* as in Def. 1) of such a blockchain of length n and let R be the relation that defines validity of its blocks. (In the construction in Sect. 5 this corresponds to R_σ from Equation (9).) We let Com be the commitment scheme for which the light client should obtain a commitment to the chain. Together, these define the language $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ for the SNACK system, where $\Gamma := (K_n, \Omega_n)_{n \geq 0}$ and Ω_n which we will define. In addition, we assume the following:

Assumption: Every block (header) of the blockchain contains a Com-commitment to all the previous blocks.

When considering the SPC commitment (Constr. 1) in GL-PoSW-augmented blockchains, this assumption trivially holds: the commitment contained in a block is simply the τ-evaluation (the

```
On input tuples $(\phi_i, n_i, \pi_i)_{i \in [N]}$ the light client does the following:
 1. For all tuples, ordered by decreasing values of $n_i$:
     (a) let $\alpha_i$ and $\Omega_i$ be as discussed in the text
     (b) check if $\pi_i$ is a valid proof for the statement $(\phi_i, n_i)$ for an $(\alpha_i, \epsilon)$-knowledge-sound SNACK
         system for $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ with weight function $\Omega_i$
     (c) if $\pi_i$ verifies then stop and return $(\phi, n) := (\phi_i, n_i)$
 2. Return $\perp$.
```

Fig. 7: Protocol 1: A Naive Light-Client Bootstrapping Protocol (informal).

"hash") of the parent labels (blocks). The assumption is also true for "FlyClient-compatible" blockchains, whose block headers need to contain *Merkle mountain range commitments* [BKLZ20].

Further note that in our (and previous [BKLZ20]) approaches to light-client protocols the commitment provided by a prover cannot be independent of the blockchain: otherwise, a malicious prover could modify (and thereby invalidate) a single block in (the stable prefix of) an honest chain, commit to the altered chain and later prove properties of the modified block to any verifier holding the commitment. As this modified blockchain does *not* constitute a (c, ℓ)-fork (neither w.r.t. Def 11 nor the original assumption [BKLZ20]), the light-client protocol cannot protect against it. (In particular, the adversary can still compute a SNACK by proving knowledge of a (heavy) path that does not go through the node.)

6.1 Naive Bootstrapping Protocols

In Protocol 1, the "naive" protocol, the light client receives from each of N full nodes a commitment ϕ_i to a purported blockchain of length n_i and a SNACK proof π_i for the statement (ϕ_i, n_i). It outputs (one of) the commitment(s) for the maximal value n that is accompanied by a valid SNACK proof (see Fig. 7).

The SNACK proofs are parametrized (via α_i and Ω_i) based on the proclaimed value of n_i so that they prove knowledge of a path which, assuming (c, ℓ, ϵ_F)-adversaries (Def. 11), must share the necessary common prefix with the honest blockchain. This can be quantified precisely, and we provide respective parameters for our final protocol in Theorem 4. The intuition behind the design of Protocol 1 is not flawed, and in fact, the same reasoning (expressed more concretely and implicitly using a specific SNACK) lies behind FlyClient. We omit the detailed treatment for Protocol 1 as its main purpose is to motivate the subsequent attack.

Protocol 1 does have some use cases, for example if the prover convincing the light client is the entity that later proves statements about the honest chain. Another application is obtaining an estimate of the honest length in the multi-prover setting (by simply outputting the length n_i of the accepted tuple).

Assuming the light client knows the current honest length n_h, a simple variant of Protocol 1, presented in Fig. 8, can then be run with a *single prover*. It guarantees that either the client again obtains a commitment to an honest-length chain or she learns that the prover is malicious or lagging behind and hence should not be trusted. In fact, also imprecise estimates of n_h (such as those obtained from Protocol 1) can be leveraged in a similar way.

On input the honest length n_h and a tuple (ϕ, n, π), if $n < n_h$, return $\perp$. Else:

1. let α_n and Ω_n be as discussed in the text;
2. if π is valid for (ϕ, n) for an (α_n, ϵ)-knowledge-sound SNACK system for $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ with weight function Ω_n then return ϕ;
3. else return $\perp$.

Fig. 8: Protocol 2: Single-Prover Bootstrapping with Length (informal).

6.2 The Void-Commitment Attack

The following attack applies to Protocols 1 and 2, as well as a naive use of the FlyClient protocol. We stress that the attack does not contradict security claims in previous work (or the informal argument given in Sect. 6.1), but rather highlights that despite these claims, protocols following the structure described in Sect. 6.1 fall short of solving a class of important practical use cases.

The attack works as follows: an adversary A first obtains from an honest full node some maximal honest chain; let h be (the header of) its terminating block. A then mines a new block h' on top of h, which it will however keep secret. It then participates as prover in one of the mentioned protocols following its specification, but using its chain terminating with h'. If A's commitment is adopted by the light client then the latter will hold a commitment to a chain to which only A knows an opening, which will be of no use for "bootstrapping" applications that would involve interactions with other full nodes.

Note that even if the protocol is run with honest full nodes only, the light client may still end up with a similarly unusable commitment if the prover's chain will eventually lose the "longest-chain race" and not become part of the prefix of future honest chains.

This observation extends to the FlyClient protocol if used naively: if the light client only keeps the last block and the contained commitment as its output from the bootstrapping (rather than the full ℓ-suffix), further interactions with other full nodes would suffer from the above attack.

6.3 A Light-Client Protocol for Common-Prefix Commitment

We now show that, using SNACKs, the original intuitive goal can still be achieved: (a) obtain a commitment guaranteed to be to (a relevant part of) an honest-length chain; (b) anyone, not only the commitment provider, can use it to prove statements about the ledger state. While it would be desirable to guarantee a commitment to the stable prefix $[s_h]_0$ with $s_h = n_h - \kappa$ of the honest chain, this cannot be achieved when only making the assumption of (c, ℓ, ϵ_F)-adversaries: the latter only precludes forks of length ℓ, so a (c, ℓ, ϵ_F)-adversary could create a differing block at position $s_h + 1$. But then it could also insert a "malicious" commitment there (cf. our discussion before Sect. 6.1). In order to fix the commitment to the honest chain, we therefore place it at latest at position $n_h - \ell + 1$.

Definition 12 (Secure common-prefix bootstrapping). *For $\kappa, \ell \in \mathbb{N}$, let Π be a blockchain protocol satisfying κ-common prefix for $\kappa < \ell$ for which each block contains a commitment to its predecessors. Fix some time t and let n_h be the honest length at time t and $(h_0, \ldots, h_{s_h})$ be the (headers of the) stable prefix of the honest chain. A light client securely* ℓ-common-prefix bootstraps *(*ℓ-CP bootstraps*) at time t if for some $m \in [n_h - \ell : s_h - 1]$ it ends up holding the commitment ϕ to $(h_0, \ldots, h_m)$ contained in h_{m+1}.*

Let Π be blockchain protocol with validity relation R, chain graph $(K_n)_{n\geq 0}$ and parameters prm. On input N tuples of the form

$$\big(\phi, n, \pi, (k_i)_{i=n-\ell+1}^{n}, (\iota_j, k_{\iota_j}, \rho_{\iota_j})_{j=1}^{q}\big) \text{ for some } q \leq \ell \cdot \deg(K_n),$$

for all tuples, ordered by decreasing values of n, check the following:

(a) Let $m := n - \ell$; check whether ϕ is contained in k_{m+1}

(b) $\mathsf{SNACK.V}\big(prm, (\phi, m), \pi\big) \overset{?}{=} 1$, where SNACK is (α_m, ϵ)-knowledge-sound for $\mathcal{L}_{(K_m, \Omega_m)_{m\geq 0}, R, \mathsf{Com}}$ with α_m and Ω_m as in Theorem 4

(c) For all $i \in [m+1 : n] : R(i, k_i, (k_j)_{j\in \text{parents}(i)}) \overset{?}{=} 1$

(d) For all $j \in [1 : q] : \mathsf{Com.ver}(cp, \phi, k_{\iota_j}, \iota_j, \rho_{\iota_j}) \overset{?}{=} 1$

(e) If all checks verify, return (ϕ, n)

Return $\perp$

Fig. 9: Light-Client Protocol 3: Multi-Prover Bootstrapping Common Commitment

In Protocol 3 , given in Fig. 9, instead of committing to the entire chain, the full nodes commit to the (stable) prefix of the honest chain of length $n - \ell$, and use a SNACK to prove knowledge of a heavy chain contained in the commitment. Now to ensure that the commitment is actually to the stable prefix, we require the provers to show that they know an extension by ℓ blocks of what was committed. For simplicity, the provers simply send (the headers of) these blocks and give commitment openings to the blocks that are necessary to check their validity. (We refrain from optimizing this further by only checking samples from the last ℓ blocks.)

We next define appropriate values α_m, Ω_m for the SNACK scheme used in our light-client Protocol 3, which will guarantee secure ℓ-CP bootstrapping.

Theorem 4. *Let Π be a blockchain protocol with underlying graph $(K_n)_{n\geq 0}$ satisfying κ-common prefix. Let $c \in (0,1]$ and $\ell \in \mathbb{N}$ with $\ell > \kappa$. For $m \in \mathbb{N}$ define α_m and $\Omega_m \colon [m]_0 \to [0,1]$ as*

$$\begin{aligned} \alpha_m &:= 1 - \big(\log_c\big(\tfrac{\ell-1}{m+\ell}\big)\big)^{-1} \\ \Omega_m(i) &:= S \cdot \tfrac{1}{m+\ell-i} \quad \text{for} \quad 0 \leq i \leq m \qquad \text{where} \quad S := \big(\textstyle\sum_{j=0}^{m} \tfrac{1}{m+\ell-j}\big)^{-1}. \end{aligned} \tag{10}$$

Then, except with probability at most ϵ_F, no (c, ℓ, ϵ_F)-adversary can create an ℓ-fork $(P, L_P, (p_v)_{v\in P})$ in K_{n^*} where P starts at 0 and ends at n^*, such that, with $m^* := n^* - \ell$:

- *P contains the last ℓ blocks, i.e., $[m^* + 1 : n^*] \subseteq P$, and*
- *P has weight at least α_{m^*} w.r.t. Ω_{m^*}, i.e., $\Omega_{m^*}(P \cap [m^*]_0) \geq \alpha_{m^*}$.*

First, we give a proof overview. Let K_n be the blockchain graph and $m := n - \ell$. Initially, we keep α_m indeterminate and define a weight function Ω_m on K_m, which is inspired by the sampling distribution of the FlyClient protocol [BKLZ20]. We then give the adversary's optimal strategy, which, given the constraints on its resources, maximizes the weight of its chain. Finally, we set α_m large enough, so that the optimal (and thus every) chain produced by a (c, ℓ, ϵ_F)-adversary has weight less than α_m.

A (c, ℓ, ϵ_F)-adversary, whose goal is to produce an ℓ-fork at time t that contains all the last ℓ blocks, is limited to choosing a point f (see Def. 11), after which it forks from some existing chain, and some length $n^* \geq n_h$, where n_h denotes the honest length at time t, and deciding which blocks after f to include in its path. (We do not make any assumptions on the blockchain graph and assume any sequence of blocks is a path.) By assumption, after the forking point there can be at most $c(n^* - f)$ valid blocks.

We will use an increasing weight distribution $\varOmega_m$, that is, later blocks in K_m weigh more. So to maximize the weight of its path, the adversary must put all its blocks just preceding the last ℓ blocks. Specifically, we adopt the hyperbolically increasing function from [BKLZ20], which assigns to the i-th block weight proportional to $\frac{1}{m+\ell-i}$. For any forking point f, the weight of the $(1-c)$-fraction of the blocks after f (which the adversary must skip) is the same, so all forking points are "equally bad".

Proof (of Theorem 4). Consider some fixed point in time t when the honest length n_h is at least ℓ. Since the weight function $\varOmega_m$ is monotonically increasing, an optimal strategy for a (c, ℓ, ϵ_F)-adversary is to choose a forking point f and some length $n^* \geq n_h$ such that $f \leq n^* - \ell$, and put no blocks in the interval $\mathsf{skp} := \big[f+1 : f + \lceil (1-c)(n^*-f) \rceil\big]$. Since its goal is an R-valid path with $[m^*+1 : n^*] \in P$, it must choose f and n^* such that $f + \lceil (1-c)(n^*-f) \rceil \leq m^*$. For an arbitrarily fixed f, the weight of the blocks in this interval is thus

$$\varOmega_{m^*}(\mathsf{skp}) = S \cdot \sum_{i=f+1}^{f+\lceil (1-c)(n^*-f) \rceil} \frac{1}{n^* - i} \ . \tag{11}$$

In order to lower-bound the above, we use the following bounds, derived by using the upper and lower (Riemann) sums of a strictly increasing function:

$$\int_a^b \frac{1}{n^*-x} dx \ < \ \sum_{i=a+1}^{b} \frac{1}{n^*-i} \ < \ \int_{a+1}^{b+1} \frac{1}{n^*-x} dx \ . \tag{12}$$

Since $\int \frac{1}{n^*-x} dx = -\ln(n^*-x)$, we get $\int_a^b \frac{1}{n^*-x} dx = \ln\big(\frac{n^*-a}{n^*-b}\big)$.

We thus have $\sum_{i=f+1}^{f+\lceil (1-c)(n^*-f) \rceil} \frac{1}{n^*-i} > \ln\big(\frac{n^*-f}{n^*-f-(1-c)(n^*-f)}\big) = -\ln(c)$. (Note that this is independent of the forking point f and the length n^*, which was the reason for choosing this function in [BKLZ20].) Using the upper bound from (12), we moreover have $\sum_{i=0}^{m^*} \frac{1}{n^*-i} < \ln\big(\frac{n^*-0}{n^*-(m^*+1)}\big) = \ln\big(\frac{n^*}{\ell-1}\big)$. Using these two bounds to lower-bound the right-hand side of (11), we get

$$\varOmega_{m^*}(\mathsf{skp}) > -\ln(c) \cdot \big(\ln\big(\tfrac{n^*}{\ell-1}\big)\big)^{-1} = \big(\log_c\big(\tfrac{\ell-1}{n^*}\big)\big)^{-1} =: \bar{\varOmega} \ .$$

The weight of the adversary's path is at most $1 - \varOmega_{m^*}(\mathsf{skp})$ and thus strictly less than $1 - \bar{\varOmega}$. Setting $\alpha_{m^*} := 1 - \bar{\varOmega}$ gives $\alpha_{m^*} = 1 - \big(\log_c\big(\frac{\ell-1}{m^*+\ell}\big)\big)^{-1}$.

We have thus shown that, except with probability ϵ_F, no (c, ℓ, ϵ_F)-adversary can produce an ℓ-fork $(P = (0, \ldots, n^*), L_P, (p_v)_{v \in P})$ such that P contains all the last ℓ blocks and has weight α_{m^*} w.r.t. $\varOmega_{m^*}$, which implies the theorem. □

Theorem 4 now allows us to prove that Protocol 3 provides a light client for ℓ-CP bootstrapping secure against (c, ℓ, ϵ_F)-adversaries.

Corollary 1 (Security of Protocol 3). *Let $\kappa, \ell \in \mathbb{N}$ with $\ell > \kappa$ and let Π be a blockchain protocol with validity relation R and graph family $(K_n)_{n \geq 0}$ that satisfies κ-common prefix. Assume the honest length $n_h > \ell$ and consider a light client running Protocol 3 (Fig. 9) with N full nodes, at least one of which is (honest and) synchronized and all others are controlled by a (c, ℓ, ϵ_F)-adversary. Let (ϕ^*, n^*) be the output of the light client. For $m \geq 0$, let α_m and Ω_m be as in Theorem 4. If*

- Com *is ϵ_C-position-binding and*
- SNACK *is $(\alpha_{n^*-\ell}, \epsilon)$-knowledge-sound for language $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ where $\Gamma := (\Gamma_m = (K_m, \Omega_m))_{m \geq 0}$,*

then the client securely ℓ-CP-bootstraps except with probability $\epsilon + \epsilon_C + \epsilon_F$.

We give a proof sketch and defer the formal proof to Appendix C.4. Assume a (c, ℓ, ϵ_F)-adversary that prevents the light client from bootstrapping and let (ϕ, n) be its output. We have $n \geq n_h$, the maximum honest length, since otherwise an honest miner would have convinced the light client.

By (α_m, ϵ)-knowledge soundness, for $m := n - \ell$, of the SNACK sent by the adversary, we can extract an R-valid path in K_m of weight at least α_m. We extend it by the sent blocks $k_{m+1}, \ldots, k_n$ to a path P in K_n, which satisfies the two requirements at the end of Theorem 4. Moreover, P is R-valid, since position-binding of Com guarantees that the sent parent labels $k_{\iota_1}, \ldots, k_{\iota_q}$ conform to those of the extracted path.

Let $s_h := n_h - \kappa$ denote the length of the honest stable prefix. Finally, P is an ℓ-fork, since either $n \geq s_h + \ell$ and it agrees with the stable prefix of the honest chain (fork of Type (2) in Def. 11), or it forks off earlier, at latest at $m + 1$ (since ϕ contained in block $m + 1$ must be different from the commitment contained in the honest prefix for the client not to bootstrap), meaning it is a fork of Type (1). By Theorem 4, no (c, ℓ, ϵ_F)-adversary can create such a fork, which proves Corollary 1.

In Appendix F, we show that when using the SNACK based on (our variant of) the PoSW by Cohen and Pietrzak [CP18] (given in Appendix D), Protocol 3 achieves virtually the same efficiency as FlyClient [BKLZ20].

7 Incremental PoSWs and SNACK Systems

A very powerful extension of non-interactive PoSW introduced in [DLM19] are *incremental* PoSWs, which additionally allow to extend a given proof π, witnessing n sequential computational steps, to a new proof for $n+n'$ steps by investing additional n' sequential steps. Note that all non-interactive PoSW schemes, including the non-interactive counterparts of Constr. 2 and 3, are technically two-message protocols, in which the verifier V_0, on input 1^λ, picks parameters prm, upon which the prover runs and outputs a proof for a value n that is verified by V_1.

Definition 13 (Incremental GL-PoSW). *A tuple of PPT algorithms* $(\mathsf{P}, \mathsf{V} := (\mathsf{V}_0, \mathsf{V}_1), \mathsf{INC})$ *is an* incremental *graph-labeling PoSW if* (P, V) *is a non-interactive graph-labeling PoSW and the following holds:*

- **Incrementality:** *For every $\lambda, n_1, n \in \mathbb{N}_0$ with $n_1 \leq n$, and every $prm \leftarrow \mathsf{V}_0(1^\lambda)$, we have* $\mathsf{V}_1(prm, n, \pi) = 1$ *for every honestly generated proof π, where we say π is* honestly generated *for parameters prm and n if either of the following holds:*
  - $\pi \leftarrow \mathsf{P}(prm, 1^n)$*, or*
  - $\pi \leftarrow \mathsf{INC}(prm, 1^{n-n_1}, n_1, \pi_1)$ *and π_1 is honestly generated for parameters prm and n_1.*

Döttling et al. [DLM19] show how to make the PoSW scheme of [CP18] (which is implicitly defined for uniform weight distributions Ω_n) incremental, and their techniques extend naturally to the augmented PoSW based on [CP18] which we gave in Constr. 3 (Appendix D). It remains open to adapt their techniques to arbitrary weight functions Ω_n. We now define incremental SNACKs.

Definition 14 (Incremental SNACK systems). *A tuple of PPT algorithms* $(\mathsf{P}, \mathsf{V}, \mathsf{INC})$ *is an* incremental SNACK *for* $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ *from Def. 6 (with witness relation* $\mathcal{R} := \mathcal{R}_{\Gamma,R,\mathsf{Com}}$*) if* (P, V) *is a SNACK system for* $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ *and the following holds:*

- **Incrementality***: For every* $\lambda, n_1, n \in \mathbb{N}_0$ *with* $n_1 \leq n$*, it holds that* $\mathsf{V}(prm, (\phi, n), \pi) = 1$ *for every honestly generated proof* π*, where we say* π *is* honestly generated *for prm and* (ϕ, n) *if either of the following holds:*
  - $\pi \leftarrow \mathsf{P}(prm, (\phi, n), w)$ *for some* w *with* $(prm, (\phi, n), w) \in \mathcal{R}$
  - $\pi \leftarrow \mathsf{INC}(prm, \pi_1, (\phi, n), w_2)$ *for some* π_1 *that is honestly generated for prm and* (ϕ_1, n_1) *via some* w_1 *and* w_2 *such that* $(prm, (\phi, n), w_1 \| w_2) \in \mathcal{R}$*.*

We remark that it follows from the definition that the running time of INC is independent of w_1.

Inspection of our SNACK construction (P, V) from a GL-PoSW in Sect. 5 suggests that if the underlying GL-PoSW is incremental, then this naturally yields an algorithm INC so that $(\mathsf{P}, \mathsf{V}, \mathsf{INC})$ is an incremental SNACK. This approach still works for augmented labels, however adding additional edges representing blockchain validity dependencies might break the incrementality property.

We can thus define incremental SNACKs for blockchains based on *simple* chain graphs. An easy fix to support a more general class of blockchains could be to add to the proofs π commitment openings of all nodes to which any future blockchain blocks will refer, which however leads to an unfavorable increase in proof size. We leave it for future work to analyze and construct incremental SNACK systems in a more formal way.

Acknowledgements. The first two authors are funded by the Vienna Science and Technology Fund (WWTF)[10.47379/VRG18002]. The first author has also received funding in part from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), the Spanish Government under projects SCUM (ref. RTI2018-102043-B-I00), the Madrid Regional Government under project BLOQUES (ref. S2018/TCS-4339), and a research grant from Nomadic Labs and the Tezos Foundation. The last author is supported in part by ERC CoG grant 724307 and conducted part of this work at ISTA, funded by the ERC under the European Union's Horizon 2020 research and innovation programme (682815 - TOCNeT).

References

AAC$^+$17. Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 357–379. Springer, Heidelberg, December 2017.

AKK$^+$19. Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 277–291. Springer, Heidelberg, May 2019.

BBBF18. 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. Springer, Heidelberg, August 2018.

BCD^{+}14. Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling Blockchain Innovations with Pegged Sidechains. https://blockstream.com/sidechains.pdf, 2014. [Online; accessed 16-August-2019].

BCKL08. Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374. Springer, Heidelberg, March 2008.

BGK^{+}18. Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 913–930. ACM Press, October 2018.

BKLZ20. Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. FlyClient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy*, pages 928–946. IEEE Computer Society Press, May 2020.

BMRS20. Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352, 2020. https://eprint.iacr.org/2020/352.

BMTZ17. Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 324–356. Springer, Heidelberg, August 2017.

CP18. 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. Springer, Heidelberg, April / May 2018.

CP19. Bram Cohen and Krzysztof Pietrzak. The Chia Network blockchain, July, 2019. https://www.chia.net/assets/ChiaGreenPaper.pdf.

DFKP15. Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Heidelberg, August 2015.

DGKR18. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018.

DKKZ20. Stelios Daveas, Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. A gas-efficient superlight bitcoin client in solidity. Cryptology ePrint Archive, Report 2020/927, 2020. https://eprint.iacr.org/2020/927.

DLM19. Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 292–323. Springer, Heidelberg, May 2019.

DPS19. Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 23–41. Springer, Heidelberg, February 2019.

FS87. 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. Springer, Heidelberg, August 1987.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

GKL15. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.

GKZ19. Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy*, pages 139–156. IEEE Computer Society Press, May 2019.

GRR21. Peter Gaži, Ling Ren, and Alexander Russell. Practical settlement bounds for proof-of-work blockchains. Cryptology ePrint Archive, Report 2021/805, 2021. https://eprint.iacr.org/2021/805.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

KKZ19. Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Compact storage of superblocks for NIPoPoW applications. In Panos M. Pardalos, Ilias S. Kotsireas, Yike Guo, and William J. Knottenbelt, editors, *MARBLE 2019*, pages 77–91. Springer, 2019.

KLS16. Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 61–78. Springer, Heidelberg, February 2016.

KLZ21. Aggelos Kiayias, Nikos Leonardos, and Dionysis Zindros. Mining in logarithmic space. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 3487–3501. ACM Press, November 2021.

KMZ20. Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 505–522. Springer, Heidelberg, February 2020.

KPZ20. Aggelos Kiayias, Andrianna Polydouri, and Dionysis Zindros. The velvet path to superlight blockchain clients. Cryptology ePrint Archive, Report 2020/1122, 2020. https://eprint.iacr.org/2020/1122.

KRDO17. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.

KZ19. Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *FC 2019 Workshops*, volume 11599 of *LNCS*, pages 21–34. Springer, Heidelberg, February 2019.

MMV13. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 373–388. ACM, January 2013.

Nak08. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008. [Online; accessed 19-June-2018].

Pie19. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.

PS17. Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 380–409. Springer, Heidelberg, December 2017.

PSs17. Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, April / May 2017.

VGS$^+$21. Psi Vesely, Kobi Gurkan, Michael Straka, Ariel Gabizon, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, and Eran Tromer. Plumo: An ultralight blockchain client. Cryptology ePrint Archive, Report 2021/1361, 2021. https://eprint.iacr.org/2021/1361.

Wes19. Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

A Vector Commitments

Definition 15 (Vector commitments). *A vector commitment scheme* Com *for universe* $\mathcal{U}$ *and message space* $\mathcal{M} = (\mathcal{M}_n)_{n\in\mathbb{N}}$ *with* $\mathcal{M}_n \subseteq \mathcal{U}^n$ *is a tuple of PPT algorithms* $(\mathsf{setup}, \mathsf{commit}, \mathsf{open}, \mathsf{ver})$ *where*

- $cp \leftarrow \mathsf{setup}(1^\lambda)$*: On input a security parameter* 1^λ*,* setup *outputs public parameters* cp*.*
- $(\phi, \mathrm{aux}) \leftarrow \mathsf{commit}(cp, (m_1, \ldots, m_n))$*: On input a tuple* $(m_1, \ldots, m_n) \in \mathcal{M}_n$*,* commit *outputs a commitment* ϕ *and an auxiliary information* aux*.*
- $\rho \leftarrow \mathsf{open}(cp, \phi, \mathrm{aux}, m, i)$*: On input a commitment* ϕ*, an auxiliary input* aux*, and a message component* m *at position* i*,* open *outputs an opening* ρ*.*
- $\mathsf{ver}(cp, \phi, m, i, \rho) \in \{0, 1\}$*: On input a commitment* ϕ*, a message* m *at position* i*, and an opening* ρ*,* ver *either accepts or rejects (by outputting* 1 *or* 0 *respectively).*

We require Com *to satisfy correctness and position-binding security:*

- Com *is correct if for every PPT adversary* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{l} cp \leftarrow \mathsf{setup}(1^\lambda) \\ (m_1, \ldots, m_n) \leftarrow \mathcal{A}(cp) \\ (\phi, \mathrm{aux}) \leftarrow \mathsf{commit}(cp, (m_1, \ldots, m_n)) \\ \rho \leftarrow \mathsf{open}(cp, \phi, \mathrm{aux}, m_i, i) \end{array} : \begin{array}{r} (m_1, \ldots, m_n) \in \mathcal{M}_n \wedge \\ \mathsf{ver}(cp, \phi, m_i, i, \rho) = 0 \end{array}\right] = 0 \ .$$

- Com *is position-binding if for every PPT adversary* $\mathcal{A}$ *the following is negligible in* λ*:*

$$\Pr\left[\begin{array}{l} cp \leftarrow \mathsf{setup}(1^\lambda) \\ (\phi, m, m', i, \rho, \rho') \leftarrow \mathcal{A}(cp) \end{array} : \begin{array}{r} \mathsf{ver}(cp, \phi, m, i, \rho) = 1 \\ \wedge\ \mathsf{ver}(cp, \phi, m', i, \rho') = 1 \\ \wedge\ m \neq m' \end{array}\right] \ .$$

For notational convenience, we generalize opening and verification to tuples of values at tuples of indices, that is we let $\rho \leftarrow \mathsf{open}(cp, \phi, \mathrm{aux}, S, \mathcal{I})$ *denote the opening of tuple* S *of values at a corresponding tuple of indices* $\mathcal{I}$*. The verification algorithm is generalized analogously.*

B Random Oracles, Consistent Queries, and τ-Sequences

The following lemma summarizes some properties of random oracles which will be useful in our setting.

Lemma 2. *Let* $\tau = (\tau_i)_{i\in[n]_0}$ *be a tuple of random oracles, with each* $\tau_i : \{0, 1\}^* \to \{0, 1\}^\lambda$*. Let* $\tilde{\mathsf{P}}^\tau$ *be an adversary with oracle access to* $\tau = (\tau_i)_{i\in[n]_0}$ *which makes at most* q *queries to* τ*. Then the following properties hold:*

1. **Collision:** *The probability of finding a collision can be bounded by*

$$\Pr\left[(p_i, p_{i'}) \leftarrow \tilde{\mathsf{P}}^\tau :\ \tau_i(p_i) = \tau_{i'}(p_{i'})\right] \leq \frac{q^2}{2^{\lambda+1}} \ .$$

2. **Lucky guess:** *Let* G_n *be a DAG. The probability that* $\tilde{\mathsf{P}}^\tau$ *finds a pair of strings* $y_i, y_{i'} \in \{0, 1\}^*$ *associated with indices* $i, i' \in [n]_0$ *such that* $y_i := p_i \| x_i$ *is consistent with* $y_{i'}$ *(cf. Def. 10) but* p_i *was* never *queried to* τ_i *can be bounded by*

$$\Pr\left[(y_i := p_i \| x_i, y_{i'}) \leftarrow \tilde{\mathsf{P}}^\tau;\ p_i \textit{ not queried to } \tau_i :\ y_i \prec y_{i'}\right] \leq \frac{1}{2^\lambda} \ .$$

3. **Consistent queries:** *Let G_n be a DAG. The probability that $\tilde{\mathsf{P}}^\tau$ finds a pair of strings $y_i, y_{i'} \in \{0,1\}^*$ associated with indices $i, i' \in [n]_0$ such that $y_i := p_i \| x_i$ is consistent with $y_{i'} := p_{i'} \| x_{i'}$ but p_i was not queried* before *$p_{i'}$ was queried can be bounded by*
$$\Pr\Big[(y_i := p_i \| x_i, y_{i'} := p_{i'} \| x_{i'}) \leftarrow \tilde{\mathsf{P}}^\tau;\ p_i \textit{ not queried before } p_{i'} :\ y_i \prec y_{i'}\Big] \leq \frac{q^2}{2^\lambda}\ .$$

Proof. For property 1, note that for any two queries the probability that their outputs collide is $\frac{1}{2^\lambda}$ since the output of the random oracles τ_i is uniform. The claim now follows by union bound over all $\binom{q}{2} = \frac{q(q-1)}{2}$ unordered pairs of queries.
For the second property, note that since p_i is not queried the output of the random oracle τ_i is uniformly random. Hence, the probability that it coincides with a specific block of size λ in $y_{i'}$ is $\frac{1}{2^\lambda}$, which proves property 2.
Let $p_i, p_{i'}$ be queried in rounds r, r' with $r' \leq r$, i.e., $p_{i'}$ is queried before the output of p_i is known. Then, similar to above, the probability that y_i is consistent with $y_{i'}$ is at most $\frac{1}{2^\lambda}$. Since there are at most q^2 possible ordered pairs of queries, property 3 follows by union bound. □

C Omitted Proofs

C.1 Proof of Theorem 1

To prove Theorem 1, we first define τ-sequences, and then prove their sequentiality and relate them to knowledge soundness.

Definition 16 (Augmented τ-sequence). *Let $G_n = ([n]_0, E_n)$ be a DAG and $\tau = (\tau_i)_{i \in [n]_0}$ a tuple of random oracles, with $\tau_i : \{0,1\}^* \to \{0,1\}^\lambda$. We call a sequence of strings $s = (y_{i_1}, \ldots, y_{i_{l+1}})$ with $y_{i_j} \in \{0,1\}^*$ for $j \in [l]$ and $y_{i_{l+1}} \in \{0,1\}^\lambda$ such that y_{i_j} is associated with index $i_j \in [n+1]$ and $y_{i_j} \prec y_{i_{j+1}}$ w.r.t. G_n for all $j \in [l]$ (cf. Def. 10) a* τ-sequence *of length l. For a weight function $\Omega_n : [n]_0 \to [0,1]$, the weight of a τ-sequence $s = (y_{i_1}, \ldots, y_{i_{l+1}})$ is defined as $\Omega_n(s) := \sum_{j=1}^{l} \Omega(i_j)$*

The notion of a τ-sequence is closely related to the language $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ when the commitment scheme and relation R are chosen appropriately.

Lemma 3 (Knowledge-soundness witnesses are τ-sequences). *Let $\Gamma_n = (G_n = ([n]_0, E_n), \Omega_n)$ be a weighted DAG, and let $\tau = (\tau_i)_{i \in [n]_0}$ be a family of random oracles. Let the relation R be defined as*
$$R(v, L_P(v), p_v) = 1 \quad \textit{iff} \quad L_P(v) = \tau_v(p_v) \| x_v \tag{13}$$
for some $x_v \in \{0,1\}^*$. Then any α-knowledge-soundness witness *for some statement $\eta := (\phi, n)$ w.r.t. the language $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ from Def. 6 implies a* τ-sequence *$s = (y_{i_1}, \ldots, y_{i_{l+1}})$ with final block $y_{i_{l+1}} = \tau_n(y_{i_l})$, last index $i_{l+1} = n+1$, penultimate index $i_l = n$, and weight $\geq \alpha$.*

If the commitment scheme Com *is defined as* SPC *(Constr. 1), then also the converse is true, i.e., any* τ-sequence *$s = (y_{i_1}, \ldots, y_{i_{l+1}})$ with final block $y_{i_{l+1}} = \phi \| x_{i_{l+1}}$ for some $x_{i_{l+1}} \in \{0,1\}^*$, last index $i_{l+1} = n+1$, penultimate index $i_l = n$, and weight $\geq \alpha$ directly implies an α-knowledge-soundness* witness *for $\eta := (\phi, n)$ w.r.t. the language $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ from Def. 6.*

Proof. Recall that an α-knowledge-soundness witness for $\eta := (\phi, n)$ consists of a path $P \subseteq G_n$, a labeling L_P on the vertices of P, values $(p_v)_{v \in P}$, and an opening ρ such that 1.) $\Omega_n(P) \geq \alpha$, and 2.) for all $v \in P$ with $(v_1, \ldots, v_k) :=$ parents(v) it holds $R(v, L_P(v), p_v) = 1$ and for all

$i \in [k]$ such that $v_i \in P$ it holds $p_v[i] = L_P(v_i)$, and 3.) $\mathsf{Com.ver}(cp, \phi, L_P, P, \rho) = 1$. Given such a witness, we immediately obtain a τ-sequence as follows: Set $(i_1, \ldots, i_l, i_{l+1}) := P \| (n+1)$ and for $j \in [l]$, $L_P(i_j) = \tau_{i_j}(p_{i_j}) \| x_{i_j} \in \{0,1\}^\lambda \times \{0,1\}^*$ set the strings $y_{i_j} := p_{i_j} \| x_{i_j}$, and $y_{i_{l+1}} = \tau_n(y_{i_l})$. Then $y_{i_j} \prec y_{i_{j+1}}$ for all $j \in [l]$ since P is a path in G_n. Since P has weight α, also the sequence $s = (y_{i_1}, \ldots, y_{i_{l+1}})$ has weight α.

Now assume the commitment scheme Com is defined as in Constr. 1. Given a τ-sequence, we extract a tuple $(P, L_P, (p'_v)_{v \in P}, \rho)$ as follows: First, set $P := (i_1, \ldots, i_l)$. Since $y_{i_j} \prec y_{i_{j+1}}$ for all $j \in [l]$, in particular $(i_j, i_{j+1}) \in E_n$ for all $j \in [l-1]$, hence $P := (i_1, \ldots, i_l)$ is a path in G_n of weight $\Omega_n(P) = \sum_{j=1}^{l} \Omega(i_j) = \alpha$. Furthermore, for $y_{i_j} = p_{i_j} \| x_{i_j}$ define labels by $L_P(i_j) := \tau_{i_j}(p_{i_j}) \| x_{i_j}$ for $j \in [l]$ and $p'_{i_j} := p_{i_j}$. Clearly, this assignment satisfies the condition on relation R from (13). Finally, set $\rho := (y_{i_1}, \ldots, y_{i_l})$. Since $i_l = n$, $i_{l+1} = n+1$, and $y_{i_{l+1}} = \phi \| x_{i_{l+1}}$ by assumption on the τ-sequence, for every $v \in P$ it holds that $\mathsf{Com.ver}(cp, \phi, L_P(v), v, \rho) = 1$. Hence $(P, L_P, (p'_v)_{v \in P}, \rho)$ is a valid α-knowledge-soundness witness for $\eta := (\phi, n)$. □

The following lemma states that τ-sequences must be computed sequentially. Without loss of generality we assume that any adversary A^τ whose aim is to output a τ-sequence does not make any redundant queries, i.e., any query to τ occurs only once.

Lemma 4 (Sequentiality of τ). *Let $\Gamma_n = (G_n, \Omega_n)$ be a weighted DAG and $\tau = (\tau_i)_{i \in [n]_0}$ be a family of random oracles with $\tau_i : \{0,1\}^* \to \{0,1\}^\lambda$. Let A^τ be an adversary that makes (parallel) queries to τ of sequential weight $< \alpha$ and makes q oracle queries in total. Then the probability that A^τ outputs a τ-sequence of weight α can be bounded by*

$$\Pr\left[s \leftarrow \mathsf{A}^\tau : \ s \textit{ is a } \tau\textit{-sequence} \wedge \Omega_n(s) = \alpha\right] \le \frac{1}{2^\lambda} + \frac{q^2}{2^\lambda} = \frac{q^2+1}{2^\lambda}.$$

Proof. Since the sequential weight of A's queries is less than α, whenever $\Omega_n(s) = \alpha$ the sequence s must either contain a string $y_{i_j} := p_{i_j} \| x_{i_j}$ such that p_{i_j} was not queried, or a pair of nodes $y_{i_j} := p_{i_j} \| x_{i_j}, y_{i_{j'}} := p_{i_{j'}} \| x_{i_{j'}}$ such that $j < j'$ and $p_{i_j}, p_{i_{j'}}$ were queried in rounds r, r' with $r' \le r$. Thus, the claim follows from basic properties of random oracles; for completeness we formalize them as Properties 2 and 3 in Lemma 2 in Appendix B. □

This allows us to prove that knowledge-soundness implies soundness of any graph-labeling PoSW.

Proof (of Theorem 1). Consider an (α, ϵ)-knowledge-sound graph-labeling PoSW. Let $\tilde{\mathsf{P}}$ be an arbitrary PPT prover that makes q queries of sequential weight $< \alpha$. Assume toward contradiction that when $\tilde{\mathsf{P}}$ interacts with V, the latter returns 1 with probability $> \epsilon'$. By (α, ϵ)-knowledge soundness, there exists an extractor E, which from $\tilde{\mathsf{P}}$ extracts a witness for $\eta := (\phi, n)$ of weight at least α with probability at least $\epsilon' - \epsilon$ (where ϕ is the output of $\tilde{\mathsf{P}}_0$). By Lemma 3, any witness for $\eta := (\phi, n)$ implies a τ-sequence of weight α with final block $y_{n+1} = \phi$. From the PPT algorithm E we can thus obtain such a sequence with probability $> \epsilon' - \epsilon$. By Lemma 4, this can happen with probability at most $(q^2+1)/2^\lambda$, as E only makes $\tilde{\mathsf{P}}$'s queries to τ, and thus their sequential weight is $< \alpha$. Since $(q^2+1)/2^\lambda = \epsilon' - \epsilon$, this is a contradiction. □

C.2 Proof of Theorem 2

We first provide a high-level intuition for the proof of Theorem 2. First, note that by the binding property of the commitment scheme, the set of challenges the prover can consistently answer is essentially fixed after it output the commitment ϕ_L. Now recall that any accepting answer to a challenge ι_i consists of a labeling of the shortest path from 0 to n which goes through node ι_i and all ancestors of this path. In the proof we show that by collision resistance of the hash function and the structure of the underlying graph all these labelings of accepting paths can be combined to derive a labeling of a long path (and its ancestors) which contains all the challenges the prover can consistently answer to. Assuming the prover can answer a single challenge with probability α, this allows us to extract a consistent labeling of a path (and its ancestors) of weight α. The claim then follows from a lemma where we prove that computing such a labeling requires queries to the random oracle of sequential weight at least α.

More formally, towards proving Theorem 2 we first establish the following lemma, which states that any τ-sequence with final block $y_{n+1} = \phi_L$ must have been computed *before* the commitment ϕ_L.

Lemma 5. *Let $\Gamma_n = (G_n, \Omega_n)$ be a weighted DAG. Let $\tilde{\mathsf{P}}_0^\tau, \tilde{\mathsf{P}}_1^\tau$ be a pair of algorithms with oracle access to $\tau = (\tau_i)_{i \in [n]_0}$, such that*

- *$\tilde{\mathsf{P}}_0$, on input 1^λ,makes q_0 queries to τ and outputs state_0 and a string $\phi \in \{0,1\}^\lambda$.*
- *$\tilde{\mathsf{P}}_1$, on input (state_0, ϕ), makes q_1 queries to τ and outputs a τ-sequence $s = (y_{i_1}, \dots, y_{i_{l+1}})$ with $y_{i_{l+1}} = (\phi, \varepsilon)$.*

Let Q be the set of queries (including responses) made by $\tilde{\mathsf{P}}_0$ and S the set of τ-sequences with final block ϕ which are extractable from Q. Then

$$\Pr\left[\begin{matrix}(\mathsf{state}_0, \phi) \leftarrow \tilde{\mathsf{P}}_0^\tau(1^\lambda); \\ \tau\text{-sequence } s = (y_{i_1}, \dots, y_{i_{l+1}} = \phi) \leftarrow \tilde{\mathsf{P}}_1^\tau(\mathsf{state}_0, \phi)\end{matrix} \ : \ s \notin S\right] \le \frac{q^2}{2^\lambda} ,$$

where $q = q_0 + q_1$.

Proof. Assume $\tilde{\mathsf{P}}_1^\tau$ outputs a sequence $s \notin S$. Since $s \notin S$, there must exist $y_{i_j} := p_{i_j} \| x_{i_j}$, for $1 \le j \le l$ in s such that $(p_{i_j}, \tau_{i_j}(p_{i_j})) \notin Q$, i.e., p_{i_j} was not queried by $\tilde{\mathsf{P}}_0$. Let y_{i_j} be the last such string in the sequence s. Then one of the following must be true: 1) p_{i_j} was never queried at all and $\tilde{\mathsf{P}}_1$ made a lucky guess, 2) $\tilde{\mathsf{P}}_1$ found a pair of consistent queries $(y_{i_j} := p_{i_j} \| x_{i_j}, y_{i_{j+1}} := p_{i_{j+1}} \| x_{i_{j+1}})$ such that p_{i_j} was queried not before $p_{i_{j+1}}$ and $y_{i_j} \prec y_{i_{j+1}}$, 3) $j = l$, $\tau(p_{i_j}) = \phi$ and p_{i_j} was only queried by $\tilde{\mathsf{P}}_1$.
Using Property 2 in Lemma 2, we can bound the probability that case 1 happens by $\frac{1}{2^\lambda}$. For case 2, similar to the proof of Property 3 in Lemma 2 we get a bound $\frac{q_0 \cdot q_1}{2^\lambda}$, where we used the fact that $p_{i_{j+1}}$ was queried by $\tilde{\mathsf{P}}_0$ and p_{i_j} was queried by $\tilde{\mathsf{P}}_1$. Finally, since $\phi \in \{0,1\}^\lambda$ is given as fixed input to $\tilde{\mathsf{P}}_1$ and p_{i_j} was queried by $\tilde{\mathsf{P}}_1$, the probability that case 3 happens is at most $\frac{q_1}{2^\lambda}$. The claim now follows by union bound since $1 + q_0 \cdot q_1 + q_1 \le q^2$ for $q = q_0 + q_1 \ge 2$. □

Proof (of Theorem 2). Completeness and succinctness of the protocol are easy to verify. Let $n \in \mathbb{N}$ be arbitrary. To prove soundness, let $\tilde{\mathsf{P}}$ be an adversary with oracle access to τ which makes queries of sequential weight at most $\alpha \in [0,1]$ before sending ϕ_L to V and makes q queries in total. First, note that for any challenge $\iota_i \in [n]_0$ its opening $\gamma_i = (o_i, \rho_i)$ gives a τ-sequence of length at most $l \le 2\log(n)$: Let $\overline{\mathrm{path}}(\iota_i) := \mathrm{path}'(\iota_i) \cup \mathrm{path}(\iota_i) = (i_0 = 0, \dots, i_l = n)$ be the shortest path in G_n from 0 to n that passes through ι_i. Then γ_i contains labels of all parents in G_n of nodes i_j, $j \in [l]$. Setting $y_{i_j} := (y_{i_j,0}, y_{i_j,1})$ with $y_{i_j,0} := L(\mathrm{parents}(i_j))$ and $y_{i_j,1} = x_{i_j}$ (i.e. the augmentation) for

$j \in [l]$, and $y_{i_{l+1}} := \phi_L$ with $i_{l+1} = n+1$ gives a τ-sequence of length l. Thus, by Lemma 5, the set of challenges which $\tilde{\mathsf{P}}$ can consistently answer is essentially fixed after it output ϕ_L. Furthermore, rewinding $\tilde{\mathsf{P}}$ to the point when it output ϕ_L allows to recover answers to all queries and we will show below that – unless $\tilde{\mathsf{P}}$ found a collision for τ (the probability of this event is bounded in Property 1 in Lemma 2) – these answers can be combined to obtain a τ-sequence that contains all challenge nodes which $\tilde{\mathsf{P}}$ can consistently open. By Lemmata 4 and 5, and the upper bound α on the sequential weight of $\tilde{\mathsf{P}}$'s queries, it then follows that with all-but-negligible probability this sequence is of weight at most α, i.e., the set of challenges $\tilde{\mathsf{P}}$ can open has weight at most α. Hence, the probability that $\tilde{\mathsf{P}}$ can answer a single challenge drawn according to distribution Ω_n is at most α, and since the t challenges are drawn independently, the success probability of $\tilde{\mathsf{P}}$ can be bounded by α^t. This proves (α, ϵ)-soundness of Constr. 2, where the soundness error ϵ is obtained by a union bound over the two bad events considered in Lemmata 4 and 5, as well as Property 1 in Lemma 2:

$$\epsilon \leq \alpha^t + \frac{q^2+1}{2^\lambda} + \frac{q^2}{2^\lambda} + \frac{q^2}{2^{\lambda+1}} \leq \alpha^t + \frac{3 \cdot q^2}{2^\lambda} \ .$$

It remains to prove that the τ-sequences extracted from verifying answers can be glued together to obtain a τ-sequence containing the openings of all challenge nodes. Recall that for each challenge ι_i the prover has to output labels $L(j)$ for every non-source node j on the path $\mathrm{path}(\iota_i)$ along with the corresponding openings $y_{j,0} := L(\mathrm{parents}(j))$. Let $\mathcal{I} \subset [n]_0$ be the set of challenges that $\tilde{\mathsf{P}}$ can consistently open. Then by the structure of the graph G_n, the subgraph induced on the set of nodes in the set of all verifying paths $\bigcup_{\iota_i \in \mathcal{I}} \overline{\mathrm{path}}(\iota_i)$ contains a path from 0 to n which includes all the properly answered challenges $\iota_i \in \mathcal{I}$; we denote this path by $\mathcal{P}$. Thus, if the openings which $\tilde{\mathsf{P}}$ outputs in response to the challenges it can consistently answer were unique, then from $\tilde{\mathsf{P}}$'s responses one could extract a τ-sequence whose weight is at least $\Omega(\mathcal{I})$. Furthermore, the verification algorithm guarantees that $L(0) = (\tau_0(\varepsilon), x_0)$ and $L(n) = (\phi_L, x_n)$ are fixed for all accepted answers γ_{ι_i}; hence, the extracted sequence must contain $y_0 = (\varepsilon, x_0)$ and $y_{n+1} = \phi_L$. In the following we will argue that – unless $\tilde{\mathsf{P}}$ found a collision for τ – the openings $y_{j,0}$ for the nodes $j \in \bigcup_{\iota_i \in \mathcal{I}} \overline{\mathrm{path}}(\iota_i)$ are indeed unique.

Uniqueness of the openings follows from the position-binding property of the commitment scheme $\mathsf{Com} = \mathsf{SPC}$, where the negligible soundness gap is bounded by Property 1 in Lemma 2: We argue uniqueness in reverse topological order along the path $\mathcal{P} = (j_0, \ldots, j_l)$ with $j_0 = 0$ and $j_l = n$: Since $L(n) = (\phi_L, x_n)$ is fixed and verification guarantees that $\tau_n(y_{n,0}) = \phi_L$ for any accepting opening $y_{n,0}$ of n, the opening $y_{n,0}$ must be unique, unless $\tilde{\mathsf{P}}$ found a collision for τ. Now all labels of nodes which are connected to node n by an edge in G_n are essentially fixed to coincide with the corresponding block in $y_{n,0}$; in particular $L(j_{l-1})$ is fixed. Unless $\tilde{\mathsf{P}}$ found a collision, this implies that the opening $y_{j_{l-1},0}$ of j_{l-1} is fixed. Proceeding analogously proves that either $\tilde{\mathsf{P}}$ found a collision or all openings of nodes along the path $\mathcal{P}$ are unique. This proves that Constr. 2 is an (α, ϵ)-sound graph-labeling PoSW. We will now argue that it is also *knowledge* sound: The above argument in particular gives an extractor $\mathsf{E}^{\tilde{\mathsf{P}}}$ which with all but negligible probability $3q^2/2^\lambda$ extracts a τ-sequence of weight α from any prover $\tilde{\mathsf{P}}$ which succeeds on a single challenge with probability α. Recall that by Lemma 3 a τ-sequence of weight α with final block $y_{n+1} = \phi_L$ and penultimate block associated with index $i_l = n$ immediately gives an α-knowledge-soundness witness for $\eta := (\phi, n)$ with respect to the language $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ with relation R from Equation (13). To prove (α, ϵ)-knowledge soundness, let $\tilde{\mathsf{P}}$ be any PPT prover and let α' be the probability that $\tilde{\mathsf{P}}$ succeeds on a single challenge. We consider two cases: First, assume $\alpha' \leq \alpha$. Then the probability

```
Verifier $\overline{\mathsf{PoSW}}.\mathsf{V} = (\mathsf{V}_0, \mathsf{V}_1, \mathsf{V}_2)$:

Stage $\mathsf{V}_0$: On input $(1^\lambda, n, \psi)$:
  1. $(\sigma, \mathsf{aux}_0) \leftarrow \mathsf{Init}(1^\lambda, \psi)$
  2. send $\sigma$ to $\mathsf{P}_0$

Stage $\mathsf{V}_1$: On input $\phi_n$:
  1. $\iota \leftarrow \mathsf{ACK.V}_1(cp, \phi_n, n)$
  2. send $\iota$ to $\mathsf{P}_1$

Stage $\mathsf{V}_2$: On input $\gamma$:
     output $\mathsf{ACK.V}_2(\gamma)$
```

```
Prover $\overline{\mathsf{PoSW}}.\mathsf{P} = (\mathsf{P}_0, \mathsf{P}_1)$:

Stage $\mathsf{P}_0$: On input $\big(1^n, (d_i)_{i \in [n]}\big)$ and $\sigma$:
  1. $L_K(0) := \sigma$
  2. $\forall i \in [n]$ do
       $(L_K(i), \mathsf{aux}_i) \leftarrow \mathsf{Mine}\big((L_K(0), \ldots, L_K(i-1)), d_i\big)$
  3. parse $L_K(n)$ as $(\ell_n \| \phi_n, h_n)$
  4. send $\phi_n$ to $\mathsf{V}_1$

Stage $\mathsf{P}_1$: On input $\iota$:
  1. $\gamma \leftarrow \mathsf{ACK.P}\big((cp, \phi_n, n), (L_K(j))_{j \in [n]_0}, \mathsf{aux}_n, \iota\big)$
  2. send $\gamma := (\gamma_i)_{i=1}^t$ to $\mathsf{V}_2$
```

Fig. 10: The augmented PoSW scheme constructed from mining algorithms and an ACK system $\mathsf{ACK.V} = (\mathsf{ACK.V}_1, \mathsf{ACK.V}_2), \mathsf{ACK.P}$.

that $\tilde{\mathsf{P}}$ succeeds is at most $(\alpha')^t + 3q^2/2^\lambda \leq \alpha^t + 3q^2/2^\lambda = \epsilon$, which implies (α, ϵ)-knowledge soundness for this case. For the case $\alpha' > \alpha$, we make use of the extractor $\mathsf{E}^{\tilde{\mathsf{P}}}$ that with all but negligible probability $3q^2/2^\lambda$ extracts an α'-knowledge soundness witness for $\eta := (\phi, n)$. Since for $\alpha < \alpha'$ an α'-knowledge soundness witness for $\eta := (\phi, n)$ in particular is an α-knowledge soundness witness for $\eta := (\phi, n)$, this extractor fails only with probability $3q^2/2^\lambda$ to output an α-knowledge soundness witness for $\eta := (\phi, n)$, which implies (α, ϵ)-knowledge soundness also for this case. □

C.3 Proof of Theorem 3

Proof. The proof strategy is simple: we use $(\mathsf{ACK.P}, \mathsf{ACK.V})$ and Alg. Mine from Fig. 4 to build an augmented PoSW $(\overline{\mathsf{PoSW}}.\mathsf{P}, \overline{\mathsf{PoSW}}.\mathsf{V})$ (see Fig. 10) whose (α, ϵ)-knowledge-soundness implies that of the SNACK. The non-interactive $(\mathsf{SNACK.P}, \mathsf{SNACK.V})$ is simply the Fiat-Shamir transformation [FS87] applied to $(\mathsf{ACK.P}, \mathsf{ACK.V})$.

Concretely, we define an X-augmented PoSW $(\overline{\mathsf{PoSW}}.\mathsf{P}, \overline{\mathsf{PoSW}}.\mathsf{V})$ for the weighted graph family $(K_n, \Omega_n)_{n \in \mathbb{N}}$ as in Fig. 10 where K_n is defined in (7). This PoSW follows the template of Fig. 2 with Com being its underlying commitment scheme and L_K (as defined by Init and Mine) being its X-augmented τ-based labeling (see Def. 4). This labeling L_K differs however from the labeling L_G of $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ in that the parents$(\cdot)$ function used in the τ-labeling is now w.r.t. graph K_n rather than $G_n = ([n]_0, E_G)$. Graph $K_n = ([n]_0, E_K)$ has the same vertex set $[n]_0$ of G_n but only differs in that its edge set is expanded as $E_K := E_G \cup E_H$ where E_H is the edge set of H_n.

However when τ is modeled as a random oracle, it is easy to see that if $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ is a τ-based (α, ϵ)-knowledge-sound PoSW for a weighted graph family $(G_n, \Omega_n)_{n \in \mathbb{N}}$, then $(\overline{\mathsf{PoSW}}.\mathsf{P}, \overline{\mathsf{PoSW}}.\mathsf{V})$ is an X-augmented τ-based (α, ϵ)-knowledge-sound PoSW for a weighted graph family $(K_n, \Omega_n)_{n \in \mathbb{N}}$ and $X = (x_0, \ldots, x_n)$, where $x_i := g_i \| h_i$ and g_i, h_i are as in Mine (and Init). To argue this we need to argue that appending x_i to the random oracle τ does not affect soundness of the PoSW. This however follows because (1) τ is a random oracle and the extra x_i's simply define x_i-salted random oracles, and crucially (2) the new edge structure E_H does not give a malicious prover $\overline{\mathsf{PoSW}}.\tilde{\mathsf{P}}$ any more power than its counterpart $\mathsf{PoSW}.\tilde{\mathsf{P}}$: this is ensured by $\mathsf{ACK.V}$ by making sure that for each challenge $\iota_i \in [n]_0$, the responses of any prover are verified w.r.t.

- the edge structure imposed by E_G, which suffices for preserving the underlying soundness of $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$, and
- the edge structure E_H of H_n, which in turn suffices to verify the validity of the augmented blockchain (using $\tilde{R}_\psi$).

Furthermore, by Def. 9, $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ is (α, ϵ)-knowledge sound with respect to witness relation $\mathcal{R}^{(\alpha)}_{\Gamma, R_\chi, \mathsf{Com}}$ as in Def. 6 and R_χ as in (5). However, due to the checks executed by $\overline{\mathsf{PoSW}}.\mathsf{V}$, the soundness of $(\overline{\mathsf{PoSW}}.\mathsf{P}, \overline{\mathsf{PoSW}}.\mathsf{V})$ is now defined w.r.t. witness relation $\mathcal{R}^{(\alpha)}_{\Gamma, R_\sigma, \mathsf{Com}}$ as in Def. 6 with R_σ defined in (9). By inspecting these witness relations, it is clear that $\mathcal{R}^{(\alpha)}_{\Gamma, R_\sigma, \mathsf{Com}}$ extends $\mathcal{R}^{(\alpha)}_{\Gamma, R_\chi, \mathsf{Com}}$ in the sense of requiring the extra check of $\tilde{R}_\psi$ imposed by R_σ. This extra check clearly does not affect soundness, but may affect completeness.[15] To make sure that completeness is also preserved, we need to make sure that an honest augmented prover $\overline{\mathsf{PoSW}}.\mathsf{P}$ gets extra information that allows it to pass the $\tilde{R}_\psi$ check, and in fact the labels $(k_i)_{i \in [n]_0}$ provided as input to $\overline{\mathsf{PoSW}}.\mathsf{P}$ by the output of (the honest) Mine ensure this.

This establishes that $(\overline{\mathsf{PoSW}}.\mathsf{P}, \overline{\mathsf{PoSW}}.\mathsf{V})$ is an (α, ϵ)-knowledge-sound X-augmented τ-based (with labeling L_K) PoSW for the weighted graph family $(K_n, \Omega_n)_{n \in \mathbb{N}}$. By absorbing the computation of $\overline{\mathsf{PoSW}}.\mathsf{V}_0$ into $\mathsf{ACK.V}_1$ and $\overline{\mathsf{PoSW}}.\mathsf{P}_0$ into $\mathsf{ACK.P}$, the pair $(\mathsf{ACK.P}, \mathsf{ACK.V})$ is syntactically a PoSW.[16] Now by Lemma 1 it holds that $(\mathsf{SNACK.P}, \mathsf{SNACK.V})$, the Fiat-Shamir transform of $(\mathsf{ACK.P}, \mathsf{ACK.V})$, is an (α, ϵ)-knowledge-sound SNACK for the language $\mathcal{L}_{\Gamma, R_\sigma, \mathsf{Com}}$ as in Def. 6 and R_σ as in (9). □

C.4 Proof of Corollary 1

Proof. Consider an adversary that makes the light client accept

$$\big(\phi^*, n^*, \pi^*, (k_i^*)_{i=m^*+1}^{n^*}, (\iota_j, k_{\iota_j}^*, \rho_{\iota_i}^*)_{j=1}^{q^*}\big)$$

with $m^* := n^* - \ell$, and the light client does *not* ℓ-CP bootstrap.

By (b), π^* is valid for (ϕ^*, m^*), so by (α_{m^*}, ϵ)-knowledge soundness we can extract, except with probability ϵ, an R-valid path $\mathcal{P}' = (P', (k_i')_{i \in P'}, (p_i')_{i \in P'})$ with $P' \subseteq [m^*]_0$ and $\Omega_{m^*}(P') \geq \alpha_{m^*}$ as well as a subset opening ρ' of ϕ^* to $(k_i')_{i \in P'}$. Except with probability ϵ_C, we have

$$\text{for all } i \in P' \cap \{\iota_1, \ldots, \iota_{q^*}\} : k_i' = k_i^*. \tag{14}$$

Otherwise, if $k_i' \neq k_i^*$ for some i, this breaks position-binding of Com, as ρ' opens ϕ^* to k_i' at position i, and ρ_i^* opens ϕ^* to k_i^* at position i by (d).

Thus $\mathcal{P} := (P, (k_i)_{i \in P}, (p_i)_{i \in P})$ is R-valid (Def. 5), where

$$\begin{aligned} P &:= P' \| [m^* + 1 : n^*] \qquad & (k_i)_{i \in P} &:= (k_i')_{i \in P'} \| (k_i^*)_{i \in [m^*+1:n^*]} \\ & & (p_i)_{i \in P} &:= (p_i')_{i \in P'} \| \big((k_j^*)_{j \in \mathrm{parents}(i)}\big)_{i \in [m^*+1:n^*]}. \end{aligned}$$

[15] If a protocol is sound when the verifier executes a check C_1, then it is clearly sound if the verifier executes an additional check C_2. However, the protocol is not guaranteed to remain complete – because the verifier would reject a proof π that verifies with respect to C_1 but fails with respect to C_2.

[16] To justify this syntactic manipulation, note that $\overline{\mathsf{PoSW}}.\mathsf{V}_0$ outputs σ which contains χ and cp which are generated identically to any graph-labeling PoSW scheme. Similarly we assume that the (honest) input to $\mathsf{ACK.P}$ was computed by an (honest) $\overline{\mathsf{PoSW}}.\mathsf{P}_0$, which is the computation of the honest mining Mine, which is in turn an honest PoSW computation.

Indeed, for all $i \in P$: $R(i, k'_i, p'_i) = 1$ by R-validity of $\mathcal{P}'$, and for $i \in [m^*{+}1 : n^*]$: $R(i, k^*_i, (k^*_j)_{j \in \mathrm{parents}(i)}) = 1$ by (c). Moreover, R-validity requires consistency of path labels appearing in some p_i, that is, for all i, letting $(i_1, \dots i_{\deg(i)}) := \mathrm{parents}(i)$ and all j: if $i_j \in P$ then $p_i[j] = k_{i_j}$. For $i \in P'$, this follows from R-validity of $\mathcal{P}'$; for $i \in [m^* + 1 : n^*]$ and $i_j \in P'$, this follows from (14); for $i, i_j \in [m^* + 1 : n^*] : p_i[j] = k^*_{i_j}$ by definition.

We have thus shown that $\mathcal{P}$ is an R-valid path with $[m^* + 1 : n^*] \subseteq P$ and $\Omega_{m^*}(P \cap [m^*]_0) = \Omega_{m^*}[P'] \geq \alpha_{m^*}$. If in addition we have

$$\mathcal{P} \text{ is an } \ell\text{-fork (Def. 11)}, \tag{15}$$

then Theorem 4 guarantees that the adversary can only create $\mathcal{P}$ with probability at most ϵ_F, which concludes the proof.

Let $(k^{(h)}_0, \dots, k^{(h)}_{s_h})$ be the stable prefix of the honest chain. We show (15) by considering two cases:

1. $n^* < s_h{+}\ell$ (and thus $m^*{+}1 < s_h{+}1$): Let $\phi^{(h)}$ be the commitment contained in $k^{(h)}_{m^*+1}$. We must have $\phi^* \neq \phi^{(h)}$, as otherwise the light client would have bootstrapped; thus $k^*_{m^*+1} \neq k^{(h)}_{m^*+1}$. We moreover have $n^* \geq n_h$; otherwise an assumed honest full node would have convinced the verifier. $\mathcal{P}$ is thus an ℓ-fork of Type (1) with $i_j = m^* + 1$ (note that $n^* \geq i_j + \ell - 1$).
2. $n^* \geq s_h + \ell$:
   (a) If for some $i \in P' \cap [s_h]_0 : k'_i \neq k^{(h)}_i$ then $\mathcal{P}'$ is again an ℓ-fork of Type (1).
   (b) If for all $i \in P' \cap [s_h]_0 : k'_i = k^{(h)}_i$ then $\mathcal{P}'$ is an ℓ-fork of Type (2). □

D A Graph-Labeling PoSW Based on CP Graphs

Our second construction is based on [CP18]. We define the PoSW construction by instantiating the unspecified parts in the blueprint given in Fig. 2, namely we define a weighted DAG family $(G_n, \Omega_n)_{n \in \mathbb{N}}$ and specify algorithms $\mathsf{PoSW.label}, \mathsf{PoSW.open}, \mathsf{PoSW.ver}, \mathsf{Com}$.

Construction 3. *We define G_n based on a slight modification of the CP DAG from [CP18]. We first define a DAG $G^{\text{CP'}}_m = (V^{\text{CP'}}_m, E^{\text{CP'}}_m)$ depicted in Fig. 11: For $m \in \mathbb{N}$, define $V^{\text{CP'}}_m = \{0,1\}^{\leq m}$ and $E^{\text{CP'}}_m = E' \cup E''$, where E' are the upward edges of the binary tree of depth m with root ε, the empty string, i.e.,*

$$E' = \left\{(x \| b, x) : b \in \{0,1\}, x \in \{0,1\}^i, 0 \leq i < m\right\} ,$$

and E'' contains for **all** *nodes v in the tree an edge (u, v) iff u is a left sibling of a node on the path from v to the root ε in the binary tree $(V^{\text{CP'}}_m, E')$, i.e.,*

$$E'' = \left\{(u, v) : v \in \{0,1\}^i, 0 < i \leq m, v = x\|1\|x', u = x\|0\right\} .$$

For completeness we note that the CP DAG [CP18] is defined identically to $G^{\text{CP'}}_m$ except that v in the definition of E'' is restricted to leaf *nodes.*

Now we define $G_n = ([n]_0, E_n)$ to be the subgraph of $G^{\text{CP}'}_m = (V^{\text{CP'}}_m, E^{\text{CP'}}_m)$ with $m = \lceil \log(n) \rceil$ induced on the first $n+1$ nodes in topological order, and $\Omega_n : [n]_0 \to [0,1]$ an arbitrary weight function. We define $\mathsf{PoSW.label}(\chi)$ *based on a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ by sampling oracles $\tau = (\tau_i(\cdot))_{i \in [n_0]}$ as $\tau_i(\cdot) := \mathcal{H}(\chi, i, \cdot)$. Finally,* $\mathsf{Com} = (\mathsf{setup}, \mathsf{commit}, \mathsf{open}, \mathsf{ver})$ *is defined by* SPC *(Construction 1). The subroutine* $\mathsf{PoSW.open}$ *and* $\mathsf{PoSW.ver}$ *do not contain any additional information/checks.*

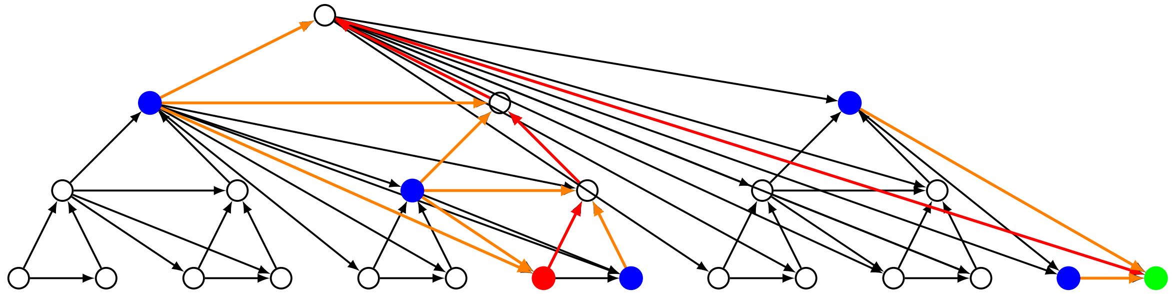

Fig. 11: Illustration of Constr. 3. The label of the last node (green) serves as the commitment. On input a challenge (red node), P opens all the siblings (blue) along the shortest path (red) from the challenge node to the final node. To verify, V evaulates the opening (red and orange edges).

The difference between Constr. 3 and the PoSW from [CP18] is that we introduce additional edges, due to allowing v in E'' to be any node and not merely a leaf node as in [CP18]. These additional edges allow for more general weight distributions: In the original CP construction, challenges are sampled uniformly at random from the leaves only. To prove knowledge soundness, [CP18] (similar to the proof for Constr. 2 above) relied on the fact that for any subset $\mathcal{I}$ of challenges there exists a path in G_n which contains $\mathcal{I}$ but only consists of vertices in $\bigcup_{\iota_i \in \mathcal{I}} \mathrm{path}(\iota_i)$. To satisfy this guarantee, in [CP18] the edge set E'' was defined to introduce edges incident on leaf nodes $v \in \{0,1\}^n$ only. For our applications, however, we will obtain slightly more efficient schemes if we allow for arbitrary weight functions, potentially assigning non-zero weight to inner nodes. Our extended definition of E'' gives a similar guarantee on the existence of a path connecting any subset of challenge nodes (as above), while not increasing the proof size.

Theorem 5. *Let $\alpha \in (0,1]$. The scheme from Constr. 3 with parameter t and arbitrary weight function Ω_n is an (α,ϵ)-knowledge sound graph-labeling PoSW with $\epsilon := \alpha^t + 3 \cdot q^2/2^\lambda$, where q is an upper bound on the number of the adversary's oracle queries.*

A proof for this theorem follows exactly the same lines as the proof of Theorem 2, taking into account the special properties of the underlying graph which were proven in [CP18] and can easily be extended to our modified construction. In the soundness proof in [CP18] (similar to our proof above) a τ-sequence is extracted which contains all challenge nodes which the prover can consistently open. While their notion of a τ-sequence is weaker than ours and does not respect the topology of the graph, adapting the proof to our stronger notion is straightforward and even allows for slightly better security loss ϵ. Also, generalizing their proof to handle arbitrary weight functions follows exactly the same lines as in the proof of Theorem 2. Here, as mentioned above, the additional edges of our graph compared to the original construction from [CP18] are necessary to allow for challenges among all nodes of the graph, not just leaf nodes.

E Instantiations and Optimizations

In this section we discuss the concrete instantiations of our SNACK system and possible optimizations.

To be able to highlight the main differences of the resulting SNACK system when instantiated with different PoSW schemes, we consider a blockchain whose underlying chain graph

$H_n = (V_H, E_H)$ is simple, i.e., $V_H = [n]_0$ and $E_H = \{(i-1, i) : i \in [n]\}$. The Bitcoin blockchain (in the fixed-difficulty setting) is an example of such a case. The two candidate α-knowledge-sound graph-labeling PoSW schemes which we use to instantiate our SNACK system are:

- the hashed skiplist scheme HSL from Constr. 2;
- the augmented scheme of [CP18] aCP from Constr. 3.

Both HSL and aCP have similar security guarantees, i.e., they are both (α, ϵ)-knowledge-sound for the same $\epsilon = \alpha^t + (3 \cdot q^2)/2^\lambda$ – see Theorems 2 and 5. For a negligible soundness error, the value α^t needs to be negligible in λ; for example for any constant $\alpha \in [0, 1)$ we set the number of challenges $t = \omega(\lambda)$.

Hence for the same security guarantees, the difference in the size of proofs in HSL and aCP is due to the difference in size of the opening of a single challenge. For a single challenge, the size of the opening γ_i is approximately:

- $|\gamma_i| \approx |k_i| + |k_i| \cdot \log(n)^2$ for HSL and
- $|\gamma_i| \approx |k_i| + |k_i| \cdot 3 \cdot \log(n)$ for aCP,

where the $\log(n)^2$ factor in HSL comes from the fact that a path that passes through a challenge vertex from the source to the sink is of length at most $2 \cdot \lceil \log(n) \rceil$ and the vertices on the path have up to $\log(n)$ parents. As for aCP an opening for a challenge is a Merkle tree opening plus the labels on the path from the challenge to the sink (necessary when using aCP in the SNACK construction in Sect. 5) and the $< \log(n)$ parents of the challenge vertex, and hence of length at most $3 \cdot \lceil \log(n) \rceil$.

It is hence clear that using aCP is preferable in terms of proof size. The advantage of using HSL, on the other hand, is its conceptual simplicity of the resulting construction.

Optimizing the proof size. The label $k_i := (\ell_i \| \phi_i, h_i)$, as computed by Mine in Fig. 4, has size that is dominated by h_i, which in turn is dominated by the data d_i of the block in the underlying blockchain. As the size of the challenge opening $|\gamma_i|$ of a challenge i crucially depends on $|k_i|$, it is natural to ask whether in some applications we can compress h_i.

Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function, we define the function $c : [n]_0 \times \{0,1\}^* \to \{0,1\}^*$ as:

$$c(j, k_i) = \begin{cases} (\ell_i \| \phi_i, \mathcal{H}(h_i)) & \text{if } i \notin \text{parents}_H(j) \\ k_i & \text{otherwise} \end{cases} \tag{16}$$

Now Mine is modified in the natural way to take this compression into account: let $\text{parents}_K(i) = (i_1, \ldots, i_m)$ be given in reverse topological order, then Mine now computes ℓ_i as

$$\ell_i = \tau_i(c(i, k_{i_1}), \ldots, c(i, k_{i_m})) \ . \tag{17}$$

This optimization affects the size of the PoSW opening, as now instead of giving full labels one gives compressed variants thereof. For a simple chain graph like the one underlying Bitcoin, this simple optimization brings the proof size of the SNACK to be almost as small as the proof size of the underlying PoSW:

- $|\gamma_i| \approx |k_i| + 2\lambda \log(n)^2$ for HSL and
- $|\gamma_i| \approx |k_i| + 6\lambda \log(n)$ for aCP.

For blockchains whose underlying chain graphs have a more complex structure, the proof size would be dominated by the h_i labels. However, even then one could also attempt applying the compression function $c(\cdot)$ above whenever the verification of the blochchain validity relation R_ψ is not affected.

F Efficiency Comparison to FlyClient

FlyClient. The number of challenges required by the FlyClient [BKLZ20] protocol is

$$t' = \lambda / \log_{1/2}\left(1 - (\log_c(\ell/n))^{-1}\right) + \ell \tag{18}$$

and is derived as follows: The authors consider an idealized "continuous" blockchain represented by the real interval $[0,1]$. The last ℓ blocks, corresponding to the interval $[1-\delta, 1]$ with $\delta := \ell/n$ are always checked, and the challenging of blocks in $[0, 1-\delta]$ is done according to the PDF $g(x) := 1/((x-1)\ln(\delta)$.

If the adversary uses the ideal strategy by deciding on a fork point f and then putting its invalid blocks in $[f, f+(1-c)(1-f)]$, the probability of catching the adversary with one challenge is $\int_f^{1+fc-c} g(x)dx = \log_\delta(c)$. The probability of an adversary breaking the soundness of the protocol that uses t'' challenges for the interval $[0, 1-\delta]$ is thus $(1-\log_\delta(c))^{t''}$. Choosing t'' so that the latter is less than $2^{-\lambda}$ yields $t'' = \lambda/\big(\log_{1/2}(1-\log_{\ell/n}(c))\big)$, to which are added the ℓ challenges for the last ℓ blocks, which yields t'.

Protocol 3. We compare the efficiency of our main application, Protocol 3 from Fig. 9 in Sect. 6.3, when instantiated with a SNACK construction from either Sect. 5 or Appendix D and using the value α from Theorem 4.

The probability that the adversary proves a false statement for the SNACK is inherited from the underlying PoSW. Thus, by Theorems 2 and 5, respectively, the adversary's success probability is α^t (plus some negligible term), where t is the number of challenges (which determines the proof size). For security parameter λ, in order to upper-bound the soundness error by $2^{-\lambda}$, we set $\alpha^t = 2^{-\lambda}$. Solving for t with α from Theorem 4 yields

$$t = \frac{\lambda}{\log_{1/2}\left(1 - \left(\log_c\left(\frac{\ell-1}{m+\ell}\right)\right)^{-1}\right)} \ . \tag{19}$$

In Protocol 3, the prover must, in addition to the SNACK proof, provide the last ℓ blocks of its chain, which are checked explicitly. Adding ℓ to the number of blocks sampled by the SNACK in (19) corresponds almost exactly to the number of challenges in the FlyClient protocol [BKLZ20], given in (18). (The differences by 1 are due to the continuous idealization in the analysis of FlyClient.) When instantiating our SNACK construction with the PoSW based on CP graphs [CP18] (Constr. 3 in Appendix D), our construction most closely resembles FlyClient, except for the paths of our commitment openings being shorter, since we embed the blockchain along all the vertices of the graph, whereas FlyClient only uses leaf nodes.