SNACKs for Proof-of-Space Blockchains

Hamza Abusalah

IMDEA Software Institute, Madrid, Spain.
hamza.abusalah@imdea.org

Abstract. SNACKs are succinct non-interactive arguments of chain knowledge. They allow for efficient and generic solutions to blockchain light-client bootstrapping. Abusalah et al. construct SNACKs in the random oracle model for any *single-chain* blockchain from any graph-labeling proof of sequential work (PoSW) scheme. Their SNACK construction is a PoSW-like protocol over the augmented blockchain.
Unlike single-chain blockchains, such as proof-of-work and proof-of-stake blockchains, proof-of-space (PoSpace) blockchains are composed of two chains: a *canonical* proof chain and a data chain. These two chains are related using a signature scheme.
In this work, we construct PoSW-enabled SNACKs for any PoSpace blockchain. Combined with the results of Abusalah et al., this gives the first solution to light-client bootstrapping in PoSpace blockchains. The space cost of our construction is *two* hash values in each augmented PoSpace block. Generating SNACK proofs for a PoSpace blockchain is identical to generating SNACK proofs for single-chain blockchains and amounts to looking up a succinct number of augmented blocks.

1 Introduction

Consider a blockchain protocol Π, say Bitcoin or the Chia Network, and a light client V, which is assumed to hold only minimal information about Π, say its genesis block ψ. A *bootstrapping* protocol [BKLZ20, AFGK22] for a blockchain allows such V to hold a commitment to its stable prefix.

A succinct non-interactive arguments of chain knowledge (SNACK) system is a computationally-sound proof system (P, V) that allows a prover P to give a succinct non-interactive proof that convinces a verifier V that P *knows* a *chain* of certain weight. Crucially, the SNACK proof is succinct, i.e., poly-logarithmic in the length of the blockchain.

In [AFGK22], secure bootstrapping is formalized and instantiated for any blockchain protocol Π for which we have (1) a secure SNACK system and (2) a natural, and previously used assumption [BKLZ20] on the adversarial mining power holds. This is captured in the (c, ℓ, ϵ)-fork assumption, which informally says that, except with probability ϵ, no adversary can produce a fork containing ℓ consecutive blocks with more than

a c-fraction of them being valid.[1] Furthermore, Abusalah et al. [AFGK22] construct SNACKs for single-chain blockchains, like Bitcoin, generically from any graph-labeling proof of sequential work (PoSW) scheme assuming the (c, ℓ, ϵ)-fork assumption holds for such chains.

SNACKs for PoSpace Blockchains. In this paper, we study SNACKs for PoSpace blockchains [PKF+18, CP19]. These blockchains are composed of two chains in tandem: a proof chain and a data chain – see Fig. 1. Both chains are bound by a signature scheme. The proof chain contains only *canonical* data, such as (unique) proofs of space [DFKP15, AAC+17] and verifiable-delay function [BBBF18] computations. The data chain contains transactions and any arbitrary data the blockchain protocol allows. The dual nature of these chains and the requirement that the proof chain must remain canonical make designing SNACKs for such chains more involved than their single-chain blockchain counterparts, say Bitcoin.

Contributions. In this work, we extend the framework of [AFGK22] and construct SNACKs for *any* PoSpace blockchain from any graph-labeling PoSW scheme. The cost of our construction is *two* hash values in each augmented block, one in the augmented proof block, and one in the augmented data block. Generating a SNACK proof is as efficient as generating a PoSW proof, which amounts to looking up a succinct[2] number of blocks.

(We mention that simply defining the PoSpace SNACK to be two SNACK systems, one for the proof chain, and one for the data chain, doesn't result in a secure SNACK, and extra care needs to be exercised in order to prove security of the SNACK and maintain the security of the underlying PoSpace blockchain.)

Therefore, by the results of [AFGK22], by plugging in our PoSpace SNACK construction into their generic bootstrapping protocol, we get, to the best of our knowledge, the first solution to bootstrapping in PoSpace blockchains. Our protocol, as outlined above, is also practically efficient.[3]

[1] This assumption was first introduced in [BKLZ20] in the PoW-blockchain setting and adopted in [AFGK22] generically, i.e., without reference to the Sybil-mechanism of the underlying blockchain protocol. Studying the (c, ℓ, ϵ)-fork assumption in various blockchain protocols, and possibly deriving it from their underlying security assumptions, is an interesting open problem that we don't address in this work.

[2] Depends on the PoSW scheme used, this number maybe $O(t \log n)$ where n is the length of the blockchain and t a security parameter

[3] SNACKs are on par with Flyclient in terms of practical efficiency [AFGK22].

2 Preliminaries

In this section, we review the SNACK-related definitions from [AFGK22].

Notation. 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 $\mathsf{parents}_G(v)$, and their number (i.e., the indegree of v) by $\mathsf{deg}_G(v)$; thus, $\mathsf{parents}_G(v) = (v_1, \ldots, v_{\mathsf{deg}_G(v)})$. We also let $\mathsf{deg}(G) := \max_{v \in V}\{\mathsf{deg}_G(v)\}$. We drop the subscript G when it's clear from context. $\mathbb{N}_0$ denotes the set of non-negative integers.

Graph labeling. A *chain graph* is a DAG on $[n]_0$ vertices such that its edge set E contains a path $P := (0, \ldots, n)$ which goes through all $[n]_0$.

Definition 1 (Chain graphs). *A DAG $G_n = ([n]_0, E_n)$ is a* chain graph *if $E_n \supseteq \{(i-1, i) : i \in [n]\}$.*

A DAG is weighted if its vertices have arbitrary weights that sum to 1. In SNACK constructions, the verifier's challenges are sampled according to the distribution induced by the weights of the underlying DAG.

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

SNACK constructions are over labeled chain graphs. An augmented data corresponding to arbitrary blockchain data is infused into the random-oracle-based labeling of chain graphs that underlie SNACKs.

Definition 3 (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 $X = (x_0, \ldots, x_n) \in (\{0,1\}^*)^{n+1}$ the X-*augmented τ-labeling *$L^\tau \colon [n]_0 \to \{0,1\}^*$ of G_n is recursively defined as*

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

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

SNACKs. A *valid path* is a labeled path whose labels are locally valid according to some poly-time relation R and globally consistent as in (2).

Definition 4 (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 $p_v = (p_v[1], \ldots, p_v[\mathsf{deg}(v)])$. We say that $(P, L_P, (p_v)_{v\in P})$ is an R-*valid *path in G_n if $\forall v \in P$ with $(v_1, \ldots, v_{\mathsf{deg}(v)}) := \mathsf{parents}(v)$, we have*

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

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

For blockchains, typically $0 \in P$, so R_ψ can verify its genesis block ψ.

The language over which SNACKs are defined $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ is parameterized with prm which is formally generated by a parameter generation G algorithm. A statement $\eta = (\phi, n)$ in $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ consists of a position-binding Com commitment ϕ to an R-valid labeling of the graph $\Gamma_n \in \Gamma$. The labeling together with an opening of ϕ constitutes a witness w for η.

Definition 5 (Chain commitment language). *Let $\Gamma = (\Gamma_n)_{n\geq 0}$ be a family of weighted DAGs and Com a vector commitment scheme, define*

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

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 $\mathcal{R}_{\Gamma,R,\mathsf{Com}}$.

A SNACK system (P, V) for $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ in a non-interactive argument system satisfying completeness, (α, ϵ)-knowledge soundness, and succinctness. Completeness guarantees that an honest P holding a witness for a statement $(\phi, n) \in \mathcal{L}_{\Gamma,R,\mathsf{Com}}$ makes V accept. For an $\alpha \in (0, 1]$, (α, ϵ)-knowledge-soundness guarantees that from any convincing prover for a statement (ϕ, n) one can extract, except with probability ϵ, an R-valid labeling of a path P in Γ_n of weight at least α. In SNACK construction, due to the use of random oracles graph graph labeling, the labels of R will be guaranteed to be computed *sequentially*. Succinctness requires that the proof size as well as its verification time are poly-logarithmic in n and polynomial in the security parameter.

Definition 6 (SNACK). *A tuple of PPT algorithms* (P, V) *is a* succinct non-interactive argument of chain knowledge (SNACK) *for* $\mathcal{L}_{\Gamma,R,\mathsf{Com}}$ *with parameter generator* G *from Def. 5 if the following properties hold:*

Completeness: $\forall \lambda \in \mathbb{N}$, $\mathsf{prm} \leftarrow \mathsf{G}(1^\lambda)$, $\eta, w \in \{0,1\}^*$ *with* $(\mathsf{prm}, \eta, w) \in \mathcal{R}_{\Gamma,R,\mathsf{Com}}$, *we have* $\Pr\left[\pi \leftarrow \mathsf{P}(\mathsf{prm}, \eta, w) : \ \mathsf{V}(\mathsf{prm}, \eta, \pi) = 1\right] = 1$.

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

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

with $\mathcal{R}^{(\alpha)}_{\Gamma,R,\mathsf{Com}}$ *from* (3).

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

Our SNACK construction relies on graph-labeling PoSW schemes. For lack of space, we give a high-level overview here and defer the definition to Sect. A.1. A graph-labeling PoSW scheme $\mathsf{PoSW} = (\mathsf{PoSW.P} := (\mathsf{PoSW.label}, \mathsf{PoSW.open}), \mathsf{PoSW.V})$ is an (interactive) proof system in which $\mathsf{PoSW.P}$ on common input a weighted DAG (G_n, Ω_n) and a statement χ sampled by the verifier, computes a proof that convinces the verifier that a certain number of *sequential* computational steps with weight 1 according to Ω_n have been performed since χ was received. In particular, $\mathsf{PoSW.label}$ computes a τ labeling L of G_n and sends a vector commitment of L to $\mathsf{PoSW.V}$, which sends challenges to $\mathsf{PoSW.P}$, where $\mathsf{PoSW.open}$ replies by giving, among other things, commitment openings to these challenges. Finally $\mathsf{PoSW.V}$ accepts or rejects. (α, ϵ)-knowledge soundness of PoSW guarantees that from any convincing prover, a sequentially-labeled path of total weight $\geq \alpha$, can be extracted except with probability ϵ.

3 SNACKs for Proof-of-Space Blockchains

We extend the PoSW-enabled SNACK construction of [AFGK22] to the context of PoSpace blockchains. Our construction in a nutshell follows the simple outline of (1) defining an appropriate DAG (2) labeling it and (3) running a PoSW-like protocol over it.

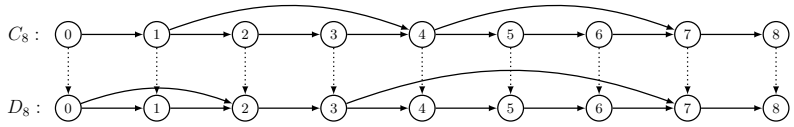

Fig. 1. Example DAGs $C_8 = ([8]_0, E_C)$ and $D_8 = ([8]_0, E_D)$.

3.1 Proof-of-Space Blockchains

The only two instances of PoSpace blockchains we are aware of are SpaceMint [PKF+18] and Chia [CP19] and our treatment covers them both.

Unlike blockchains based on either proofs of work (PoW) or proofs of stake (PoS), proofs of space (PoSpace) based blockchains are composed of two chains: a *canonical* proof chain and a data chain. The proof chain contains unique proofs and hence is canonical, while the data chain contains transactions and any arbitrary data that the blockchain permits. The data chain is bound to the proof chain by means of digital signatures.

Without loss of generality we can view a *PoSpace blockchain* as a tuple of labeled chains whose underlying DAG is $B_n := (C_n, D_n)$ where $C_n := ([n]_0, E_C)$ and $D_n := ([n]_0, E_D)$ are the chain graphs underlying the canonical proof and data chains, respectively. Both C_n and D_n are chain graphs in the sense of Def. 1, and we stress that E_C and E_D need not be equal. Furthermore, C_n is bound to D_n by a digital signature scheme $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ in a simple manner that we explain shortly below. Example chain graphs for C_n, D_n are shown in Fig. 1.

We view blockchain mining as the process of labeling the vertices of these chains. We let $(b_i := (c_i, d_i))_{i \in [n]_0}$ denote the labels of these chains, where c_i and d_i denote the ith labels of the canonical and data blocks, respectively. Although our treatment allows for arbitrary labeling, c_i and d_i, for simplicity of exposition, can be assumed to have the following format (which is faithful to existing PoSpace blockchains):

- $c_i := (i, \pi_i)$ where π_i is a canonical computation that depends on the labels of parent proof blocks $(c_{i_1}, \ldots, c_{i_q})$ where $(i_1, \ldots, i_q) := \mathsf{parents}_C(i)$. For simplicity, we assume that $\pi := (\delta_i, (\mathsf{VDF}^i_v, \mathsf{VDF}^i_p))$ where δ_i is a proof of space [DFKP15, AAC+17] and $(\mathsf{VDF}^i_v, \mathsf{VDF}^i_p)$ is a verifiable delay function [BBBF18] computation/proof pair.
- $d_i := (s_i, \mathsf{data}_i)$ where $s_i \leftarrow \mathsf{Sign}_{\mathsf{sk}}(d_{i_1} \| \ldots \| d_{i_p} \| \mathsf{data}_i \| c_i)$ is a signature on the parents data blocks $d_{i_1} \| \ldots \| d_{i_p}$ where $(i_1, \ldots, i_p) := \mathsf{parents}_D(i)$, the current data data_i, and the current proof block c_i.

These simplifying assumptions are without loss of generality. For example, in both Chia and SpaceMint, π_i contains a *unique* PoSpace δ_i. The PoSpace challenge chal_i for δ_i is uniquely determined by the labels of its parents $\mathsf{parents}_C(i)$. The value of δ_i is defined by chal_i and the public key pk associated with the signing key sk of SIG. In Chia, π_i additionally contains $(\mathsf{VDF}_v^i, \mathsf{VDF}_p^i)$ where VDF_v^i is a verifiable delay function evaluation on input x_i for a time parameter t_i and VDF_p^i is a unique proof of correctness of VDF_v^i; both x_i and t_i are uniquely defined by $\mathsf{parents}_C(i)$.[4]

3.2 SNACKs for PoSpace Blockchains: An Overview

Constructing SNACKs for PoSpace blockchains is more subtle than for PoW blockchains, mainly due to the requirement that proof chain blocks must remain canonical. That the proof chain must be canonical (non-grindable) is crucial for the security of PoSpace blockchains [PKF+18, CP19].[5] (If such a requirement is relaxed, then simpler solutions would be possible – see Sect. B.2.)

In Sect. A.2 we give a detailed account of the generic PoSW-enabled SNACK for single chain blockchains of [AFGK22]. We give here a high-level overview. Let $H_n := ([n]_0, E_H)$ be the underlying chain graph of the blockchain in question and $G_n := ([n]_0, E_G)$ the chain graph of a PoSW scheme. Then, the SNACK construction works by first defining an augmented chain graph $K_n := ([n]_0, E_K := E_H \cup E_G)$ whose ith augmented label is $k_i := (g_i, h_i)$ where g_i is defined by the underlying PoSW scheme and h_i contains the actual content of the block including the publicly verifiable (say PoW) proof π_i. The SNACK then would essentially be a non-interactive augmented PoSW on this labeled K_n. The (α, ϵ)-knowledge soundness guarantees imply that from any successful prover, we can extract, except with probability ϵ, an (α, R)-valid path $(P, L_P, (p_v)_{v \in P})$ as defined in Def. 4, such that the labels of L_P are sequentially computed and have total weight α.

For notational simplicity, we refer (α, R)-valid paths $(P, L_P, (p_v)_{v \in P})$ by (P, L_P), and when R is either clear from the context or irrelevant for the discussion, we call an (α, R)-valid path α-valid.

In Sec B.2, we explore a few (insecure) natural approaches that resemble the PoSW-enabled SNACK construction for single-chain blockchains.

[4] In fact, in Chia, the pair $(\mathsf{VDF}_v^i, \mathsf{VDF}_p^i)$ is a pair of tuples, i.e., $\mathsf{VDF}_v^i = (y_{i_1}, \ldots, y_{i_k})$ and $\mathsf{VDF}_p^i = (\rho_{i_1}, \ldots, \rho_{i_k})$ where (y_{i_j}, ρ_{i_j}) is a VDF evaluation/proof pair on a challenge and time parameter pair (x_{i_j}, t_{i_j}), which is uniquely defined by the proof chain so far $(c_0, \ldots, c_{i-1})$.

[5] In Sec. B, we highlight the need for canonical proofs in PoSpace blockchains.

Our SNACK. We construct a SNACK for $B_n = (C_n, D_n)$, by constructing two SNACKs simultaneously, one for C_n, call it CS, and one for the data chain D_n, call it DS. Both CS and DS are generic PoSW-enabled constructions following the blueprint ofh [AFGK22]. They also satisfy:

1. CS and DS both use PoSW, i.e., the same underlying PoSW scheme
2. PoSW uses a *deterministic* Com, and
3. CS is embedded into DS.

The final PoSpace SNACK is simply DS. The soundness guarantees of DS is that from any convincing prover, we can extract an α-valid path (P, L_P) where $L_P = ((c_{i_1}, d_{i_1}), \ldots, (c_{i_k}, d_{i_m}))$ is such that (c_{i_j}, d_{i_j}) is a valid blockchain block and that $(c_{i_1}, \ldots, c_{i_m})$ and $(d_{i_1}, \ldots, d_{i_m})$ are both sequentially computed. These are the guarantees that a SNACK should provide for a blockchain: sequentiality of its blocks.

Let's justify the design choice made above. Note that assuming the same PoSW chain graph in CS and DS simplifies the final construction, and requiring Com to be deterministic is necessary to preserve the canonical nature of the augmented proof chain.[6] To see the necessity of embedding CS into DS, let's see what guarantees one would get from these SNACKs individually, and why these guarantees falls short of our goal of ensuring the sequentiality of the combined PoSpace blockchain blocks.

From α-valid paths (P_c, L_{P_c}) and (P_d, L_{P_d}) extracted from CS and DS respectively, we would like to construct an α-valid path (P, L_P) as above. However, as it may be the case that $P_c \neq P_d$, i.e., P_c, P_d may not coincide, constructing (P, L_P) with weight α out of (P_c, L_{P_c}) and (P_d, L_{P_d}) may not be possible.

A natural first idea towards ensuring $P_c = P_d$ would be to fix the same PoSW scheme in both CS and DS. This way, we augment both C_n and D_n with the same PoSW chain graph G_n to arrive at augmented chain graphs K_n^c and K_n^d, respectively. However, this doesn't mean that $K_n^c = K_n^d$ as C_n need not be equal to D_n. But as we will show in Lemma 1 below, in any PoSW-enabled SNACK, over an augmented chain graph, say $K_n^c := ([n]_0, E_K := E_C \cup E_G)$, any *extractable* α-valid path (P_c, L_{P_c}) is such that P_c lies in G_n. Still, that P_c and P_d lie in G_n doesn't mean they coincide, but now we are a step closer towards ensuring they do.

To be able to compose an α-valid path (P, L_P) from α-valid paths (P_c, L_{P_c}) and (P_d, L_{P_d}), not only we want to ensure that $P_c = P_d$, but also that their labels are valid and bound, i.e., let $L_{P_c} = (c_{i_1}, \ldots, c_{i_m})$ and

[6] We remark that all PoSW schemes in the ROM [MMV13, CP18, AKK+19, DLM19, AFGK22] use deterministic Com anyway.

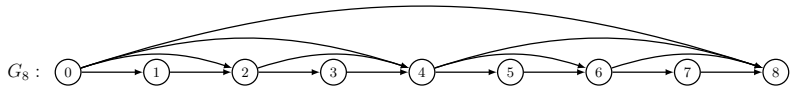

Fig. 2. An illustrative example DAG of a skiplist-based PoSW [AKK+19, AFGK22].

$L_{P_d} = (d_{i_1}, \ldots, d_{i_m})$, then it must hold that (c_{i_j}, d_{i_j}) is a valid blockchain block including that d_{i_j} contains a signature on c_{i_j}. Recall that SIG binds C_n to D_n. Now because c_{i_j} is needed to validate d_{i_j}, DS can't simply be independent of CS.

To resolve all issues at once, that is, to make sure $P_c = P_d$ and that $L_P := ((c_{i_1}, d_{i_1}), \ldots, (c_{i_m}, d_{i_m}))$ is valid, where $P := P_c = P_d$, we require that CS and DS use the *same* underlying PoSW scheme, and furthermore, *embed* CS into DS. By embedding the augmented labeled proof chain K_n^c into the augmented labeled data chain K_n^d, and relying on Lemma 1 below, we ensure that the same labeled path in K_n^d contains a valid labeling in K_n^c at the same time.

3.3 SNACK for PoSpace Blockchains: The Main Construction

For simplicity, fix an integer n and let PoSW be any GL-PoSW scheme and $\Gamma_n = (G_n = ([n]_0, E_G), \Omega_n)$ its underlying weighted chain graph where G_n is a chain graph and $\Omega_n : [n]_0 \to [0,1]$ s.t. $\Omega_n([n]_0) = 1$ is a weight function. We will use the PoSW from [AKK+19, AFGK22] in our illustrative examples. Its underlying DAG is depicted in Fig. 2

Furthermore, let $B_n = (C_n, D_n)$ be a PoSpace blockchain with ψ being its genesis block, and R_ψ^c and R_ψ^d be the polynomial-time validity relations for the (labeled) proof and data chains, respectively. That is, let $(i_1, \ldots, i_p) := \mathsf{parents}_C(i)$, then

$$R_\psi^c(i, c_i, (c_{i_1}, \ldots, c_{i_p})) = 1 \tag{5}$$

iff c_i is a valid proof chain block, and for $(i_1, \ldots, i_q) := \mathsf{parents}_D(i)$:

$$R_\psi^d(i, (c_i, d_i := (s_i, \mathsf{data}_i)), (d_{i_1}, \ldots, d_{i_q})) = 1 \tag{6}$$

iff d_i is a valid data chain block. In particular, (6) implies that

$$\mathsf{Vrfy}_{\mathsf{pk}}(d_{i_1} \| \ldots \| d_{i_q} \| \mathsf{data}_i \| c_i, s_i) = 1 \ . \tag{7}$$

The validity relation for B_n is simply the relation that checks that both (5) and (6) hold simultaneously. These validity relations are blockchain specific and they can be augmented or redefined to suite the specific instantiation of the PoSpace blockchain in question.

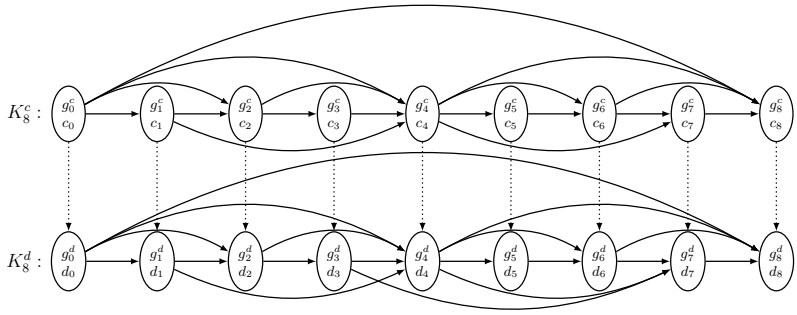

Fig. 3. Example K_8^c and K_8^d defined in (8) and (9) where C_n and D_n are from Fig. 1 and G_8 is from Fig. 2. CS and DS are w.r.t. K_8^c and K_8^d, respectively. The dashed arrows indicate the labels of the source are embedded into the labels of the target. The labels of these graphs are computed by SInit and SMine of Fig. 4.

Augmented Blockchains. We define *augmented chain graphs* K_n^c and K_n^d that respectively underlie CS and DS as follows:

$$K_n^c := ([n]_0, E_{K^c}) \quad \text{with} \quad E_{K^c} := E_G \cup E_C \tag{8}$$

$$K_n^d := ([n]_0, E_{K^d}) \quad \text{with} \quad E_{K^d} := E_G \cup E_D \cup E_C \tag{9}$$

Note that while K_n^c augments C_n with G_n, K_n^d augments the union of C_n and D_n with G_n. The reason for this is that we would like the DS extractor to succeed in extracting a labeled path in (the labeled) K_n^d such that it contains a labeled path in K_n^c, and for this to be possible, we make sure that (1) the (augmented) labels of K_n^c are embedded (as data) in the (augmented) labels of K^d and that (2) $E_{K^d} \supseteq E_{K^c}$. Examples of these graphs are depicted in Fig. 3.

The *augmented blockchain* is obtained by labeling the augmented chain graphs K_n^c and K_n^d at once and furthermore embedding K_n^c in K_n^d as data. This labeling is formalized by the augmented mining algorithms Init and SMine, 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 pp.

SMine, on input $(k_j^d := (g_j^d, d_j)_{j \in [i-1]_0}$, i.e., the augmented labels of the first i vertices of K_n^d, as well as the current data data_i including transactions and the signing/verification key pair $(\mathsf{sk}, \mathsf{pk})$ of an arbitrary space farmer, and some auxiliary information $\mathsf{aux}_{i-1}^c, \mathsf{aux}_{i-1}^d$ related to the commitment opening (which we explicitly state in Fig. 4, but ignore in

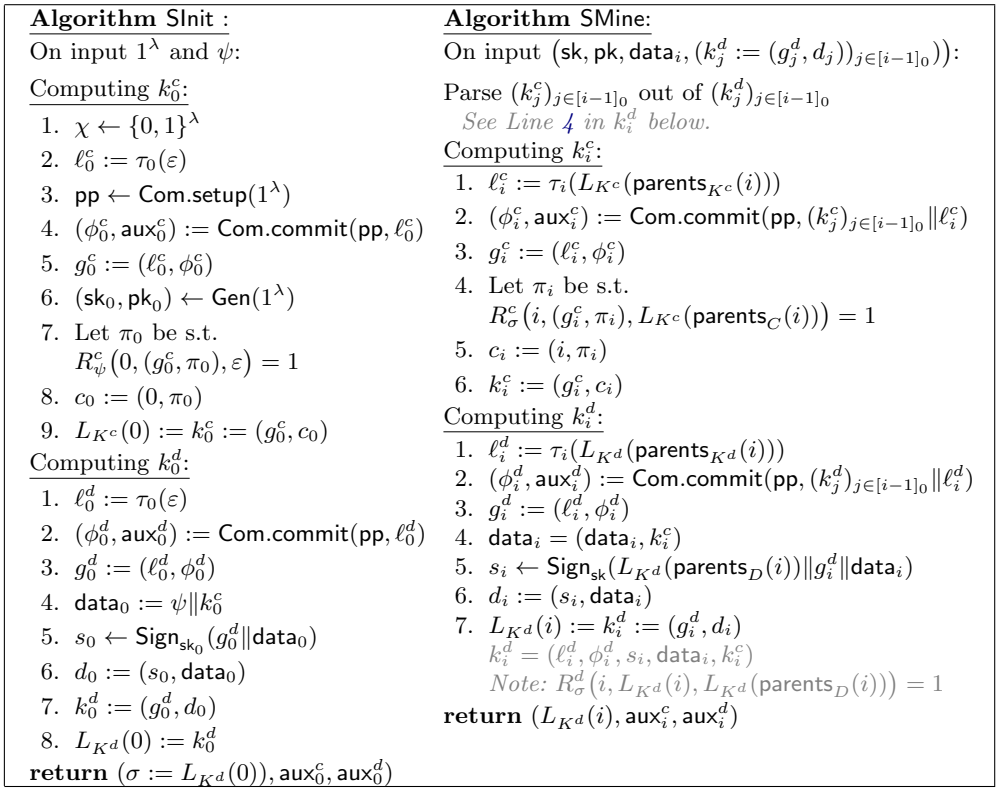

Fig. 4. The mining algorithms Init and SMine for PoSpace-augmented blockchains.

this informal discussion for simplicity), computes the augmented labels of the both the proof and data chain as follows.

As for the ith augmented proof chain label, SMine computes the PoSW label ℓ_i^c using the random oracle τ_i and the graph structure of K_n^c, computes a *deterministic* commitment ϕ_i^c of the labels $(k_j^c := (g_j^c, c_j))_{j\in[i-1]_0}$ and ℓ_i^c, and defines the label $g_i^c := (\ell_i^c, \phi_i^c)$. We stress that the commitment ϕ_i^c must be deterministic, for otherwise the proof chain becomes grindable. The label of the proof chain is defined as $c_i := (i, \pi_i)$ and the augmented label is defined as $L_{K^c}(i) := k_i^c := (g_i^c, c_i)$. For the sequentiality guarantees of the PoSW to carry on to the SNACKs, we must ensure that in the augmented blockchain, π_i is computed *after* ϕ_i^c – see [AFGK22] for details. However, note that Line 4 in Fig. 4 doesn't make this explicit, as ensuring this condition is blockchain-specific.

As for the ith augmented data chain label, SMine computes the PoSW label ℓ_i^d using τ_i and the graph structure of K_n^d, computes a commit-

ment ϕ_i^d of the labels $(k_j^d := (g_j^d, d_j))_{j\in[i-1]_0}$ and ℓ_i^d, and defines the label $g_i^d := (\ell_i^d, \phi_i^d)$. A space farmer/miner who generates δ_i using pk, computes a digital signature s_i, using the corresponding signing key sk, on $(L_{K^d}(\mathsf{parents}_D(i))\|g_i^d\|\mathsf{data}_i)$, where $L_{K^d}(\mathsf{parents}_D(i))$ are the augmented labels of the parents of i in D, data_i is the current data including (the now embedded) k_i^c, transactions and any arbitrary data that the original mining protocol allows. Finally set $d_i := (s_i, \mathsf{data}_i)$ and $L_{K^d} := k_i^d := (g_i^d, d_i)$.

Blockchain validity must be adapted to accommodate the augmentation. Therefore, we define the *augmented validity relations* R_σ^c and R_σ^d of the proof and data chains, by overriding R_ψ^c and R_ψ^d, respectively. Both R_σ^c and R_σ^d still consider the same graph structure as R_ψ^c and R_ψ^d but expect augmented labels. Concretely, we override R_ψ^c from (5) as

$$R_\sigma^c\big(i, L_{K^c}(i), L_{K^c}(\mathsf{parents}_C(i))\big) = 1 \ , \tag{10}$$

iff the augmented block is valid. Note that σ is defined by SInit and is the augmented genesis block. As before, this validity is blockchain specific. Similarly, we override R_ψ^d from (6) as

$$R_\sigma^d\big(i, L_{K^d}(i), L_{K^d}(\mathsf{parents}_D(i))\big) = 1 \tag{11}$$

iff the augmented data block is valid, which in particular, implies that

$$\mathsf{Vrfy}_{\mathsf{pk}}(L_{K^d}(\mathsf{parents}_D(i))\|g_i^d\|\mathsf{data}_i, s_i) = 1 \ . \tag{12}$$

Note that R_σ^d doesn't verify transaction consistency in data_i. The consistency of data_i is assumed, i.e., we assume honest miners would not finalize block i that contains data_i that is inconsistent with $\mathsf{data}_0, \ldots, \mathsf{data}_{i-1}$. The consistency of data in orthogonal to the SNACK constructions.

As the ith augmented (proof/data) block contains, not only blockchain-specific, but also PoSW-specific data, we define augmented validity relations $\tilde{R}_\sigma^c$ and $\tilde{R}_\sigma^d$ that check the validity of (a) the blockchain-specific data using R_σ^c and R_σ^d, respectively, and check (b) the PoSW data as formally defined in (17) (in Def. 9, Sect. A.1). Concretely, define $\tilde{R}_\sigma^c, \tilde{R}_\sigma^d$ and R_σ:

$$\begin{aligned}&\tilde{R}_\sigma^c\big(i, L_{K^c}(i), L_{K^c}(\mathsf{parents}_{K^c}(i))\big) = 1 \Leftrightarrow \exists\, x_i \text{ s.t.} \\ &R_\sigma^c\big(i, L_{K^c}(i), L_{K^c}(\mathsf{parents}_C(i))\big) = 1 \wedge L_{K^c}(i) = \tau_i(L_{K^c}(\mathsf{parents}_{K^c}(i)))\|x_i.\end{aligned} \tag{13}$$

$$\begin{aligned}&\tilde{R}_\sigma^d\big(i, L_{K^d}(i), L_{K^d}(\mathsf{parents}_{K^d}(i))\big) = 1 \Leftrightarrow \ \exists\, x_i \text{ s.t.} \\ &L_{K^d}(i) = \tau_i(L_{K^d}(\mathsf{parents}_{K^d}(i)))\|x_i \wedge R_\sigma^d\big(i, L_{K^d}(i), L_{K^d}(\mathsf{parents}_D(i))\big) = 1.\end{aligned} \tag{14}$$

$$R_\sigma\big(i, L_{K^d}(i), L_{K^d}(\mathsf{parents}_{K^d}(i))\big) = 1 \Leftrightarrow \tag{15}$$
$$\tilde{R}^c_\sigma\big(i, L_{K^c}(i), L_{K^c}(\mathsf{parents}_{K^c}(i))\big) = \tilde{R}^d_\sigma\big(i, L_{K^d}(i), L_{K^d}(\mathsf{parents}_{K^d}(i))\big) = 1.$$

where by construction (see Fig. 4), $L_{K^c}(i)$ is contained in $L_{K^d}(i)$ as part of data_i and $\mathsf{parents}_{K^c}(i)$ is contained in $\mathsf{parents}_{K^d}(i)$ by definition of K^d_n.

Arguments for Augmented PoSpace Blockchains. We first define the witness relation $\mathcal{R}$ with respect to which we construct our SNACK.

Definition 7 (Augmented PoSpace chain language). *We define the* augmented PoSpace chain relation $\mathcal{R}^{(1)}_{\Gamma^d_n, R_\sigma, \mathsf{Com}}$ *for any* $\alpha \in (0,1]$ *as*

$$\mathcal{R}^{(\alpha)}_{\Gamma^d_n, R_\sigma, \mathsf{Com}} = \left\{ \begin{array}{l} \big((\mathsf{prm} := (\sigma, \mathsf{pp}), \eta := ((\phi^c, \phi^d), n), \\ w := (P, L^d_P, (p^d_i)_{i\in P}, \rho^c, \rho^d)\big) \ \textit{s.t.} \\ (P, L^d_P, (p^d_i)_{i\in P}) \ \textit{is an } (\alpha, R_\sigma)\textit{-valid path in } \Gamma^d_n \\ \wedge\ \mathsf{Com.ver}(\mathsf{pp}, \phi^c, L^c_P, P, \rho^c) = 1 \\ \wedge\ \mathsf{Com.ver}(\mathsf{pp}, \phi^d, L^d_P, P, \rho^d) = 1 \end{array} \right\}. \tag{16}$$

where R_σ *is defined in* (15) *and* $\Gamma^d_n := (K^d_n, \Omega_n)$ *for* K^d_n *as in* (9) *and* Ω_n *from the underlying PoSW scheme* PoSW. *We let* $\mathcal{L}_{\Gamma^d, R_\sigma, \mathsf{Com}}$ *(cf. Def 5) be the language associated with* $\mathcal{R}^{(1)}_{\Gamma^d_n, R_\sigma, \mathsf{Com}}$

Having formalized the language $\mathcal{L}_{\Gamma^d, R_\sigma, \mathsf{Com}}$, we now build for it an ACK system $\mathsf{ACK} := (\mathsf{ACK.P}, \mathsf{ACK.V})$, which we later Fiat-Shamir to get our final DS. The ACK is given in Fig. 6. Note that the prover takes as input the labeling of K^d_n as its witness. Additionally, the SNACK parameter generator G outputs $\mathsf{prm} := \sigma$. Syntactically, ACK can be seen as two copies of the respective ACK construction given in [AFGK22] and recalled in Sect. A.2, one for the proof chain and one for the data chain. The difference is that we embed K^c_n into K^d_n, use the same underlying PoSW scheme as well as the same verifier challenges in these two copies.

Theorem 1. *Let* $\mathsf{SNACK} := (\mathsf{SNACK.P}, \mathsf{SNACK.V})$ *be the non-interactive counterpart of* ACK *from Fig. 6, then in the ROM,* SNACK *is an* (α, ϵ)*-knowledge-sound SNACK for* $\mathcal{L}_{\Gamma^d, R_\sigma, \mathsf{Com}}$ *as in Def. 7 if* PoSW *is an* (α, ϵ)*-knowledge-sound* τ*-based GL-PoSW as in Def. 9 with* Com *being its underlying deterministic 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.*

Algorithm $\mathsf{PoSW.ver}_{K^c}$:
On input (χ, ι_i, o_i):

1. Run $b_i := \mathsf{PoSW.ver}(\chi, \iota_i, o_i)$ modified as follows: whenever it queries $\tau_j(L_{K^c}(\mathsf{parents}_G(j)))$ for some j, issue query $\tau_j(L_{K^c}(\mathsf{parents}_{K^c}(j)))$ instead. *(Missing labels are provided in $o_{i,2}$.)*
2. **return** b_i

Algorithm $\mathsf{PoSW.open}_{K^c}$:
On input $(\chi, \mathsf{pp}, \phi_n, \mathsf{aux}_n, L, \iota_i)$:

1. $o_{i,1} \leftarrow \mathsf{PoSW.open}(\chi, \mathsf{pp}, \phi_n, \mathsf{aux}_n, L, \iota_i)$ *(*$\mathsf{PoSW.open}$ *acts based on edges E_G.)*
2. $\mathcal{J} := \{j \in [n]_0 : L_{K^c}(\mathsf{parents}_G(j))$ which appear in $o_{i,1}\}$
3. $o_{i,2} := \{(j, L_{K^c}(\mathsf{parents}_C(j)))\}_{j \in \mathcal{J}}$
4. **return** $o_i := (o_{i,1}, o_{i,2})$

Fig. 5. Algorithms $\mathsf{PoSW.open}_{K^c}$ and $\mathsf{PoSW.ver}_{K^c}$ defined based on $\mathsf{PoSW.open}$ and $\mathsf{PoSW.ver}$, respectively. Algorithms $\mathsf{PoSW.open}_{K^d}$ and $\mathsf{PoSW.ver}_{K^d}$ are defined analogously by changing every occurrence of c to d and C to D.

SNACK's Cost and Guarantees. We started with an underlying chain $B_n = (C_n, D_n)$ of a PoSpace blockchain, and augmented it to (K_n^c, K_n^d), on which we run ACK, whose non-interactive counterpart SNACK is the SNACK construction for the PoSpace blockchain. The space cost of SNACK is storing in each block in K_n^c a PoSW label and a commitment pair (ℓ_i^c, ϕ_i^c). The same holds for K_n^d. (Note that the embedding of $L_{K^c}(i)$ into $L_{K^d}(i)$ doesn't mean that we actually store $L_{K^c}(i)$ twice, in K_n^c and K_n^d; all what it means is that computing $L_{K^d}(i)$ and verifying $L_{K^d}(i)$ requires having $L_{K^c}(i)$ explicitly given as input.) If we instantiate PoSW from either PoSW schemes in [AFGK22], then $\phi_i^c = \ell_i^c$ and $\phi_i^d = \ell_i^d$ and hence the space cost is ℓ_i^c and ℓ_i^d per (PoSpace) block. Setting $|\ell_i^c| = |\ell_i^d| = 256$ bits is a reasonable instantiation. Furthermore, modeling a hash function as a RO, ℓ_i^c and ℓ_i^d are efficient hash computations.

Generating SNACK proofs is identical to generating PoSW proofs in the underlying PoSW scheme when $E_C = E_D = \{(i-1, i) : i \in [n]\}$. The more edges E_C and E_D contain, the bigger the proof size. Furthermore optimizations similar to those given in [AFGK22] are possible.

At this storage and computation costs, we get sequentiality guarantees. Fix prm and a statement $(\phi^c, \phi^d, n) \in \mathcal{L}_{\Gamma^d, R_\sigma, \mathsf{Com}}$, then from any convincing SNACK prover, a witness w can be extracted, and such a witness contains an (α, R_σ)-valid path in Γ_n^d, which is sequentially computed. That the extracted path lies in K_n^d is clear, but that it also lies in K_n^c is not. In the sequel, we show that extracted paths must be in K_n^c as well, and hence this shows that the extracted path contains valid blocks in the combined augmented blockchain. This, in turn, allows us to talk of the PoSpace blockchain as a single sequentially mined chain.

Lemma 1 says that if P is a path extracted by the knowledge-soundness of SNACK from Theorem 1, then the edges of P lie in G_n.

Verifier $\mathsf{ACK.V} = (\mathsf{V}_1, \mathsf{V}_2)$

Stage V_1: On input $(1^\lambda, \eta)$:

1. $\forall i \in [t]$ **do** $\iota_i \xleftarrow{\$} \Omega_n$
2. $\iota_0 := 0$
3. **send** $\iota := (\iota_i)_{i\in[t]_0}$ **to** P

Stage V_2: On input $\gamma := (\gamma_i)_{i\in[t]_0}$:

1. $\forall i \in [t]$ **do:**
   (a) $b^c_{i,1} := \tilde{R}^c_\sigma(\iota_i, k^c_{\iota_i}, p^c_{\iota_i})$
   (b) $b^c_{i,2} := \mathsf{PoSW.ver}_{K^c}(\chi, \iota_i, o^c_i)$
   (c) $b^c_{i,3} := \mathsf{Com.ver}(\mathsf{pp}, \phi^c_n, k^c_{\iota_i}, \iota_i, \rho^c_i)$
   (d) $b^d_{i,1} := \tilde{R}^d_\sigma(\iota_i, k^d_{\iota_i}, p^d_{\iota_i})$
   (e) $b^d_{i,2} := \mathsf{PoSW.ver}_{K^d}(\chi, \iota_i, o^d_i)$
   (f) $b^d_{i,3} := \mathsf{Com.ver}(\mathsf{pp}, \phi^d_n, k^d_{\iota_i}, \iota_i, \rho^d_i)$
2. **output** $\bigwedge_{i=0}^t b^c_{i,1} \wedge b^c_{i,2} \wedge b^c_{i,3} \wedge b^d_{i,1} \wedge b^d_{i,2} \wedge b^d_{i,3}$

Prover ACK.P:

On input $(1^\lambda, (\eta, (k^d_j)_{j\in[n]_0}, (\mathsf{aux}^c_n, \mathsf{aux}^d_n), \iota)$:

1. Parse $\eta := (\sigma, \phi^c_n, \phi^d_n, n)$
2. Parse $(k^c_j)_{j\in[n]_0}$ out of $(k^d_j)_{j\in[n]_0}$
3. $\forall i \in [t]_0$ **do:**
   (a) $o^c_i \leftarrow \mathsf{PoSW.open}_{K^c}(\chi, \mathsf{pp}, \phi^c_n, \mathsf{aux}^c_n, (k^c_j)_{j\in[n]_0}, \iota_i)$
   (b) $o^d_i \leftarrow \mathsf{PoSW.open}_{K^d}(\chi, \mathsf{pp}, \phi^d_n, \mathsf{aux}^d_n, (k^d_j)_{j\in[n]_0}, \iota_i)$
   We assume o^d_i contains both $L_{K^d}(\iota_i)$ and $p^d_{\iota_i} := L_{K^d}(\mathsf{parents}_{K^d}(\iota_i))$. Similarly for o^c_i, see Fig. 7.
   (c) $\rho^c_i \leftarrow \mathsf{Com.open}(\mathsf{pp}, \phi^c_n, \mathsf{aux}^c_n, k^c_{\iota_i}, \iota_i)$
   (d) $\rho^d_i \leftarrow \mathsf{Com.open}(\mathsf{pp}, \phi^d_n, \mathsf{aux}^d_n, k^d_{\iota_i}, \iota_i)$
   (e) $\gamma_i := ((o^c_i, o^d_i), (\rho^c_i, \rho^d_i))$
4. **send** $\gamma := (\gamma_i)_{i\in[t]_0}$ **to** V_2

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

Lemma 1. *Let* SNACK *be as in Theorem 1 and let*

$$\Big(\sigma, \eta := ((\phi^c, \phi^d), n), w := (P, L^d_P, (p^d_i)_{i\in P}, \rho^c, \rho^d)\Big) \in \mathcal{R}^{(\alpha)}_{\Gamma^d_n, R_\sigma, \mathsf{Com}}$$

be such that w is output by the extractor guaranteed by the (α, ϵ)-knowledge soundness of SNACK*, then P is a path in G_n.*

The following lemma shows that SNACK contains an embedded SNACK system for $\mathcal{L}_{\Gamma^c_n, \tilde{R}^c_\sigma, \mathsf{Com}}$, where $\mathcal{L}_{\Gamma^c_n, \tilde{R}^c_\sigma, \mathsf{Com}}$ is as in Def. 5 and $\tilde{R}^c_\sigma$ as in (13).

Lemma 2. *Let $\big(\sigma, ((\phi^c, \phi^d), n), (P, L^d_P, (p^d_i)_{i\in P}, \rho^c, \rho^d)\big) \in \mathcal{R}^{(\alpha)}_{\Gamma^d_n, R_\sigma, \mathsf{Com}}$ and P be a path in G_n, then $(\sigma, (\phi^c, n), w := (P, L^c_P, (p^c_i)_{i\in P}, \rho^c)) \in \mathcal{R}^{(\alpha)}_{\Gamma^c_n, \tilde{R}^c_\sigma, \mathsf{Com}}$, where $L^c(i), p^c_i$ are embedded in and extracted from $L^d(i), p^d_i$.*

Theorem 1 and Lemma 2 imply Corollary 1. which shows that SNACK proves knowledge of sequentially computed and valid PoSpace blockchain.

Corollary 1. *Let* SNACK *be as in Theorem 1 and let*

$$\Big(\sigma, \eta := ((\phi^c, \phi^d), n), w := (P, L^d_P, (p^d_i)_{i\in P}, \rho^c, \rho^d)\Big) \in \mathcal{R}^{(\alpha)}_{\Gamma^d_n, R_\sigma, \mathsf{Com}}$$

be s.t. w is output by the extractor guaranteed by the (α, ϵ)-knowledge soundness of SNACK*, then (P, L^d_P) is a sequentially-computed path containing valid PoSpace blocks, i.e., let $P = (i_0, \ldots, i_k)$, then $\forall j \in [k]_0, L^d_P(i_j)$*

contains a valid augmented data chain block which contains a signature on a valid augmented proof chain block $L_P^c(i_j)$*, and for every* $j \in [k]$*,* $L_P^c(i_j)$ *is computed* after $L_P^c(i_{j-1})$ *and* $L_P^d(i_j)$ after $L_P^d(i_{j-1})$*.*

Proofs of Theorem 1 and Lemmas 1 and 2 are, due to space constrains, given in Sect. C.

Acknowledgements. I want to thank Karen Klein for a fruitful discussion at the beginning of the project and for her feedback on an early version of this paper. Parts of this work was done while the author was at TU Wien supported by the Vienna Science and Technology Fund (WWTF) through project VRG18-002. The project 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.

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.

AFGK22. Hamza Abusalah, Georg Fuchsbauer, Peter Gaži, and Karen Klein. Snacks: Leveraging proofs of sequential work for blockchain light clients. Cryptology ePrint Archive, Paper 2022/240, 2022. https://eprint.iacr.org/2022/240.

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.

BKLZ20. Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 928–946. IEEE, 2020.

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 9, 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.

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.

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.

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.

PKF+18. Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gazi, Joël Alwen, and Krzysztof Pietrzak. SpaceMint: A cryptocurrency based on proofs of space. In Sarah Meiklejohn and Kazue Sako, editors, *FC 2018*, volume 10957 of *LNCS*, pages 480–499. Springer, Heidelberg, February / March 2018.

A Omitted Preliminaries

A.1 Graph-Labeling Proofs of Sequential Work Schemes

In this section, we review the definition of Augmented GL-PoSW schemes. Informally, a PoSW scheme is an (interactive) proof system in which the prover on common input an integer parameter n and a statement χ sampled by the verifier, computes a proof that convinces the verifier that n *sequential* computational steps have been performed since χ was received. Existing PoSW constructions [MMV13, CP18, AKK+19, DLM19, AFGK22] are based on a random-oracle induced labeling of a particular weighted DAG $(G_n = ([n]_0, E_G), \Omega_n)$.[7] The prover, upon receiving a statement χ from the verifier, uses χ to refresh a random oracle $\mathcal{O} : \{0,1\}^* \to \{0,1\}^\lambda$ to compute a labeling $L : [n]_0 \to \{0,1\}^\lambda$ of G_n, where the label of $i \in [n]_0$ is simply defined as $L(i) := \mathcal{O}(\chi, i, L(\mathsf{parents}(i)))$, where $\mathsf{parents}(i)$ denotes the parents of i in G_n in any fixed ordering. The prover then sends to the verifier a (position-binding) commitment ϕ to the labeling L. The verifier then engages with the prover in a challenge response protocol in which for each challenge $i \in [n]_0$ drawn by the verifier according to distribution induced by Ω_n, the prover responds with protocol- and challenge-specific labels from L, which the verifier then check for consistency. (A part of the prover's response to challenge i is a ϕ-opening at position i.) This interactive protocol is then made non-interactive in the ROM by applying the Fiat-Shamir [FS87] transform.

The formal definition of augmented graph-labeling PoSW schemes [AFGK22] utilizes the notion of sequential weights of oracle queries.

Definition 8 (Sequential weight [AFGK22]). *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 $\Omega_n : [n]_0 \to [0,1]$ as*

$$\Omega_{\mathrm{seq}}(Q) := \sum_{i=1}^{\ell} \max\left\{\Omega_n(j) \,:\, Q_i \text{ contains a query to } \tau_j\right\} .$$

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

Definition 9 (Augmented GL-PoSW [AFGK22]). *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 $(\mathsf{P} := (\mathsf{P}_0, \mathsf{P}_1), \mathsf{V} := (\mathsf{V}_0, \mathsf{V}_1, \mathsf{V}_2))$,*

[7] In the literature, where Ω_n is not explicitly defined, one can think of it as the uniform distribution of $[n]_0$.

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. 7 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*

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

(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* $\Omega_{\mathrm{seq}}(Q) < \alpha$*, it holds that*

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

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* $\mathsf{poly}(\lambda, \log n)$*. The running time of* P *is* $\mathsf{poly}(\lambda, n)$ *and that of* V *is* $\mathsf{poly}(\lambda, \log n)$*.*

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*

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

where prm *is as sampled by* V_0*,* $\mathsf{out}_{\tilde{\mathsf{P}}_0}$ *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*

$$R\big(i, L(i), p_i\big) = 1 \quad \textit{iff} \quad L(i) = \tau_i(p_i)\|x_i \textit{ for some } x_i \in \{0,1\}^* \ . \tag{17}$$

The connection between SNACKs and PoSW schemes is captured by the following lemma, which will be needed for the proof of the main theorem.

Lemma 3 ([AFGK22]). *Let* Π *be an (interactive)* (α, ϵ)*-knowledge-sound GL-PoSW based on a family of weighted DAGs* Γ *and a commitment scheme* Com*. Then applying the Fiat-Shamir [FS87] transformation* Π *results in an* (α, ϵ)*-knowledge-sound SNACK system for* $\mathcal{R}^{(\alpha)}_{\Gamma,R,\mathsf{Com}}$ *when* R *is defined as in (17).*

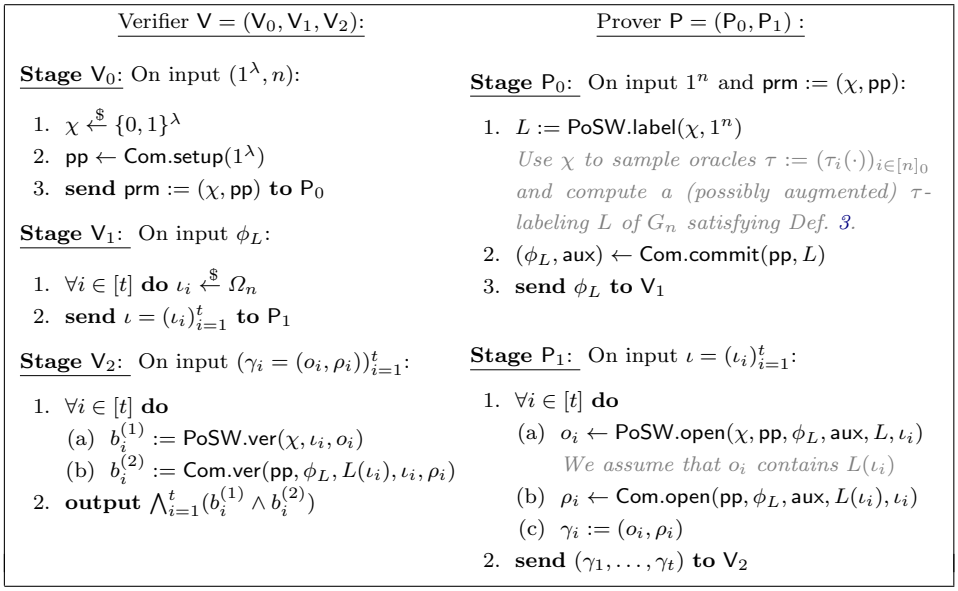

Fig. 7. The template of a GL-PoSW, parametrized by a family of weighted DAGs $(\Gamma_n)_{n \in \mathbb{N}}$, a vector commitment scheme Com and the number of challenges t.

A.2 PoSW-Enabled SNACKs

We review the single-chain PoSW-enabled SNACK of [AFGK22].

Blockchains are viewed as labeled *chain graphs*, where a chain graph $H_n = ([n]_0, E_H)$ is a DAG with edge set $E_H \supseteq \{(i-1, i) : i \in [n]\}$, i.e., the edge set contains the line graph with some additional edges. Hence, we use the terms block and label interchangeably. Furthermore, we assume that a blockchain is equipped with a polynomial-time relation R that checks validity, i.e., on input a label/block and the labels of its parents, R outputs a bit indicating validity of the block. For example, the edge set of the underlying chain graph of fixed-difficulty Bitcoin is simply $\{(i-1, i) : i \in [n]\}$ and R takes the ith label and the label of its parent and output 1 if and only if the proof of work checks. The generality of the notion of a chain graph, which allows for extra edges, is well suited for other blockchains for which the validity of a block may require checking many parent blocks, rather than the immediate parent.

The PoSW-enabled SNACK construction works by first defining an *augmented* chain graph $K_n := ([n]_0, E_G \cup E_H)$ where E_G is the edge sets of the underlying chain graph G_n of a PoSW scheme $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ and E_H is the edge set of the chain graph underlying a blockchain. Then

an augmented *mining* algorithm that labels K_n by infusing PoSW-related data into blockchain labels is defined. The validity of labels in K_n is now an augmented validity relation $\tilde{R}$, which in addition to using R to check blockchain block validity, also checks the consistency of the added PoSW data. The augmentation is carefully done such that running (the Fiat-Shamir-transformed non-interactive counterpart of) $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ over the augmented labeled K_n gives rise to a SNACK system with respect to a chain commitment language $\mathcal{L}_{\Gamma,\tilde{R},\mathsf{Com}}$.

For blockchain augmentation to work, the blockchain protocol needs to allow it, i.e., the blockchain uses an augmented mining protocol. The cost of such augmentation is minimal: the edge structure of the blockchain changes from E_H to E_K and each augmented block contains an extra λ-bit hash value, say a 256 bit value. The benefit of this augmentation is that a SNACK system for the augmented blockchain is as efficient as its underlying PoSW scheme. To put things in perspective, a full node miner holding the augmented blockchain generates a SNACK proof by simply looking up a poly-logarithmic number of blocks and sends them to the verifier, which in turns, simply checks their consistency. An illustration of example graphs is given in Fig. 8.

Now (α, ϵ)-knowledge soundness of a SNACK system for the language $\mathcal{L}_{\Gamma,\tilde{R},\mathsf{Com}}$ guarantees that from any successful prover for a statement (ϕ, n), one can extract a labeled path P in $\Gamma_n := (K_n, \Omega_n)$ that is $\tilde{R}$-valid and of weight α, and as $\tilde{R}$ does PoSW checks, one is also assured that the labels of P are sequentially computed. By construction, (α, ϵ)-knowledge soundness of the SNACK system follows directly from (α, ϵ)-knowledge soundness of the underlying PoSW scheme; for PoSW, (α, ϵ)-knowledge soundness means that from any PoSW successful prover, one can extract a sequentially labeled path P of weight α [AFGK22]. If a graph-labeling PoSW scheme is secure in the random oracle model, so is its PoSW-enabled SNACK counterpart.

Augmented Blockchains. For simplicity, fix an integer n and consider a blockchain with underlying chain graph $H_n = ([n]_0, E_H)$, genesis block $h_0 := \psi$, and an associated validity relation R_ψ, where a label h_i is valid if and only if

$$R_\psi(i, h_i, (h_{i_1}, \ldots, h_{i_p})) = 1 \tag{18}$$

where $h_{i_1}, \ldots, h_{i_p}$ are the labels of the parents of i.

Furthermore, let $\Gamma_n = (G_n = ([n]_0, E_G), \Omega_n)$ be the underlying weighted chain graph a (GL)PoSW scheme $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ adhering to Fig. 7.

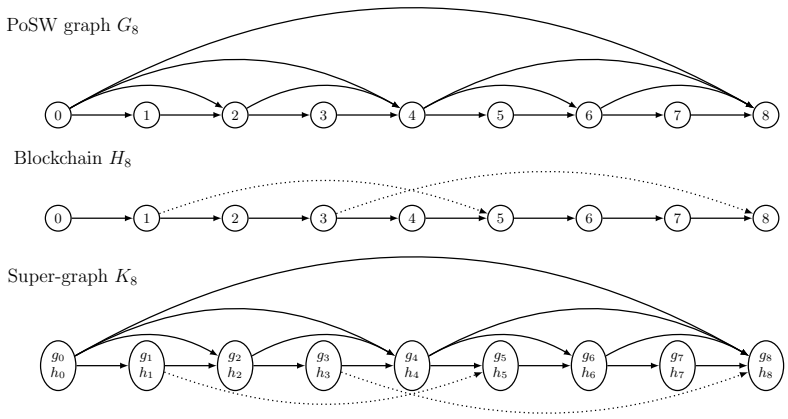

Fig. 8. Example graphs H_8, G_8, K_8, where G_8 is the chain graph underlying the PoSW scheme of [AFGK22]. The labeling $(k_i)_{i\in[8]_0}$ of K_8 is computed such that $k_i = (g_i, h_i)$ where g_i is PoSW-related and essentially contains a hash of all parent labels in K_8, and h_i is a standard blockchain block, where additionally the publicly verifiable proof in it, say PoW in Bitcoin, must depend on all parent labels in K_8 as well as g_i. The augmented blockchain is now the labeled K_8 and therefore contains in its ith block, compared to the initial blockchain, an additional PoSW-related g_i, which is typically small, say 256 bits. At this price of augmentation, one can get SNACKs as efficient as PoSW schemes.

Recall that the PoSW computation mainly involves computing labels of vertices by using an oracle $\tau := (\tau_i(\cdot))_{i\in[n]_0}$.

The *augmented* chain graph K_n is defined as

$$K_n := ([n]_0, E_K) \quad \text{with} \quad E_K := E_G \cup E_H \ . \tag{19}$$

The augmented blockchain is simply the labeled K_n where the labeling is done according to algorithms Init and Mine of Fig. 9.

In augmented blockchains, the validity relation R_ψ is augmented to $\tilde{R}_\psi$ which still considers the same graph structure H but takes *augmented* labels as input, and hence (18) becomes:

$$\tilde{R}_\psi\big(i, L_K(i), L_K(\mathsf{parents}_H(i))\big) = 1 \ . \tag{20}$$

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

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

Algorithm Init: On input 1^λ and ψ:[8]

1. $\chi \xleftarrow{\$} \{0,1\}^\lambda$
2. $\ell_0 := \tau_0(\varepsilon)$
3. $\mathsf{pp} \leftarrow \mathsf{Com.setup}(1^\lambda)$
4. $(\phi_0, \mathsf{aux}_0) \leftarrow \mathsf{Com.commit}(\mathsf{pp}, \ell_0)$
5. $g_0 := (\ell_0, \phi_0)$
6. $h_0 := (0, d_0 := \psi \| \chi \| \mathsf{pp}, \pi_0 := \varepsilon)$
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. $\ell_i := \tau_i(L_K(\mathsf{parents}_K(i)))$
2. $(\phi_i, \mathsf{aux}_i) \leftarrow \mathsf{Com.commit}(\mathsf{pp}, (k_j)_{j \in [i-1]_0} \| \ell_i)$
3. $g_i := (\ell_i, \phi_i)$
4. Compute π_i s.t. $\tilde{R}_\psi\big(i, (g_i, h_i := (i, d_i, \pi_i)), L_K(\mathsf{parents}_H(i))\big) = 1$
5. **return** $(L_K(i) := k_i := (g_i, h_i), \mathsf{aux}_i)$

Fig. 9. The mining algorithm Mine for augmented blockchains.

data ia defined in (17) is defined as

$$R_\sigma(i, L_K(i), L_K(\mathsf{parents}_K(i))) = 1 \Leftrightarrow \tag{21}$$
$$\tilde{R}_\psi(i, L_K(i), L_K(\mathsf{parents}_H(i))) = 1 \ \wedge \ \exists\, x_i \text{ s.t. } L_K(i) = \tau_i(L_K(\mathsf{parents}_K(i))) \| x_i \ .$$

In order to verify that $L_K(0)$ contains the blockchain genesis block ψ on which R_σ depends, $\iota_0 = 0$ is always included as a challenge in our protocols, see Fig. 10.

Arguments of Knowledge for Augmented Blockchains. The SNACK system $\Pi = (\mathsf{SNACK.P}, \mathsf{SNACK.V})$ for the language $\mathcal{L}_{\Gamma, R_\sigma, \mathsf{Com}}$ as in Def. 5 with R_σ being as in (21) is constructed by first constructing 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. 10. The SNACK parameter generator G outputs $\mathsf{prm} := \sigma$ that defines R_σ. Π is then obtained by a applying the Fiat-Shamir transformation [FS87] to $(\mathsf{ACK.P}, \mathsf{ACK.V})$ from Fig. 10.

Theorem 2 ([AFGK22]). *Π is an (α, ϵ)-knowledge-sound SNACK for $\mathcal{L}_{\Gamma, R_\sigma, \mathsf{Com}}$ if $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ is an (α, ϵ)-knowledge sound PoSW 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 is modeled as a random oracle.*

B Complementary Material

B.1 On the canonical nature of proof chains of PoSpace blockchains

In this exposition, we only give an intuition on why this is so and refer the curious reader to the respective PoSpace blockchain protocols.

In a PoSpace blockchain, given a proof chain of length i, each PoSpace farmer can propose at most one valid proof label c_{i+1} for the next block, which either gets adopted or abandoned by the network of miners. This is in contrast to PoW blockchains, where a miner, given a blockchain of length i, can in principle generate infinitely many valid next blocks by tweaking transaction data and using different PoW nonces. As the mining resource in a PoW blockchain is exactly this PoW computation and a miner's success in appending the next valid block to the chain needs to be proportional to its mining resource, this process of generating many PoWs is the *honest* mining strategy. However, in PoSpace blockchains, the mining resource is *space*, and therefore, being able to generate more than one valid PoSpace value δ_{i+1} for the next block for a single unit of space is a *malicious* mining strategy that must be prevented. To make sure that a space miner/farmer when given a proof chain of length i can only propose at most a single next block, it must be ensured that each PoSpace farmer for each space unit gets a single PoSpace challenge chal_i *and* that all used primitives such as the PoSpace scheme itself, signatures (if used in the proof chain), VDFs, etc. are *unique*. This ensures that a miner can generate and propose at most a single block for each space unit per index i.

B.2 Simple SNACK Approaches that Fail

In this section, we explore a few (insecure) natural approaches that resemble the PoSW-enabled SNACK construction for single-chain blockchains,

Verifier $\mathsf{ACK.V} = (\mathsf{V}_1, \mathsf{V}_2)$

Stage V_1: On input η:

1. $\forall i \in [t]$ **do** $\iota_i \xleftarrow{\$} \Omega_n$
2. $\iota_0 := 0$
3. **send** $\iota := (\iota_i)_{i\in[t]_0}$ **to** P

Stage V_2: On input $\gamma = \big(o_i, \rho_i\big)_{i\in[t]_0}$:

1. $\forall i \in [t]_0$ **do:**
   (a) $b_i^{(1)} := \mathsf{PoSW.ver}_K(\chi, \iota_i, o_i)$
   (b) $b_i^{(2)} := R_\sigma(\iota_i, L_K(\iota_i), p_i)$
   (c) $b_i^{(3)} := \mathsf{Com.ver}(\mathsf{pp}, \phi, L_K(\iota_i), \iota_i, \rho_i)$
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 ι:

1. **parse** η **as** $\eta = (\mathsf{pp}, \phi, n)$
2. $\forall i \in [t]_0$ **do:**
   (a) $o_i \leftarrow \mathsf{PoSW.open}_K($
   $\chi, \mathsf{pp}, \phi, \mathsf{aux}_n, (L_K(j))_{j\in[n]_0}, \iota_i)$
   We assume o_i contains both $L_K(\iota_i)$ and $p_i := L_K(\mathsf{parents}_K(\iota_i))$
   (b) $\rho_i \leftarrow \mathsf{Com.open}(\mathsf{pp}, \phi, \mathsf{aux}_n, L_K(\iota_i), \iota_i)$
   (c) $\gamma_i := (o_i, \rho_i)$
3. **send** $\gamma := (\gamma_i)_{i\in[t]_0}$ **to** V_2

Fig. 10. The interactive proof system ACK which underlies the SNACK construction.

Algorithm $\mathsf{PoSW.ver}_K$:
On input (χ, ι_i, o_i):

1. Run $b_i := \mathsf{PoSW.ver}(\chi, \iota_i, o_i)$ modified as follows: whenever it queries $\tau_j(L_K(\mathsf{parents}_G(j)))$ for some j, issue query $\tau_j(L_K(\mathsf{parents}_K(j)))$ instead. *(Missing labels are provided in $o_{i,2}$.)*
2. **return** b_i

Algorithm $\mathsf{PoSW.open}_K$:
On input $(\chi, \mathsf{pp}, \phi_n, \mathsf{aux}_n, L_K, \iota_i)$:

1. $o_{i,1} \leftarrow \mathsf{PoSW.open}(\chi, \mathsf{pp}, \phi_n, \mathsf{aux}_n, L_K, \iota_i)$ *(*$\mathsf{PoSW.open}$ *acts based on edges E_G.)*
2. $\mathcal{J} := \{j \in [n]_0 \colon L_K(\mathsf{parents}_G(j))$ which appear in $o_{i,1}\}$
3. $o_{i,2} := \{(j, L_K(\mathsf{parents}_H(j)))\}_{j\in\mathcal{J}}$
4. **return** $o_i := (o_{i,1}, o_{i,2})$

Fig. 11. $\mathsf{PoSW.open}_K$ and $\mathsf{PoSW.ver}_K$ defined based on $\mathsf{PoSW.open}$ and $\mathsf{PoSW.ver}$.

as considered in [AFGK22] and recalled in Sect. A.2. The formal SNACK construction for PoSpace blockchains is given in Sect. 3.3.

A Single Merged SNACK. One simple idea towards constructing SNACKs for a PoSpace blockchain $B_n = (C_n, D_n)$ is to view a PoSpace blockchain as a *single* chain with an underlying chain graph U_n, which is the union of both C_n and D_n, i.e., $U_n := ([n]_0, E_U := E_C \cup E_D)$, and then apply the generic SNACK construction with input chain graph U_n. In this case, the augmented chain graph is $K_n := ([n]_0, E_K := E_U \cup E_G)$ and the ith augmented label is $k_i := (g_i, u_i)$ where g_i is defined by the underlying PoSW scheme and $u_i := (c_i, d_i)$ contains the actual content of the block including the proof and data blocks, c_i and d_i. As required by the generic SNACK construction, u_i must depend on g_i. A pictorial illustrative example of this approach is depicted in Fig. 1. Then, the PoSpace SNACK is identical to the generic SNACK construction for single-chain blockchains.

This simple approach doesn't work for PoSpace blockchains. The reason is that the PoSpace blockchain itself would no longer be secure as the canonical nature of the proof chain would no longer be maintained: as c_i would now depend on $(u_{i_1}, \ldots, u_{i_p})$ for $(i_1, \ldots, i_p) := \mathsf{parents}_K(i)$ and each u_{i_j} contains data_{i_j}, and hence c_i is clearly grindable, and therefore, the canonical nature of the proof chain would be lost.

SNACK Once, Check Twice. To try to salvage the above simple approach, one could apply a generic SNACK construction to U_n as above, and simultaneously maintain the canonical nature of C_n. This is done by defining, as above, the augmented label $k_i := (g_i, u_i)$ and the label $u_i := (c_i, d_i)$ such that c_i is now *independent* of all $(d_j, g_j)_{j\in[i-1]_0}$, while d_i is computed according to the generic SNACK construction on U_n, and

hence, depends on $k_{i_1}, \ldots, k_{i_p}$ where $(i_1, \ldots, i_p) := \mathsf{parents}_K(i)$. This can be thought of as a SNACK for the data chain. If on top of that, whenever in the SNACK verification, a data block d_i is checked for validity, both the validity of the corresponding proof block c_i, as well as the validity of the signature s_i contained in d_i are checked, then we are guaranteed that for any data block d_i on an extracted path, its corresponding c_i is valid and that these two blocks are correctly bound by the signature s_i.

The problem with this approach is that the sequentiality guarantees implied by the SNACK on the data chain don't translate into sequentiality guarantees on the proof chain. To see this, assume we have an (α, ϵ)-knowledge-sound SNACK and let $(d_{i_1}, \ldots, d_{i_m})$ be the data blocks on an extracted (α, R)-valid path, where R is the relation that validates the augmented chain – see Def. 6. By the SNACK guarantees, this implies that $(d_{i_1}, \ldots, d_{i_m})$ must have been computed sequentially. Thanks to the additional checks we made to the SNACK, the corresponding $(c_{i_1}, \ldots, c_{i_m})$ as well as signatures $(s_{i_1}, \ldots, s_{i_m})$ are valid. This however doesn't guarantee that $(c_{i_1}, \ldots, c_{i_m})$ were computed sequentially, and this is why these additionally checks are not enough for the SNACK security.[9]

(For completeness we remark that, the SNACK definition, Def. 6, implies that an (α, R)-valid path can be extracted, but this doesn't directly mean that the path must have been sequentially computed. However, the (PoSW) relation R that we use encompasses verifying PoSW computation, and hence, guarantees sequentially of the labeled path. For more on these sequentiality guarantees, see [AFGK22].)

This simply shows that having a SNACK on the data chain would not guarantee sequentiality on the proof chain, and by symmetry, having a SNACK on the proof chain would not guarantee sequentiality on the data chain. This problem persists simply because validity of labels in C_n, respectively D_n, does not guarantee their sequentiality even if their respective labels in D_n, respectively C_n, are sequentially computed.

[9] For example, consider the chain graphs in Fig. 1, and let $P = (0, 2, \ldots, 8)$ be the index path of an extracted labeled path. By the SNACK guarantees, its corresponding $(d_0, d_2, \ldots, d_8)$ data blocks must have been computed sequentially. The additional checks guarantee that the corresponding $(c_0, c_2, \ldots, c_8)$ proof blocks as well as signatures $(s_0, s_2, \ldots, s_8)$ are valid. This however doesn't guarantee that $(c_0, c_2, \ldots, c_8)$ were computed sequentially: in fact, as the edge $(0, 2) \notin E_C$, there is no guarantee that c_2 was computed after c_0, even if c_0, c_2 are valid and $(0, 2) \in E_D$ as is the case in this example.

C Omitted Proofs

Proof (of Theorem 1). The proof strategy is simple: we use $(\mathsf{ACK.P}, \mathsf{ACK.V})$ and Alg. SMine from Fig. 4 to build an augmented PoSW $\bar{\Pi} := (\overline{\mathsf{PoSW}}.\mathsf{P}, \overline{\mathsf{PoSW}}.\mathsf{V})$ whose knowledge-soundness implies that of $(\mathsf{ACK.P}, \mathsf{ACK.V})$. 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, in Fig. 12 we define an augmented PoSW $\bar{\Pi}$ whose

1. labeling is X-augmented τ-labeling, for an X that we specify later,
2. underlying weighted graph family is $(K_n^d, \Omega_n)_{n\in\mathbb{N}}$ where K_n^d is defined in (9),
3. witness relation $\mathcal{R}^{(1)}_{\Gamma_n^d, R_\sigma, \mathsf{Com}}$ as in Def. 7 is defined with respect to a relation R_σ as defined in (15).

We will then show that $\bar{\Pi}$ is (α, ϵ)-knowledge sound if $\Pi := (\mathsf{PoSW.P}, \mathsf{PoSW.V})$ is (α, ϵ)-knowledge sound, i.e., we need to argue that the security guarantees of Π are preserved under the modifications in the above Items 1, 2, and 3.

As for Items 1 and 2, note that the X-augmented labeling L_{K^d} differs from the labeling L_G of $(\mathsf{PoSW.P}, \mathsf{PoSW.V})$ in that the $\mathsf{parents}(\cdot)$ function used in the τ-labeling is now w.r.t. graph K_n^d rather than $G_n = ([n]_0, E_G)$. Recall that $K_n^d = ([n]_0, E_{K^d})$ has the same vertex set $[n]_0$ of G_n but differs in that its edge set is expanded as $E_{K^d} = E_G \cup E_D \cup E_C$ where E_D and E_C are the edge sets of D_n and C_n, respectively. However, when τ is modeled as a random oracle, it is easy to see that if Π is a τ-based (α, ϵ)-knowledge-sound PoSW for the weighted graph family $(G_n, \Omega_n)_{n\in\mathbb{N}}$, then $\bar{\Pi}$ is an X-augmented τ-based (α, ϵ)-knowledge-sound PoSW for the weighted graph family $(K_n^d, \Omega_n)_{n\in\mathbb{N}}$ and $X = (x_0, \ldots, x_n)$, where x_i can be any bitstring, in our case $x_i = (\phi_i^d, s_i, \mathsf{data}_i)$. Note that $k_i^d = (\ell_i^d, \phi_i^d, s_i, \mathsf{data}_i)$ as computed by SMine (and SInit). 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 input x_i simply defines an x_i-salted random oracle, and crucially (2) the new edge structure E_{K^d} 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 Π, and

– the edge structure E_{K^d}, which in turn suffices to verify the validity of the augmented blockchain (using R_σ).

As for Item 3, recall that the witness relation for Π is $\mathcal{R}^{(\alpha)}_{\Gamma,R,\mathsf{Com}}$ as in Def. 5 and R as in (17), while the witness relation for $\bar{\Pi}$ is $\mathcal{R}^{(\alpha)}_{\Gamma_n^d,R_\sigma,\mathsf{Com}}$ as in Def. 7 and R_σ is defined in (15). This imposes the extra checks in $\mathsf{ACK.V}_2$ of Fig. 6: Lines 1a, 1b, 1c, and 1d. However, these extra checks don't affect soundness: if a protocol Π is sound when the verifier executes a check C_1, then it is clearly sound if the verifier executes an additional check C_2 and accepts if and only if both checks pass. However, Π is not guaranteed to remain complete due to the extra checks: clearly the verifier would reject a proof π that verifies with respect to C_1 but fails with respect C_2. 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 extra these extra checks: in fact, $\eta, (L_{K^d}(j))_{j\in[n]_0}, \mathsf{aux}^c_n, \mathsf{aux}^d_n$ provided as input to $\overline{\mathsf{PoSW}}.\mathsf{P}$ by the output of (the honest) SMine ensure this: to show this, we show that Line 2 of Fig. 6 holds, i.e., that we can parse $(k^c_j)_{j\in[n]_0}$ out of $(k^d_j)_{j\in[n]_0}$: it suffices to observe that, by construction (see SMine and Sinit), $\forall j \in [n]_0$, $L_{K^d}(j)$ contains $L_{K^c}(j)$, and that, by construction of (9), $E_{K^c} \subseteq E_{K^d}$. This allows $\overline{\mathsf{PoSW}}.\mathsf{P}$ via invoking $\mathsf{ACK.P}$ to run all subroutines corresponding to the extra checks.

This establishes that $\bar{\Pi}$ is an (α,ϵ)-knowledge-sound X-augmented τ-based (with labeling L_{K^d}) PoSW for the weighted graph family $(K^d_n, \Omega_n)_{n\in\mathbb{N}}$. 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 an augmented PoSW.[10]

Now by Lemma 3 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 $\mathcal{L}_{\Gamma^d,R_\sigma,\mathsf{Com}}$ as in Def. 7. □

Proof (of Lemma 1). The SNACK construction starts with a PoSW scheme Π with an underlying weighted DAG is (G_n, Ω_n) and validity relation is R,[11] and augments it into X-augmented PoSW $\bar{\Pi}$ whose underlying weighted DAG is (K^d_n, Ω_n) where K^d_n is as in (9) and validity relation

[10] To justify this syntactic manipulation, note that $\overline{\mathsf{PoSW}}.\mathsf{V}_0$ outputs σ which contains χ and pp 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.

[11] This R is the PoSW relation in (17).

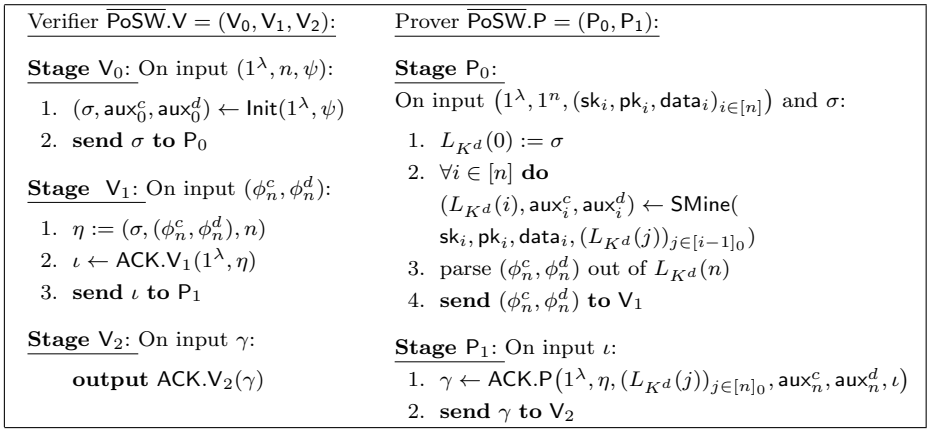

Fig. 12. The augmented PoSW scheme constructed from mining algorithms and an ACK system $\mathsf{ACK.V} = (\mathsf{ACK.V}_1, \mathsf{ACK.V}_2), \mathsf{ACK.P}$.

R_σ. Now the (α, ϵ)-knowledge soundness of Π implies that from any convincing prover an (α, R)-valid path in G_n can be extracted. Similarly for $\bar{\Pi}$, i.e., its (α, ϵ)-knowledge soundness implies that from any convincing prover an (α, R_σ)-valid path in K_n^d can be extracted. However, by construction, $\bar{\Pi}$ runs (see Fig. 6) the augmented PoSW procedures $\mathsf{PoSW.open}_{K^d}, \mathsf{PoSW.ver}_{K^d}, \mathsf{PoSW.open}_{K^c}, \mathsf{PoSW.ver}_{K^c}$.[12] Although these procedures consider augmented labelings L_{K^d} and L_{K^c}, they still, and crucially so,[13] consider the edge structure E_G of G_n, and not E_{K^c} or E_{K^d}. Therefore, the extracted path must still lie in G_n even though its labels are augmented. This dependency on E_G is highlighted in Fig. 5, which is invoked by these augmented PoSW algorithms. This implies that the extracted $(\alpha, \tilde{R}_\sigma^d)$-valid path in K_n^d must actually lie G_n. □

Proof (of Lemma 2). We need to show that an (α, R_σ)-valid path in Γ_n^d contains an $(\alpha, \tilde{R}_\sigma^c)$-valid path in Γ_n^c, where R_σ and $\tilde{R}_\sigma^c$ are defined in (15) and (13), respectively. By design, $L^d(i)$ contains $L^c(i)$ as a substring – see Fig. 4. By (2), it holds that $p^d(i) = L_P^d(i)$ and therefore $p^c(i)$ in contained in $p^d(i)$ as a substring. This, together with the fact that R_σ internally checks $\tilde{R}_\sigma^c$, implies that $(P, L_P^c, (p_i^c)_{i\in P})$ is an $(\alpha, \tilde{R}_\sigma^c)$-valid in Γ_n^c. Furthermore, by (16), it holds that $\mathsf{Com.ver}(\mathsf{pp}, \phi^c, L_P^c, P, \rho^c) = 1$,

[12] Note that these augmented PoSW algorithms work identically to their underlying $\mathsf{PoSW.open}, \mathsf{PoSW.ver}$ from Π, except that the labels are augmented and some extra parent labels need to be provided by $\mathsf{PoSW.open}_{K^d}$ and $\mathsf{PoSW.open}_{K^c}$ for R_σ^c and R_σ^d to be checked – and these extra parents are defined by K^d and K^c.

[13] See the proof of Theorem 1.

where pp is contained in σ. Consequently, $(\sigma, (\phi^c, n), w) \in \mathcal{R}^{(\alpha)}_{\Gamma^c_n, \tilde{R}^c_\sigma, \mathsf{Com}}$. $\square$