CAPSS: A Framework for SNARK-Friendly Post-Quantum Signatures

Thibauld Feneuil and Matthieu Rivain

CryptoExperts, Paris, France
thibauld.feneuil@cryptoexperts.com
matthieu.rivain@cryptoexperts.com

Abstract. In this paper, we present a general framework for constructing SNARK-friendly post-quantum signature schemes based on minimal assumptions, specifically the security of an arithmetization-oriented family of permutations. The term "SNARK-friendly" here refers to the efficiency of the signature verification process in terms of SNARK constraints, such as R1CS or AIR constraints used in STARKs. Within the CAPSS framework, signature schemes are designed as proofs of knowledge of a secret preimage of a one-way function, where the one-way function is derived from the chosen permutation family. To obtain compact signatures with SNARK-friendly verification, our primary goal is to achieve a hash-based proof system that is efficient in both proof size and arithmetization of the verification process.

To this end, we introduce SmallWood, a hash-based polynomial commitment and zero-knowledge argument scheme tailored for statements arising in this context. The SmallWood construction leverages techniques from Ligero, Brakedown, and Threshold-Computation-in-the-Head (TCitH) to achieve proof sizes that outperform the state of the art of hash-based zero-knowledge proof systems for witness sizes ranging from 2^5 to 2^{16}.

From the SmallWood proof system and further optimizations for SNARK-friendliness, the CAPSS framework offers a generic transformation of any arithmetization-oriented permutation family into a SNARK-friendly post-quantum signature scheme. We provide concrete instances built on permutations such as Rescue-Prime, Poseidon, Griffin, and Anemoi. For the Anemoi family, achieving 128-bit security, our approach produces signatures of sizes ranging from 9 to 13.3 KB, with R1CS constraints between 19 K and 29 K. This represents a 4–6× reduction in signature size and a 5–8× reduction in R1CS constraints compared to Loquat (CRYPTO 2024), a SNARK-friendly post-quantum signature scheme based on the Legendre PRF.

1 Introduction

The advent of a quantum computer capable of breaking classical public-key cryptosystems (RSA, ECC) urges the cryptography research community to find reliable alternatives to common cryptosystems. One of the most ubiquitous cryptographic primitives that will need to be replaced is digital signature. While some post-quantum signature schemes have been proposed for general-purpose application, such as TLS communications, they might not be tailored to certain use-cases with specific constraints. One such area that requires specialized design is SNARK-friendly signatures. A SNARK (Succinct Non-Interactive Argument of Knowledge) is a type of cryptographic proof that enables a party to succinctly demonstrate the correctness of a potentially large computation, without the verifier needing to re-execute the entire computation themselves. For a digital signature to be SNARK-friendly, its verification process must be tweaked in a way to make it efficiently wrappable into a SNARK proof.

SNARK-friendly signatures have numerous potential applications, such as blind signatures and anonymous credentials, where zero-knowledge proofs for signatures are paramount [Cha82, Fis06, CL01]. Another particularly promising use case is aggregate signatures [BGLS03] –a cryptographic technique that combines multiple individual signatures into a single, compact signature for more efficient storage, transmission, and verification. In blockchains, aggregate signatures play a crucial role in enhancing scalability by allowing more transactions to fit into each block, optimizing bandwidth and reducing data overhead. They also speed up

verification by enabling a single operation to validate multiple signatures at once, saving computational resources.

In this paper, we present a general framework for constructing SNARK-friendly post-quantum signature schemes based on a minimal assumption: the security of an arithmetization-oriented permutation modeled as an ideal permutation. SNARK-friendliness refers to the signature verification process being efficient in terms of SNARK constraints, such as R1CS or AIR constraints used in STARKs. This property enables efficient and generic aggregation schemes using post-quantum SNARKs, such as Aurora [BCR+19] or STARK [BBHR19].

The design of a SNARK-friendly post-quantum signature scheme was recently explored in [ZSE+24]. In this work, the authors introduced Loquat, a post-quantum signature scheme built on the Legendre PRF, explicitly tailored for SNARK compatibility. Loquat produces signatures of approximately 57 KB, with their verification requiring 149 K R1CS constraints. Another recent study proposed a signature scheme leveraging a STARK-based proof system applied to the Rescue-Prime permutation [AdSGK24a] though it does not specifically target SNARK-friendliness. This scheme achieves signatures of either 80 KB or 100 KB, depending on the decoding regime used in the underlying STARK proof.

In the present work, to achieve SNARK-friendly signatures, we leverage hash-based proof systems with verification processes that are amenable to arithmetization. Specifically, the Merkle-tree variant of the Threshold-Computation-in-the-Head (TCitH) framework [FR23], which borrows and extends techniques from Ligero [AHIV17, AHIV23], is particularly well-suited to our goal. Compared to other MPC-in-the-Head schemes (e.g., TCitH-GGM [FR23], VOLE-in-the-Head [BBD+23a], and earlier MPCitH schemes [KKW18, BDK+21]), the Merkle-tree approach achieves faster verification by involving some (light) arithmetic computation and verifying a small number of Merkle paths. When paired with arithmetization-oriented hash functions for their core Merkle trees, these systems are ideal for SNARK-friendly verification.

Building on this, we adapt techniques from Brakedown [GLS+23] to extend the framework into a full polynomial commitment scheme. While the original framework only supported small-domain polynomial commitments, our approach generalizes it to a broader range which further allows us to build more compact zero-knowledge arguments for arithmetic circuits and "LPPC" statements considered in Ligero and TCitH.[1] The resulting scheme, named SmallWood,[2] achieves the best proof sizes among hash-based zero-knowledge proof systems in the literature for witness sizes ranging from 2^5 to 2^{16}.

The CAPSS framework builds on the SmallWood proof system with additional tweaks for SNARK-friendliness to transform any arithmetization-oriented permutation family into a SNARK-friendly post-quantum signature scheme. Specifically, the signature scheme is constructed by applying SmallWood to prove knowledge of a preimage x for an output of a one-way function $y = f(x)$. Using the Fiat-Shamir transform, we derive a signature scheme where y serves as the public key and x as the secret key. Both the one-way function f and the core hash function used in SmallWood are instantiated from the same permutation family, ensuring that the scheme's security relies solely on the underlying permutation. This approach provides a conservative candidate for post-quantum security.

We provide general arithmetization techniques for arithmetization-oriented permutations in the LPPC syntax, which is ideally suited for this type of statement. Thanks to this, we can achieve signature sizes below 10 KB in a "short signature regime". We provide concrete instances built on arithmetization-oriented permutations such as Rescue-Prime [SAD20, AKM+22], Poseidon [GKR+21, GKS23], Griffin [GHR+23], and Anemoi [BBC+23], carefully selected for their suitability in SNARK-friendly contexts. For the Anemoi family, achieving 128-bit security, our approach produces signature sizes ranging from 9 to 13.3 KB, with the number of R1CS constraints between 19 K and 29 K. These results represent a 4–6× reduction in signature size and a 5–8× reduction in R1CS constraints compared to Loquat [ZSE+24].

Paper organization. After giving some technical preliminaries in Section 2, Section 3 introduces the SmallWood proof system. We first describe the polynomial commitment scheme (SmallWood-PCS) and then

[1] The LPPC terminology stands for *(global) linear and parallel polynomial constraints* which was introduced in [FR23]. See Section 3.4 for a formal definition.

[2] Wood because this scheme is about hash and trees and Small because it targets "small" witnesses compared to schemes such as Ligero [AHIV17], FRI [BBHR18] or Brakedown [GLS+23] which target "larger" witnesses.

zero-knowledge argument scheme for LPPC statements (SmallWood-ARK), and provide some comparison with state-of-the-art hash-based zero-knowledge proof systems. Subsequently, Section 4 formally defines the CAPSS framework. We provide a comprehensive description of the underlying permutation-based cryptographic primitives, the arithmetization of the one-way function statements in LPPC syntax, the tweaks for SNARK-friendly verification, and a detailed description of the resulting signature scheme. The section concludes with the specific instances derived from the four considered families of permutations as well as a comparison to the state of the art.

2 Preliminaries

2.1 Polynomials

Along this paper, we shall call $\boldsymbol{P} = (P_1, \ldots, P_n) \in (\mathbb{F}[X])^n$ a *vector polynomial*. Its degree is defined as $\deg \boldsymbol{P} = (\deg P_1, \ldots, \deg P_n) \in \mathbb{N}^n$ and is said to be lower than or equal to $\boldsymbol{d} = (d_1, \ldots, d_n) \in \mathbb{N}^n$ if $\deg P_i \leq d_i$ for every $i \in [1, n]$. The set of polynomials with coefficient from $\mathbb{F}$ and degree at most d shall be denoted $\mathbb{F}[X]^{(\leq d)}$.

The evaluation of a vector polynomial $\boldsymbol{P}$ in a point $e \in \mathbb{F}$ is defined as the vector $\boldsymbol{P}(e) = (P_1(e), \ldots, P_n(e)) \in \mathbb{F}^n$. For any set $E \subseteq \mathbb{F}$, we denote $\boldsymbol{P}|_E$ the set of evaluations of $\boldsymbol{P}$ over E, that is $\boldsymbol{P}|_E := \{\boldsymbol{P}(e)\}_{e \in E}$. We further denote, $\boldsymbol{P} = \mathsf{Interpol}(E, \boldsymbol{P}|_E)$ the interpolation which returns the lowest-degree vector polynomial $\boldsymbol{P}$ matching $\boldsymbol{P}|_E$.

The *vanishing polynomial* of a set $E \subseteq \mathbb{F}$ is the polynomial $V_E(X) \in \mathbb{F}[X]$ of degree $|E|$ defined as:

$$V_E(X) = \prod_{e \in E} (X - e) \ .$$

2.2 Interactive Commitment Schemes

In this paper, we should consider transparent interactive commitment schemes. The transparency feature means that no (trusted) setup is required by those schemes. The scheme parameters which vary according to the security level are left implicit in the exposition. The interactive feature means that those schemes are composed of *public-coin verifier* (PCV) interactive protocol, namely interactive protocols between a stateful (PPT) prover and a stateful public-coin (PPT) verifier. In such a protocol Π, the prover $\Pi.\mathcal{P}$ and the verifier $\Pi.\mathcal{V}$ exchange messages. The protocol might terminate with either $\Pi.\mathcal{P}$ or $\Pi.\mathcal{V}$ producing the final message. All the messages from $\Pi.\mathcal{V}$ are fresh uniform random values (over some space depending on the definition of the protocol) except for the final message of the protocol which is a string from $\{\textsc{Accept}, \textsc{Reject}\}$ (when coming from $\Pi.\mathcal{V}$). We denote

$$(out_{\mathcal{P}}, out_{\mathcal{V}}, \pi) \leftarrow \Pi(in_{\mathcal{P}}, in_{\mathcal{V}})$$

for running Π with $in_{\mathcal{P}}$ and $in_{\mathcal{V}}$ as input of $\Pi.\mathcal{P}$ and $\Pi.\mathcal{V}$ respectively, getting $out_{\mathcal{P}}$ and $out_{\mathcal{V}}$ as output of $\Pi.\mathcal{P}$ and $\Pi.\mathcal{V}$ respectively, and π as protocol transcript.

We now introduce the notion of Polynomial Commitment Scheme (PCS) and Linear-Map Vector Commitment Scheme (LVCS) that we use in our work. While the notion of PCS is well-established, the notion of LVCS was recently introduced in [CNR+22]. The definition provided below adapts the formalization from that work to a transparent and interactive setting.

Definition 1 (Polynomial Commitment Scheme). *Let $n \in \mathbb{N}$ and $\boldsymbol{d} = (d_1, \ldots, d_n) \in \mathbb{N}^n$. Let $\mathbb{F}$ a finite field. A* polynomial commitment scheme *with parameters $(\mathbb{F}, n, \boldsymbol{d})$ is a pair of PCV protocols* $(\mathsf{Commit}, \mathsf{Eval})$*:*

- Commit*: In this protocol,* $\mathsf{Commit}.\mathcal{P}$ *takes a vector polynomial $\boldsymbol{P} \in \mathbb{F}[X]^{(\leq d_1)} \times \cdots \times \mathbb{F}[X]^{(\leq d_n)}$ as input and produces an opening key* key *as output while* $\mathsf{Commit}.\mathcal{V}$ *takes no input and produces no output. The transcript of the protocol is the commitment denoted* com*. We abuse notations and denote*

$$(\mathsf{com}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{P})$$

to mean $(\mathsf{key}, \perp, \mathsf{com}) := (out_{\mathcal{P}}, out_{\mathcal{V}}, \pi) \leftarrow \mathsf{Commit}(in_{\mathcal{P}} := \boldsymbol{P}, in_{\mathcal{V}} := \perp)$.

– Eval*: In this protocol,* $\mathsf{Eval}.\mathcal{P}$ *takes an opening key* key *as input and* $\mathsf{Eval}.\mathcal{V}$ *takes a commitment* com *as input. Both parties additionally take a set of evaluation points* $E \subseteq \mathbb{F}$ *and a set of evaluations* $\boldsymbol{P}|_E := \{\boldsymbol{P}(e)\}_{e\in E}$*. The protocol ends with the verifier returning* $out_{\mathcal{V}} \in \{\textsc{Accept}, \textsc{Reject}\}$*. We abuse notations and denote*

$$(out_{\mathcal{V}}, \pi) \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}, E, \boldsymbol{P}|_E)$$

to mean $(\perp, out_{\mathcal{V}}, \pi) \leftarrow \mathsf{Eval}\big(in_{\mathcal{P}} := (\mathsf{key}, E, \boldsymbol{P}|_E), in_{\mathcal{V}} := (\mathsf{com}, E, \boldsymbol{P}|_E)\big)$*, sometimes skipping* π *from the output if only* $out_{\mathcal{V}}$ *is relevant.*

The scheme is

– correct: *for any vector polynomial* $\boldsymbol{P} = (P_1, \ldots, P_n) \in \mathbb{F}[X]^{(\leq d_1)} \times \cdots \times \mathbb{F}[X]^{(\leq d_n)}$ *and any set of evaluation points* $E \subseteq \mathbb{F}$*, we have:*

$$\Pr\left[out = \textsc{Accept} \;\middle|\; \begin{array}{l} (\mathsf{com}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{P}) \\ out \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}, E, \boldsymbol{P}|_E) \end{array}\right] = 1 \ ,$$

where $\boldsymbol{P}|_E := \{\boldsymbol{P}(e)\}_{e\in E}$*.*

– (t, ε)-polynomial-binding: *for any stateful adversary* $\mathcal{A}$ *running in time* t*, playing the role of the prover in the above protocols, and for any integer* $K \in \mathbb{N}$*, we have:*

$$\Pr\left[\begin{array}{l} out_1 = \cdots = out_K = \textsc{Accept} \ \wedge \\ \quad \nexists\, \boldsymbol{P} \text{ with } \deg \boldsymbol{P} \leq \boldsymbol{d} \text{ s.t.} \\ \boldsymbol{P}(e) = \boldsymbol{p}_e^{(k)} \ \forall e \in E^{(k)}, \ \forall k \in [1, K] \end{array} \;\middle|\; \begin{array}{l} (\mathsf{com}, *) \leftarrow \mathsf{Commit}^{\mathcal{A}}(*) \\ \big(E^{(k)}, \{\boldsymbol{p}_e^{(k)}\}_{e\in E^{(k)}}\big)_{k\in[1,K]} \leftarrow \mathcal{A}() \\ \forall k \in [1, K] : \ out_k \leftarrow \mathsf{Eval}^{\mathcal{A}}\big(*, \mathsf{com}, E^{(k)}, \{\boldsymbol{p}_e^{(k)}\}_{e\in E^{(k)}}\big) \end{array}\right] \leq \varepsilon \ ,$$

where $*$ *denote arbitrary inputs/outputs of the adversary which do not affect the definition. The above probability is over the randomness of* $\mathcal{A}$*,* $\mathsf{Commit}.\mathcal{V}$ *and* $\mathsf{Eval}.\mathcal{V}$*.*

– (t, ξ)-honest-verifier zero knowledge ((t, ξ)-HVZK)*: there exists a PPT algorithm* Sim *(the simulator) such that for any stateful adversary* $\mathcal{A}$ *running in time* t*, for any vector polynomial* $\boldsymbol{P} = (P_1, \ldots, P_n) \in \mathbb{F}[X]^{(\leq d_1)} \times \cdots \times \mathbb{F}[X]^{(\leq d_n)}$ *and for any set of evaluation points* $E \subseteq \mathbb{F}$*, we have:*

$$\Pr\left[\hat{b} = b \;\middle|\; \begin{array}{l} (\mathsf{com}^{(0)}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{P}) \\ (out, \pi^{(0)}) \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}^{(0)}, E, \boldsymbol{P}|_E) \\ (\mathsf{com}^{(1)}, \pi^{(1)}) \leftarrow \mathsf{Sim}(E, \boldsymbol{P}|_E) \\ b \leftarrow \{0, 1\} \\ \hat{b} \leftarrow \mathcal{A}(\mathsf{com}^{(b)}, \pi^{(b)}) \end{array}\right] \leq \frac{1}{2} + \xi \ .$$

Definition 2 (Linear-Map Vector Commitment Scheme). *Let* n_{rows}, n_{cols} *be some integers. Let* $\mathbb{F}$ *a finite field. A* linear-map vector commitment scheme *with parameters* $(\mathbb{F}, n_{rows}, n_{cols})$ *is a pair of PCV protocols* $(\mathsf{Commit}, \mathsf{Eval})$*:*

– Commit*: In this protocol,* $\mathsf{Commit}.\mathcal{P}$ *takes* n_{rows} *vectors* $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{rows}} \in \mathbb{F}^{n_{cols}}$ *as input and produces an opening key* key *as output while* $\mathsf{Commit}.\mathcal{V}$ *takes no input and produces no output. The transcript of the protocol is the commitment denoted* com*. We abuse notations and denote*

$$(\mathsf{com}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{rows}})$$

to mean $(\mathsf{key}, \perp, \mathsf{com}) := (out_{\mathcal{P}}, out_{\mathcal{V}}, \pi) \leftarrow \mathsf{Commit}(in_{\mathcal{P}} := (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{rows}}), in_{\mathcal{V}} := \perp)$*.*

– Eval*: In this protocol,* $\mathsf{Eval}.\mathcal{P}$ *takes an opening key* key *as input and* $\mathsf{Eval}.\mathcal{V}$ *takes a commitment* com *as input. For some* $m \in \mathbb{N}$*, both parties additionally take a set of coefficients* $\mathcal{C} := \{c_{k,j}\}$ *with* $1 \leq k \leq m$ *and* $1 \leq j \leq n_{rows}$*, and a set of vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m \in \mathbb{F}^{n_{cols}}$*. The protocol ends with the verifier returning* $out_{\mathcal{V}} \in \{\textsc{Accept}, \textsc{Reject}\}$*. We abuse notations and denote*

$$(out_{\mathcal{V}}, \pi) \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}, \mathcal{C}, (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m))$$

to mean $(\perp, out_{\mathcal{V}}, \pi) \leftarrow \mathsf{Eval}\big(in_{\mathcal{P}} := (\mathsf{key}, \mathcal{C}, (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)), in_{\mathcal{V}} := (\mathsf{com}, \mathcal{C}, (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m))\big)$*, sometimes skipping* π *from the output if only* $out_{\mathcal{V}}$ *is relevant.*

The scheme is

- correct: *for any vectors* $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{rows}} \in \mathbb{F}^{n_{cols}}$ *and any set of coefficients* $\mathcal{C} := \{c_{k,j}\}$*, we have:*

$$\Pr\left[out = \textsc{Accept} \;\middle|\; \begin{array}{l} (\mathsf{com}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{rows}}) \\ out \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}, \mathcal{C}, (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)) \end{array}\right] = 1 \ ,$$

  where $\boldsymbol{v}_k := \sum_{j=1}^{n_{rows}} c_{k,j} \cdot \boldsymbol{r}_j$ *for every* $k \in [1, m]$*.*

- (t, ε)-function-binding: *for any stateful adversary* $\mathcal{A}$ *running in time* t*, playing the role of the prover in the above protocols, and for any integer* $I \in \mathbb{N}$*, we have:*

$$\Pr\left[\begin{array}{c} out_1 = \cdots = out_I = \textsc{Accept} \ \wedge \\ \nexists\, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{rows}} \in \mathbb{F}^{n_{cols}} \ s.t. \\ \sum_j c_{k,j}^{(i)} \cdot \boldsymbol{r}_j = \boldsymbol{v}_k^{(i)} \ \forall k \in [1,m], \ \forall i \in [1, I] \end{array} \;\middle|\; \begin{array}{l} (\mathsf{com}, *) \leftarrow \mathsf{Commit}^{\mathcal{A}}(*) \\ \left(\mathcal{C}^{(i)} := \{c_{k,j}^{(i)}\}, (\boldsymbol{v}_1^{(i)}, \ldots, \boldsymbol{v}_m^{(i)})\right)_{i \in [1,I]} \leftarrow \mathcal{A}() \\ \forall i \in [1, I]: \\ \quad out_i \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}, \mathcal{C}^{(i)}, (\boldsymbol{v}_1^{(i)}, \ldots, \boldsymbol{v}_m^{(i)})) \end{array}\right] \leq \ \varepsilon \ ,$$

  where $*$ *denote arbitrary inputs/outputs of the adversary which do not affect the definition. The above probability is over the randomness of* $\mathcal{A}$*,* $\mathsf{Commit}.\mathcal{V}$ *and* $\mathsf{Eval}.\mathcal{V}$*.*

- (t, ξ)-honest-verifier zero knowledge ((t, ξ)-HVZK)*: there exists a PPT algorithm* Sim *(the simulator) such that for any stateful adversary* $\mathcal{A}$ *running in time* t*, for any vectors* $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{rows}} \in \mathbb{F}^{n_{cols}}$ *and any set of coefficients* $\mathcal{C} := \{c_{k,j}\}$ *and corresponding linear combination vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$*, we have:*

$$\Pr\left[\hat{b} = b \;\middle|\; \begin{array}{l} (\mathsf{com}^{(0)}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{rows}}) \\ (out, \pi^{(0)}) \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}^{(0)}, \mathcal{C}, (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)) \\ (\mathsf{com}^{(1)}, \pi^{(1)}) \leftarrow \mathsf{Sim}(\mathcal{C}, (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)) \\ b \leftarrow \{0,1\} \\ \hat{b} \leftarrow \mathcal{A}(\mathsf{com}^{(b)}, \pi^{(b)}) \end{array}\right] \leq \ \frac{1}{2} + \xi \ .$$

3 SmallWood: Merkle-Tree PCS and ZK-ARK from Degree Enforcement

This section introduces new polynomial commitment schemes from Merkle trees. These schemes are reminiscent of techniques from Ligero [AHIV17, AHIV23], STARKs [BBHR18, BBHR19], Brakedown [GLS+23] and Threshold-Computation-in-the-Head [FR23]. In particular, we build upon the recent technique of *degree-enforcing commitment scheme* (DECS) put forward in [FR23]. We formalize this protocol as a (binding and hiding) *small-domain polynomial commitment scheme*. Using techniques from Brakedown [GLS+23], we then build a standard polynomial commitment scheme (SmallWood-PCS) and zero-knowledge arguments (SmallWood-ARK) from the DECS protocol. We show that those schemes improve the proof sizes of state-of-the-art hash-based schemes for polynomials or arithmetic circuits of size $\leq 2^{16}$.

3.1 Degree Enforcing Commitment Scheme

A common approach to defining a *somewhat* polynomial commitment scheme from hash functions is to rely on Merkle trees. The idea is to use a Merkle tree-based vector commitment to commit to the evaluations of a (vector) polynomial $\boldsymbol{P}$ on an evaluation domain $\mathbb{E}$. However, such a somewhat PCS has two limitations. First, it does not allow proving the evaluation of $\boldsymbol{P}$ for any point $e \in \mathbb{F}$, but for $e \in \mathbb{E}$ only. Second, a malicious prover could commit to evaluations that do not correspond to a polynomial of the right degree. This second limitation can be addressed by equipping the commitment with a protocol that statistically proves the low degree of the committed polynomial or its proximity to a low-degree polynomial. Several protocols have been proposed to target asymptotic efficiency, such as Ligero [AHIV17, AHIV23] or FRI [BBHR18, BGKS20]. A

more recent work has proposed the notion of *degree-enforcing commitment scheme* (DECS) [FR23] which achieves promising results for the "small to medium size" regime. This scheme guaranties to the verifier that the committed evaluations are of degree at most d.

While the notion of DECS introduced in [FR23] keeps the binding property implicit for the committed evaluations, we note here that combining the notion of degree enforcing from [FR23] together with the binding of committed evaluations yields the notion of polynomial binding as in a standard PCS with main difference that here only evaluations on a small domain $\mathbb{E}$ can be opened. We formalize this notion as a *small-domain PCS* in the next definition.[3] We stress that the notion of small-domain PCS can also be interpreted as a vector commitment scheme with the additional property that the committed values are enforced to correspond to the evaluations of a polynomial of degree d.

Definition 3 (Small-Domain Polynomial Commitment Scheme). *Let $N, n \in \mathbb{N}$, $\boldsymbol{d} = (d_1, \ldots, d_n) \in \mathbb{N}^n$ and $\varepsilon \in [0, 1)$. Let $\mathbb{F}$ a finite field and $\mathbb{E} = \{e_1, \ldots, e_N\} \subseteq \mathbb{F}$. An interactive ε-*small-domain polynomial commitment scheme *(SD-PCS) with parameters $(\mathbb{F}, \mathbb{E}, n, \boldsymbol{d})$ is a a pair of PCV protocols* $(\mathsf{Commit}, \mathsf{Eval})$*:*

- Commit*: (This protocol has the same interface as in a standard PCS –see Definition 1.) In this protocol, $\mathsf{Commit}.\mathcal{P}$ takes a vector polynomial $\boldsymbol{P} \in \mathbb{F}[X]^{(\leq d_1)} \times \cdots \times \mathbb{F}[X]^{(\leq d_n)}$ as input and produces an opening key key as output while $\mathsf{Commit}.\mathcal{V}$ takes no input and produces no output. The transcript of the protocol is the commitment denoted com. We abuse notations and denote*

$$(\mathsf{com}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{P})$$

to mean $(\mathsf{key}, \perp, \mathsf{com}) := (out_{\mathcal{P}}, out_{\mathcal{V}}, \pi) \leftarrow \mathsf{Commit}(in_{\mathcal{P}} := \boldsymbol{P}, in_{\mathcal{V}} := \perp)$.

- Eval*: (This protocol has the same interface as in a standard PCS –see Definition 1– except that the evaluation points must be from $\mathbb{E}$.) In this protocol, $\mathsf{Eval}.\mathcal{P}$ takes an opening key key as input and $\mathsf{Eval}.\mathcal{V}$ takes a commitment com as input. Both parties additionally take a set of evaluation points $E \subseteq \mathbb{E}$ and a set of evaluations $\boldsymbol{P}|_E := \{\boldsymbol{P}(e)\}_{e \in E}$. The protocol ends with the verifier returning $out_{\mathcal{V}} \in \{\textsc{Accept}, \textsc{Reject}\}$. We abuse notations and denote*

$$(out_{\mathcal{V}}, \pi) \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}, E, \boldsymbol{P}|_E)$$

to mean $(\perp, out_{\mathcal{V}}, \pi) \leftarrow \mathsf{Eval}\big(in_{\mathcal{P}} := (\mathsf{key}, E, \boldsymbol{P}|_E), in_{\mathcal{V}} := (\mathsf{com}, E, \boldsymbol{P}|_E)\big)$, , sometimes skipping π from the output if only $out_{\mathcal{V}}$ is relevant.

The scheme satisfies the following properties:

- correct: *for any vector polynomial $\boldsymbol{P} = (P_1, \ldots, P_n) \in \mathbb{F}[X]^{(\leq d_1)} \times \cdots \times \mathbb{F}[X]^{(\leq d_n)}$ and any set of evaluation points $E \subseteq \mathbb{E}$, we have:*

$$\Pr\left[out = \textsc{Accept} \;\middle|\; \begin{array}{l} (\mathsf{com}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{P}) \\ out \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}, E, \boldsymbol{P}|_E) \end{array}\right] = 1 \ ,$$

where $\boldsymbol{P}|_E := \{\boldsymbol{P}(e)\}_{e \in E}$.

- (t, ε)-polynomial-binding: *for any stateful adversary $\mathcal{A}$ running in time t, playing the role of the prover in the above protocols, and for any integer $K \in \mathbb{N}$, we have:*

$$\Pr\left[\begin{array}{l} out_1 = \cdots = out_K = \textsc{Accept} \ \wedge \\ \quad \nexists \boldsymbol{P} \text{ with } \deg \boldsymbol{P} \leq \boldsymbol{d} \text{ s.t.} \\ \boldsymbol{P}(e) = \boldsymbol{p}_e^{(k)} \ \forall e \in E^{(k)}, \ \forall k \in [1, K] \end{array} \;\middle|\; \begin{array}{l} (\mathsf{com}, *) \leftarrow \mathsf{Commit}^{\mathcal{A}}(*) \\ \big(E^{(k)}, \{\boldsymbol{p}_e^{(k)}\}_{e \in E^{(k)}}\big)_{k \in [1,K]} \leftarrow \mathcal{A}() \\ \forall k \in [1, K] : \ out_k \leftarrow \mathsf{Eval}^{\mathcal{A}}\big(*, \mathsf{com}, E^{(k)}, \{\boldsymbol{p}_e^{(k)}\}_{e \in E^{(k)}}\big) \end{array}\right] \leq \varepsilon \ ,$$

where $*$ denote arbitrary inputs/outputs of the adversary which do not affect the definition. The above probability is over the randomness of $\mathcal{A}$, $\mathsf{Commit}.\mathcal{V}$ and $\mathsf{Eval}.\mathcal{V}$.

[3] Our definition can be thought of as a more general and more formal definition of the degree-enforcing commitment scheme from [FR23] which further integrates the binding of evaluations, considers general Commit and Eval protocols (whereas the original definition assumed fixed number of rounds), and further includes the hiding property.

– (t,ξ)-honest-verifier zero knowledge ((t,ξ)-HVZK)*: there exists a PPT algorithm* Sim *(the simulator) such that for any stateful adversary* $\mathcal{A}$ *running in time* t*, for any vector polynomial* $\boldsymbol{P} = (P_1, \ldots, P_n) \in \mathbb{F}[X]^{(\leq d_1)} \times \cdots \times \mathbb{F}[X]^{(\leq d_n)}$ *and for any set of evaluation points* $E \subseteq \mathbb{E}$*, we have:*

$$\Pr\left[\hat{b} = b \;\middle|\; \begin{array}{l} (\mathsf{com}^{(0)}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{P}) \\ (out, \pi^{(0)}) \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}^{(0)}, E, \boldsymbol{P}|_E) \\ (\mathsf{com}^{(1)}, \pi^{(1)}) \leftarrow \mathsf{Sim}(E, \boldsymbol{P}|_E) \\ b \leftarrow \{0,1\} \\ \hat{b} \leftarrow \mathcal{A}(\mathsf{com}^{(b)}, \pi^{(b)}) \end{array}\right] \leq \frac{1}{2} + \xi \ .$$

We recall the degree-enforcing commitment scheme from [FR23] in Figure 1. This scheme restricts the vector degree parameter $\boldsymbol{d}$ to be of the form $\boldsymbol{d} = (d, \ldots, d)$ for some $d \in \mathbb{N}$. The parameter $\eta \in \mathbb{N}$ is chosen to reach the desired polynomial-biding soundness. Our description uses a distribution $\mathcal{D}(\mathbb{F}^{\eta \times n})$ over the space $\mathbb{F}^{\eta \times n}$ which might be different from uniformity. It further integrates masking polynomials to make the scheme hiding.

DECS.Commit:

On input a vector polynomial $\boldsymbol{P} \in \left(\mathbb{F}[X]^{(\leq d_{\mathsf{decs}})}\right)^{n_{\mathsf{decs}}}$ for the prover, $\mathsf{DECS.Commit}$ consists of the following 3-pass protocol:

1. $\mathcal{P}$ samples uniformly at random η degree-d_{decs} polynomials $\boldsymbol{M} = (M_1, \ldots, M_\eta) \leftarrow (\mathbb{F}[X]^{(\leq d_{\mathsf{decs}})})^\eta$ and N random hash commitment tapes $(\rho_1, \ldots, \rho_N) \leftarrow (\{0,1\}^\lambda)^N$. $\mathcal{P}$ computes $u_j = \mathsf{Hash}(\boldsymbol{P}(e_j), \boldsymbol{M}(e_j), j, \rho_j)$ for every $e_j \in \mathbb{E}$ and $\mathsf{root} = \mathsf{MerkleRoot}(u_1, \ldots, u_N)$. $\mathcal{P}$ sends root to $\mathcal{V}$.
2. $\mathcal{V}$ samples a random matrix $\Gamma = (\gamma_{k,i})_{k,i} \leftarrow \mathcal{D}(\mathbb{F}^{\eta \times n_{\mathsf{decs}}})$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes the polynomials

$$R_k(X) = M_k(X) + \sum_{i=1}^{n_{\mathsf{decs}}} \gamma_{k,i} \cdot P_i(X) \quad \text{for all } k \in [1, \eta]$$

and sends the vector polynomial $\boldsymbol{R} := (R_1, \ldots R_\eta)$ to $\boldsymbol{V}$.

$\mathcal{P}$ returns $\mathsf{key} := (\boldsymbol{P}, \boldsymbol{M})$ while the commitment (i.e., the protocol transcript) is defined as $\mathsf{com} := (\mathsf{root}, \Gamma, \boldsymbol{R})$.

DECS.Eval:

The prover takes as input an opening key $\mathsf{key} = (\boldsymbol{P}, \boldsymbol{M})$, and the verifier takes as input a commitment $\mathsf{com} = (\mathsf{root}, \Gamma, \boldsymbol{R})$. Both parties additionally take a set of evaluation points $E \subseteq \mathbb{E}$ and a set of evaluations $\boldsymbol{P}|_E := \{\boldsymbol{P}(e)\}_{e \in E}$. The evaluation protocol works as follows:

1. $\mathcal{P}$ sends the evaluations $\boldsymbol{M}|_E$, the random tapes $\{\rho_j \ ; \ e_j \in E\}$ and the proof π_{MT} made of the authentication paths from the leaves $\{u_j\}_{e_j \in E}$ to the Merkle root root.
2. $\mathcal{V}$ performs the following checks:
   – For all j s.t. $e_j \in E$, verify the authentication path for $u_j = \mathsf{Hash}(\boldsymbol{P}(e_j), \boldsymbol{M}(e_j), j, \rho_j)$ in π_{MT} w.r.t. root.
   – For all $e \in E$ and all $k \in [1, \eta]$, verify the equality $R_k(e) = M_k(e) + \sum_{i=1}^{n_{\mathsf{decs}}} \gamma_{k,i} \cdot P_i(e)$.
   – For all $k \in [1, \eta]$, verify the degree $\deg R_k \leq d$.

Fig. 1: Degree-enforcing commitment scheme (DECS) from [FR23]. This is a small-domain PCS with parameters $(\mathbb{F}, \mathbb{E}, n_{\mathsf{decs}}, \boldsymbol{d})$ with $\boldsymbol{d} = (d_{\mathsf{decs}}, \ldots, d_{\mathsf{decs}})$ for $d_{\mathsf{decs}} \in \mathbb{N}$. Additional parameters: the number degree-enforcing test repetitions $\eta \in \mathbb{N}$ and the challenge distribution $\mathcal{D}(\mathbb{F}^{\eta \times n})$ (a probability distribution over $\mathbb{F}^{\eta \times n_{\mathsf{decs}}}$).

We have the following theorem. The proof, is a direct adaptation from [FR23]. For the sake of simplicity, we assume that the adversary cannot produce hash collisions. The polynomial-binding soundness is then information theoretic: whatever the adversary $\mathcal{A}$ and its running time $t_{\mathcal{A}}$, a break of the polynomial-binding property can only happen with probability $\varepsilon_{\mathsf{decs}}$ as long as $\mathcal{A}$ cannot produce a hash collision.

Theorem 1. *For an adversary $\mathcal{A}$ unable to produce hash collisions and running in time $t_{\mathcal{A}}$, the DECS protocol depicted in Figure 1 is a $(t_{\mathcal{A}}, \varepsilon_{\mathsf{decs}})$-polynomial-binding small-domain PCS with*

$$\varepsilon_{\mathsf{decs}} = \binom{N}{d_{\mathsf{decs}}+2} \cdot \varepsilon_{\mathcal{D}} \quad \textit{where} \quad \varepsilon_{\mathcal{D}} := \max_{v \in \mathbb{F}^n, u \in \mathbb{F}^\eta} \Pr_{\Gamma \leftarrow \mathcal{D}(\mathbb{F}^{\eta \times n})}[\Gamma \cdot v + u = 0] \ .$$

When $\mathcal{D}(\mathbb{F}^{\eta \times n})$ is the uniform distribution over $\mathbb{F}^{\eta \times n}$, we have

$$\varepsilon_{\mathsf{decs}} = \frac{\binom{N}{d_{\mathsf{decs}}+2}}{|\mathbb{F}|^\eta} \ .$$

The following theorem states the hiding property of the DECS protocol.

Theorem 2. *In the ROM (i.e., modelling* Hash *as a random oracle), the DECS protocol depicted in Figure 1 is a (t, ξ)-HVZK small-domain PCS, with*

$$t = Q \quad \textit{and} \quad \xi \leq \frac{Q}{2^\lambda}$$

for every $Q \in \mathbb{N}$, where Q represents the number of RO queries made by the adversary.

Proof. We first describe the HVZK simulator Sim taking $(E, \boldsymbol{P}|_E)$ as input. If $|E| \geq d_{\mathsf{decs}} + 1$, then Sim reconstructs $\boldsymbol{P}$ from $\boldsymbol{P}|_E$ and runs a genuine execution:

$$(\mathsf{com}^{(1)}, \mathsf{key}) \leftarrow \mathsf{Commit}(\boldsymbol{P})$$
$$(\mathit{out}, \pi^{(1)}) \leftarrow \mathsf{Eval}(\mathsf{key}, \mathsf{com}^{(1)}, E, \boldsymbol{P}|_E)$$

and returns $(\mathsf{com}^{(1)}, \pi^{(1)})$. We then have that $(\mathsf{com}^{(1)}, \pi^{(1)})$ is identically distributed to $(\mathsf{com}^{(0)}, \pi^{(0)})$ and hence $\xi = 0$. Now let us assume that $|E| \leq d_{\mathsf{decs}}$. The simulator runs as follows:

1. Sample a random matrix $\Gamma = (\gamma_{k,i})_{k,i} \leftarrow \mathcal{D}(\mathbb{F}^{\eta \times n_{\mathsf{decs}}})$,
2. Sample a random vector polynomial $\boldsymbol{R} = (R_1, \ldots, R_\eta) \leftarrow (\mathbb{F}[X]^{(\leq d_{\mathsf{decs}})})^\eta$,
3. Sample N random hash commitment tapes $(\rho_1, \ldots, \rho_N) \leftarrow (\{0,1\}^\lambda)^N$,
4. For all $e \in \mathbb{E}$, sample a random vector $\boldsymbol{P}(e) \leftarrow \mathbb{F}^{n_{\mathsf{decs}}}$,
5. For all $e \in E$, and all $k \in [1, \eta]$ defines

$$M_k(e) = R_k(e) - \sum_{i=1}^{n_{\mathsf{decs}}} \gamma_{k,i} \cdot P_i(e) \ .$$

   For all $e \in \mathbb{E} \setminus E$, sample a random vector $\boldsymbol{M}(e) \leftarrow \mathbb{F}^\eta$.
6. Compute $u_j = \mathsf{Hash}(\boldsymbol{P}(e_j), \boldsymbol{M}(e_j), j, \rho_j)$ for every $e_j \in \mathbb{E}$ and $\mathsf{root} = \mathsf{MerkleRoot}(u_1, \ldots, u_N)$.
7. Let $\mathsf{com}^{(1)} = (\mathsf{root}, \Gamma, \boldsymbol{R})$. Let $\pi^{(1)}$ the evaluation protocol transcript made of the evaluations $\boldsymbol{M}|_E$, the random tapes $\{\rho_j \ ; \ e_j \in E\}$ and the proof π_{MT} (i.e. the authentication paths from the leaves $\{u_j\}_{e_j \in E}$ to the Merkle root root). Return $(\mathsf{com}^{(1)}, \pi^{(1)})$.

We now use a hybrid argument to bound the adversary's advantage in distinguishing a proof $(\mathsf{com}^{(1)}, \pi^{(1)})$ returned by the above simulator from a genuine proof generated using the full polynomial $\boldsymbol{P}$. Let us denote Game_0, the standard HVZK game depicted in Definition 3 and let Game_1 the game which is similar to Game_0

but stopping whenever the adversary makes a RO query of the form $\mathsf{Hash}(\boldsymbol{p}, \boldsymbol{m}, j, \rho_j)$ for any $\boldsymbol{p} \in \mathbb{F}^{n_{\mathsf{decs}}}$ and $\boldsymbol{m} \in \mathbb{F}^{\eta}$, for j such that $e_j \notin E$ and for ρ_j the random tape generated by the simulator. We have:

$$\Pr[\mathsf{Game}_1 \text{ stops}] \leq \frac{Q}{2^\lambda} .$$

Indeed, for each RO query of the right format, the adversary has probability $1/2^\lambda$ of using the right ρ_j. Then, conditioned to the event that Game_1 does not stop, we have that $(\mathsf{com}^{(1)}, \pi^{(1)})$ is identically distributed to a genuine proof $(\mathsf{com}^{(0)}, \pi^{(0)})$ computed from $\boldsymbol{P}$. Specifically,

- Γ is freshly random from $\mathcal{D}(\mathbb{F}^{\eta \times n_{\mathsf{decs}}})$,
- $\boldsymbol{M}|_E$ and $\boldsymbol{R}$ are uniformly random from $(\mathbb{F}^{\eta})^{|E|}$ and $(\mathbb{F}[X])^{\eta}$ satisfying the degree-enforcing test,
- $\{\rho_j \,;\, e_j \in E\}$ are uniformly random from $\{0,1\}^\lambda$,
- root is the Merkle root corresponding to leaves $u_j = \mathsf{Hash}(\boldsymbol{P}(e_j), \boldsymbol{M}(e_j), j, \rho_j)$ for every j such that $e_j \in E$ and for uniformly random leaves u_j for every j such that $e_j \notin E$,
- π_{MT} is made of correct authentication paths from $\{u_j \,;\, e_j \in E\}$ to root.

We deduce:

$$\Pr[\mathcal{A} \text{ wins } \mathsf{Game}_0] \leq \underbrace{\Pr[\mathcal{A} \text{ wins } \mathsf{Game}_1 \mid \mathsf{Game}_1 \text{ does not stop}]}_{\leq 1/2} + \underbrace{\Pr[\mathsf{Game}_1 \text{ stops}]}_{\leq Q/2^\lambda}$$

which concludes the proof.

□

3.2 Polynomial Commitment Scheme from DECS

This section introduces further commitment schemes constructed from a small-domain PCS. Although our concrete schemes are based on the DECS protocol pictured in Figure 1, we stress that our constructions and analysis can rely on any small-domain PCS and could benefit future constructions of small-domain PCS. We first introduce SmallWood-LVCS a linear-map vector commitment scheme (LVCS) and then introduce SmallWood-PCS, a polynomial commitment scheme (PCS) constructed from SmallWood-LVCS. The proposed constructions are based on techniques from [BCG20, Lee21, GLS+23]. Our main contribution is to formally describe and analyze the LVCS and PCS obtained by applying these techniques with the DECS protocol and to extend these schemes to inherently integrate the zero-knowledge property.

SmallWood-LVCS. Let $\Omega = \{\omega_1, \ldots, \omega_{n_{\mathsf{cols}}}\} \subseteq \mathbb{F}$. The high-level principle of the LVCS is to commit the rows $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}} \in \mathbb{F}^{n_{\mathsf{cols}}}$ as polynomials $P_1, \ldots, P_{n_{\mathsf{rows}}}$ using the DECS protocol. Specifically, for all j, we define P_j by interpolation as a polynomial satisfying $(P_j(\omega_1), \ldots, P_j(\omega_{n_{\mathsf{cols}}})) = \boldsymbol{r}_j$. Later, to prove $\boldsymbol{v}_k = \sum_j c_{k,j} \cdot \boldsymbol{r}_j$ for some vector $\boldsymbol{v}_k \in \mathbb{F}^{n_{\mathsf{cols}}}$ and coefficients $c_{k,1}, \ldots, c_{k,n_{\mathsf{rows}}} \in \mathbb{F}$, the prover sends the polynomial $Q_k(X) = \sum_j c_{k,j} \cdot P_j(X)$ to the verifier who checks $(Q_k(\omega_1), \ldots, Q_k(\omega_{n_{\mathsf{cols}}})) = \boldsymbol{v}_k$. The verifier then challenges the prover to open some evaluations $\boldsymbol{P}|_E$ for $\boldsymbol{P} = (P_1, \ldots, P_{n_{\mathsf{cols}}})$ and $E \subseteq \mathbb{E}$ a subset of the DECS evaluation domain, and checks $Q_k(e) = \sum_j c_{j,k} \cdot P_j(e)$ for all $e \in E$ (thus verifying that Q_k has been correctly constructed by the prover).

To achieve the zero-knowledge property, the set Ω must be disjoint from $\mathbb{E}$. In addition, we must ensure that opening ℓ evaluations of $P_1, \ldots, P_{n_{\mathsf{rows}}}$ leaks no information about the rows' coefficients, where $\ell := |E|$. To this purpose, we randomly pick vectors $\bar{\boldsymbol{r}}_j \in \mathbb{F}^\ell$ and build the P_j as the polynomial of degree $\leq d := n_{\mathsf{cols}} + \ell - 1$ satisfying both $(P_j(\omega_1), \ldots, P_j(\omega_{n_{\mathsf{cols}}})) = \boldsymbol{r}_j$ and $(P_j(\omega'_1), \ldots, P_j(\omega'_\ell)) = \bar{\boldsymbol{r}}_j$ for some set $\Omega' = \{\omega'_1, \ldots, \omega'_\ell\} \subseteq \mathbb{F}$ disjoint from Ω.

The opening proof is sound thanks to the Schwartz-Zippel lemma: if there exists i such that $(\boldsymbol{v}_k)_i \neq \sum_j c_{k,j} \cdot (\boldsymbol{r}_j)_i$, while $(Q_k(\omega_1), \ldots, Q_k(\omega_{n_{\mathsf{cols}}})) = \boldsymbol{v}_k$ (which is necessary for the verifier to accept), then the polynomial Q_k sent by the prover does not equal $\sum_j c_{k,j} \cdot P_j$ (since it differs at least in point ω_i). Then, the probability that the verifier accepts the proof is at most $\binom{d}{\ell} / \binom{|\mathbb{E}|}{\ell}$, where $d = \max\{\deg P_j\} \leq n_{\mathsf{cols}} + \ell - 1$.

In practice, the prover does not need to send the polynomials $\{Q_k\}_k$ in full but only the vectors $\bar{\boldsymbol{v}}_k = \sum_{j=1}^{n_{\mathsf{rows}}} c_{k,j} \cdot \bar{\boldsymbol{r}}_j$ which, together with the vectors $\boldsymbol{v}_k$'s, enable the verifier to recover $\{Q_k\}_k$ by interpolation.

The obtained LVCS is formally described in Figure 2 and its security is given by the following theorem.

LVCS.Commit:

On input vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}}$ from $\mathbb{F}^{n_{\mathsf{cols}}}$ for the prover, run the following protocol:

1. $\mathcal{P}$ samples uniformly at random $\bar{\boldsymbol{r}}_1, \ldots, \bar{\boldsymbol{r}}_{n_{\mathsf{rows}}} \leftarrow \mathbb{F}^{\ell}$ and build by interpolation the degree-$(n_{\mathsf{cols}}+\ell-1)$ polynomial $P_j(X)$ such that
$$(P_j(\omega_1), \ldots, P_j(\omega_{n_{\mathsf{cols}}})) = \boldsymbol{r}_j \quad \text{and} \quad (P_j(\omega_{n_{\mathsf{cols}}+1}), \ldots, P_j(\omega'_\ell)) = \bar{\boldsymbol{r}}_j \ .$$
Let $\boldsymbol{P} = (P_1, \ldots, P_{n_{\mathsf{rows}}})$ the associated vector polynomial.
2. $\mathcal{P}$ and $\mathcal{V}$ run the protocol $\mathsf{DECS.Commit}(\boldsymbol{P})$ resulting in the commitment $\mathsf{com}_{\mathsf{decs}}$ and the opening key $\mathsf{key}_{\mathsf{decs}}$.

$\mathcal{P}$ returns $\mathsf{key} := (\mathsf{key}_{\mathsf{decs}}, (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}}), (\bar{\boldsymbol{r}}_1, \ldots, \bar{\boldsymbol{r}}_{n_{\mathsf{rows}}}))$ while the commitment (i.e., the protocol transcript) is defined as $\mathsf{com} := \mathsf{com}_{\mathsf{decs}}$.

LVCS.Eval:

In this protocol, the prover takes as input an LVCS opening key $\mathsf{key} = (\mathsf{key}_{\mathsf{decs}}, (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}}), (\bar{\boldsymbol{r}}_1, \ldots, \bar{\boldsymbol{r}}_{n_{\mathsf{rows}}}))$, and the verifier takes as input a commitment com. Both parties additionally take a set of coefficients $\mathcal{C} := \{c_{k,j}\}$ with $1 \le k \le m$ and $1 \le j \le n_{\mathsf{rows}}$, and a set of vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m \in \mathbb{F}^{n_{\mathsf{cols}}}$. The protocol runs as follows:

1. $\mathcal{P}$ computes $\bar{\boldsymbol{v}}_k = \sum_{j=1}^{n_{\mathsf{rows}}} c_{k,j} \cdot \bar{\boldsymbol{r}}_j$ for every $k \in [1, m]$ and sends $\bar{\boldsymbol{v}}_1, \ldots, \bar{\boldsymbol{v}}_m$ to $\mathcal{V}$.
2. $\mathcal{V}$ samples ℓ distinct random points $E \subseteq \mathbb{E}$ and send them to $\mathcal{P}$.
3. $\mathcal{P}$ sends $\boldsymbol{P}|_E := \{\boldsymbol{P}(e)\ ;\ e \in E\}$ to $\mathcal{V}$.
4. $\mathcal{P}$ and $\mathcal{V}$ run the protocol $\mathsf{DECS.Eval}(\mathsf{key}_{\mathsf{decs}}, \mathsf{com}_{\mathsf{decs}}, E, \boldsymbol{P}|_E)$. If the output is Reject, $\mathcal{V}$ stops and outputs Reject. Otherwise, the protcol continues.
5. For every $k \in [1, m]$, $\mathcal{V}$ computes $Q_k(X)$ such that
$$(Q_k(\omega_1), \ldots, Q_k(\omega_{n_{\mathsf{cols}}})) = \boldsymbol{v}_k \quad \text{and} \quad (Q_k(\omega'_1), \ldots, Q_k(\omega'_\ell)) = \bar{\boldsymbol{v}}_k \ .$$
$\mathcal{V}$ checks $Q_k(e) = \sum_{j=1}^{n_{\mathsf{rows}}} c_{k,j} \cdot \boldsymbol{P}_j(e)$ for all $k \in [1, m]$ and $e \in E$. If all the equalities are verified, then $\mathcal{V}$ outputs Accept, otherwise $\mathcal{V}$ outputs Reject.

Fig. 2: SmallWood-LVCS: Linear-map vector commitment scheme from the DECS protocol.

Theorem 3. *The LVCS depicted in Figure 2 is (t, ε)-function-binding with*

$$\varepsilon = \varepsilon_{\mathsf{decs}} + \frac{\binom{n_{\mathsf{cols}}+\ell-1}{\ell}}{\binom{N}{\ell}} \ ,$$

where $\varepsilon_{\mathsf{decs}}$ is the advantage of breaking the polynomial-binding property of the underlying DECS protocol in time t.

Proof. To break the function-binding property, the adversary should first run the Commit protocol with the verifier, resulting in a commitment com, and then produce some openings $\left(\mathcal{C}^{(i)} := \{c^{(i)}_{k,j}\}, (\boldsymbol{v}^{(i)}_1, \ldots, \boldsymbol{v}^{(i)}_m)\right)_{i \in [1,I]}$ such that (1) $\mathsf{Eval}(\mathsf{key}, \mathsf{com}, \mathcal{C}^{(i)}, (\boldsymbol{v}^{(i)}_1, \ldots, \boldsymbol{v}^{(i)}_m))$ outputs Accept for every $i \in [1, I]$, and (2) there exist no vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}} \in \mathbb{F}^{n_{\mathsf{cols}}}$ such that

$$\begin{cases} \sum_j c^{(i)}_{1,j} \cdot \boldsymbol{r}_j = \boldsymbol{v}^{(i)}_1 \\ \quad \vdots \\ \sum_j c^{(i)}_{m,j} \cdot \boldsymbol{r}_j = \boldsymbol{v}^{(i)}_m \end{cases}$$

for every $i \in [1, I]$. This second condition implies that there exist no vectors $(\boldsymbol{r}_1 \parallel \bar{\boldsymbol{r}}_1), \ldots, (\boldsymbol{r}_{n_{\mathsf{rows}}} \parallel \bar{\boldsymbol{r}}_{n_{\mathsf{rows}}}) \in \mathbb{F}^{n_{\mathsf{cols}}+\ell}$ such that

$$\begin{cases} \sum_j c_{1,j}^{(i)} \cdot (\boldsymbol{r}_j \parallel \bar{\boldsymbol{r}}_j) = (\boldsymbol{v}_1^{(i)} \parallel \bar{\boldsymbol{v}}_1^{(i)}) \\ \quad \vdots \\ \sum_j c_{m,j}^{(i)} \cdot (\boldsymbol{r}_j \parallel \bar{\boldsymbol{r}}_j) = (\boldsymbol{v}_m^{(i)} \parallel \bar{\boldsymbol{v}}_m^{(i)}) \end{cases}$$

for every $i \in [1, I]$, which further translates to: there exist no polynomials $P'_1, \ldots, P'_{n_{\mathsf{cols}}}$ of degree $\leq n_{\mathsf{cols}}+\ell-1$ satisfying

$$\begin{cases} \sum_j c_{1,j}^{(i)} \cdot P'_j = Q_1^{(i)} \\ \quad \vdots \\ \sum_j c_{m,j}^{(i)} \cdot P'_j = Q_m^{(i)} \end{cases} \tag{1}$$

for every $i \in [1, I]$ where $Q_1^{(i)}, \ldots, Q_m^{(i)}$ denote the polynomials computed by the verifier in the fifth step of the i-th run of the Eval protocol (i.e. on input $\mathsf{key}, \mathsf{com}, \mathcal{C}^{(i)}, (\boldsymbol{v}_1^{(i)}, \ldots, \boldsymbol{v}_m^{(i)})$).

Now let us denote the challenged evaluation points in the i-th run of the Eval protocol by $E^{(i)}$. Also, let DEB be the event that the opened evaluations $\{\boldsymbol{P}|_{E^{(i)}}\}_i$ form a break of the polynomial-binding property of the DECS scheme. We have

$$\varepsilon \leq \underbrace{\Pr[\mathsf{DEB}]}_{\varepsilon_{\mathsf{decs}}} + \Pr[\mathcal{A} \text{ breaks polynomial-binding} \mid \neg\mathsf{DEB}] \ .$$

Given that DEB does not occur, all the opened evaluations from the DECS must correspond to a vector polynomial $\boldsymbol{P} = (P_1, \ldots, P_{n_{\mathsf{cols}}})$ of degree $\leq n_{\mathsf{cols}} + \ell - 1$. Then, we obtain a break of the function-binding property if and only if all the Eval protocols accept, implying that the opened evaluations verify the system in (1), while there exist no $\boldsymbol{P}$ satisfying this system. In other words, for at least one $k \in [1, m]$ and one $i \in [1, I]$, we have

$$\sum_j c_{k,j}^{(i)} \cdot P_j \neq Q_k^{(i)} \ ,$$

yet the opended evaluations must satisfy $\sum_j c_{k,j}^{(i)} \cdot P_j(e) = Q_k^{(i)}(e)$ for all $e \in E^{(i)}$. According to the Schwartz-Zippel lemma, this can only occur with probability

$$\frac{\binom{n_{\mathsf{cols}}+\ell-1}{\ell}}{\binom{N}{\ell}} \ .$$

□

Theorem 4. *The LVCS depicted in Figure 2 is (t, ξ)-HVZK provided that the underlying DECS protocol is (t, ξ)-HVZK.*

Proof. By definition, the vectors $\bar{\boldsymbol{v}}_1, \ldots, \bar{\boldsymbol{v}}_m$ and the evaluation points $\boldsymbol{P}|_E := \{\boldsymbol{P}(e) \, ; \, e \in E\}$ are uniformly random and independent from the input vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}}$. The HVZK simulator simply samples these values randomly (for a random evaluation set E of size ℓ) and call the HVZK simulator of the DECS for the rest. □

SmallWood-PCS. We now describe how to construct a PCS from the LVCS. The idea is to arrange the coefficients of the committed polynomial as rows of the LVCS commitment. Then a polynomial evaluation can be obtained from a linear combination of the rows with the appropriate powers of the evaluation point as coefficients. Additional care must be given so that the opened LVCS evaluations do not leak information in order to ensure the zero-knowledge property.

We explain the principle with a simple example. For a polynomial $P(X) = \sum_{i=0}^{d} a_i \cdot X^i$ and some parameters $\mu, \nu \in \mathbb{N}$ such that $\nu\mu = d+1$, consider the matrix

$$\boldsymbol{A} = \begin{bmatrix} a_0 & a_\mu & \cdots & a_{(\nu-1)\mu} \\ a_1 & a_{\mu+1} & \cdots & a_{(\nu-1)\mu+1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{\mu-1} & a_{2\mu-1} & \cdots & a_{\nu\mu-1} \end{bmatrix} .$$

We can express an evaluation $P(e)$ from a linear combination $\boldsymbol{v}$ of the rows of the matrix:

$$P(e) = \langle \boldsymbol{v}, (1, e^\mu, e^{2\mu}, \ldots, e^{(\nu-1)\mu}) \rangle \quad \text{with} \quad \boldsymbol{v} := (1, e, e^2, \ldots, e^{\mu-1}) \cdot \boldsymbol{A} .$$

This way, a linear-map vector commitment of the rows of $\boldsymbol{A}$ yields a polynomial commitment of P. However, this is not zero-knowledge since opening the evaluation $P(e)$ implies opening the vector $\boldsymbol{v}$ which reveals additional information on the coefficients of P, specifically ν degrees of freedom (i.e. linear combinations of the coefficients) instead of 1. To make the construction zero-knowledge, we shall mask these $\nu - 1$ revealed degrees of freedom using randomness. For this, the matrix $\boldsymbol{A}$ is replaced by the following $(\mu+1) \times \nu$ matrix:

$$\boldsymbol{A'} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{0} \end{bmatrix} + \begin{bmatrix} 0 & -r_1 & -r_2 & \cdots & -r_{\nu-2} & -r_{\nu-1} \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ r_1 & r_2 & r_3 & \cdots & r_{\nu-1} & 0 \end{bmatrix} .$$

In words, we append a $\boldsymbol{0}$ (all-0 row vector) to $\boldsymbol{A}$, then we add $(0, -r_1, \ldots, -r_{\nu-1})$ to the first row and $(r_1, \ldots, r_{\nu-1}, 0)$ to the last row. We now have:

$$P(e) = \langle \boldsymbol{v'}, (1, e^\mu, e^{2\mu}, \ldots, e^{(\nu-1)\mu}) \rangle \quad \text{with} \quad \boldsymbol{v'} := (1, e, e^2, \ldots, e^{\mu-1}, e^\mu) \cdot \boldsymbol{A'} .$$

Moreover, it can be checked that $\boldsymbol{v'}$ is uniformly random among the vector of $\mathbb{F}^\nu$ satisfying the relation $P(e) = \sum_i v'_i \cdot e^{i \cdot \mu}$ which implies that $\boldsymbol{v'}$ reveal no more information on P than the evaluation $P(e)$. To open ℓ' evaluations $\{P(e)\}$, one shall add ℓ' such rows of randomness to the above matrix.

Our PCS follows the above approach which we now describe for the general setting of a vector polynomial $\boldsymbol{P} = (P_1, \ldots, P_{n_{\mathsf{pcs}}}) \in \mathbb{F}[X]^{(\leq d_1)} \times \cdots \times \mathbb{F}[X]^{(\leq d_{n_{\mathsf{pcs}}})}$ which aims to be opened in ℓ' points $E' = \{e_1, \ldots, e_{\ell'}\}$ (here we use the notation E' to avoid confusion with the set E of the DECS evaluation points arising in the LVCS description). Each polynomial $P_j(X) = \sum_i a_{j,i} \cdot X^i$ gives rise to a matrix $\boldsymbol{A}_j$ with $\mu + \ell'$ rows and ν_j columns such that

$$\nu_j := \lceil (d_j + 1 - \ell')/\mu \rceil .$$

For the sake of simplicity, we shall first assume that $d_j + 1 - \ell'$ is a multiple of μ, so that $\nu_j = (d_j + 1 - \ell')/\mu$. We also require $\ell' \leq \mu$. The matrix $\boldsymbol{A}_j$ is then defined as:

$$\boldsymbol{A}_j := \left[\begin{array}{ccc|c} a_{j,0} & \cdots & a_{j,(\nu_j-2)\mu} & a_{j,(\nu_j-1)\mu} \\ \vdots & \ddots & \vdots & \vdots \\ a_{j,\mu-1} & \cdots & a_{j,(\nu_j-1)\mu-1} & a_{j,\nu_j\mu-1} \\ \hline 0 & \cdots & 0 & a_{j,\nu_j\mu} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{j,\nu_j\mu+\ell'-1} \end{array}\right] + \left[\begin{array}{cccc|c} 0 & -r_{j,1,1} & \cdots & -r_{j,1,\nu_j-2} & -r_{j,1,\nu_j-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & -r_{j,\ell',1} & \cdots & -r_{j,\ell',\nu_j-2} & -r_{j,\ell',\nu_j-1} \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ \hline r_{j,1,1} & r_{j,1,2} & \cdots & r_{j,1,\nu_j-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ r_{j,\ell',1} & r_{j,\ell',2} & \cdots & r_{j,\ell',\nu_j-1} & 0 \end{array}\right] .$$

In the above equation, the horizontal line marks the frontier between the first μ rows and the last ℓ' rows (which are added for the zero-knowledge property). We optimize the mapping by overflowing the high-degree coefficients of P_j on the ℓ' extra rows of the last column. Using this tweak, one can verify that the $\boldsymbol{A}_j$ matrix satisfies:

$$P_j(e) = \langle \boldsymbol{v}^{(e)}, (1, e^{\mu}, e^{2\mu}, \ldots, e^{(\nu_j-1)\mu}) \rangle \quad \text{with} \quad \boldsymbol{v}^{(e)} := (1, e, e^2, \ldots, e^{\mu+\ell'}) \cdot \boldsymbol{A}_j$$

for every evaluation point $e \in \mathbb{F}$. Moreover, one can check that for any ℓ' evaluation points $E' = \{e_1, \ldots, e_{\ell'}\}$, the vectors $\{\boldsymbol{v}^{(e)}\}_{e \in E'}$ do not reveal any more information than the evaluations $\{P_j(e)\}_{e \in E'}$.

Let us now consider the case where $d_j + 1 - \ell'$ is not a multiple of μ. Since ν_j is defined as $\nu_j := \lceil (d_j + 1 - \ell')/\mu \rceil$, it implies that

$$\mu \cdot \nu_j + \ell' > d_j + 1 \ .$$

We might consider building $\boldsymbol{A}_j$ as previously by writing $P_j(X)$ as $\sum_{i=0}^{d'_j} a_{j,i} \cdot X^i$, with $d'_j = \mu \cdot \nu_j + \ell' - 1$ and $(a_{j,d_j+1}, \ldots, a_{j,d'_j}) = (0, \ldots, 0)$. However, in doing so, the verifier would only be guaranteed that the prover committed to a degree-d'_j polynomial, rather than a degree-d_j polynomial, since there would be no assurance that the coefficients $(a_{j,d+1}, \ldots, a_{j,d'})$ are actually zero. To avoid this issue, we can define $\boldsymbol{A}_j$ as

$$\boldsymbol{A}_j := \left[\begin{array}{ccc|c} a_{j,0} & \cdots & a_{j,(\nu_j-2)\mu} & \left.\begin{matrix}0\\ \vdots \\ 0\end{matrix}\right\} \begin{matrix}\delta_j \\ \text{times}\end{matrix} \\ \vdots & \ddots & \vdots & \\ \vdots & \ddots & \vdots & \\ \vdots & \ddots & \vdots & \\ \hline a_{j,\mu-1} & \cdots & a_{j,(\nu_j-1)\mu-1} & a_{j,(\nu_j-1)\mu} \\ \hline 0 & \cdots & 0 & a_{j,d_j-\ell'+1} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{j,d_j} \end{array}\right] + \left[\begin{array}{cccc|c} 0 & -r_{j,1,1} & \cdots & -r_{j,1,\nu_j-2} & \left.\begin{matrix}0\\ \vdots \\ 0\end{matrix}\right\} \begin{matrix}\delta_j \\ \text{times}\end{matrix} \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & -r_{j,\ell',1} & \cdots & -r_{j,\ell',\nu_j-2} & \\ 0 & 0 & \cdots & 0 & -r_{j,1,\nu_j-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & -r_{j,\ell',\nu_j-1} \\ \hline r_{j,1,1} & r_{j,1,2} & \cdots & r_{j,1,\nu_j-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ r_{j,\ell',1} & r_{j,\ell',2} & \cdots & r_{j,\ell',\nu_j-1} & 0 \end{array}\right] \tag{2}$$

with $\delta_j := (\mu \cdot \nu_j + \ell') - (d_j + 1)$, namely we shift the last column of $\boldsymbol{A}_j$ to the bottom by δ_j coefficients. Then one can verify that this new $\boldsymbol{A}_j$ matrix satisfies

$$P_j(e) = \langle \boldsymbol{v}^{(e)}, (1, e^{\mu}, e^{2\mu}, \ldots, e^{(\nu_j-2)\mu}, e^{(\nu_j-1)\mu-\delta_j}) \rangle \quad \text{with} \quad \boldsymbol{v}^{(e)} := (1, e, e^2, \ldots, e^{\mu+\ell'}) \cdot \boldsymbol{A}_j \ .$$

Let us now define the global matrix $\boldsymbol{A}$ of the LVCS which embeds all the matrices $\boldsymbol{A}_j$. Remind that all the $\boldsymbol{A}_j$'s have the same number of rows $\mu + \ell'$ but different number of columns ν_j (depending on the degree d_j of P_j). We introduce an additional parameter β for the number of $(\mu + \ell')$-row layers in the matrix $\boldsymbol{A}$. The global matrix $\boldsymbol{A}$ is then defined as:

$$\underbrace{\boldsymbol{A}}_{\beta(\mu+\ell') \times \lceil \nu/\beta \rceil} := \mathsf{Stack}_{\beta}\big(\underbrace{\boldsymbol{A}_1 | \ldots | \boldsymbol{A}_{n_{\mathsf{pcs}}}}_{(\mu+\ell') \times \nu} \big)$$

where $\nu = \sum_j \nu_j$ is the total number of columns of the $\boldsymbol{A}_j$ matrices and Stack_{β} is the mapping which consists in splitting the columns of a matrix in β subsequent groups and stacking them so that the resulting matrix has $\beta(\mu + \ell')$ rows (instead of $\mu + \ell'$) and $\lceil \nu/\beta \rceil$ columns (instead of ν), possibly padded with all-0 columns.

To open the evaluations $P_1(e), \ldots, P_{n_{\mathsf{pcs}}}(e)$ for a point $e \in \mathbb{F}$, one opens the LVCS evaluations with the coefficient vectors:

$$\boldsymbol{u}_1 \otimes (1, e, \ldots, e^{\mu+\ell'}), \ldots, \boldsymbol{u}_{\beta} \otimes (1, e, \ldots, e^{\mu+\ell'}) \ ,$$

where $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{\beta}$ is the canonical basis of $\mathbb{F}^{\beta}$ and $\otimes$ is the tensor product. Denoting $\boldsymbol{v}_1^{(e)}, \ldots, \boldsymbol{v}_{\beta}^{(e)}$ the LVCS openings associated to these coefficients, i.e. $\boldsymbol{v}_k^{(e)} := \big(\boldsymbol{u}_k \otimes (1, e, \ldots, e^{\mu+\ell'})\big) \cdot \boldsymbol{A}$ for every $k \in [1, \beta]$, one can check that we get

$$(\boldsymbol{v}_1^{(e)} \mid \ldots \mid \boldsymbol{v}_{\beta}^{(e)}) = (\hat{\boldsymbol{v}}_1^{(e)} \mid \ldots \mid \hat{\boldsymbol{v}}_{n_{\mathsf{pcs}}}^{(e)})$$

PCS.Commit:

On input a vector polynomial $\boldsymbol{P} = (P_1, \ldots, P_{n_{\mathsf{pcs}}}) \in \mathbb{F}[X]^{(\leq d_1)} \times \cdots \times \mathbb{F}[X]^{(\leq d_n)}$ for the prover, PCS.Commit consists in the following interactive protocol:

1. For all $1 \leq j \leq n_{\mathsf{pcs}}$, $\mathcal{P}$ picks $\ell'(\nu_j - 1)$ values $r_{j,1,1}, \ldots, r_{j,\nu_j-1,\ell'}$ uniformly at random from $\mathbb{F}$ and defines the matrix $\boldsymbol{A}_j$ following Equation (2) from the those random values and the coefficients $\{a_{j,i}\}_{j,i}$ of the polynomials $P_j(X) = \sum_{i=0}^{d_j} a_{j,i} X^i$.
2. $\mathcal{P}$ and $\mathcal{V}$ run the protocol $\mathsf{LVCS.Commit}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}})$ with

$$\begin{bmatrix} \boldsymbol{r}_1^\top \\ \boldsymbol{r}_2^\top \\ \vdots \\ \boldsymbol{r}_{n_{\mathsf{rows}}}^\top \end{bmatrix} = \mathsf{Stack}_\beta\big(\boldsymbol{A}_1 | \ldots | \boldsymbol{A}_{n_{\mathsf{pcs}}}\big) \ ,$$

   resulting in the commitment $\mathsf{com}_{\mathsf{LVCS}}$ and opening key $\mathsf{key}_{\mathsf{LVCS}}$.

$\mathcal{P}$ returns $\mathsf{key} := (\mathsf{key}_{\mathsf{LVCS}}, \boldsymbol{A}_1, \ldots \boldsymbol{A}_{n_{\mathsf{pcs}}})$ while the commitment (i.e., the protocol transcript) is defined as $\mathsf{com} := \mathsf{com}_{\mathsf{LVCS}}$.

PCS.Eval:

In this protocol, the prover takes as input a PCS opening key $\mathsf{key} = (\mathsf{key}_{\mathsf{LVCS}}, \boldsymbol{A}_1, \ldots \boldsymbol{A}_{n_{\mathsf{pcs}}})$, and the verifier takes as input a commitment com. Both parties additionally take a set of evaluation points $E' \subseteq \mathbb{F}$ and a set of evaluations $\boldsymbol{P}|_{E'} := \{\boldsymbol{P}(e)\}_{e \in E'}$. The protocol runs as follows:

1. $\mathcal{P}$ computes

$$\boldsymbol{v}_k^{(e)} := \big(\boldsymbol{u}_k \otimes (1, e, \ldots, e^{\mu+\ell'})\big) \cdot \boldsymbol{A}$$

   for every $k \in [1, \beta]$ and $e \in E'$, where $\boldsymbol{A} = \mathsf{Stack}_\beta\big(\boldsymbol{A}_1 | \ldots | \boldsymbol{A}_{n_{\mathsf{pcs}}}\big)$ and sends $\{\boldsymbol{v}_1^{(e)}, \ldots, \boldsymbol{v}_\beta^{(e)}\}_{e \in E'}$ to $\mathcal{V}$.
2. For every $j \in [1, n_{\mathsf{pcs}}]$ and $e \in E'$, $\mathcal{V}$ checks that the following equality correctly holds:

$$P_j(e) = \langle \hat{\boldsymbol{v}}_j^{(e)}, (1, e^\mu, \ldots, e^{(\nu_j-1)\mu}, e^{(\nu_j-1)\mu-\delta_j}) \rangle$$

   where $\hat{\boldsymbol{v}}_1^{(e)} \ldots, \hat{\boldsymbol{v}}_{n_{\mathsf{pcs}}}^{(e)}$ are vectors of size $\nu_1, \ldots, \nu_{n_{\mathsf{pcs}}}$ defined as $(\hat{\boldsymbol{v}}_1^{(e)} \mid \ldots \mid \hat{\boldsymbol{v}}_{n_{\mathsf{pcs}}}^{(e)}) = (\boldsymbol{v}_1^{(e)} \mid \ldots \mid \boldsymbol{v}_\beta^{(e)})$ and where $\delta_j := (\mu \cdot \nu_j + \ell') - (d_j + 1)$. If one of those check fails, then $\mathcal{V}$ stops and returns Reject. Otherwise, the protocol continues.
3. $\mathcal{P}$ and $\mathcal{V}$ run the protocol $\mathsf{LVCS.Eval}(\mathsf{key}_{\mathsf{LVCS}}, \mathsf{com}, \mathcal{C}, \{\boldsymbol{v}_1^{(e)}, \ldots, \boldsymbol{v}_\beta^{(e)}\}_{e \in E'})$ with coefficients

$$\mathcal{C} = \{\boldsymbol{u}_k \otimes (1, e, \ldots, e^{\mu+\ell'})\}_{k \in [1,\beta], e \in E'} \ .$$

   If the output is Reject, $\mathcal{V}$ outputs Reject. Otherwise (the output is Accept), $\mathcal{V}$ outputs Accept.

Fig. 3: SmallWood-PCS: Polynomial Commitment Scheme from SmallWood-LVCS.

where $\hat{\boldsymbol{v}}_1^{(e)} \ldots, \hat{\boldsymbol{v}}_{n_{\mathsf{pcs}}}^{(e)}$ are vectors of size $\nu_1, \ldots, \nu_{n_{\mathsf{pcs}}}$ such that $P_j(e) = \langle \hat{\boldsymbol{v}}_j^{(e)}, (1, e^\mu, \ldots, e^{(\nu_j-2)\mu}, e^{(\nu_j-1)\mu-\delta_j}) \rangle$ for every $j \in [1, n_{\mathsf{pcs}}]$. The obtained PCS is formally described in Figure 3 and its security is stated in the following theorem.

Theorem 5. *The PCS depicted in Figure 3 is (t, ε)-polynomial-binding assuming that the underlying LVCS is (t, ε)-function-binding.*

Proof. By definition of the matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{n_{\mathsf{pcs}}}$, a polynomial-binding break of the PCS directly translates to a function-binding break of the LVCS.

Theorem 6. *The LVCS depicted in Figure 3 is (t, ξ)-HVZK provided that the underlying LVCS is (t, ξ)-HVZK.*

Proof. By definition, the $\hat{\boldsymbol{v}}_k^{(e)}$ are uniformly distributed vectors satisfying

$$P_j(e) = \langle \hat{\boldsymbol{v}}_j^{(e)}, (1, e^{\mu}, \ldots, e^{(\nu_j-2)\mu}, e^{(\nu_j-1)\mu-\delta_j}) \rangle$$

for every $j \in [1, n_{\mathsf{pcs}}]$ and $e \in E'$. The HVZK simulator simply samples these vectors randomly and call the HVZK simulator of the LVCS for the rest. □

3.3 Comparison with the State of the Art

In this section, we compare our polynomial commitment scheme with prior works on hash-based polynomial commitments. Our scheme aims to be used with for "small" polynomials, typically of degree less than 2^{16}. This parameter regime differs from the target of succint polynomial commitments with applications to SNARKs/STARKs. For instance, the authors of Brakedown [GLS+23] compare the state of the art for polynomials of degrees ranging from 2^{13} to 2^{29}. This difference of regime has two implications:

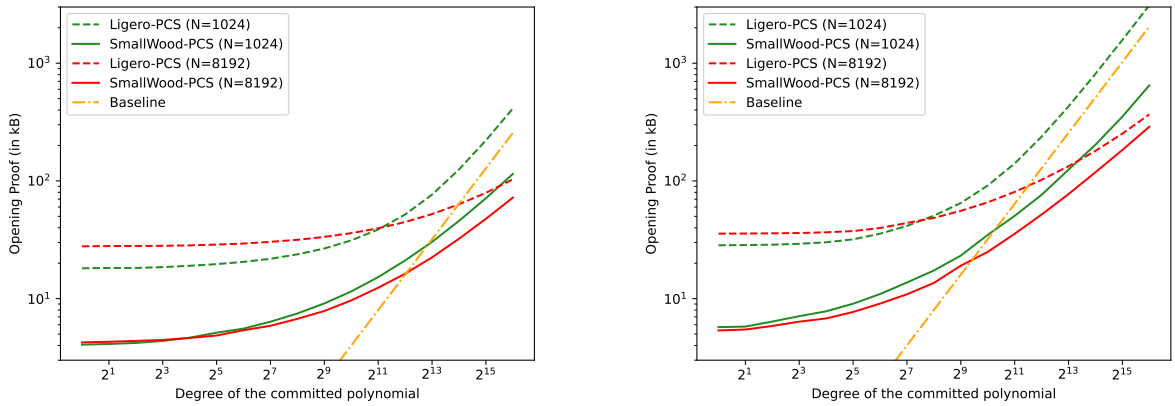
- While focusing on small polynomials, the resulting commitment is often not "succinct" in the sense that it is larger than the committed polynomial. This is not a concern in our context, as our primary goal is to achieve zero-knowledge arguments with small concrete sizes rather than asymptotic succinctness.
- To fairly compare different PCS based on Merkle trees, we need to consider a given size for the evaluation domain $N = |\mathbb{E}|$ (i.e., the number of leaves in the Merkle tree). The policy commonly used in the prior works is to fix the rate of the underlying linear code, i.e., to fix the ratio between the number of committed coefficients per row and the size of the evaluation domain. For example, the authors of Brakedown [GLS+23] use rates equal to 1/4, 1/2 and 38/39. While such a policy makes sense asymptotically, it is irrelevant while focusing on small polynomials, for which it would lead to small Merkle trees and hence larger commitments (because more evaluations must then be opened). For this reason, we consider a fixed size of the evaluation domain $N = |\mathbb{E}|$.

We compare our scheme to Ligero-PCS, the polynomial commitment scheme described in [GLS+23] based on Ligero [AHIV17, AHIV23]. Our scheme is similar to this PCS but relies on the DECS protocol instead of Ligero's proximity test with further adaptation to achieve the hiding property. For a fair comparison, we also integrate our tweaks to make Ligero-PCS hiding (these tweaks are summarized in Appendix A). Figure 4 provides the obtain sizes for the commitment plus one evaluation opening with a 128-bit soundness for 32-bit and 256-bit fields. Let us mention that n_{rows}, the number of rows in the underlying LVCS, is a flexible parameter which we select to minimize the proof size.

We observe that SmallWood-PCS leads to smaller sizes than Ligero-PCS in this parameter regime. This comes from the fact that the DECS outperforms the Ligero's proximity test for small parameters [FR23]. We also observe that the smallest sizes achievable by Ligero-PCS are around 20 KB and around 30 KB (for very small polynomials) for respectively 32-bit and 256-bit fields, while SmallWood-PCS can go as low as 4–6 KB. We also observe that SmallWood-PCS crosses the "succinctness threshold" (size for which is becomes more compact than the committed polynomial) around 2^{12} and 2^{13} for the selected parameters (i.e., $N \in [2^{10}, 2^{13}]$) for 32-bit fields and around 2^9 and 2^{10} for 256-bit fields.

We can explain why Ligero-PCS do not perform well for (very) small polynomials. This is because the number of opened leaves in the Merkle tree is at least 128 when targeting a 128-bit soundness. Indeed, opening ℓ leaves leads to a soundness error of $(\frac{1+\rho}{2})^{\ell}$, where ρ is the rate of the underlying linear code. In our context, the rate is $\rho = \frac{n_{\mathsf{cols}}+\ell}{N}$ where n_{cols} is the row-size parameter of the LVCS and ℓ the number of opened evaluation in the Merkle tree.

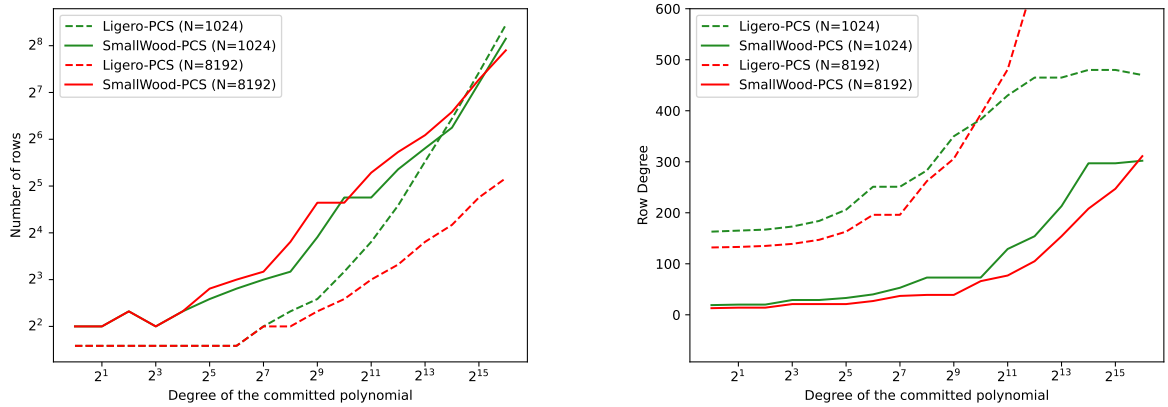
While SmallWood-PCS leads to smaller size than Ligero-PCS, let us stress that the comparison made in Figure 4 is yet in favor of Ligero-PCS. Indeed, while for a given N the computation cost of the Merkle tree is the same for both PCS, we can argue that building the data committed as leaves in the Merkle tree is more expensive in Ligero-PCS than in the DECS. Indeed, as illustrated in Figure 5, while optimizing the internal parameters of the LVCS to minimize the size, Ligero-PCS selects fewer rows leading to row polynomial of higher degrees. Evaluating these larger polynomial is hence computationally more intensive. For example, to commit a degree-64 polynomial over a 256-bit field with $N = 8192$, SmallWood-PCS requires evaluating 8 degree-27 polynomials while Ligero-PCS requires evaluating 3 degree-196 polynomials.



(a) Over a 32-bit field.

(b) Over a 256-bit field.

Fig. 4: Comparison of hash-based hiding PCS with 128-bit soundness. The baseline curve gives the cost of sending the entire committed polynomial.



(a) Number of rows in the LVCS.

(b) Degree of the row polynomials in the LVCS.

Fig. 5: Comparison of the optimal LVCS parameters for SmallWood-PCS and Ligero-PCS over a 256-bit field. Those are the parameters minimizing the sizes which are used for the results depicted in Figure 4b.

Comparison with further schemes. In [GLS+23], the authors propose Brakedown-PCS. This polynomial commitment scheme is a variant of Ligero-PCS in which the underlying Reed-Solomon code is replace by a linear code with linear-time encoding. This tweak does not enables them to achieve smaller sizes (in fact, it tends to produce larger sizes in our regime since their code is not a maximum distance separable (MDS) code), but it leads to a scheme for which the prover/committer's running time in linear in the size of the committed polynomial. Therefore, the committer's running time is asymptotically better than Ligero-PCS and SmallWood-PCS, but in the regime of committing to small polynomials, it presents no advantage.

In [VP19], Vlasov and Panarin propose a polynomial commitment scheme based on the FRI protocol [BBHR18]. However, their scheme is not hiding. As explained in their article, hiding is often "achieved at the application level" in FRI-based protocols,. We can thus not compare our scheme with this PCS, because

our main goal is the zero-knowledge property since we are focusing on small polynomials, and making their scheme hiding would require a non-trivial analysis.

3.4 Zero-Knowledge Arguments from DECS

In this section, we describe zero-knowledge arguments built by the standard approach of combining a *polynomial interactive oracle proof* (PIOP) with a polynomial commitment scheme, and making the result non-interactive using the Fiat-Shamir transform. We formalize the LPPC PIOP protocol which is an abstraction of Ligero's protocol to check arithmetic circuits [AHIV17]. The LPPC terminology stands for *(global) linear and parallel polynomial constraints* which was introduced in [FR23]. By combining the LPPC PIOP with SmallWood-PCS (and Fiat-Shamir), we obtain SmallWood-ARK, a hash-based zero-knowledge argument scheme achieving the smallest sizes of the state of the art for arithmetic circuits of sizes up to 2^{16}.

We first recall the syntax of LPPC statements and formally introduce the LPPC PIOP. We then discuss its combination with different hash-based PCS and argue about the interest of SmallWood-PCS in this context. We further address the parameter selection and the proof size for SmallWood-ARK, the zero-knowledge argument scheme obtained by combining the LPPC PIOP and SmallWood-PCS. We finally showcase the application of SmallWood-ARK to arithmetic circuits.

LPPC Statements. As in [FR23], we consider a general LPPC syntax where polynomial constraints are of arbitrary degrees. In the LPPC PIOP, the prover aims to prove knowledge of a witness w arranged as a matrix

$$w = \begin{bmatrix} w_{1,1} & \dots & w_{1,s} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \dots & w_{n,s} \end{bmatrix}$$

which satisfies two types of constraints:

1. *(parallel polynomial constraints)* they satisfy some polynomial relations:

$$\forall j \in [1, m_1], \forall k \in [1, s], f_j(w_{1,k}, \dots, w_{n,k}, \theta_{j,1,k}, \dots, \theta_{j,n_c,k}) = 0$$

   where $f_1, \dots, f_{m_1}$ are polynomials from $\mathbb{F}[X_1, \dots, X_n, Y_1, \dots, Y_{n_c}]$ of total degree at most d and $\{\theta_{j,i,k}\}$ are constants from $\mathbb{F}$;

2. *(global linear constraints)* they satisfy some linear relations:

$$\forall j \in [1, m_2], \sum_{k \in [1,s], i \in [1,n]} a_{j,i,k} \cdot w_{i,k} = t_k$$

   where $\{a_{j,i,k}\}$ and $\{t_k\}$ are constants from $\mathbb{F}$.

The LPPC statement is hence defined by the m_1 polynomials $\{f_j\}$ and their associated constants $\{\theta_{j,i,k}\}$ and the m_2 linear relations determined by the coefficients $\{a_{j,i,k}\}$ and $\{t_k\}$. From this definition, we see that each polynomial constraint f_j is verified in parallel on each column of the witness matrix (possibly with different constants), while the global linear constraints are verified on all the coefficients of the matrix (i.e. as linear combinations of the flattened witness matrix).

The LPPC Polynomial IOP. We now describe the LPPC PIOP to check such a statement on the witness w. A PIOP is an interactive proof in which the prover can send a *polynomial oracle* $[P_1, \dots, P_n]$ to the verifier for polynomials $P_1, \dots, P_n \in \mathbb{F}[X]$. From such a polynomial oracle, the verifier can then query some evaluations. Namely, for any point $e \in \mathbb{F}$, a query e to the oracle provides the verifier with the polynomial evaluations $P_1(e), \dots, P_n(e)$. The number of queries made by the verifier is fixed by the definition of the PIOP protocol.

Let $\Omega = \{\omega_1, \ldots, \omega_s\}$ some fixed points of $\mathbb{F}$ and let $V_\Omega(X) = \prod_{\omega \in \Omega}(X - \omega)$ the vanishing polynomial of Ω. Let $\mathbb{S} \subseteq \mathbb{F} \setminus \Omega$, the set of evaluation points that can be queried by the verifier (in practice the definition of $\mathbb{S}$ depends on the underlying PCS; $\mathbb{S}$ must exclude Ω for zero-knowledge to be achieved). Let $\tilde{A}_{j,i}(X)$ be the degree-$(s-1)$ polynomial defined by interpolation such that $\tilde{A}_{j,i}(\omega_k) = a_{j,i,k}$ for all k, and let $\Theta_{j,i}(X)$ be the degree-$(s-1)$ polynomial defined by interpolation such that $\Theta_{j,i}(\omega_k) = \theta_{j,i,k}$ for all k.

The LPPC PIOP is detailed in Protocol 6. Below, we explain its high-level idea. The prover interpolates each row $(w_{i,1}, \ldots, w_{i,s})$ of the witness matrix as a polynomial P_i (using Ω as support), i.e. builds $P_i \in \mathbb{F}[X]$ such that

$$P_i(\omega_1) = w_{i,1}, \ldots, P_i(\omega_s) = w_{i,s} \ .$$

The prover then demonstrates the existence of $\hat{\boldsymbol{Q}}_1 = (\hat{Q}_{1,1}, \ldots, \hat{Q}_{1,m_1})$ and $\hat{\boldsymbol{Q}}_2 = (\hat{Q}_{2,1}, \ldots, \hat{Q}_{2,m_2})$ such that

$$\forall 1 \leq j \leq m_1, \quad \hat{Q}_{1,j}(X) \cdot V_\Omega(X) = f_j\left(P_1(X), \ldots, P_n(X), \Theta_{j,1}(X), \ldots, \Theta_{j,n_c}(X)\right) \tag{3}$$

and

$$\forall 1 \leq j \leq m_2, \quad \begin{cases} \hat{Q}_{2,j} & = \sum_{i=1}^{n} \tilde{A}_{j,i}(X) \cdot P_i(X) \\ t_j & = \sum_{h=1}^{s} \hat{Q}_{2,j}(\omega_h) \end{cases} \tag{4}$$

If Equation (3) holds, then:

$$\begin{aligned} f_j(w_{1,k}, \ldots, w_{n,k}, \theta_{j,1,k}, \ldots, \theta_{j,n_c,k}) &= f_j(P_1(\omega_k), \ldots, P_n(\omega_k), \Theta_{j,1}(\omega_k), \ldots, \Theta_{j,n_c}(\omega_k)) \\ &= \hat{Q}_{1,j}(\omega_k) \cdot V_\Omega(\omega_k) && \text{by (3)} \\ &= \hat{Q}_{1,j}(\omega_k) \cdot 0 = 0 \ . \end{aligned}$$

for every $j \in [1, m_1]$ and every $k \in [1, s]$. Namely, the witness satisfies the parallel polynomial constraints. If Equation (4) holds, then:

$$\begin{aligned} \sum_{h \in [1,s], i \in [1,n]} a_{j,i,h} \cdot w_{i,h} &= \sum_{h \in [1,s], i \in [1,n]} \tilde{A}_{j,i}(\omega_h) \cdot P_i(\omega_h) \\ &= \sum_{h \in [1,s]} \hat{Q}_{2,j}(\omega_h) && \text{by (4), first equality} \\ &= t_j \ . && \text{by (4), second equality} \end{aligned}$$

for every $j \in [1, m_1]$ and every $k \in [1, s]$. Namely, the witness satisfies the global linear constraints.

To convince the verifier that (3) and (4) indeed hold, the prover could send $\hat{\boldsymbol{Q}}_1$ and $\hat{\boldsymbol{Q}}_2$. The verifier could then check these relations on randomly chosen evaluation points using the Schwartz-Zippel Lemma. However, sending $\hat{\boldsymbol{Q}}_1$ and $\hat{\boldsymbol{Q}}_2$ directly is inefficient in terms of communication (or proof size). Instead, we rely on batching. The prover computes and sends batched polynomials $\boldsymbol{Q}_1 = (Q_{1,1}, \ldots, Q_{1,\rho})$ and $\boldsymbol{Q}_2 = (Q_{2,1}, \ldots, Q_{2,\rho})$ defined as:

$$Q_{1,k} := \sum_{j=1}^{m_1} \gamma'_{k,j} \cdot \hat{Q}_{1,j} \quad \text{and} \quad Q_{2,k} := \sum_{j=1}^{m_2} \gamma'_{k,j} \cdot \hat{Q}_{2,j}$$

for all $1 \leq k \leq \rho$, where ρ is a flexible parameter and $\{\gamma'_{k,j}\}_{k,j}$ are random verifier-chosen values of $\mathbb{F}$. If $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ satisfy the batched variant of Equations (3) and (4), then $\hat{\boldsymbol{Q}}_1$ and $\hat{\boldsymbol{Q}}_2$ satisfy the original equation with high probability. Upon receiving the batched polynomials, the verifier challenges the prover to open evaluations of the witness polynomials to check the batched equations using the Schwartz-Zippel Lemma.

To ensure zero-knowledge, the witness polynomials $P_1, \ldots, P_n$ incorporate randomness so that the revealed evaluations do not leak information about the witness matrix. Additionally, the batched polynomials $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are masked with random polynomials $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, which are provided to the verifier through the polynomial oracle alongside the witness polynomials.

Soundness. The soundness of the LPPC PIOP holds for the following reasons:

1. $\mathcal{P}$ builds random degree-$(\ell' + s - 1)$ polynomials $P_1, \ldots, P_n$ defined such that
$$\forall\, i \in [1, n],\ P_i(\omega_1) = w_{i,1}, \ldots, P_i(\omega_s) = w_{i,s}\ .$$
2. $\mathcal{P}$ samples and commits ρ random degree-$(d \cdot (\ell' + s - 1) - s)$ polynomials $M_{1,1}, \ldots, M_{\rho,1}$ and ρ random degree-$(\ell' + 2s - 2)$ polynomials $M_{1,2}, \ldots, M_{\rho,2}$ such that $\sum_{k=1}^{s} M_{2,j}(\omega_k) = 0$ for all j.
3. $\mathcal{P}$ sends a polynomial oracle $[\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2]$ to $\mathcal{V}$, where $\boldsymbol{P} = (P_1, \ldots, P_n)$, $\boldsymbol{M}_1 = (M_{1,1}, \ldots, M_{\rho,1})$ and $\boldsymbol{M}_2 = (M_{1,2}, \ldots, M_{\rho,2})$.
4. $\mathcal{V}$ samples a random matrix $\Gamma' = (\gamma'_{k,i})_{k,i} \leftarrow \mathcal{D}(\mathbb{F}^{\rho \times \max(m_1, m_2)})$ and sends it to $\mathcal{P}$, where $\mathcal{D}(\mathbb{F}^{\rho \times \max(m_1, m_2)})$ is the uniform probability distribution over $\mathbb{F}^{\rho \times \max(m_1, m_2)}$.
5. For all $1 \leq k \leq \rho$, $\mathcal{P}$ computes the polynomials:
$$Q_{k,1}(X) := M_{k,1}(X) + \frac{\sum_{j=1}^{m_1} \gamma'_{k,j} \cdot f_j(P_1(X), \ldots, P_n(X), \Theta_{j,1}(X), \ldots, \Theta_{j,n_c}(X))}{V_\Omega(X)}$$
and
$$Q_{k,2}(X) \cdot X + c_k := M_{k,2}(X) + \sum_{j=1}^{m_2} \gamma'_{k,j} \cdot \sum_{i=1}^{n} \tilde{A}_{j,i}(X) \cdot P_i(X)\ ,$$
for c_k the constant term of the right-hand side polynomial.
6. $\mathcal{P}$ sends $\boldsymbol{Q}_1 = (Q_{1,1}, \ldots, Q_{\rho,1})$ and $\boldsymbol{Q}_2 = (Q_{1,2}, \ldots, Q_{\rho,2})$ to $\mathcal{V}$.
7. $\mathcal{V}$ picks ℓ' random points $E' = \{e_1, \ldots, e_{\ell'}\} \subseteq \mathbb{S}$ and queries the oracle $[\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2]$ for evaluations of the polynomials at those points.
8. $\mathcal{V}$ checks the correctness of the following relations for the queried evaluations: for all $1 \leq k \leq \rho$,
$$\begin{cases} V_\Omega(X) \cdot Q_{k,1}(X) = V_\Omega(X) \cdot M_{k,1}(X) \\ \qquad\qquad + \sum_{j=1}^{m_1} \gamma'_{k,j} \cdot f_j(P_1(X), \ldots, P_n(X), \Theta_{j,1}(X), \ldots, \Theta_{j,n_c}(X)) \\ Q_{k,2}(X) \cdot X + c_k = M_{k,2}(X) + \sum_{j=1}^{m_2} \gamma'_{k,j} \cdot \sum_{i=1}^{n} \tilde{A}_{j,i}(X) \cdot P_i(X) \end{cases}$$
where
$$c_k := \frac{1}{s} \cdot \left(\sum_{j=1}^{m_2} \gamma'_{k,j} \cdot t_j - \sum_{h=1}^{s} Q_{2,k}(\omega_h) \cdot \omega_h \right).$$

Fig. 6: The LPPC Polynomial IOP.

1. If (at least) one parallel polynomial constraint is not satisfied, then the polynomial relation involving $Q_{k,1}$ holds with probability at most $1/|\mathbb{F}|$ (over the random choice of the $\{\gamma'_{k,j}\}_j$). Similarly, if (at least) one global linear constraint is not satisfied, then the polynomial relation involving $Q_{k,2}$ can only hold with probability $1/|\mathbb{F}|$ (still over the random choice of the $\{\gamma'_{k,j}\}_j$). To wrap up and since the checking is performed ρ times in parallel using independent randomness, if (at least) one constraint is not satisfied then then both relations hold with probability at most $1/|\mathbb{F}|^\rho$.
2. The relations are checked on ℓ' random points of $\mathbb{S}$. By the Schwartz-Zippel lemma, if one of the relations does not hold then the check passes with probability at most $\binom{d_{\mathsf{REL}}}{\ell'} / \binom{|\mathbb{S}|}{\ell'}$ for d_{REL} the degree of the relation.

Formally, we have the following result:

Theorem 7 (Soundness of the LPPC PIOP). *The LPPC PIOP described in Figure 6 has soundness error:*
$$\varepsilon \leq \frac{1}{|\mathbb{F}|^\rho} + \frac{\binom{d_{\mathsf{REL}}}{\ell'}}{\binom{|\mathbb{S}|}{\ell'}} \quad \textit{with} \quad d_{\mathsf{REL}} = \max(d_{\mathsf{PP}}, d_{\mathsf{GL}}) \quad \textit{for} \quad \begin{cases} d_{\mathsf{PP}} := d \cdot (\ell' + s - 1) \\ d_{\mathsf{GL}} := \ell' + 2s - 2 \end{cases}.$$

Namely, for any polynomial oracle $[\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2]$ with underlying witness $(w_{i,k}) = (P_i(\omega_k))$ that fails to satisfy at least one of the LPPC constraints, the verifier accepts with probability at most ε.

Proof. The verification process of this protocol consists in verifying the polynomial relations

$$V_\Omega(X) \cdot Q_{k,1}(X) = V_\Omega(X) \cdot M_{k,1}(X) + \sum_{j=1}^{m_1} \gamma'_{k,j} \cdot f_j(P_1(X), \ldots, P_n(X), \Theta_{j,1}(X), \ldots, \Theta_{j,n_c}(X)) \ , \quad (5)$$

and

$$Q_{k,2}(X) \cdot X + c_k = M_{k,2}(X) + \sum_{j=1}^{m_2} \gamma'_{k,j} \cdot \sum_{i=1}^{n} \tilde{A}_{j,i}(X) \cdot P_i(X) \ , \quad (6)$$

for all k, by asking the prover to reveal the evaluations of $\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2$ for some random points ($\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ is known by the verifier). The soundness of the protocol hold for the following reasons:

- If (at least) one parallel polynomial constraint is not satisfied, then there exists (j', i') such that $f_{j'}(w_{1,i'}, \ldots, w_{n,i'}, \theta_{j',1,i'}, \ldots) \neq 0$. In that case, for all $1 \leq k \leq \rho$, we have

$$\begin{aligned} &V_\Omega(\omega_{i'}) \cdot M_{k,1}(\omega_{i'}) + \sum_{j=1}^{m_1} \gamma'_{k,j} \cdot f_j(P_1(\omega_{i'}), \ldots, P_n(\omega_{i'}), \Theta_{j,1}(\omega_{i'}), \ldots) \\ &\quad = 0 + \gamma'_{k,j'} \cdot \underbrace{f_{j'}(P_1(\omega_{i'}), \ldots, P_n(\omega_{i'}), \theta_{j',1,i'}, \ldots)}_{\neq 0} + \sum_{j=1, j\neq j'}^{m_1} \gamma'_{k,j} \cdot f_j(P_1(\omega_{i'}), \ldots, P_n(\omega_{i'}), \theta_{j,1,i'}, \ldots) \end{aligned}$$

  which is uniformly random in $\mathbb{F}$ since $\gamma'_{k,j'}$ has been chosen uniformly at random in $\mathbb{F}$ by the verifier. Therefore, since the left-hand term of (5) is always zero when evaluating into $\omega_{i'}$ ($V_\Omega(\omega_{i'}) \cdot Q_{k,1}(\omega_{i'}) = 0 \cdot Q_{k,1}(\omega_{i'}) = 0$), the probability that the relation (5) holds (for a specific k) is at most the probability that it holds for the evaluation point $\omega_{i'}$, and so it is at most $1/|\mathbb{F}|$. By independency of the challenges accross repetitions, the probability that the relation (5) holds for all k is then at most $1/|\mathbb{F}|^\rho$.

- If (at least) one global linear constraint is not satisfied, there exists j' such that $\sum_{h\in[1,s], i\in[1,n]} a_{j',i,h} \cdot w_{i,h} \neq t_k$. In that case, for all $1 \leq k \leq \rho$, we have

$$\begin{aligned} &\sum_{h\in[1,s]} \Big[(Q_{k,2}(\omega_h) \cdot \omega_h + c_k) - \Big(M_{k,2}(\omega_h) + \sum_{j=1}^{m_2} \gamma'_{k,j} \cdot \sum_{i=1}^{n} \tilde{A}_{j,i}(\omega_h) \cdot P_i(\omega_h) \Big) \Big] \\ &\quad = \sum_{j=1}^{m_2} \gamma'_{k,j} \cdot t_j - \sum_{h\in[1,s]} M_{k,2}(\omega_h) - \sum_{j=1}^{m_2} \gamma'_{k,j} \cdot \sum_{i=1}^{n} \sum_{h\in[1,s]} a_{j,i,h} \cdot w_{i,h} \\ &\quad = \gamma'_{k,j'} \cdot \underbrace{\Big(t_{j'} - \sum_{i=1}^{n} \sum_{h\in[1,s]} a_{j',i,h} \cdot w_{i,h} \Big)}_{\neq 0} \\ &\qquad + \sum_{j=1, j\neq j'}^{m_2} \gamma'_{k,j} \cdot \Big(t_j - \sum_{i=1}^{n} \sum_{h\in[1,s]} a_{j,i,h} \cdot w_{i,h} \Big) - \sum_{h\in[1,s]} M_{k,2}(\omega_h) \end{aligned}$$

  which is uniformly random in $\mathbb{F}$ since $\gamma'_{k,j'}$ has been chosen uniformly at random in $\mathbb{F}$ by the verifier. Therefore, the probability that the relation (6) holds for all k is at most the probability that it holds when summing for evaluations in the witness support, and so it is at most $1/|\mathbb{F}|^\rho$.

To sum up, when the witness does not satisfy the LPPC statement, the two relations (5) and (6) hold with probability at most $1/|\mathbb{F}|^\rho$. Then, at the end of the protocol, the verifier checks that both relations are

satisfied for random evaluation points. The Schwartz-Zippel lemma states that if a degree-d_{REL} polynomial relation is not satisfied, then the probability that it holds for ℓ' random distinct evaluation points is at most $\binom{d_{\mathsf{REL}}}{\ell'}/\binom{|\mathbb{S}|}{\ell'}$. By definition of the LPPC protocol, the first relation (checking the parallel polynomial constraints) has degree $d_{\mathsf{PP}} := d \cdot (\ell' + s - 1)$ while the second relation (checking the global linear constraints) has degree $d_{\mathsf{GL}} := \ell' + 2s - 2$. □

Remark 1. In [AHIV17] and [FR23], different verifier challenges are used for batching polynomial constraints and linear constraints. In contrast, as shown in Figure 6, we employ the same challenges for both. This approach does not compromise soundness, as formally proven above.

Zero Knowledge. The zero-knowledge property stands from the masking of the polynomial $\{Q_{k,1}\}$ and $\{Q_{k,2}\}$ (with $\{M_{k,1}\}$ and $\{M_{k,2}\}$) and the randomness of the polynomials $\{P_i\}$ which makes the revealed evaluations independent of the witness. Formally we have:

Theorem 8 (Zero-Knowledge of the LPPC PIOP). *The LPPC PIOP described in Figure 6 is perfectly honest-verifier zero-knowledge. Namely, there exists a simulator which perfectly simulates a transcript of the protocol together with the answers to the polynomial oracle queries without knowledge of the witness w.*

Proof. We build a simulator $\mathcal{S}$ which outputs a perfect distribution of the protocol transcript and associated answers to the polynomial oracle queries for any challenged evaluation points $E' = \{e_1, \ldots, e_{\ell'}\}$ (given to the similator as input):

1. $\mathcal{S}$ samples random degree-$(\ell' + s - 1)$ polynomials $P_1, \ldots, P_n$.
2. For all $1 \leq k \leq \rho$,
   - $\mathcal{S}$ samples a random degree-$(d \cdot (s + \ell' - 1) - s)$ polynomial $Q_{k,1}$ and a random degree-$(\ell' + 2s - 3)$ polynomial $Q_{k,2}$.
   - $\mathcal{S}$ samples random values $\gamma'_{k,1}, \ldots, \gamma'_{k,\max(m_1,m_2)}$.
   - For all $1 \leq h \leq \ell'$, $\mathcal{S}$ computes $M_{k,1}(e_h)$ and $M_{k,2}(e_h)$ such that

$$M_{k,1}(e_h) := Q_{k,1}(e_h) - \frac{\sum_{j=1}^{m_1} \gamma'_{k,j} \cdot f_j(P_1(e_h), \ldots, P_n(e_h), \Theta_{j,1}(e_h), \ldots, \Theta_{j,n_c}(e_h))}{V_\Omega(e_h)}$$

$$M_{k,2}(e_h) := Q_{k,2}(e_h) \cdot e_h + c_k - \sum_{j=1}^{m_2} \gamma'_{k,j} \cdot \sum_{i=1}^{n} \tilde{A}_{j,i}(e_h) \cdot P_i(e_h)$$

     where $c_k := \frac{1}{s} \cdot \left(\sum_{j=1}^{m_2} \gamma'_{k,j} \cdot t_j - \sum_{k=1}^{s} Q_2(\omega_k) \cdot \omega_k \right)$. We stress that $M_{k,1}$ is well defined since $e_h \notin \Omega$, implying that $V_\Omega(e_h) \neq 0$ for all h.
   - $\mathcal{S}$ samples a random degree-$(d \cdot (s + \ell' - 1) - s)$ polynomial $M_{k,1}$ and a random degree-$(\ell' + 2s - 2)$ polynomial $M_{k,2}$ that satisfy the previous evaluations $\{M_{k,1}(e_h)\}_{1 \leq h \leq \ell'}$ and $\{M_{k,2}(e_h)\}_{1 \leq h \leq \ell'}$.
3. $\mathcal{S}$ returns the transcript made of

$$\left\{ \{\gamma'_{k,j}\}_j, Q_{k,1}, Q_{k,2} \right\}_{1 \leq k \leq \rho}, \{e_h, P_1(e_h), \ldots P_n(e_h), M_{1,1}(e_h), \ldots, M_{\rho,2}(e_h)\}_{1 \leq h \leq \ell'} .$$

First, we can observe that the transcript produced by this simulator is valid, i.e. it passes the verification. Now, let us briefly explained why it is perfectly indistinguishable from a real transcript. In both cases (real transcript and simulated transcript):

- the opened evaluations of $P_1, \ldots, P_n$ are uniformly random since these polynomials are interpolated from ℓ' random evaluations (in addition to witness coordinates);
- the polynomials $Q_{k,1}$ and $Q_{k,2}$ are uniformly random. While it is direct in the simulated transcript, it comes from the randomness of $M_{k,1}$ and $M_{k,2}$ in the real transcripts.
- knowing the opened evaluations of $P_1, \ldots, P_n$ and the polynomials $Q_{k,1}$ and $Q_{k,2}$, the opened evaluations of $M_{k,1}$ and $M_{k,2}$ are totally determined thanks to the verification equations.

Underlying PCS. The performance of the proof system obtained by combining the LPPC PIOP with a concrete PCS highly depends on the latter. This composition involves committing to the polynomials P_1, $\ldots$, P_n, $\{M_{k,1}\}_k$, $\{M_{k,2}\}_k$ using the PCS, in place of the polynomial oracle. Subsequently, the evaluation protocol of the PCS is executed, allowing the prover to reveal the evaluations of these polynomials at random points $e_1, \ldots, e_{\ell'}$ while proving to the verifier that these evaluations are consistent with the commitment. This process replaces the verifier's direct queries to the polynomial oracle. To ensure the zero-knowledge property of the PIOP, the underlying PCS must be hiding, such that the verifier learns only the revealed evaluations.

In Ligero [AHIV17], to commit to a vector polynomial $\boldsymbol{P}(X)$, the prover commits to N evaluations $\boldsymbol{P}(e_1), \ldots, \boldsymbol{P}(e_N)$ into a Merkle tree of N leaves. Opening an evaluation then consists in revealing a leaf of the tree along with its authentication path with respect to the committed root. However, the prover should prove that the committed evaluations correspond to the evaluations of a low-degree polynomial. Indeed, without further check, nothing would prevent the prover to commit a polynomial of degree up to $N-1$ which would make the protocol unsound. To proceed, in Ligero the prover runs a proximity test in parallel of the proof system. This approach is pushed further in TCitH-MT [FR23] which relies on the degree-enforcing commitment scheme (DECS) as recalled in Section 3.1. This scheme ensures that the committed evaluations correspond to a polynomial of the right degree which improves the performance of the LPPC proof system for "small-to-medium" instances (while asymptotically Ligero remains better).

In both Ligero and TCitH, only the N committed evaluations of the polynomials (as leaves of the Merkle tree) can be opened. This implies that the set $\mathbb{S}$ in the LPPC PIOP has a cardinality of N. In contrast, SmallWood-PCS supports the opening of committed polynomials at any point within the field. When the field is large enough, this approach offers three significant advantages:

- *Reduced verification cost.* Using the DECS directly as a PCS restricts openings to $|\mathbb{S}| = N$ evaluation points, with N being the size of the Merkle tree. To achieve λ-bit security, at least ℓ' evaluations must be opened, such that $\binom{N}{\ell'} > 2^\lambda$. Employing a standard PCS, however, enables the prover to open committed polynomials at any point in $\mathbb{F}$. Since $\mathbb{F}$ is substantially larger than N in our context, fewer evaluations are required to maintain the same security level. This reduction translates into lower computational costs for the verifier when checking the polynomial relations (Equations (5) and (6)) for the opened evaluations.
- *Lower polynomial degrees.* Using the DECS directly requires increasing the number of evaluations, i.e., using a larger value of ℓ', which leads to a higher degree $s-1+\ell'$ for the committed polynomials. In contrast, employing our PCS reduces the degrees of the committed polynomials which in turns reduces the degrees of the polynomials $\{Q_{k,1}\}_k$, $\{Q_{k,2}\}_k$. This degree reduction lowers the communication overhead associated to these polynomials as well as the computational costs for the verifier; it further improves the soundness.
- *Cheaper DECS opening.* Although SmallWood-PCS is built uppon the DECS, the latter is not used to directly commit the witness polynomials of the LPPC PIOP. Note that for PIOP checking relations of degree d, employing the DECS direcly to commit polynomials results in a soundness error $\binom{d\cdot\ell}{\ell}/\binom{N}{\ell}$. Instead, SmallWood-PCS employs the DECS within an LVCS checking linear relations, i.e., relations of degree $d=1$. This way, fewer evaluations need to be opened in the DECS to reach a given soundness.

Parameters and Proof Sizes. SmallWood-ARK is a zero-knowledge argument scheme for LPPC statements obtained by compiling the LPPC PIOP with SmallWood-PCS and applying the Fiat-Shamir transform. We provide in Appendix C a sketch of the non-interactive scheme after application of the Fiat-Shamir transform. We now discuss the choice of parameters for this scheme and provide underlying proof sizes.

The different parameters of the scheme are summarized in Table 1. Besides the parameters of the LPPC statement, which are given as input of the scheme, the parameters are either chosen (to satisfy the target soundness while, e.g., reducing the proof size) or defined from the other parameters. This is depicted in the last column of the table. For the basic version of SmallWood-ARK, we use the uniform distribution over $\mathcal{D}(\mathbb{F}^{\eta\times n})$ for the DECS and the number of evaluation set for the LPPC PIOP has cardinality $|\mathbb{S}| = |\mathbb{F}| - s$.

Table 1: Parameters of SmallWood-ARK.

Parameters of the LPPC Statement:		
$\mathbb{F}$	Base field	
s	Number of columns in the witness matrix (packing factor)	
n	Number of rows in the witness matrix	
d	Maximal degree of parallel polynomial constraints	
m_1	Number of parallel polynomial constraints	
m_2	Number of global linear constraints	
Parameters of the LPPC PIOP:		
ℓ'	Number of oracle queries (polynomial evaluations)	*chosen*
ρ	Number of parallel repetitions	*chosen*
Parameters of the PCS:		
n_{pcs}	Number of committed polynomials	$n_{\mathsf{pcs}} = n + 2\rho$
$\{d_j\}_{1\le j\le n}$	Degrees of the witness polynomials	$d_j = \ell' + s - 1$
$\{d_j\}_{n<j\le n+\rho}$	Degrees of the masking polynomials $\{M_{k,1}\}_k$	$d_j = d\cdot(\ell'+s-1)-s$
$\{d_j\}_{n+\rho<j\le n+2\rho}$	Degrees of the masking polynomials $\{M_{k,2}\}_k$	$d_j = \ell' + 2s - 2$
μ	Number of coefficient rows in the matrices $\{\boldsymbol{A}_j\}$	*chosen*
$\{\nu_j\}$	Number of columns of the matrices $\{\boldsymbol{A}_j\}$	$\nu_j = \lceil (d_j + 1 - \ell')/\mu \rceil$
β	Stacking factor (number of $(\mu+m)$-row layers in $\boldsymbol{A}$)	*chosen*
Parameters of the LVCS:		
n_{rows}	Number of committed row vectors	$n_{\mathsf{rows}} = \beta(\mu+\ell')$
n_{cols}	Size of committed row vectors	$n_{\mathsf{cols}} = \lceil (\sum_j \nu_j)/\beta \rceil$
m	Number of LVCS queries	$m = \beta\cdot\ell'$
ℓ	Number of DECS evaluation queries	*chosen*
Parameters of the DECS:		
n_{decs}	Number of committed polynomials	$n_{\mathsf{decs}} = n_{\mathsf{rows}}$
d_{decs}	Degree of committed polynomials	$d_{\mathsf{decs}} = n_{\mathsf{cols}} + \ell - 1$
N	Size of the evaluation domain (number of Merkle leaves)	*chosen*
η	Number of parallel repetitions of the degree-enforcing round	*chosen*
κ	Number of bits of proof of work for grinding	*chosen*

Before applying Fiat-Shamir, the 9-round interactive proof obtained by composing the PIOP and the PCS achieves round-by-round soundness with the following soundness errors:

$$\varepsilon_1 = \frac{\binom{N}{d_{\mathsf{decs}}+2}}{|\mathbb{F}|^\eta}\ , \quad \varepsilon_2 = \frac{1}{|\mathbb{F}|^\rho}\ , \quad \varepsilon_3 = \frac{\binom{d(s+\ell'-1)}{\ell'}}{\binom{|\mathbb{F}|}{\ell'}}\ , \quad \varepsilon_4 = \frac{\binom{n_{\mathsf{cols}}+\ell-1}{\ell}}{\binom{N}{\ell}}\ ,$$

where ε_1 corresponds to degree-enforcing soundness error (arising in the DECS commitment), ε_2 corresponds to the first-round soundness error of the PIOP (batching of constraints), ε_3 corresponds to the second-round soundness error of the PIOP (evaluation queries) and ε_4 corresponds to the PCS/LVCS opening soundness error. To achieve a λ-bit security, we constrain the chosen parameters of Table 1 to ensure that these soundness errors are at most $2^{-\lambda}$.

Grinding. To further reduce the size of the DECS opening and minimize the verifier's running time, we employ *grinding* [Sta21]. Specifically, we incorporate a κ-bit proof-of-work into the Fiat-Shamir hash computation for the DECS opening challenge (see Appendix B for details). This approach relaxes the constraint on the final soundness error, which now satisfies $\varepsilon_4 \le 2^{-\lambda+\kappa}$. Consequently, fewer Merkle tree leaves need to be opened, leading to a more compact proof.

Proof size. We employ a standard optimization technique to reduce the size of the proof by selectively revealing elements. Specifically, when a polynomial ($\boldsymbol{R}$ in DECS or $\boldsymbol{Q}_1, \boldsymbol{Q}_2$ in the PIOP) needs to be disclosed

prior to specific evaluations that enable partial reconstruction of the polynomial, a hash commitment to the polynomial is sent instead. Subsequently, after some evaluations have been revealed and used to partially reconstruct the committed polynomial, additional complementary evaluations are disclosed. These additional evaluations allow the full polynomial to be reconstructed and the hash commitment to be verified. The two hash commitments (the one for $\boldsymbol{R}$ and the one for $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$ in the PIOP) are further combined in a single hash commitment.

A SmallWood-ARK proof is composed of the prover-to-verifier messages of the LPPC PIOP (i.e., the polynomials $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$), the answers to the evaluation queries (i.e., polynomials $\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2$ evaluated in E') and underlying PCS proof (commitment and evaluation proof):[4]

$$|\pi| \;=\; \underbrace{2\lambda}_{\text{poly. hash com.}} \;+\; \underbrace{\big[(d-1)\cdot(s+\ell'-1)+2s-2\big]\cdot\rho\cdot\log_2|\mathbb{F}|}_{\boldsymbol{Q}_1,\boldsymbol{Q}_2} \;+\; \underbrace{\ell'\cdot(n_{\mathsf{pcs}}+2\rho)\cdot\log_2|\mathbb{F}|}_{\text{Answers to evaluation queries}} \;+\; |\pi_{\mathsf{pcs}}|$$

where $|\pi_{\mathsf{pcs}}|$ denotes the size of the PCS proof. As explained above, the polynomials $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are first sent through a hash commitment and then recomputed by interpolation using the polynomial evaluation queries. This tweak saves ℓ' coefficients per polynomial while adding a 2λ-bit hash digest to the proof.

Then, the PCS proof π_{pcs} is made of the linear combinations $\{\boldsymbol{v}_1^{(e)},\dots,\boldsymbol{v}_\beta^{(e)}\}_{e\in E'}$ and underlying LVCS proof (commitment and evaluation proof):

$$|\pi_{\mathsf{pcs}}| \;=\; \underbrace{\Big[\ell'\cdot\sum_{i=1}^{n_{\mathsf{pcs}}}(\nu_i-1)\Big]\cdot\log_2|\mathbb{F}|}_{\{\boldsymbol{v}_1^{(e)},\dots,\boldsymbol{v}_\beta^{(e)}\}_{e\in E'}} \;+\; |\pi_{\mathsf{lvcs}}|$$

where $|\pi_{\mathsf{lvcs}}|$ denotes the size of the LVCS proof. While the total size of the vectors $\boldsymbol{v}_1^{(e)},\dots,\boldsymbol{v}_\beta^{(e)}$ for one evaluation point $e\in E'$ is $\sum_{j=1}^{n_{\mathsf{pcs}}}\nu_j$, we can save one coefficient per polynomial which can be retrieve using the underlying polynomial evaluation (already sent in the PIOP proof). This explains the -1 in the above formula.

Then, the LVCS proof π_{lvcs} is made of the vectors $\bar{\boldsymbol{v}}_1,\dots,\bar{\boldsymbol{v}}_m$ (where $m=\beta\cdot\ell'$) and the polynomial evaluations $\boldsymbol{P}|_E$ and underlying DECS proof (commitment and evaluation proof):

$$|\pi_{\mathsf{lvcs}}| \;=\; \underbrace{m\cdot\ell\cdot\log_2|\mathbb{F}|}_{\bar{\boldsymbol{v}}_1,\dots,\bar{\boldsymbol{v}}_m} \;+\; \underbrace{\ell\cdot(n_{\mathsf{rows}}-m)\cdot\log_2|\mathbb{F}|}_{\boldsymbol{P}|_E} \;+\; |\pi_{\mathsf{decs}}|$$

where $|\pi_{\mathsf{decs}}|$ denotes the size of the DECS proof. Simarly to $\boldsymbol{v}_1^{(e)},\dots,\boldsymbol{v}_\beta^{(e)}$ in PCS, the verifier can retrive $\ell\cdot m\cdot\log_2|\mathbb{F}|$ coefficients of $\boldsymbol{P}|_E$, so the prover does not need to send them.

Finally, the DECS proof is made of the degree-enforcing vector polynomial $\boldsymbol{R}$, the masking polynomial evaluations $\boldsymbol{M}|_E$, the authentication paths to the opened leaves, and the opened random tapes $\{\rho_j\}_{e_j\in E}$:

$$|\pi_{\mathsf{decs}}| \;=\; \underbrace{\eta\cdot(d_{\mathsf{decs}}+1-\ell)\cdot\log_2|\mathbb{F}|}_{\boldsymbol{R}} \;+\; \underbrace{\ell\cdot\eta\cdot\log_2|\mathbb{F}|}_{\boldsymbol{M}|_E} \;+\; \underbrace{2\lambda\cdot|\mathsf{auth}_{\ell,N}|}_{\text{Authentication paths}} \;+\; \underbrace{\ell\cdot\lambda}_{\text{Random tapes}}$$

where $\mathsf{auth}_{\ell,N}$ denotes the number of nodes in the authentication paths for the opening of ℓ leaves in a Merkle tree with N leaves (omitting the root). As for the polynomials sent by the prover in the PIOP, the vector polynomial $\boldsymbol{R}$ is also first sent through a hash commitment and then recomputed by interpolation using the revealed evaluation queries. This saves ℓ coefficients per polynomial R_k.

Application to Arithmetic Circuits. Similarly to Ligero [AHIV17] and TCitH-MT [FR23], our scheme provide sublinear arguments for arithmetic circuits. For this purpose, an arithmetic circuit statement $C(X)=$

[4] The proof size for $\boldsymbol{Q}_1,\boldsymbol{Q}_2$ is obtained as follows. $\boldsymbol{Q}_1$ has $d(\ell'+s-1)-s+1$ coefficients. Q_2 has $\ell'+2s-2$ coefficients. For both polynomials ℓ' coefficients are saved using the opened evaluations (as discussed above).

y is expressed as an LPPC statement as follows: the witness w is defined as the concatenation of three vectors a, b and c whose coordinates a_j, b_j and c_j are respectively the first operand, the second operand and the result of the j^{th} multiplication gate of C on input x. Then proving $C(x) = y$ is similar to proving $c = a \circ b$, where $\circ$ is the coordinate-wise multiplication, and proving the linear constraints of the circuit, which can be expressed as $a = A_1 \cdot w$, $b = A_2 \cdot w$ and $y = A_3 \cdot w$ for some matrices A_1, A_2, A_3. Denoting $|C|$ the number of multiplication gates in C and assuming that $|C|$ is a multiple of s, the witness (of size $|w| = 3|C|$) is arranged so that we have $m_1 = |C|/s$ quadratic polynomial constraints of the form $f(a, b, c) = c - a \cdot b$ to verify (each of them verifying s multiplication gates in parallel). We further have $m_2 = 2|C| + |y|$ global linear constraints to verify (for the computation of a, b and y).

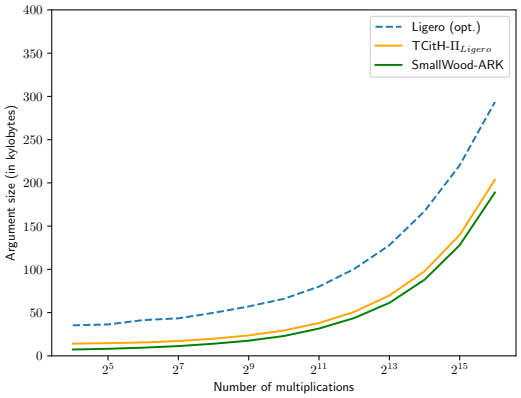
We provide hereafter the proof sizes of SmallWood-ARK applied to arithmetic circuits over 32-bit and 256-bit fields. We further compare these results to Ligero [AHIV17, AHIV23] (with optimization from [FR23]) and TCitH-Π_{Ligero}, i.e., the Threshold-Computation-in-the-Head framework applied to the Ligero protocol [FR23] (which makes a direct use of the DECS as polynomial commitment scheme). Since the number of Merkle leaves N is the parameter that impacts the most the prover running time (most of the computation being devoted to the evaluation and hashing of the leaves of the Merkle tree), we use the same value of N for the three schemes and rely on Ligero's formula for the selection of N with respect to the circuit size $|C|$. For the other parameters of SmallWood-ARK, we fix μ to $d_j + 1 - \ell' = s$, for $d_j = \ell' + s - 1$ the degree of the PIOP witness polynomials. This ensures that each witness polynomial fits into exactly one column of the LVCS (i.e., $\nu_j = 1$ for witness polynomials). This choice eliminates the need for additional randomness while committing the witness polynomials in the LVCS. We note that the masking polynomials of the PIOP are still committed through several columns (since they have larger degrees) and still require additional randomness. The parameters β and s are then chosen such that the proof size is minimized. Here, we do not use grinding (i.e., we set $\kappa = 0$) to get a fair comparison with prior works.

Figures 7a and 7c provides the proof sizes of the three schemes for 32-bit fields and 256-bit fields respectively. Table 2 further provides some concrete parameters and sizes for circuits of size $|C| = 2^8$. We observe that TCitH and SmallWood-ARK clearly outperform Ligero. While SmallWood-ARK is only slightly better than TCitH for 32-bit fields, the improvement becomes more significant for larger fields. For example, it almost halves the proof size for circuits of size 2^8 on 256-bit fields. The reason why the gain is mitigated on small fields is because there is less advantage of using a full PCS as in SmallWood-ARK instead of directly using the DECS as PCS as in TCitH (since N is closer to $|\mathbb{F}|$).

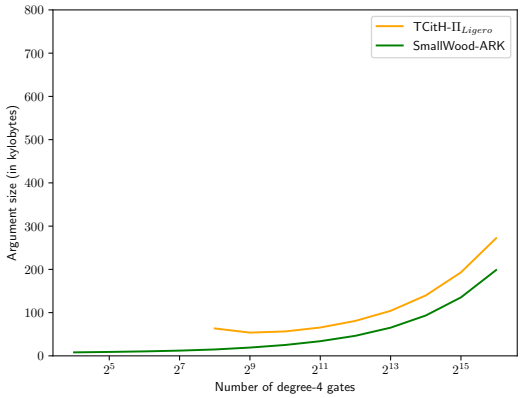
Table 2: Parameters and proof sizes for Ligero [AHIV17], TCitH-Π_{Ligero} [FR23] and SmallWood-ARK for arithmetic circuits of size $|C| = 2^8$.

Scheme	$\lvert\mathbb{F}\rvert$	$\lvert x\rvert$	$\lvert C\rvert$	Degree	N	ℓ	s	η	ℓ'	ρ	Size
Ligero (opt.)	2^{32}	100	2^8	2	975	219	106	–	–	4	49.7 kB
TCitH-Π_{Ligero}					975	40	23	15	–	4	19.8 kB
SmallWood-ARK					975	27	19	13	5	4	14.0 kB
Ligero (opt.)	2^{256}	100	2^8	2	975	219	106	–	–	1	147.6 kB
TCitH-Π_{Ligero}					975	56	55	3	–	1	56.0 kB
SmallWood-ARK					975	37	12	3	1	1	33.4 kB
TCitH-Π_{Ligero}	2^{32}	100	2^8	4	975	79	8	18	–	4	63.4 kB
SmallWood-ARK					975	28	16	14	5	4	14.4 kB
TCitH-Π_{Ligero}	2^{256}	100	2^8	4	975	79	8	3	–	1	295.9 kB
SmallWood-ARK					975	37	12	3	1	1	34.9 kB

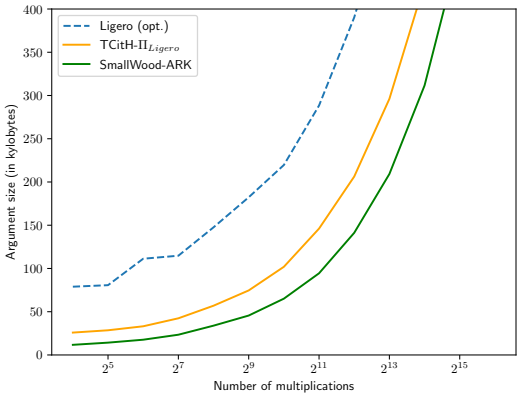
Besides performing better on large fields, another key advantage of SmallWood-ARK over TCitH emerges when the degree d of LPPC constraints exceeds 2. While this scenario does not apply to standard arithmetic circuits discussed earlier, it is precisely relevant in the CAPSS framework introduced hereafter (and to many further contexts where the proof size needs to be optimized). The degree d of LPPC polynomial constraints



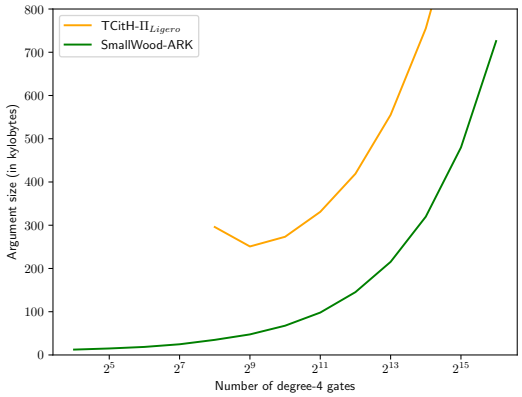
(a) Proof sizes for 32-bit fields.



(b) Proof sizes for 32-bit fields (with degree-4 gates).



(c) Proof sizes for 256-bit fields.



(d) Proof sizes for 256-bit fields (with degree-4 gates).

Fig. 7: Comparison of Ligero [AHIV17], TCitH-Π_{Ligero} [FR23] and SmallWood-ARK for arithmetic circuits (with input $x \in \mathbb{F}^{100}$).

significantly impacts the proof size for TCitH, while having only a minimal effect on SmallWood-ARK. This allows for statements with higher degrees to be handled without substantial increases in proof size. To illustrate, consider arithmetic circuits featuring non-linear gates of degree 4 (with fan-in 2) instead of the usual degree-2 multiplication gates. Using the same parameters as before, we compare SmallWood-ARK and TCitH under these conditions in Figures 7b and 7d, as well as Table 2. (Results for Ligero are omitted, as it is defined only for degree-2 gates.) In this context, we observe that SmallWood-ARK achieves significantly better proof sizes compared to TCitH. For instance, with $|C| = 2^8$ and $|\mathbb{F}| = 2^{32}$, the proof size for SmallWood-ARK is 14.4 KB, an increase of less than 1 KB over the degree-2 case, whereas the proof size for TCitH triples, growing from 19 KB to 62 KB. For a 256-bit field, the disparity is even more significant: TCitH's proof size increases from 56 KB to 295 KB, while SmallWood-ARK's proof size remains almost unchanged, with only a 1 KB increase.

Remark 2. We can observe a strange behavior for the proof sizes of TCitH-Π_{Ligero} when considering degree-4 gates for a small number of gates. This behavior is inherited from the choice of the parameter N which scales with the size of the circuit as specified in Ligero [AHIV17]. For small circuits, it is not possible to find a number ℓ of opened leaves such that the soundness error $\binom{4(s+\ell-1)}{\ell} / \binom{N}{\ell}$ is negligible, because N is too small.

Beyond $|C| = 2^8$, N is large enough but there is no flexibility on the choice of the packing parameter s, which explains why we observe greater proof sizes than for $|C| = 2^9$.

Comparison with zero-knowledge arguments based on GGM trees. Recently, very compact zero-knowledge arguments based on symmetric cryptography (PRG and hash functions) have been constructed using the MPC-in-the-Head paradigm and GGM trees. The most prominent examples include the VOLE-in-the-Head framework [BBD+23a] and its variant TCitH-GGM [FR23]. While these schemes achieve highly compact proofs for small circuits and statements –such as those used in post-quantum signature designs (e.g., [FR23, BFG+24, OTX24, HJ24])– their linear proof size quickly renders them non-competitive compared to SmallWood-ARK as the witness size increases. Specifically, for 128-bit soundness, the proof size of VOLEitH (or TCitH-GGM) can be lower bounded by $\tau \cdot |w|$, where τ represents the number of parallel repetitions (typically $\tau \approx 10$ for GGM trees of size $\approx 2^{12}$). For standard arithmetic circuits, this leads to a proof size of approximately $30 \cdot |C| \cdot \log |\mathbb{F}|$. To illustrate this with the examples presented in Table 2, the resulting proof sizes would be 31 KB for $|C| = 2^8$ and $|\mathbb{F}| = 2^{32}$ and 246 KB for $|C| = 2^8$ and $|\mathbb{F}| = 2^{256}$.

4 The CAPSS Framework

4.1 Overview

The CAPSS framework compiles an arithmetization-oriented family of permutations $\mathcal{P}$ into a signature scheme. The transformation is overviewed in Figure 8. The family of permutations $\mathcal{P}$ is used to define three cryptographic primitives: a one-way function (OWF), Merkle trees (MT) and an extendable output hash function (XOF). The one-way function is the underlying hard problem for the signature scheme. Specifically, for an initialization value iv and secret input value w (for witness), the secret and public keys (sk, pk) of the scheme are defined as:

$$sk = w \quad \text{and} \quad pk = (iv, y) \quad \text{where} \quad y = \mathsf{OWF}_{iv}(w)\,.$$

A signature is a non-interactive zero-knowledge (ZK) argument of knowledge (ARK) of a secret input w which maps to the public output y through OWF_{iv}.

For the latter we use SmallWood-ARK, the ZK-ARK scheme for LPPC statements introduced in Section 3, which combines the LPPC polynomial IOP with our DECS-based polynomial commitment scheme, SmallWood-PCS. The LPPC syntax is ideal to efficiently arithmetize a permutation-based OWF; we provide general arithmetization techniques for such primitives. The Merkle trees in SmallWood-PCS and the XOF used to hash the leaves and for the Fiat-Shamir transform are both built from $\mathcal{P}$. The obtained ZK-ARK is further tweaked in several ways towards SNARK friendliness. The goal of those tweaks is to make the underlying verification algorithm efficient to arithmetize while giving rise to a low number of R1CS constraints.

The rest of the section is organized as follows. We first describe the three permutation-based primitives underlying the CAPSS framework (Section 4.2). We then describe the LPPC arithmetization of the permutation-based OWF (Section 4.3). Next, we introduce different tweaks to make the verification process of the proof system SNARK-friendly (Section 4.4). We finally depict the obtained signature scheme in detail (Section 4.5) and provide concrete instances and their performances (Section 4.6).

4.2 Permutation-Based Primitives

As overviewed previously, the CAPSS framework relies on the following modes and permutation-based primitives:

- extendable output hash functions (XOF) using the Sponge mode [BDPV08],
- Merkle trees (MT) with XOF compression functions using the Jive mode [BBC+23],
- one-way functions (OWF) using truncation.

We formally introduce the considered families of permutations and the aforementioned modes of operations hereafter.

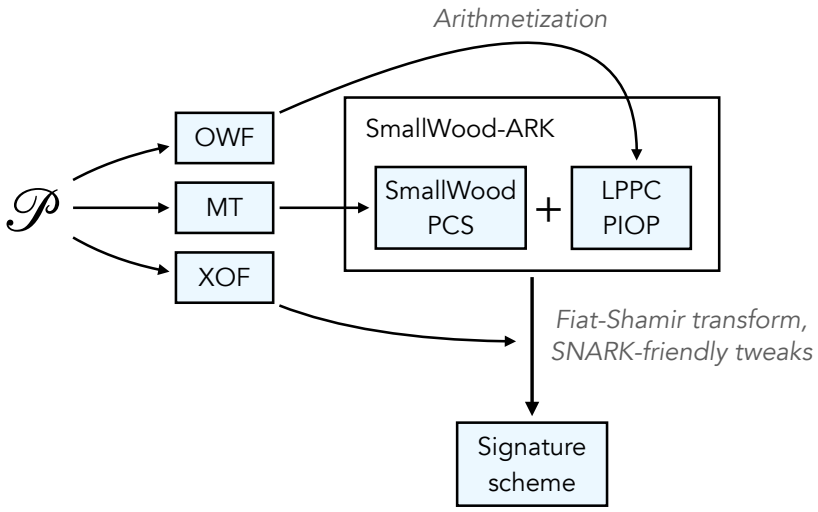


Fig. 8: Overview of the CAPSS framework to compile a permutation family $\mathcal{P}$ into a signature scheme.

Family of permutations. An arithmetization-oriented family of permutations is a set of bijective functions

$$\mathcal{P} = \{P_{t,q} : \mathbb{F}_q^t \to \mathbb{F}_q^t \mid q \in \mathcal{Q}, t \in \mathcal{T}\}$$

defined with respect to a set of admissible field sizes $\mathcal{Q} \subseteq \mathbb{N}$ and a set of admissible state sizes $\mathcal{T} \subseteq \mathbb{N}$. In the following, we will sometimes keep the state size t and the field size q implicit and simply denote P the considered permutation from $\mathcal{P}$.

We specifically consider permutations which are constructed by iterating a number of rounds. We say that the permutation has a *regular iterated construction* with n_r rounds if it can be expressed as

$$P(x) = R_{n_r-1} \circ \cdots \circ R_1 \circ R_0(x) \quad \text{with} \quad R_i(x) = F(x, c_i)$$

for a round permutation $F : \mathbb{F}_q^t \times \mathbb{F}_q^{n_c} \to \mathbb{F}_q^t$ and round constants $c_0, \ldots, c_{n_r-1} \in \mathbb{F}_q^{n_c}$. We say that the permutation has an *irregular iterated construction* with n_p partial rounds and n_f full rounds if it can be expressed as

$$P(x) = R_{n_f+n_p-1} \circ \cdots \circ R_0(x) \quad \text{with} \quad R_i(x) = \begin{cases} F_f(x, c_i) & \text{if } i \in [0, n_f/2) \cup [n_f/2 + n_p, n_f + n_p) \\ F_p(x, c_i) & \text{if } i \in [n_f/2, n_f/2 + n_p) \end{cases}$$

for a full round permutation $F_f : \mathbb{F}_q^t \times \mathbb{F}_q^{n_c} \to \mathbb{F}_q^t$, a partial round permutation $F_p : \mathbb{F}_q^t \times \mathbb{F}_q^{n_c} \to \mathbb{F}_q^t$ and round constants $c_0, \ldots, c_{n_r-1} \in \mathbb{F}_q^{n_c}$. A round permutation F (resp. F_f, F_p) has verification function $G : \mathbb{F}_q^{|v|} \times \mathbb{F}_q^t \times \mathbb{F}_q^t \times \mathbb{F}_q^{n_c} \to \mathbb{F}_q^{n_G}$ (resp. G_f, G_p) if for all $x, y \in \mathbb{F}_q^t$ and $c \in \mathbb{F}^{|c|}$, we have

$$y = F(x, c) \quad \Leftrightarrow \quad \exists v \in \mathbb{F}_q^{|v|} : G(v, x, y, c) = 0 \ .$$

The arithmetization-oriented feature of a permutation $P \in \mathcal{P}$ is informally captured by requiring that a equality $y = P(x)$ can be efficiently verified using a small number of arithmetic constraints. In the iterated setting, with underlying verification function G, verifying $y = P(x)$ translates to verifying the constraint system

$$\begin{cases} x_0 = x, \ x_{n_r} = y \\ \forall i \in [0, n_r) \ , \ \exists v_i \ : \ G(v_i, x_i, x_{i+1}, c_i) = 0 \end{cases}$$

We now introduce the three modes that we consider in our framework (XOF, MT and OWF). They will all make use of the truncation function

$$\mathsf{Tr}_u : \mathbb{F}_q^t \to \mathbb{F}_q^u$$

which returns the u first coordinates of its input vector.

Sponge-based (extendable output) hash functions. We consider the Sponge mode of operation [BDPV08] with the tweak from [Hir18].[5] Given a permutation family $\mathcal{P}$, a target security level λ and a field size q, we select a permutation $P \in \mathcal{P}$ of state size $t \geq \log_2 q/2\lambda + 1$ and define the *capacity* and the *rate* as the parameters

$$c = \left\lceil \frac{\log_2 q}{2\lambda} \right\rceil \quad \text{and} \quad r = t - c \ .$$

For these parameters and for an input length n_{in} and an output length n_{out}, the Sponge-based XOF function

$$\mathsf{XOF}_{\mathcal{P}} : m \in \mathbb{F}_q^{n_{\text{in}}} \mapsto z \in \mathbb{F}_q^{n_{\text{out}}}$$

is defined as follows. The message is split into $n'_{\text{in}} := \lceil n_{\text{in}}/r \rceil$ blocks $m_0, \ldots, m_{n'_{\text{in}}-1} \in \mathbb{F}_q^r$ which are defined as:

$$\begin{cases} (m_0 \parallel m_1 \parallel \ldots \parallel m_{n'_{\text{in}}-1}) = m & \text{if } r \mid n_{\text{in}} \\ (m_0 \parallel m_1 \parallel \ldots \parallel m_{n'_{\text{in}}-1}) = (m \parallel (1, 0, \ldots, 0)) & \text{if } r \nmid n_{\text{in}} \end{cases}$$

From these blocks, the Sponge mode iteratively processes state variables $s_0, s_1, \ldots, s_{n'_{\text{in}}+n'_{\text{out}}} \in \mathbb{F}_q^t$ as follows:

$$s_i = \begin{cases} (m_0 \parallel 0^c) & \text{if } i = 0 \\ P(s_{i-1}) + (m_i \parallel 0^c) & \text{if } i \in [1, n'_{\text{in}}) \\ P(s_{n'_{\text{in}}-1}) + (m_{n'_{\text{in}}} \parallel \sigma) & \text{if } i = n'_{\text{in}} \\ P(s_{i-1}) & \text{if } i \in (n'_{\text{in}}, n'_{\text{in}} + n'_{\text{out}}] \end{cases} \quad \text{with} \quad \sigma = \begin{cases} 1 & \text{if } r \mid n_{\text{in}} \\ 0 & \text{if } r \nmid n_{\text{in}} \end{cases}$$

Finally the output is composed of $n'_{\text{out}} := \lceil n_{\text{out}}/r \rceil$ blocks $z_0, \ldots, z_{n'_{\text{out}}-1} \in \mathbb{F}_q^r$ such that

$$z = \mathsf{Tr}_{n_{\text{out}}}(z_0 \parallel \ldots \parallel z_{n'_{\text{out}}-1}) \quad \text{with} \quad z_i = \mathsf{Tr}_r(s_{n'_{\text{in}}+1+i}) \ , \ \forall i \in [0, n'_{\text{out}}) \ .$$

It is well known that if the permutation P is modeled as a random permutation, then $\mathsf{XOF}_{\mathcal{P}}$ is indifferentiable from a random oracle [BDPV08]. The security of our signature schemes shall thus hold under a random oracle assumption for $\mathsf{XOF}_{\mathcal{P}}$.

Domain separation. To enforce domain separation in CAPSS, we additionally tweak the sponge mode by redefining σ as follows:

$$\sigma^{(i)} = 2 \cdot i + \begin{cases} 1 & \text{if } r \mid n_{\text{in}} \\ 0 & \text{if } r \nmid n_{\text{in}} \end{cases}$$

for the i-th call to the XOF denoted $\mathsf{XOF}_{\mathcal{P}}^{(i)}$ (assuming that $i < q/2$).

Merkle trees with Jive compression. Merkle trees are derived from $\mathcal{P}$ by using two underlying primitives:

- a (fixed-output) hash function for the leaves: we use the Sponge-based $\mathsf{XOF}_{\mathcal{P}}$ function as introduced above,
- a (α-arity) compression function for the nodes of the tree.

[5] The tweak proposed in [Hir18] consists in the introduction of the constant $\sigma \in \{0, 1\}$ in the capacity to deal with message of length n_{in} divisible by the rate r without adding a full additional block $(1, 0 \ldots, 0)$.

Several approaches have been considered in the literature to build such a compression function from aritmetization-oriented permutations. Such a function is a collision-resistant function which maps α hash digests $x_0, \ldots, x_{\alpha-1} \in \mathbb{F}_q^c$ to 1 hash digest $y \in \mathbb{F}_q^c$ (where here hash digests are tuples of field elements). A possible strategy proposed by Poseidon [GKR+21] is to rely on the Sponge mode with a single permutation, with capacity c and rate $r = bc$, so that $y = \mathsf{Tr}_c(P(x_0 \parallel \ldots \parallel x_{\alpha-1} \parallel 0^c))$, this strategy is not optimal as it requires a permutation of state size $t = (\alpha + 1)c$ to deal compress an input of size bc, which is due to the capacity parameter of the Sponge mode. A better strategy is to rely on Davies-Meyer construction to avoid this loss. This is the approach followed by the Jive mode [BBC+23].

For a compression parameter α, and a state size t divisible by α, the Jive mode turns a arithmetization-oriented permutation $P : \mathbb{F}_q^t \to \mathbb{F}_q^t$ into a α-to-1 compression function $\mathsf{Jive}_{\mathcal{P}} : \mathbb{F}_q^t \to \mathbb{F}_q^c$ with $c = t/\alpha$. It can be summarized as summing the α blocks of size c composing the output of P', the Davies-Meyer transformation of P. Formally:

$$\mathsf{Jive}_{\mathcal{P}}(x) = \sum_{i=0}^{\alpha-1} P'_i(x)$$

with $P'_i : \mathbb{F}_q^t \to \mathbb{F}_q^c$ the coordinate functions of P' defined as:

$$(P'_0(x), \ldots, P'_{\alpha-1}(x)) = P'(x) = P(x) + x \ .$$

For a security parameter λ, our instantiations of the Jive mode must have an output size of at least 2λ bits. In other words, the output of the $\mathsf{Jive}_{\mathcal{P}}$ compression function must be of size $c = \left\lceil \frac{\log_2 q}{2\lambda} \right\rceil$ (which coincides with the capacity parameter of the Sponge mode, hence we keep the same notation). We then select a permutation with state size $t = \alpha \cdot c$ given the target compression parameter α. We might use several instantiations of the Jive mode corresponding to different compression parameter α. We shall denote $\mathsf{Jive}_P^{(\alpha)}$ the compression function with compression parameter α when we wish to make it explicit.

We consider Merkle trees using Jive compression functions of possibly different arities at the different layer of the tree. Namely, a Merkle tree with H layers is defined with respect to arities $\alpha_1, \ldots, \alpha_L$. It hashes $N = \prod_{i=1}^{H} \alpha_i$ leaves from $\mathbb{F}_q^c$ into one root in $\mathbb{F}_q^c$:

$$\mathsf{MerkleTree} : (x_0, \ldots, x_{N-1}) \in (\mathbb{F}_q^c)^N \mapsto y \in \mathbb{F}_q^c \ .$$

At layer $j \in [0, H]$, the state of the tree is composed of N_j digests on $\mathbb{F}_q^c$ where $N_0 = 1$ (the root), $N_L = N$ (the leaves) and $N_j = \prod_{i=1}^{j} \alpha_j$ for $j \in [1, H]$. The N_{j-1} digests of layer $j-1$, denoted $s_0^{(j-1)}, \ldots, s_{N_{j-1}-1}^{(j-1)}$, are computed from the N_j digests of layer j, denoted $s_0^{(j)}, \ldots, s_{N_j-1}^{(j)}$, as follows

$$s_i^{(j-1)} = \mathsf{Jive}_{\mathcal{P}}(s_{i\alpha_j}^{(j)} \parallel s_{i\alpha_j+1}^{(j)} \parallel \ldots \parallel s_{(i+1)\alpha_j-1}^{(j)}) \ , \ \forall i \in [0, N_{j-1}) \ ,$$

with $(s_0^{(H)}, \ldots, s_{N-1}^{(H)}) = (x_0, \ldots, x_{N-1})$ and $y = s_0^{(0)}$.

One-way function using truncation. To derive a one-way function from $\mathcal{P}$, we use truncation. Specifically, for an input size $|x|$ and an output size $|y|$, the one-way function is defined with respect to an initialization value $iv \in \mathbb{F}_q^{|iv|}$ as follows:

$$\mathsf{OWF}_{\mathcal{P},iv} \ : \ x \in \mathbb{F}_q^{|x|} \ \mapsto \ y = \mathsf{Tr}_{|y|}(P(iv, x)) \in \mathbb{F}_q^{|y|}$$

where $P \in \mathcal{P}$ is of state size $t = |iv| + |x|$. We define those sizes as follows:

$$|x| = |y| = |iv| = \left\lceil \frac{\lambda}{\log_2 q} \right\rceil \ .$$

These paramters ensure the λ-bit security of the above one-way function under the hardness of solving the *constrained-input constrained-output* (CICO) problem [BDPA11, BBL+24]. Moreover, assuming a number of users (substantially) lower than $2^{\lambda/2}$, we should get no (or very few) collisions on the initialization value and hence further obtain nearly λ bits of multi-user security for the one-way function.

4.3 Arithmetization

As overviewed in Section 4.1, the CAPSS framework relies on SmallWood-ARK, the zero-knowledge argument scheme for LPPC statements introduced in Section 3. In the LPPC syntax (see Section 3.4), the witness is an $n \times s$ matrix which is proved to satisfy polynomial constraints applied to each column of the witness matrix in parallel and linear constraints applied globally on the (flattened) witness matrix. In our context, the *arithmetization* of a permutation $P \in \mathcal{P}$ is the process of expressing a statement $(y, z) = P(iv, x)$ as an LPPC statement, where x and z are secret (part of the witness) and iv and y are public (part of the LPPC boundary constraints). In what follows, we propose two different LPPC arithmetization techniques applying to a wide-range of arithmetization-oriented families of permutations $\mathcal{P}$.

Arithmetization of Regular Permutations. We focus on the case of *regular permutations* which we define as permutations with a regular iterated round structure. Precisely, a regular permutation P has the following form:

$$P(\,\cdot\,) = R_{n_r-1} \circ \ldots R_1 \circ R_0(\,\cdot\,) \quad \text{with} \quad R_i(\,\cdot\,) = F(\,\cdot\,, c_i)$$

for a round permutation F and round constants $c_0, \ldots, c_{n_r-1}$, where n_r is the number of rounds. Many arithmetization-oriented families of permutations match this format, such as RescuePrime [AAB+20, SAD20], Griffin [GHR+23] and Anemoi [BBC+23]. In the following, we shall denote $x_0, \ldots, x_{n_r} \in \mathbb{F}^t$ the successive states arising in a statement $(y, z) = P(iv, x)$. Namely, we have:

$$\forall\, 0 \le i \le n_r,\ x_i = \begin{cases} (iv, x) & \text{if } i = 0 \\ F(x_{i-1}, c_{i-1}) & \text{otherwise} \end{cases}$$

and $x_{n_r} = (y, z)$. We further let G be a *verification function* of the round function F, that is a polynomial function satisfying:

$$x_{i+1} = F(x_i, c_i) \iff \exists\, v_i \in \mathbb{F}^{|v|} : G(v_i, x_i, x_{i+1}, c_i) = 0\ ,$$

and denote α the degree of G. It is usual for arithmetization-oriented permutations to have such a verification function with low degree α. While an obvious verification function is $G(x_i, x_{i+1}, c_i) = x_{i+1} - F(x_i, c_i)$, which has same degree as F, some permutation designs rely on a round function F of large degree that has a verification function G of low degree α (making the SNARK verification of the function efficient).

For the sake of simplicity, let us first consider that the LPPC packing factor s (i.e., the number of columns in the LPPC witness matrix) divides the number of rounds. We will relax this assumption later on. Namely, there exists $b \in \mathbb{N}$ such that $n_r = b \cdot s$. We then define n, the number of rows in the LPPC witness matrix, to be

$$n := (b+1) \cdot t + b \cdot |v|\ .$$

The LPPC witness matrix is defined as:

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,s} \\ w_{2,1} & \ldots & & w_{2,s} \\ \vdots & & & \vdots \\ w_{n,1} & w_{n,2} & \ldots & w_{n,s} \end{bmatrix} := \begin{bmatrix} x_0 & x_b & \ldots & x_{(s-1)b} \\ x_1 & x_{b+1} & \ldots & x_{(s-1)b+1} \\ \vdots & \vdots & & \vdots \\ x_b & x_{2b} & \ldots & x_{s \cdot b} \\ v_0 & v_b & \ldots & v_{(s-1)b} \\ \vdots & \vdots & & \vdots \\ v_{b-1} & v_{2b-1} & \ldots & v_{s \cdot b-1} \end{bmatrix}\ .$$

In the right-hand side of the above equation, the x_i's are column vectors of length t and the v_i's are column vectors of length $|v_i| := |v|$, while the $w_{i,j}$ coefficients of the left-hand side are field elements. We well have a matrix with $n = (b+1) \cdot t + b \cdot |v|$ rows.

Let us now define the LPPC constraints to be verified on the above witness matrix to ensure that the statement $(y, z) = P(iv, x)$ is satisfied. We have $m_1 = b$ parallel polynomial constraints $f_0, \ldots, f_{b-1}$, defined as:

$$f_j : (\underbrace{\hat{x}_0, \ldots, \hat{x}_b, \hat{v}_0, \ldots, \hat{v}_{b-1}}_{\text{witness column}}, \underbrace{\hat{c}_0, \ldots, \hat{c}_{b-1}}_{\text{constants}}) \mapsto G(\hat{v}_j, \hat{x}_j, \hat{x}_{j+1}, \hat{c}_j)$$

for every $j \in [0, b-1]$. Applied to the first column of the witness matrix with constants $c_0, \ldots, c_{b-1}$, these polynomial constraints check the correctness of the b first state transitions $x_0 \to x_1 \to \cdots \to x_b$. Indeed, by definition, we have:

$$f_j(x_0, \ldots, x_b, v_0, \ldots, v_{b-1}, c_0, \ldots, c_{b-1}) \quad \Leftrightarrow \quad x_{j+1} = F(x_j, c_j) \ .$$

In the same way, applied to the k^{th} column of the witness matrix with constants $c_{(k-1)b}, \ldots, c_{kb-1}$, these polynomial constraints check the correctness of state transitions $x_{(k-1)b} \to x_{(k-1)b+1} \to \cdots \to x_{kb}$.

This way, using $m_1 = b$ parallel polynomial constraints, we verify all the state transitions. However, we should still make sure that subsequent columns are consistent namely that the vector x_b of the first witness column is equal to the vector x_b of the second witness column and, more generally, that the vector x_{kb} in the k^{th} column equals the vector x_{kb} in the $(k+1)^{\text{th}}$ column for every $k \in [1, s-1]$. To this purpose, we can use the following global linear constraints:

$$\forall\, i \in [0, t-1], \forall\, k \in [1, s-1] \ : \ w_{bt+i,k} - w_{i,k+1} = 0 \ .$$

Combining the previous parallel polynomial constraints and the above global linear constraints, the witness is ensured to satisfy $x_{n_r} = R_{n_r-1} \circ \ldots R_1 \circ R_0(x_0)$. An additional $|x| + |iv|$ global linear constraints, the *boundary constraints*, need to be added in order to check

$$\begin{cases} \mathsf{Tr}_{|iv|}(x_0) = iv \\ \mathsf{Tr}_{|y|}(x_{n_r}) = y \end{cases}$$

which finally ensures that there exists $x \in \mathbb{F}^{|x|}$ and $z \in \mathbb{F}^{t-|y|}$ such that $(y, z) = P(iv, x)$. We this obtain an LPPC statement for $(y, z) = P(iv, x)$ with $m_1 = b$ parallel polynomial constraints and $m_2 = (s-1)\cdot t + |iv| + |y|$ global linear constraints.

Let us now assume that the packing factor s does not divide the number n_r of rounds. In that case, we take $b \in \mathbb{N}$ minimal such that $n_r \leq b \cdot s$ and we proceed exactly as previously while padding the witness for $x_{n_r+1}, \ldots, x_{b\cdot s}$. We should just be careful that the padded values do not prevent to the witness matrix of satisfying the polynomial constraints. One possible option is to define $x_{n_r+1}, \ldots, x_{b\cdot s}$ as

$$\begin{aligned} x_{n_r+1} &= F(x_{n_r}, 0), \\ &\vdots \\ x_{b\cdot s} &= F(x_{b\cdot s-1}, 0). \end{aligned}$$

S-Box-Centric Arithmetization. We now propose an alternative, simpler, arithmetization technique for permutations which do not have a regular structure but rely on a single unitary S-box $S : \mathbb{F} \to \mathbb{F}$. Namely, we consider a permutation P which is solely composed of $\mathbb{F}$-linear operations and n_{sbx} calls to S. We can for instance express the Poseidon permutation [GKR+21] in this format (while it is not a regular permutation due to the usage of two different type of rounds: the full rounds and the partial rounds).

Let G be a degree-α verification function of the S-box S, which satisfies:

$$y_i = S(x_i) \iff \exists\, v_i \in \mathbb{F}^{|v|} : G(v_i, x_i, y_i) = 0$$

for some $|v| \in \mathbb{N}$. The idea of the S-box-centric arithmetization is that the witness contains the inputs and outputs of all the S-box calls in P. Then using the parallel polynomial constraints, we can batch the

verification of the S-box relations, while using global linear constraints we can check the $\mathbb{F}$-linear operations and the boundary relations.

Let us assume that the packing factor divides the number of S-boxes, i.e. there exists $n' \in \mathbb{N}$ such that $n_{\mathsf{sbx}} = s \cdot n'$. We will relax this assumption later on. For the $((i-1)\cdot s + (j-1) + 1)^{\text{th}}$ S-box with $1 \leq i \leq n'$ and $1 \leq j \leq s$, we denote its input $x_{i,j}$, its output $y_{i,j}$ and its verification witness $v_{i,j}$ such that $G(v_{i,j}, x_{i,j}, y_{i,j}) = 0$. The LPPC witness matrix is defined as follows:

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,s} \\ w_{2,1} & \dots & & w_{2,s} \\ \vdots & & & \vdots \\ w_{n,1} & w_{n,2} & \dots & w_{n,s} \end{bmatrix} := \left[\begin{array}{c|c|c|c} v_{1,1} & v_{1,2} & & v_{1,s} \\ x_{1,1} & x_{1,2} & \dots & x_{1,s} \\ y_{1,1} & y_{1,2} & & y_{1,s} \\ \hline v_{2,1} & v_{2,2} & & v_{2,s} \\ x_{2,1} & x_{2,2} & \dots & x_{2,s} \\ y_{2,1} & y_{2,2} & & y_{2s} \\ \hline \vdots & \vdots & & \vdots \\ \hline v_{n',1} & v_{n',2} & & v_{n',s} \\ x_{n',1} & x_{n',2} & \dots & x_{n',s} \\ y_{n',1} & y_{n',2} & & y_{n',s} \end{array}\right] ,$$

with $n := (2 + |v|) \cdot n'$. For $1 \leq j \leq n'$, the j^{th} parallel polynomial constraint is

$$\forall 1 \leq k \leq s,\ G(v_{j,k}, x_{j,k}, y_{j,k}) = 0 \ .$$

The j^{th} parallel polynomial constraint simultaneously implies $y_{j,1} = S(x_{j,1}), \dots, y_{j,s} = S(x_{j,s})$. The n' polynomial constraints thus verify the input-output relation for all the S-boxes. It remains to check the $\mathbb{F}$-linear operations of the permutations. By definition of the permutation P, there exists two matrices $A_1^{(P)}, A_2^{(P)} \in \mathbb{F}^{(n_{\mathsf{sbx}}-t)\times n_{\mathsf{sbx}}}$ and a vector $b^{(P)} \in \mathbb{F}^{n_{\mathsf{sbx}}-t}$ such that

$$A_1^{(P)} \cdot (x_{1,1} \dots x_{1,s} \dots x_{n',1} \dots x_{n',s})^\top = A_2^{(P)} \cdot (y_{1,1} \dots y_{1,s} \dots y_{n',1} \dots y_{n',s})^\top + b^{(P)} \ . \tag{7}$$

The global linear constraints check the n_{sbx} linear relations induced by (7), together with the boundary linear constraints of the form

$$\begin{cases} iv = A_3^{(P)} \cdot (x_{1,1} \dots x_{1,s} \dots x_{n',1} \dots x_{n',s})^\top + b'_P \\ y \ = A_4^{(P)} \cdot (y_{1,1} \dots y_{1,s} \dots y_{n',1} \dots y_{n',s})^\top + b''_P \end{cases}$$

for some matrices $A_3^{(P)} \in \mathbb{F}^{|iv|\times n_{\mathsf{sbx}}}$, $A_4^{(P)} \in \mathbb{F}^{|y|\times n_{\mathsf{sbx}}}$ and vectors $b'_P \in \mathbb{F}^{|iv|}$, $b''_P \in \mathbb{F}^{|y|}$. This makes a total of $m_1 = n'$ parallel polynomial constraints and $m_2 = n_{\mathsf{sbx}} - t + |iv| + |y|$ global linear constraints.

Let us now assume that the packing factor s does not divide the number n_{sbx} of calls to S. In that case, we take $n' \in \mathbb{N}$ minimal such that $n_{\mathsf{sbx}} \leq s \cdot n'$ and we proceed exactly as previously while padding the witness for $(v_{i,j}, x_{i,j}, y_{i,j})$ when $(i-1)\cdot s + (j-1) + 1 > n_{\mathsf{sbx}}$. We should just be careful that the padded values do not prevent to the witness matrix of satisfying the polynomial constraints. One possible option is to define those $(v_{i,j}, x_{i,j}, y_{i,j})$ as

$$(v_{i,j}, x_{i,j}, y_{i,j}) = (v_0, 0, y_0) \ ,$$

where $y_0 := S(0)$ and v_0 is the value satisfying $G(v_0, 0, y_0) = 0$.

4.4 SNARK-Friendly Verification

As mentionned previously, we use the SmallWood-ARK scheme introduced in Section 3.4 in the CAPSS framework. The goal of this framework is to build SNARK-friendly post-quantum signature schemes featuring lightweight verification algorithm while expressed as an arithmetic circuit or in terms of SNARK constraints for some arithmetization system, typically R1CS. By inspecting the verification process in a CAPSS signature (see Section 4.5), one can see that the bootleneck comes from the verification of the Merkle authentication paths as well as the XOF calls to hash the Merkle leaves and generate verifier challenges through Fiat-Shamir. In what follows, we propose several design tweaks to mitigate this bottleneck and reduce the number of SNARK constraints arising in the verification algorithm.

Parameter Trade-offs in Merkle Trees. The main part of the verification algorithm in terms of SNARK constraints is the decommitment of the polynomial evaluations in the DECS, i.e., the verification of the underlying Merkle authentication paths. Let us denote $P_1, \ldots, P_{n_{\mathsf{decs}}}$ the polynomials given as inputs of the DECS (which differ from the witness polynomials following the PCS design – see Section 3). The prover needs to compute

$$u_i \leftarrow \mathsf{XOF}(P_1(e_i), \ldots, P_{n_{\mathsf{decs}}}(e_i), M_1(e_i), \ldots, M_\eta(e_i)) \tag{8}$$

for all $i \in [1, N]$, for some polynomials $M_1, \ldots, M_\eta$. Then, the prover computes a Merkle tree which has $u_1, \ldots, u_N$ as leaves. From their side, the verifier receives ℓ evaluations

$$\{P_1(e_i), \ldots, P_{n_{\mathsf{decs}}}(e_i), M_1(e_i), \ldots, M_\eta(e_i)\}_{i \in I} \ ,$$

with $|I| = \ell$, and computes their hash digests $\{u_i\}_{i \in I}$ using (8). Then, they need to verify that $\{u_i\}_{i \in I}$ are indeed the committed values, by checking their authentication paths in the Merkle tree.

This verification computation cost can be lowered in two different ways:

– *Lowering the leaf hash computation.* In the verification algorithm, since we open ℓ evaluations in the DECS (used as subroutine of the PCS), we need to recompute ℓ hash digests. To mitigate this cost, we can use the packing factor to decrease the size of the hash input. Namely, we increase the degree d_{decs} of the witness polynomials to reduce their number n_{decs}.

More precisely, since we commit a small number of witness polynomials and since the packing factor s is flexible with our arithmetization technique (see Section 4.3), we will take the stacking factor β equal to 1 and use the packing factor s to parametrize the shape of the matrix committed through SmallWood-LVCS. Moreover, to minimize the number of coefficients in this matrix, we take $\nu_j = 1$ and $\nu_j \cdot \mu + m = (s + \ell' - 1) + 1$ for $1 \leq j \leq n$, i.e. each degree-$(s + \ell')$ witness polynomial can be encoded using a single column for $\boldsymbol{A}_j$. Since $m = \ell'$, this implies that $\mu = s$, $\nu_j \approx d$ [6] for $n < j \leq n + \rho$, and $\nu_j = 2$ for $n + \rho < j \leq n + 2\rho$. Therefore, we have

$$\begin{aligned} n_{\mathsf{decs}} &= n_{\mathrm{rows}} = \beta \cdot (\mu + m) = 1 \cdot (s + \ell') = s + \ell' \\ d_{\mathsf{decs}} &= n_{\mathrm{cols}} + \ell - 1 = \left\lceil \frac{\sum_j \nu_j}{\beta} \right\rceil + (\ell - 1) = \sum_{j=1}^{n} \nu_j + \sum_{j=n+1}^{n+\rho} \nu_j + \sum_{j=n+\rho+1}^{n+2\rho} \nu_j + (\ell - 1) \\ &\approx n \cdot 1 + \rho \cdot d + \rho \cdot 2 + \ell - 1 = n + \rho \cdot (d + 2) + \ell - 1 \ . \end{aligned}$$

As detailed in Section 4.3, in the arithmetization for regular permutations, we obtain $n = \left\lceil \frac{n_r}{s} \right\rceil (t + |v|) + t$ where n_r, t and $|v|$ are the number of permutation rounds, the permutation state size and the size of the witness for the verification of the round function, respectively. In the S-box-centric arithmetization, we have $n = (2 + |v|) \cdot \left\lceil \frac{n_{\mathrm{sbx}}}{s} \right\rceil$ where n_{sbx} and $|v|$ are the number of S-boxes and the size of the witness for the verification of the S-box, respectively. In both cases, when s decreases, the number n_{decs} of the DECS' input polynomials decreases while their degree d_{decs} increases. Thus, we can decrease the packing factor to lower the cost of recomputing the ℓ Merkle leaves (which scales with n_{decs}). The main drawback of this approach is to degrade the PCS opening soundness error $\varepsilon_4 = \binom{d_{\mathsf{decs}}}{\ell} / \binom{N}{\ell}$. One then needs to take a larger ℓ to compensate this security loss, thus increasing the cost of recomputating the leaves and increasing the cost of checking authentication paths in the Merkle tree. For this reason, the packing factor s can only be increased until reaching this threshold.

– *Shortening the Merkle authentication paths.* While using binary Merkle trees is optimal is terms of size (lowering the number of hash digests in an authentication path), using Merkle trees with larger arity decreases the length of the path in terms of number of hash nodes and hence in terms of SNARK constraints in the verification. We hence consider Merkle trees of arity possibly greater than 2, which provides a trade-off between signature size (shortened with binary trees) and SNARK constraints (reduced with larger arity). We further consider that the arity parameter might vary with the node depth in the tree for more flexibility in the parameter selection.

[6] We have the equality when $\ell' = 1$.

Trimming Authentication Paths. Once the verifier recomputed all the opened leaves (by hashing the opened values), he needs to check those leaves using the authentication paths. Since verifying authentication paths in a Merkle tree is not constant time due to the path merging, preventing us to write the verification algorithm as an arithmetic circuit, we will need to verify each path one by one. To proceed, one needs to decompress the merged authentication paths into ℓ individual authentication paths. This decompression can be made by the verifier before calling the verification algorithm. Then the arithmetization circuit will take those ℓ individual authentication paths as inputs, each path enabling us to recompute the Merkle root from one opened leaf.

However, this strategy is not optimal in terms of computation. Since we check each leaf independently, we are recomputing somes nodes several times. For example, we recompute the root from its children ℓ times (at each independent checking). In fact, more a node is close to the root, more often it will be recomputed. We propose an alternative strategy. Let us have an additional parameter $\gamma_{\mathsf{MT}} \in \{0, \ldots, H\}$. Instead of taking the ℓ individual authentication paths, the circuit inputs will contains all the nodes of depth γ_{MT}, together with the ℓ truncated authentication paths that enables to recompute the corresponding depth-γ_{MT} nodes from the opened leaves. The verification then recomputes the root from the depth-γ_{MT} nodes *only once* and checks each truncated individual authentication path independently. While the simplest strategy computes $\gamma_{\mathsf{MT}} \cdot \ell$ nodes of depth lower than γ_{MT}, the alternative strategy computes $\sum_{i=0}^{\gamma_{\mathsf{MT}}-1} \prod_{j=0}^{i-1} \alpha_j$ nodes, where α_j denotes the arity of the nodes at depth j in the tree.

To able the verifier to compute all the nodes at depth γ_{MT}, we should slightly tweak the algorithm that generates the authentication paths. Indeed, without tweaking it, the verifier has no guarantee to be able to recompute all those nodes because the paths could merge very quickly (i.e. close to the leaves). We describe all the routines for the Merkle tree in Figure 9:

- The routine MerkleTree computes the root from the N leaves. It does not require to be tweaked.
- The routine GetAuthPath computes the authentication paths to able the verifier to recompute the depth-γ_{MT} nodes. The only difference with the non-tweaked algorithm is that the loops (Items 2 and 4) over the Merkle layers should stop earlier, at the depth-γ_{MT} layer instead than at the root layer. Even if those tweaked path enables the verifier to recompute the depth-γ_{MT} nodes, let us stress that it does not prevent him to recompute the root (by simply recomputing the head of the Merkle tree).
- The routine RetrieveRootFromPath recomputes the root from the tweaked authentication paths. As the previous routine, the only difference is that the layer loops at Items 2 and 5 stop earlier.

Decomposition of the DECS Opening Challenge. Let us investigate how to arithmetize the DECS opening when working over a 256-bit field and targeting a 128-bit soundness. In that setting, the verifier's challenge will be a single field element v, i.e. the corresponding sponge-based hash function will output a single field element v. We want to transform v as the opening of ℓ leaves of the Merkle tree, while supporting a κ-bit proof-of-work as mentioned in Section 3.4. We decompose v as follows:

$$v = \sum_{j=0}^{\ell-1} \left(\sum_{i=0}^{h_{\mathsf{MT}}-1} \left(\sum_{k=0}^{\alpha_i - 1} b_{j,i,k} \cdot k \right) \cdot \prod_{k=0}^{i-1} \alpha_i \right) N^j + \left(\sum_{j=0}^{n_{\text{decomp}}-1} b'_j \cdot 2^j \right) N^\ell$$

where

- $b_{j,i,k}$ and b'_j are binary values for all (i, j, k),
- $b_{j,i,0} + \ldots + b_{j,i,\alpha_i} = 1$ for all (i, j),
- $\sum_{i=0}^{h_{\mathsf{MT}}-1} \left(\sum_{k=0}^{\alpha_i - 1} b_{j,i,k} \cdot k \right) \cdot \prod_{k=0}^{i-1} \alpha_i$ are distinct for all j,
- $N = \alpha_0 \times \ldots \times \alpha_{h_{\mathsf{MT}}-1}$ is the number of leaves of each Merkle tree,
- n_{decomp} is the larger value such that $2^{\kappa + n_{\text{decomp}}} \cdot N^\ell \leq |\mathbb{F}|$.

$\mathsf{MerkleTree}(c_1, \ldots, c_N)$:

On input N leaves $c_1, \ldots, c_N$:

1. Set $\mathsf{tree}[h_{\mathsf{MT}}][0] = c_1, \ldots, \mathsf{tree}[h_{\mathsf{MT}}][N-1] = c_N$.
2. For h from $h_{\mathsf{MT}} - 1$ downto 0, for i from 0 to N_h, compute
$$\mathsf{tree}[h][i] = \mathsf{Jive}\left(\mathsf{tree}[h+1][i \cdot \alpha_h] \parallel \ldots \parallel \mathsf{tree}[h+1][(i+1) \cdot \alpha_h - 1]\right).$$
3. Set $\mathsf{root} = \mathsf{tree}[0][0]$.
4. Return $(\mathsf{tree}, \mathsf{root})$.

$\mathsf{GetAuthPath}(\mathsf{tree}, I)$:

On input a Merkle tree tree and a query index set $I := \{i_1, \ldots, i_\ell\}$:

1. Set $\mathsf{missing} \leftarrow \{(h_{\mathsf{MT}}, i), i \notin I\}$.
2. For h from $h_{\mathsf{MT}} - 1$ downto γ_{MT}, for i from 0 to N_h,
   - If $L := \{(h+1, i \cdot \alpha_h + j), 0 \leq j < \alpha_h\} \subset \mathsf{missing}$, update $\mathsf{missing}$ as
$$\mathsf{missing} \leftarrow (\mathsf{missing} \backslash L) \cup \{(h, i)\}.$$
3. Set $\mathsf{auth} \leftarrow \emptyset$.
4. For h from $h_{\mathsf{MT}} - 1$ downto γ_{MT}, for i from 0 to N_h,
   - If $(h, i) \in \mathsf{missing}$, update auth as
$$\mathsf{auth} \leftarrow (\mathsf{auth} \parallel \mathsf{tree}[h][i]).$$
5. Return auth.

$\mathsf{RetrieveRootFromPath}(\{c_{i_j}\}_j, \mathsf{auth}, I)$:

On input a query index set $I := \{i_1, \ldots, i_\ell\}$, the opened leaves $\{c_{i_j}\}_j$ and an authentication path:

1. Set $\mathsf{missing} \leftarrow \{(h_{\mathsf{MT}}, i), i \notin I\}$.
2. For h from $h_{\mathsf{MT}} - 1$ downto γ_{MT}, for i from 0 to N_h,
   - If $L := \{(h+1, i \cdot \alpha_h + j), 0 \leq j < \alpha_h\} \subset \mathsf{missing}$, update $\mathsf{missing}$ as
$$\mathsf{missing} \leftarrow (\mathsf{missing} \backslash L) \cup \{(h, i)\}.$$
3. Set $\mathsf{tree}[h][i] = \mathsf{null}$ for all (h, i).
4. Set $\mathsf{tree}[h_{\mathsf{MT}}][i_1 - 1] = c_{i_1}, \ldots, \mathsf{tree}[h_{\mathsf{MT}}][i_\ell - 1] = c_{i_\ell}$.
5. For h from $h_{\mathsf{MT}} - 1$ downto γ_{MT}, for i from 0 to N_h,
   - If $(h, i) \in \mathsf{missing}$, update auth as
$$(\mathsf{tree}[h][i] \parallel \mathsf{auth}) \leftarrow \mathsf{auth}.$$
6. For h from $h_{\mathsf{MT}} - 1$ downto 0, for i from 0 to N_h
   - If $\mathsf{tree}[h+1][i \cdot \alpha_h] \neq \mathsf{null}$, compute
$$\mathsf{tree}[h][i] = \mathsf{Jive}\left(\mathsf{tree}[h+1][i \cdot \alpha_h] \parallel \ldots \parallel \mathsf{tree}[h+1][(i+1) \cdot \alpha_h - 1]\right).$$
7. Set $\mathsf{root} = \mathsf{tree}[0][0]$.
8. Return root.

Fig. 9: Merkle-tree routines in the CAPSS framework. We denote N the number of leaves, H the height, α_j the arity at depth j, N_j the number of nodes at depth j ($N_0 = 1, N_1 = \alpha_0, \ldots, N_H = \alpha_1 \times \ldots \times \alpha_{H-1} = H$). Moreover γ_{MT} is an additional parameter allowing the verifier to recover all the depth-γ_{MT} nodes from the authentication paths.

Using this decomposition, the j^{th} opened leaf in the Merkle tree is the leaf of index

$$\mathsf{ind}_j := \sum_{i=0}^{h_{\text{MT}}-1} \left(\sum_{k=0}^{\alpha_i - 1} b_{j,i,k} \cdot k \right) \cdot \prod_{k=0}^{i-1} \alpha_i \ .$$

The goal of having a more refined decomposition is to ease the verification of the authentication path: the revealed child of the node at depth i is the child number $\sum_{k=0}^{\alpha_i-1} b_{j,i,k} \cdot k$, where $(b_{j,i,0}, \ldots, b_{j,i,\alpha_i})$ is an elementary vector (all the coefficients are zero, except one which is one). This decomposition enables us to efficiently verify a Merkle node: let us denote x the value of the child number $\sum_{k=0}^{\alpha_i-1} b_{j,i,k} \cdot k$ and $y_1, \ldots, y_{\alpha_i}$ all the children of the parent node. To check that the child is well-located as the other children, we can check that

$$\forall 0 \leq k < \alpha_i, \ y_k = b_{j,i,k} \cdot x \ .$$

We can observe that we can decompose any number from $\mathcal{S} := \{0, \ldots, 2^{n_{\text{decomp}}} \cdot N^\ell - 1\}$ except those leading two identical opened leaves (i.e. where $\mathsf{ind}_i = \mathsf{ind}_j$ for some $i \neq j$). It means that all the other values will be rejected as proof-of-work, as described in Section 3.4. The rejection rate is then $\frac{|\mathcal{S}|}{|\mathbb{F}|} \approx \frac{N^\ell \cdot 2^{n_{\text{decomp}}}}{|\mathbb{F}|}$, which satisfied

$$2^{-\kappa-1} \leq \frac{N^\ell \cdot 2^{n_{\text{decomp}}}}{|\mathbb{F}|} \leq 2^{-\kappa}$$

by definition of n_{decomp}.

Batching using Powers. Besides the DECS opening challenge (which we addressed above) and the PCS opening challenge which is fairly simple to handle (only one or a few points of $\mathbb{F}$), the proof system includes a few random combination challenges from the verifier: the challenge of the degree-enforcing round and the challenges for the two relations of the LPPC PIOP protocol. After application of the Fiat-Shamir transform, all the underlying random coefficients are generated through a (sponge-based) extendable-output function (XOF). The lower the number of those generated random coefficients, the lower the number of permutation calls in the Fiat-Shamir XOF computation. The degree-enforcing test in the DECS consists of computing and revealing (η times) a polynomial R built as

$$R(X) := M(X) + \sum_{j=1}^{n_{\text{dec}}} \gamma_j \cdot P_j(X) \ ,$$

where $\{\gamma_j\}_j$ is the required randomness. While this technique has been introduced where $\{\gamma_j\}_j$ are chosen uniformly at random, we here use the common strategy to define $\{\gamma_j\}_j$ as $\{\gamma^j\}_j$ for some random value γ.[7] This strategy drastically reduces the number of random coefficients generated by the XOF but slightly impact the soundness of the commitment. Namely, the degree-enforcing soundness error becomes

$$\varepsilon_1 = \binom{N}{d_\beta + 2} \cdot \left(\frac{n_{\text{dec}}}{|\mathbb{F}|} \right)^\eta$$

instead of $\binom{N}{d_\beta+2} / |\mathbb{F}|^\eta$ before – see Theorem 1.

We can use the same strategy for the randomness of the LPPC PIOP. For all $1 \leq k \leq \rho$, the verifier samples a random value γ'_k and the prover computes the polynomials $Q_{k,1}$ and $Q_{k,2}$ such that

$$Q_{k,1}(X) := M_{k,1}(X) + \frac{\sum_{j=1}^{m_1} \gamma'^j_k \cdot f_j(P_1(X), \ldots, P_n(X), \Theta_{j,1}(X), \ldots, \Theta_{j,n_c}(X))}{V_\Omega(X)} \ ,$$

$$Q_{k,2}(X) \cdot X + c_k := M_{k,2}(X) + \sum_{j=1}^{m_2} \gamma'^j_k \cdot \sum_{i=1}^{n} \tilde{A}_{j,i}(X) \cdot P_i(X) \ .$$

[7] This strategy is frequent in SNARKs to obtain a verification time independent of (or sublinear in) the size of the circuit. This strategy was not considered in the TCitH framework since its main target was building arguments for small circuits (typically for signature scheme) without much care about asymptotic complexity.

The first-round soundness error the PIOP becomes

$$\varepsilon_2 = \left(\frac{\max(m_1, m_2)}{|\mathbb{F}|}\right)^{\rho}$$

instead of $1/|\mathbb{F}|^{\rho}$.

For a large enough field $\mathbb{F}$, the loss of soundness is very moderate while the XOF workload is reduced to

- generate η field elements instead of $\eta \cdot n_{\text{dec}}$ for the degree-enforcing round,
- generate ρ field elements instead of $\rho \cdot \max(m_1, m_2)$ for the PIOP first round.

Illustration of the Saving. Let us give some concrete numbers to illustrate the typical saving in terms of SNARK constraints we obtain using our tweaks. We consider the Anemoi permutation family [BBC+23] over a 256-bit prime field and round verification degree $\alpha = 3$, with a Merkle tree of 4096 leaves. Without any tweaks for SNARK-friendliness, an application of SmallWood-ARK would lead to signature size of around 9.6 KB with 28 700 R1CS constraints, from which around 3300 are due to hashing the leaves and around 18 000 are due to the verification of the authentication paths.

Using the parameter trade-offs in Merkle trees (less hashing for the leaves and greater node arity), we can reduce the number of constraints to 23 000 at the cost of increasing the signature size to 11.4 KB. From now on, 12 300 are due to the verification of the authentication paths in the Merkle tree. While further applying the trimming tweak, the signature size remains 11.4 KB but the number of constraints for verifying the authentication paths drop to 9300, making a total of 20 000 constraints. Finally, using powers as random challenges for the DECS and PIOP batching saves an additional 1000 constraints without changing the signature size.

To summarize, the tweaks decreased the number of constraints by 34%, from 28 700 to 19 000 while increasing the signature size of 19%, from 9.6 KB to 11.4 KB. Of course, a lot of trade-offs are possible given the many different parameters and tweaks but this gives an illustration of a typical trade-off.

4.5 General Description of the Signature Scheme

In this section, we provide a detailed description of the general signature scheme produced by the CAPSS framework. By general, we mean that the description remains compatible with any LPPC statement and does not assume a particular permutation family or arithmetization technique.

The reader is referred to Table 1 for a summary of the different parameters of SmallWood-ARK, which are also used in the signature scheme. We further denote by c the "capacity parameter" introduced in Section 4.2 which is the smallest integer such that $|\mathbb{F}^c| \leq 2^{2\lambda}$. Hash digests arising in the application of Fiat-Shamir belong to $\mathbb{F}^c$. The signature description further uses three fixed subsets of $\mathbb{F}$, namely the DECS evaluation domain $\mathbb{E} = \{e_1, \ldots, e_N\}$, the LVCS witness support $\Omega = \{\omega_1, \ldots, \omega_{n_{\text{cols}}}\}$ and the LVCS randomness support $\Omega' = \{\omega'_1, \ldots, \omega'_\ell\}$. Also, the DECS evaluations to be opened are queried through an index set $I = \{i_1, \ldots, i_\ell\}$, which means that the queried set of evaluation points is $E = \{e_{i_1}, \ldots, e_{i_\ell}\}$. For the sake of simplicity, the domain separation index as well as the output format is left implicit in the calls to the XOF primitive.

We first describe the signing and verification algorithms and then the underlying PCS and PIOP routines.

Signing and Verification Algorithms. The signing and verification algorithms are depicted in Figure 10. The public key pk consists of the LPPC statement, which is composed of the polynomial constraints $\{f_j\}_j$ and the global linear constraints $\{(A_j, t_j)\}_j$. The secret key sk is composed of the latter LPPC statement and the associated witness matrix $(w_{i,j})_{i,j}$.

As explained in Section 3.4, we employ a standard optimization technique to reduce the signature size by selectively revealing elements. Specifically, the polynomials $\boldsymbol{R}$ from the DECS commitment and $\boldsymbol{Q}_1, \boldsymbol{Q}_2$ from the PIOP proof are only partially disclosed in the signature as the opened polynomial evaluations enable to reconstruct them in full. We must still provide hash commitment of these polynomials for further application of the Fiat-Shamir transform. This is done through the variable trans_hash (for transcript hash) for $\boldsymbol{R}$ while

its done through the variable h_{piop} for $\boldsymbol{Q}_1, \boldsymbol{Q}_2$. The PCS and PIOP proofs are then implicitly checked in the verification algorithm by recomputing the proof transcript (including $\boldsymbol{R}$) for the former and by recomputing h_{piop} (hashing $\boldsymbol{Q}_1, \boldsymbol{Q}_2$) for the latter.

$\underline{\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})}$:

1. *Initialization.*
   (a) Sample a random salt salt.
   (b) Parse sk as $(\mathsf{lppc\_stat}, \mathsf{lppc\_wit})$.
   (c) Pares $\mathsf{lppc\_wit}$ as $(w_{i,j})_{i,j}$.
2. *Polynomial commitment.*
   (a) *Building witness polynomials.* For all $j \in [1, n]$, generate a random degree-$(s+\ell'-1)$ polynomial $P_j(X)$ such that $P_j(\omega_1) = w_{j,1}, \ldots, P_j(\omega_s) = w_{j,s}$. Let $\boldsymbol{P} = (P_1, \ldots, P_n)$.
   (b) *Sampling masking polynomials.* Sample ρ random degree-$(d\cdot(s+\ell'-1)-s)$ polynomials $(M_{1,1}, \ldots, M_{\rho,1})$ and ρ random degree-$(2s-1)$ polynomials $(M_{1,2}, \ldots, M_{\rho,2})$ such that $\sum_{i=1}^{s} M_{k,2}(\omega_i) = 0$ for all k. Let $\boldsymbol{M}_1 = (M_{1,1}, \ldots, M_{\rho,1})$ and $\boldsymbol{M}_2 = (M_{1,2}, \ldots, M_{\rho,2})$.
   (c) *Computing polynomial commitment.* Run:
   $$(\mathsf{transcript}_{\mathsf{pcs}}, \mathsf{key}_{\mathsf{pcs}}) \leftarrow \mathsf{PCS.Commit}(\mathsf{salt}, \boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2) \ .$$
3. *Polynomial IOP.* Let $\mathsf{transcript} = \mathsf{msg} \parallel \mathsf{transcript}_{\mathsf{pcs}}$. Run:
   $$(\mathsf{transcript}_{\mathsf{piop}}, \pi_{\mathsf{piop}}) = \mathsf{PIOP.Run}(\mathsf{lppc\_stat}, \boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2, \mathsf{transcript}) \ .$$
   Then derive the PIOP challenge:
   $$h_{\mathsf{piop}} = \mathsf{XOF}(\mathsf{transcript}_{\mathsf{piop}}) \in \mathbb{F}^c \ ,$$
   $$E' = \mathsf{XOF}(h_{\mathsf{piop}}) \in \mathbb{F}^{\ell'} \ .$$
4. *PCS opening.* Let $\mathsf{transcript} = h_{\mathsf{piop}}$. Run:
   $$((\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}, \pi_{\mathsf{pcs}}) = \mathsf{PCS.Open}(\mathsf{key}_{\mathsf{pcs}}, E', \mathsf{transcript}) \ .$$
5. *Signature assembling.* Return
   $$\sigma := (\mathsf{salt}, h_{\mathsf{piop}}, \pi_{\mathsf{piop}}, (\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}, \pi_{\mathsf{pcs}}).$$

$\underline{\mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma)}$:

1. *Initialization.* Parse σ as $(\mathsf{salt}, h_{\mathsf{piop}}, \pi_{\mathsf{piop}}, (\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}, \pi_{\mathsf{pcs}})$.
2. *PIOP challenge recomputation.* Run:
   $$E' = \mathsf{XOF}(h_{\mathsf{piop}}) \in \mathbb{F}^{\ell'} \ .$$
3. *Polynomial commitment recomputation.* Let $\mathsf{transcript} = h_{\mathsf{piop}}$. Run:
   $$\mathsf{transcript}_{\mathsf{pcs}} = \mathsf{PCS.RecomputeTranscript}(\mathsf{salt}, E', (\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}, \pi_{\mathsf{pcs}}, \mathsf{transcript})$$
4. *PIOP transcript recomputation.* Let $\mathsf{transcript} = \mathsf{msg} \parallel \mathsf{transcript}_{\mathsf{pcs}}$. Run:
   $$\mathsf{transcript}_{\mathsf{piop}} = \mathsf{PIOP.RecomputeTranscript}(\mathsf{lppc\_stat}, E', (\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}, \pi_{\mathsf{piop}}, \mathsf{transcript})$$
5. *Final verification.* If $h_{\mathsf{piop}} = \mathsf{XOF}(\mathsf{transcript}_{\mathsf{piop}})$, return ACCEPT. Otherwise return REJECT.

Fig. 10: Signature scheme in the CAPSS framework.

Polynomial Commitment Routines. We describe hereafter the routines for the non-interactive version of the SmallWood-PCS scheme introduced in Section 3. This scheme is based on three layers, from bottom to top: the DECS layer, the LVCS layer and the PCS layer. We describe hereafter the routines of these three layers successively.

DECS routines. The DECS routines are formally described in Figure 11.The routine DECS.Commit is called in the signing algorithm (through the LVCS layer) to compute the DECS commitment transcript (including the degree-enforcing round). The DECS.Open routine is called in the signing algorithm (through the LVCS layer) to compute the opened evaluations and their opening proof. The DECS.RecomputeTranscript routine is called in the verification algorithm (through the LVCS layer) to recompute the commitment transcript –in particular the polynomial $\boldsymbol{R}$– from the opened evaluations. The DECS.OpeningChallenge is called in the signing algorithm (through the LVCS layer) to derive the opening challenge, i.e., the set of open DECS evaluations. The DECS.RecomputeOpeningChallenge is called in the verification algorithm (through the LVCS layer) with the same purpose as the latter routine, but taking the grinding counter as argument to avoid the proof-of-work computation.

LVCS routines. The LVCS routines are formally described in Figure 12. The LVCS.Commit routine is used in the signing algorithm (through the PCS layer) to compute the LVCS commitment transcript. The LVCS.Open routine is used in the signing algorithm (through the PCS layer) to compute the opened evaluations and their opening proof. The LVCS.RecomputeTranscript is used in the verification algorithm (through the PCS layer) to recompute the LVCS commitment transcript.

PCS routines. The PCS routines are formally described in Figure 13. The PCS.Commit routine is used in the signing algorithm to compute the PCS commitment transcript. The PCS.Open routine is used in the signing algorithm to compute the opened evaluations and their opening proof. The PCS.RecomputeTranscript is used in the verification algorithm to recompute the PCS commitment transcript. These routines satisfy the following property: for any $\mathsf{salt}, \boldsymbol{P}, E', \mathsf{transcript}$ of the right format, running:

$$
\begin{aligned}
&(\mathsf{transcript}_{\mathsf{pcs}}, \mathsf{key}_{\mathsf{pcs}}) \leftarrow \mathsf{PCS.Commit}(\mathsf{salt}, \boldsymbol{P}) \\
&(\boldsymbol{P}|_{E'}, \pi_{\mathsf{pcs}}) \leftarrow \mathsf{PCS.Open}(\mathsf{key}_{\mathsf{pcs}}, E', \mathsf{transcript}) \\
&\mathsf{transcript}'_{\mathsf{pcs}} \leftarrow \mathsf{PCS.RecomputeTranscript}(\mathsf{salt}, E', \boldsymbol{P}|_{E'}, \pi_{\mathsf{pcs}}, \mathsf{transcript})
\end{aligned}
$$

always yields $\mathsf{transcript}_{\mathsf{pcs}} = \mathsf{transcript}'_{\mathsf{pcs}}$. Checking the latter equality amounts to implicitly checking the correctness of the PCS opening proof π_{pcs} for the opened evaluations $\boldsymbol{P}|_{E'}$. Namely, the equality only holds if the opening proof is correct.

DECS.Commit($\mathsf{salt}, \boldsymbol{P}$):

On input a salt salt and a vector polynomial $\boldsymbol{P} := (P_1, \ldots, P_{n_{\mathsf{decs}}}) \in \left(\mathbb{F}[X]^{(\leq d_{\mathsf{decs}})}\right)^{n_{\mathsf{decs}}}$:

1. Sample $\boldsymbol{M} = (M_1, \ldots, M_\eta)$ from $\left(\mathbb{F}[X]^{(\leq d_{\mathsf{decs}})}\right)^{\eta}$.
2. For all $i \in [1, N]$, compute the leaves $u_i = \mathsf{XOF}(\mathsf{salt}, \boldsymbol{P}(e_i), \boldsymbol{M}(e_i))$.
3. Compute $\mathsf{tree}, \mathsf{root} = \mathsf{MerkleTree}(u_1, \ldots, u_N)$.
4. Compute $h_{\mathrm{mt}} = \mathsf{XOF}(\mathsf{salt}, \mathsf{root}) \in \mathbb{F}^c$.
5. Derive the challenge $\{\gamma_k\}_{k \in [1,\eta]} = \mathsf{XOF}(h_{\mathrm{mt}}) \in \mathbb{F}^\eta$.
6. For all $k \in [1, \eta]$, compute the polynomial $R_k(X) = M_k(X) + \sum_{i=1}^{n_{\mathsf{decs}}} \gamma_k^i \cdot P_i(X) \in \mathbb{F}[X]^{(\leq d_{\mathsf{decs}})}$.
7. Set $\mathsf{transcript}_{\mathsf{decs}} = (h_{\mathrm{mt}}, \boldsymbol{R})$ and $\mathsf{key}_{\mathsf{decs}} = (\boldsymbol{M}, \boldsymbol{P}, \boldsymbol{R}, \mathsf{tree})$, where $\boldsymbol{R} = (R_1, \ldots, R_\eta)$.
8. Return $(\mathsf{transcript}_{\mathsf{decs}}, \mathsf{key}_{\mathsf{decs}})$.

DECS.Open($\mathsf{key}_{\mathsf{decs}}, I$):

On input an opening key $\mathsf{key}_{\mathsf{decs}} := (\boldsymbol{M}, \boldsymbol{P}, \boldsymbol{R}, \mathsf{tree})$ and a query index set $I := \{i_1, \ldots, i_\ell\}$:

1. Compute $\mathsf{auth} = \mathsf{GetAuthPath}(\mathsf{tree}, I)$.
2. For all $j \in [1, \ell]$, compute $\boldsymbol{p}^{(\mathrm{eval},j)} = \boldsymbol{P}(e_{i_j})$ and $\boldsymbol{m}^{(\mathrm{eval},j)} = \boldsymbol{M}(e_{i_j})$.
3. Set $\boldsymbol{r}^{(\mathrm{high})}$ as the $d_{\mathsf{decs}} + 1 - \ell$ higher coefficients of $\boldsymbol{R}$, and $\pi_{\mathsf{decs}} = (\mathsf{auth}, \{\boldsymbol{m}^{(\mathrm{eval},j)}\}_j, \boldsymbol{r}^{(\mathrm{high})})$.
4. Return $(\{\boldsymbol{p}^{(\mathrm{eval},j)}\}_j, \pi_{\mathsf{decs}})$.

DECS.RecomputeTranscript($\mathsf{salt}, I, \{\boldsymbol{p}^{(\mathrm{eval},j)}\}_j, \pi_{\mathsf{decs}}$):

On input a salt salt, a query index set $I := \{i_1, \ldots, i_\ell\}$, a set of evaluations $\{\boldsymbol{p}^{(\mathrm{eval},j)}\}_j$ and a proof $\pi_{\mathsf{decs}} := (\mathsf{auth}, \{\boldsymbol{m}^{(\mathrm{eval},j)}\}_j, \boldsymbol{r}^{(\mathrm{high})})$:

1. For all $j \in [1, \ell]$, compute $u_{i_j} = \mathsf{XOF}(\mathsf{salt}, \boldsymbol{p}^{(\mathrm{eval},j)}, \boldsymbol{m}^{(\mathrm{eval},j)})$.
2. Compute $\mathsf{root} = \mathsf{RetrieveRootFromPath}(\{u_{i_j}\}_j, \mathsf{auth}, I)$.
3. Compute $h_{\mathrm{mt}} = \mathsf{XOF}(\mathsf{salt}, \mathsf{root}) \in \mathbb{F}^c$.
4. Derive the challenge $\{\gamma_k\}_{k \in [1,\eta]} = \mathsf{XOF}(h_{\mathrm{mt}}) \in \mathbb{F}^\eta$.
5. For all k, compute $r_k^{(\mathrm{eval},j)} = m_k^{(\mathrm{eval},j)} + \sum_{i=1}^{n_{\mathsf{decs}}} \gamma_k^i \cdot p_i^{(\mathrm{eval},j)}$.
6. Restore $\boldsymbol{R}$ from $\boldsymbol{r}^{(\mathrm{high})}$ and $\{\boldsymbol{r}^{(\mathrm{eval},j)}\}_j$.
7. Set $\mathsf{transcript}_{\mathsf{decs}} = (h_{\mathrm{mt}}, \boldsymbol{R})$.
8. Return $\mathsf{transcript}_{\mathsf{decs}}$.

DECS.OpeningChallenge($\mathsf{trans\_hash}$):

On input a transcript hash $\mathsf{trans\_hash}$:

1. Initialize $\mathsf{counter} = 0$.
2. Compute $v = \mathsf{XOF}(\mathsf{counter}, \mathsf{trans\_hash})$. If $v > t_{\mathrm{pow}}$, increment $\mathsf{counter}$ and repeat.
3. Decompose v as $\left(\sum_{j=1}^{\ell} v_j \cdot N^{j-1}\right) + N^\ell \cdot v'$, with $v_1, \ldots, v_\ell \in \{0, \ldots, N-1\}$ and $v' \in \{0, \ldots, t_{\mathrm{pow}}/N^\ell - 1\}$.
4. If there exists $i \neq j$ such that $v_i = v_j$, increment $\mathsf{counter}$ and repeat.
5. Set $I = \{v_j\}_{j \in [1,\ell]}$.
6. Return $(I, \mathsf{counter})$.

DECS.RecomputeOpeningChallenge($\mathsf{counter}, \mathsf{trans\_hash}$):

On input a counter $\mathsf{counter}$ and a transcript hash $\mathsf{trans\_hash}$:

1. Compute $v = \mathsf{XOF}(\mathsf{counter}, \mathsf{trans\_hash})$. Check that $v \leq t_{\mathrm{pow}}$.
2. Decompose v as $\left(\sum_{j=1}^{\ell} v_j \cdot N^{j-1}\right) + N^\ell \cdot v'$, with $v_1, \ldots, v_\ell \in \{0, \ldots, N-1\}$ and $v' \in \{0, \ldots, t_{\mathrm{pow}}/N^\ell - 1\}$.
3. Check that there are no $i \neq j$ such that $v_i = v_j$.
4. Set $I = \{v_j\}_{j \in [1,\ell]}$.
5. Return $(I, \mathsf{counter})$.

Fig. 11: DECS routines in the CAPSS framework.

$\underline{\mathsf{LVCS.Commit}(\mathsf{salt}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}})}$

On input a salt salt and n_{rows} row vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}} \in \mathbb{F}^{n_{\mathsf{cols}}}$:

1. For all $j \in [1, n_{\mathsf{rows}}]$, sample $\bar{\boldsymbol{r}}_j$ from $\mathbb{F}^{\ell}$.
2. For all $j \in [1, n_{\mathsf{rows}}]$, interpolate the degree-$(n_{\mathsf{cols}} + \ell - 1)$ polynomial $P_j(X)$ such that
$$(P_j(\omega_1), \ldots, P_j(\omega_{n_{\mathsf{cols}}})) = \boldsymbol{r}_j \quad \text{and} \quad (P_j(\omega'_1), \ldots, P_j(\omega'_{\ell})) = \bar{\boldsymbol{r}}_j.$$
3. Run $(\mathsf{transcript}_{\mathsf{decs}}, \mathsf{key}_{\mathsf{decs}}) = \mathsf{DECS.Commit}(\mathsf{salt}, \boldsymbol{P})$.
4. Set $\mathsf{transcript}_{\mathsf{lvcs}} = \mathsf{transcript}_{\mathsf{decs}}$ and $\mathsf{key}_{\mathsf{lvcs}} = (\mathsf{key}_{\mathsf{decs}}, \{\boldsymbol{r}_j, \bar{\boldsymbol{r}}_j\}_j)$.
5. Return $(\mathsf{transcript}_{\mathsf{lvcs}}, \mathsf{key}_{\mathsf{lvcs}})$.

$\underline{\mathsf{LVCS.Open}(\mathsf{key}_{\mathsf{lvcs}}, \mathcal{C}, \mathsf{transcript})}$:

On input an opening key $\mathsf{key}_{\mathsf{lvcs}} := (\mathsf{key}_{\mathsf{decs}}, \{\boldsymbol{r}_j, \bar{\boldsymbol{r}}_j\}_j)$, a set of coefficients $\mathcal{C} := \{c_{k,1}, \ldots, c_{k,n_{\mathsf{rows}}}\}_{k \in \{1,\ldots,m\}}$ and a transcript $\mathsf{transcript}$:

1. For all $k \in [1, m]$, compute $\boldsymbol{v}_k = \sum_{j=1}^{n_{\mathsf{rows}}} c_{k,j} \cdot \boldsymbol{r}_j$ and $\bar{\boldsymbol{v}}_k = \sum_{j=1}^{n_{\mathsf{rows}}} c_{k,j} \cdot \bar{\boldsymbol{r}}_j$.
2. Compute $\mathsf{trans\_hash} = \mathsf{XOF}(\mathsf{transcript}, \{\boldsymbol{v}_k, \bar{\boldsymbol{v}}_k\}_k) \in \mathbb{F}^c$.
3. Compute $(I, \mathsf{counter}) = \mathsf{DECS.OpeningChallenge}(\mathsf{trans\_hash})$.
4. Run $(\{\boldsymbol{p}^{(\mathrm{eval},j)}\}_j, \pi_{\mathsf{decs}}) = \mathsf{DECS.Open}(\mathsf{key}_{\mathsf{decs}}, I)$.
5. Set $\pi_{\mathsf{lvcs}} = (\mathsf{counter}, \pi_{\mathsf{decs}}, \{\bar{\boldsymbol{v}}_k\}_k, \{\mathsf{drop}_m(\boldsymbol{p}^{(\mathrm{eval},j)})\}_j)$.
6. Return $(\{\boldsymbol{v}_k\}_k, \pi_{\mathsf{lvcs}})$.

$\underline{\mathsf{LVCS.RecomputeTranscript}(\mathsf{salt}, \mathcal{C}, \{\boldsymbol{v}_k\}_k, \pi_{\mathsf{lvcs}}, \mathsf{transcript})}$:

On input a salt, a query set $\mathcal{C} = \{(c_{k,1}, \ldots, c_{k,n_{\mathsf{rows}}})\}_{k \in \{1,\ldots,m\}}$, a set of LVCS evaluations $\{\boldsymbol{v}_k\}_k$ and a proof $\pi_{\mathsf{lvcs}} := (\mathsf{counter}, \pi_{\mathsf{decs}}, \{\bar{\boldsymbol{v}}_k\}_k, \{\mathsf{drop}_m(\boldsymbol{p}^{(\mathrm{eval},j)})\}_j)$:

1. Compute $\mathsf{trans\_hash} = \mathsf{XOF}(\mathsf{transcript}, \{\boldsymbol{v}_k, \bar{\boldsymbol{v}}_k\}_k) \in \mathbb{F}^c$.
2. Compute $I = \mathsf{DECS.RecomputeOpeningChallenge}(\mathsf{counter}, \mathsf{trans\_hash})$.
3. For all $k \in [1, m]$, compute Q_k as the degree-$(n_{\mathsf{cols}} + \ell + 1)$ polynomial satisfying:
$$(Q_k(\omega_1), \ldots, Q_k(\omega_{n_{\mathsf{cols}}})) = \boldsymbol{v}_k \quad \text{and} \quad \big(Q_k(\omega'_1), \ldots, Q_k(\omega'_{\ell})\big) = \bar{\boldsymbol{v}}_k.$$
4. For all $k \in [1, m]$, compute $\boldsymbol{v}'_k = (Q_k(e_{i_1}), \ldots, Q_k(e_{i_\ell}))$.
5. Compute $\{(\boldsymbol{p}_h^{(\mathrm{eval},j)})_{h \le m}\}_j$ such that $(\boldsymbol{v}'_k)_j = \sum_h c_{k,h} \cdot \boldsymbol{p}_h^{(\mathrm{eval},j)}$ for all $j \in [1, \ell]$ and $k \in [1, m]$.
6. Return $\mathsf{DECS.RecomputeTranscript}(\mathsf{salt}, I, \{\boldsymbol{p}^{(\mathrm{eval},j)}\}_j, \pi_{\mathsf{decs}})$.

Fig. 12: LVCS routines in the CAPSS framework.

$\mathsf{PCS.Commit}(\mathsf{salt}, (P_1, \ldots, P_{n_{\mathsf{pcs}}}))$:

1. For all $1 \leq j \leq n_{\mathsf{pcs}}$, sample $\ell'(\nu_j - 1)$ values $r_{j,1,1}, \ldots, r_{j,\nu_j-1,\ell'}$ uniformly at random from $\mathbb{F}$ and compute:

$$\boldsymbol{A}_j := \left[\begin{array}{ccc|c} a_{j,0} & \cdots & a_{j,(\nu_j-2)\mu} & \left.\begin{matrix}0\\ \vdots \\ 0\end{matrix}\right\} \begin{matrix}\delta_j \\ \text{times}\end{matrix} \\ \vdots & \ddots & \vdots & \\ \vdots & \ddots & \vdots & \\ \vdots & \ddots & \vdots & \\ & & & a_{j,(\nu_j-1)\mu} \\ & & & \vdots \\ \hline a_{j,\mu-1} & \cdots & a_{j,(\nu_j-1)\mu-1} & \\ \hline 0 & \cdots & 0 & a_{j,d_j-\ell'+1} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{j,d_j} \end{array}\right] + \left[\begin{array}{ccc|c} 0 & -r_{j,1,1} & \cdots & -r_{j,1,\nu_j-2} & \left.\begin{matrix}0\\ \vdots \\ 0\end{matrix}\right\} \begin{matrix}\delta_j \\ \text{times}\end{matrix} \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & -r_{j,\ell',1} & \cdots & -r_{j,\ell',\nu_j-2} & \\ 0 & 0 & \cdots & 0 & -r_{j,1,\nu_j-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & -r_{j,\ell',\nu_j-1} \\ \hline r_{j,1,1} & r_{j,1,2} & \cdots & r_{j,1,\nu_j-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ r_{j,\ell',1} & r_{j,\ell',2} & \cdots & r_{j,\ell',\nu_j-1} & 0 \end{array}\right]$$

   with $\delta_j := (\mu \cdot \nu_j + \ell') - (d_j + 1)$ and where $\{a_{j,i}\}_i$ are the coefficients of P_j, i.e., $P_j := \sum_{i=0}^{d_j} a_{j,i} X^i$.
2. Set $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}} \in \mathbb{F}^{n_{\mathsf{cols}}}$ such that

$$\begin{bmatrix} \boldsymbol{r}_1^\top \\ \boldsymbol{r}_2^\top \\ \vdots \\ \boldsymbol{r}_{n_{\mathsf{rows}}}^\top \end{bmatrix} = \mathsf{Stack}_\beta\left(\boldsymbol{A}_1 | \ldots | \boldsymbol{A}_{n_{\mathsf{pcs}}}\right) ,$$

   where Stack_β is the mapping which consists in splitting the columns of a matrix in β subsequent groups and stacking them (c.f. Section 3.2).
3. $(\mathsf{transcript}_{\mathsf{lvcs}}, \mathsf{key}_{\mathsf{lvcs}}) = \mathsf{LVCS.Commit}(\mathsf{salt}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathsf{rows}}})$.
4. Return $(\mathsf{transcript}_{\mathsf{pcs}}, \mathsf{key}_{\mathsf{pcs}}) = (\mathsf{transcript}_{\mathsf{lvcs}}, \mathsf{key}_{\mathsf{lvcs}})$

$\mathsf{PCS.Open}(\mathsf{key}_{\mathsf{lvcs}}, E', \mathsf{transcript})$:

1. Set $\mathcal{C} = \{\boldsymbol{u}_k \otimes (1, r, \ldots, r^{\mu+m})\}_{r \in E', k \in [1,\beta]}$.
2. Compute $(\{\boldsymbol{v}_k^{(r)}\}_{r \in E', k \in [1,\beta]}, \pi_{\mathsf{lvcs}}) = \mathsf{LVCS.Open}(\mathsf{key}_{\mathsf{lvcs}}, \mathcal{C}, \mathsf{transcript})$.
3. For all $r \in E'$, parse $(\boldsymbol{v}_1^{(r)} \mid \ldots \mid \boldsymbol{v}_\beta^{(r)})$ as $(\hat{\boldsymbol{v}}_1^{(r)} \mid \ldots \mid \hat{\boldsymbol{v}}_{n_{\mathsf{pcs}}}^{(r)})$, where $\hat{\boldsymbol{v}}_1^{(r)}, \ldots, \hat{\boldsymbol{v}}_n^{(r)}$ are vectors of size $\nu_1, \ldots, \nu_{n_{\mathsf{pcs}}}$.
4. For all $r \in E'$ and all $1 \leq j \leq n_{\mathsf{pcs}}$, $p_j^{(r)} = \langle \hat{\boldsymbol{v}}_j^{(r)}, (1, r^\mu, \ldots, r^{(\nu_j-2)\mu}, r^{(\nu_j-1)\mu-\delta_j}) \rangle$.
5. Set $\pi_{\mathsf{pcs}} = (\pi_{\mathsf{lvcs}}, \{\mathsf{drop\_first}(\hat{\boldsymbol{v}}_j^{(r)})\}_{r \in E', 1 \leq j \leq n_{\mathsf{pcs}}})$, where $\mathsf{drop\_first}$ removes the first coordinate of the input vector.
6. Return $(\{p_j^{(r)}\}_{r \in E', 1 \leq j \leq n_{\mathsf{pcs}}}, \pi_{\mathsf{pcs}})$.

$\mathsf{PCS.RecomputeTranscript}(\mathsf{salt}, E', \{p_j^{(x)}\}_{x \in E', 1 \leq j \leq n_{\mathsf{pcs}}}, \pi_{\mathsf{pcs}}, \mathsf{transcript})$:

1. Set $\mathcal{C} = \{\boldsymbol{u}_k \otimes (1, r, \ldots, r^{\mu+m})\}_{r \in E', k \in [1,\beta]}$.
2. Parse π_{pcs} as $(\pi_{\mathsf{lvcs}}, \{\mathsf{drop\_first}(\hat{\boldsymbol{v}}_j^{(r)})\}_j)$.
3. For $1 \leq j \leq n_{\mathsf{pcs}}$, compute $(\hat{\boldsymbol{v}}_j)_1 = p_j^{(r)} - \left[(\hat{\boldsymbol{v}}_j)_{\nu_j} \cdot (r^{(\nu_j-1)\mu-\delta_j}) + \sum_{i=2}^{\nu_j-1} (\hat{\boldsymbol{v}}_j)_i \cdot (r^\mu)^{i-1}\right]$.
4. For all $r \in E'$, parse $(\boldsymbol{v}_1^{(r)} \mid \ldots \mid \boldsymbol{v}_\beta^{(r)})$ as $(\hat{\boldsymbol{v}}_1^{(r)} \mid \ldots \mid \hat{\boldsymbol{v}}_{n_{\mathsf{pcs}}}^{(r)})$.
5. Return $\mathsf{LVCS.RecomputeTranscript}(\mathsf{salt}, \mathcal{C}, \{\boldsymbol{v}_k^{(r)}\}_{r \in E', k \in [1,\beta]}, \pi_{\mathsf{lvcs}}, \mathsf{transcript})$.

Fig. 13: PCS routines in the CAPSS framework.

Polynomial IOP Routines. We provide in Figure 14 a formal description of the LPPC PIOP routines. The PIOP.Run routine is called in the signing algorithm. It processes the PIOP protocol by computing the polynomials $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$. It returns π_{piop}, a proof made of partial values of $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$, together with the PIOP transcript. The PIOP.RecomputeTranscript is called in the verification algorithm. It reconstructs $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$ from π_{piop} and the opened evaluations of the witness polynomials $(\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}$. From the reconstructed polynomials, it regenerates the PIOP transcript. By checking that the regenerated transcript matches the original one, the verification algorithm implicitly checks the correctness of the PIOP proof.

PIOP.Run($\mathsf{lppc\_stat}, \boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2, \mathsf{transcript}$)

On input the LPPC statement lppc_stat, vector polynomials $\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2$ and a transcript transcript:

1. Parse lppc_stat as $\{f_j\}_j$, $\{(A_j, t_j)\}_j$.
2. Compute $h_{\mathrm{fpp}} = \mathsf{XOF}(\mathsf{transcript})$.
3. Compute $(\gamma'_1, \ldots, \gamma'_\rho) = \mathsf{XOF}(h_{\mathrm{fpp}})$.
4. For all $1 \leq k \leq \rho$,
   - Compute $Q_{k,1}(X) = M_{k,1}(X) + \frac{\sum_{j=1}^{m_1} \gamma'^j_k \cdot f_j(P_1(X),\ldots,P_n(X),\Theta_{j,1}(X),\ldots,\Theta_{j,n_c}(X))}{V_\Omega(X)}$
   - Compute $Q_{k,2}(X)$ such that $Q_{k,2}(X) \cdot X + c_k = M_{k,2}(X) + \sum_{j=1}^{m_2} \gamma'^j_k \sum_{i=1}^{n} \tilde{A}_{j,i}(X) \cdot P_i(X)$ for some c_k.

   Let $\boldsymbol{Q}_1 = (Q_{1,1}, \ldots, Q_{\rho,1})$ and $\boldsymbol{Q}_2 = (Q_{1,2}, \ldots, Q_{\rho,2})$.
5. Set $\pi_{\mathsf{piop}} = (q^{(H)}_{k,1}, q^{(H)}_{k,2})_k$, where $q^{(H)}_{k,1}$ is the $\deg Q_{k,1}+1-\ell'$ higher coefficients of $Q_{k,1}$ and $q^{(H)}_{k,2}$ is the $\deg Q_{k,2}+1-\ell'$ higher coefficients of $Q_{k,2}$.
6. Set $\mathsf{transcript}_{\mathsf{piop}} = (h_{\mathrm{fpp}}, \boldsymbol{Q}_1, \boldsymbol{Q}_2)$.
7. Return $(\mathsf{transcript}_{\mathsf{piop}}, \pi_{\mathsf{piop}})$

PIOP.RecomputeTranscript($\mathsf{lppc\_stat}, E', (\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}, \pi_{\mathsf{piop}}, \mathsf{transcript}$)

On input the LPPC statement lppc_stat, a set of evaluation points E', polynomial evaluations $(\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}$, a PIOP proof π_{piop}, and a transcript transcript:

1. Parse lppc_stat as $\{f_j\}_j$, $\{(A_j, t_j)\}_j$.
2. Compute $h_{\mathrm{fpp}} = \mathsf{XOF}(\mathsf{transcript})$.
3. Compute $(\gamma'_1, \ldots, \gamma'_\rho) = \mathsf{XOF}(h_{\mathrm{fpp}})$.
4. For all $1 \leq k \leq \rho$,
   - Compute $Q_{k,1}(q) = M_{k,1}(q) + \frac{\sum_{j=1}^{m_1} \gamma'^j_k \cdot f_j(P_1(q),\ldots,P_n(q),\Theta_{j,1}(q),\ldots,\Theta_{j,n_c}(q))}{V_\Omega(q)}$
   - Restore $Q_{k,1}(X)$ from $q^{(H)}_{k,1}$ and $Q_{k,1}(q)$
   - Compute $\bar{Q}_{k,2}(q) = M_{k,2}(q) + \sum_{j=1}^{m_2} \gamma'^j_k \sum_{i=1}^{n} \tilde{A}_{j,i}(q) \cdot P_i(q)$
   - Restore $Q_{k,2}(X) \cdot X + c'_k$ from $q^{(H)}_{k,2}$ and $\bar{Q}_{k,2}(q)$ knowing that $\sum_{h=1}^{s}(Q_2(\omega_h) \cdot \omega_h + c'_k) = \sum_{j=1}^{m_2} \gamma'_j \cdot t_j$.
5. Set $\mathsf{transcript}_{\mathsf{piop}} = (h_{\mathrm{fpp}}, \boldsymbol{Q}_1, \boldsymbol{Q}_2)$.
6. Return $\mathsf{transcript}_{\mathsf{piop}}$

Fig. 14: PIOP routines in the CAPSS framework.

4.6 Instances

This section presents concrete instances of signature schemes built from the CAPSS framework and compare their performances to other signature schemes from the literature. We apply the CAPSS framework to four families of permutations: Anemoi [BBC+23], Griffin [GHR+23], Poseidon [GKR+21], and RescuePrime [AAB+20, SAD20]. We denote "Perm-α" the family of permutations Perm over a 256-bit field $\mathbb{F}$ such that $x \mapsto x^\alpha$ is invertible on $\mathbb{F}$ and α is the degree of the round verification function introduced in Section 4.3. For each family, we propose three instances with different trade-offs between signature size, signing time and verification complexity (running time and number of constraints). Each trade-off corresponds to a different size for the Merkle tree $N \in \{2^{10}, 2^{12}, 2^{14}\}$:

- "Short" trade-off: We use $N = 2^{14}$ and trees of small arities. This trade-off aims at small signature sizes.
- "Default" trade-off: We use $N = 2^{12}$. This trade-off aims at a good balance between signing time, signature size and verification complexity.
- "Fast" trade-off: We use $N = 2^{10}$. This trade-off aims at fast signing time.

We stress that for any of the above trade-offs, verification is much faster than signing.

Performance of our instances. Table 3 summarizes the performance of our instances. The signature size is mainly determined by the witness size, i.e., the size $s \cdot n$ of the witness matrix. For Anemoi, Griffin and RescuePrime, we rely on the arithmetization for regular permutations. This arithmetization benefits from small witnesses due to its low redundancy and the small number of rounds in these permutations. Specifically, the number of witness coefficients is 48 for Anemoi-3, 56 for Anemoi-5, 60 for Griffin-3/5, 75 for RescuePrime-3, and 60 for RescuePrime-5. Anemoi yields slightly smaller witnesses because we can use a permutation with state of size 2 to instantiate the one-way function (while we need at least 3 for the others). Unfortunately, we cannot use the same arithmetization for Poseidon because of its irregular structure. We then use the S-box-centric arithmetization for this family, but it leads to larger witnesses: Poseidon-3/5 requires 180 witness coefficients. This is due to the higher degree of redundancy in the S-box-centric arithmetization which includes both the input and output of each S-box to the witness. For this reason, we achieve signature sizes between 9 KB and 14 KB for Anemoi, Griffin and RescuePrime, while we achieve sizes between 15 KB and 24 KB for Poseidon.

We also provide estimated running times for our instances. Since the bottleneck in our signature and verification algorithms is the computation of the permutations, we approximate the timings by counting the number of permutation calls and using existing benchmarks for the running times of the permutations. We use the benchmarks provided in Table 5 of [BBC+23] for the running time of a single permutation call for Anemoi, Griffin, Poseidon and RescuePrime on a 381-bit field. We should hence obtain upper estimations of the timings of our instances which are defined on 256-bit fields. While these benchmarks are for a single state size, we extrapolate further timings by assuming that the running time of each permutation scales linearly in $t \cdot n_r$, where t is the state size and n_r is the number of rounds. Let us further note that the benchmarks of [BBC+23, Table 5] have been obtained on an Intel(R) Core(TM) i7-8565U CPU running at 1.80 GHz. For Anemoi, Griffin and RescuePrime, the signing time ranges between 200 and 600 ms for the fast instances, around 1 second for the default instances and between 5 and 14 seconds for the short instances. For these permutations, the verification takes one to few dozens milliseconds. On the other hand, Poseidon provides the fastest instances ranging from 100 ms (fast instances) to 2–3 seconds (short instances) for signing, and around 5 ms for verification.

Table 3 also provides the SNARK-friendliness of each instance by measuring the number of R1CS constraints for the verification algorithms. These constraints have been counted based on a Python proof-of-concept implementation explicitly building the verification arithmetic circuit. We can see that the instances derived from Anemoi and Griffin have a number of R1CS constraints ranging between 17 K and 20 K, while the instances derived from Poseidon and RescuePrime are between 30 K and 60 K. This is not surprising, as the verification complexity is primarily dictated by the underlying permutations. Consequently, the number

Table 3: Performance of CAPSS signature instances. The running times are given in milliseconds. Asterisks indicate estimated timings.

Permutation	**LPPC Statement**					**Proof System**						**Sig. Size**	**#R1CS**	**Signing**	**Verif.**
Family	t	n_r	s	b	n	Trade-off	ℓ	N	Arity	η	w			**Time**	**Time**
Anemoi-3	2	21	3	7	16	Short	13	16 384	2^{14}	2	8	8 980 B	20 888	6 939*	27*
						Default	17	4 096	4^6	2	7	11 402 B	19 013	1 458*	22*
						Fast	24	1 024	4^5	2	8	12 239 B	23 603	407*	25*
Anemoi-5	2	21	4	6	14	Short	13	16 384	2^{14}	2	8	9 780 B	26 263	6 939*	28*
						Default	17	4 096	4^6	2	7	12 330 B	23 811	1 458*	23*
						Fast	24	1 024	4^5	2	8	13 396 B	29 275	407*	26*
Griffin-3	3	16	4	4	15	Short	12	19 683	3^9	2	13	9 950 B	17 353	5 690*	13*
						Default	17	3 888	$4^2 \cdot 3^5$	2	7	11 095 B	20 578	1 000*	14*
						Fast	25	972	$4 \cdot 3^5$	2	5	12 586 B	26 788	252*	17*
Griffin-5	3	14	5	3	12	Short	12	19 683	3^9	2	13	10 685 B	18 866	5 083*	12*
						Default	17	3 888	$4^2 \cdot 3^5$	2	7	11 994 B	21 944	896*	13*
						Fast	25	972	$4 \cdot 3^5$	2	5	13 735 B	28 414	226*	16*
Poseidon-3	3	-	15	-	16	Short	12	19 683	3^9	2	13	15 678 B	34 469	2 987*	5*
						Default	17	3 888	$4^2 \cdot 3^5$	2	7	18 589 B	39 456	562*	5*
						Fast	25	972	$4 \cdot 3^5$	2	5	22 884 B	49 992	141*	7*
Poseidon-5	3	-	15	-	12	Short	12	19 683	3^9	2	13	16 447 B	41 231	2 089*	4*
						Default	17	3 888	$4^2 \cdot 3^5$	2	7	19 354 B	47 189	393*	4*
						Fast	25	972	$4 \cdot 3^5$	2	5	23 656 B	59 565	99*	5*
RescuePrime-3	3	18	5	4	15	Short	12	19 683	3^9	2	13	10 459 B	31 872	14 051*	36*
						Default	17	3 888	$4^2 \cdot 3^5$	2	7	11 768 B	36 179	2 385*	38*
						Fast	25	972	$4 \cdot 3^5$	2	5	13 513 B	45 805	602*	47*
RescuePrime-5	3	14	5	3	12	Short	12	19 683	3^9	2	13	10 687 B	40 658	12 990*	31*
						Default	17	3 888	$4^2 \cdot 3^5$	2	7	11 992 B	46 238	2 264*	34*
						Fast	25	972	$4 \cdot 3^5$	2	5	13 733 B	58 380	572*	41*

of R1CS constraints in signature verification is largely determined by the constraints dedicated to these permutations. For example, in the default trade-off of Anemoi-3, 84% of the R1CS constraints are devoted to verifying the permutations. Since Anemoi and Griffin are more efficient in terms of R1CS constraints compared to Poseidon and RescuePrime, the resulting CAPSS signature naturally inherits this efficiency hierarchy.

To provide further insights on the R1CS cost, we give in Table 4 the distribution of R1CS constraints among the different components of the verification algorithm for the CAPSS-Anemoi instances. We observe that the bottleneck is the verification of the authentication paths in the Merkle tree –from 41% to 63%– despite the tweaks described in Section 4.4. The next bottlenecks come from hashing the leaves and generating the Fiat-Shamir challenges. While hashing the leaves is expensive (12%–23%), generating the Fiat-Shamir challenges is also far from being negligible, notably the one that require hashing the polynomials $\boldsymbol{R}$ in the DECS (8%–10%) because of the relatively high degree of this vector polynomial.

Comparison to the state of the art. Table 5 compares our signature schemes with the other symmetric-based post-quantum signature schemes from the state of the art. Besides the CAPSS instances, all the numbers of R1CS constraints are taken from the estimates given in [ZSE+24].

The first part of the table provides known schemes which do not specifically target at SNARK-friendliness. As our signature schemes, SPHINCS$^+$ verification also involves verifying Merkle paths. While this scheme could be also made SNARK-friendly by using an arithmetization-oriented hash function, this would result in

Table 4: Distribution of R1CS constrains for CAPSS-Anemoi instances.

		Anemoi-3			Anemoi-5		
		Short	Default	Fast	Short	Default	Fast
Verify	$\mathsf{XOF}(\mathsf{transcript}_{\mathrm{piop}})$	320	320	320	800	800	800
PCS	Computation of $\{(\hat{\boldsymbol{v}}_j)_1\}_j$	5	5	5	10	10	10
LVCS	$\mathsf{RecomputeOpeningChallenge}(\mathsf{counter}, \mathsf{trans\_hash})$	255	292	395	283	320	423
	Computation of $\{\bar{\boldsymbol{v}}_k\}_k$	897	1309	2184	910	1326	2208
	$\mathsf{XOF}(\mathsf{transcript}, \{\boldsymbol{v}_k, \bar{\boldsymbol{v}}_k\}_k)$	960	960	1280	1200	1200	1600
DECS	$\mathsf{XOF}(\boldsymbol{p}^{(\mathrm{eval},j)}, \boldsymbol{m}^{(\mathrm{eval},j)})$	2496	3264	4608	3575	4675	6600
	$\mathsf{RetrieveRootFromPath}(\{u_{i_j}\}_j, \mathsf{auth}, I)$	13 116	9264	9968	16 161	11 308	12 124
	$\mathsf{XOF}(\mathsf{salt}, \mathsf{root})$	118	118	118	148	148	148
	Computation of $\boldsymbol{R}$	988	1428	2352	1014	1462	2400
PIOP	$\mathsf{XOF}(\mathsf{msg} \parallel \mathsf{transcript}_{\mathrm{pcs}})$	1655	1975	2295	2059	2459	2859
	Computation of $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$	76	76	76	101	101	101
Total		20 888	19 013	23 603	26 263	23 811	29 275

very slow signing times because of the large number of hash computations required in a SPHINCS$^+$ signature computation.[8]

The next schemes in the first part of the table are all based on the MPC-in-Head paradigm [IKOS07]. In these schemes, the signature verification requires the re-computation of GGM trees, which results in a large number of R1CS constraints. FAEST [BBD$^+$23b] is the most recent scheme in this category, which achieves signature sizes around 5 KB using the VOLE-in-the-Head technique [BBD$^+$23a]. While the number of R1CS constraints for this scheme is not estimated in [ZSE$^+$24], it should be above the other ones, given the fact that FAEST makes greedier use of GGM trees.

The second part of the table includes two recent schemes. The first one, Loquat [ZSE$^+$24], is a signature scheme based on the Legndre PRF and targeting SNARK-friendliness. Our schemes clearly outperforms Loquat in terms of signature size, timings and R1CS constraints for the verification. For instance, our CAPSS-Anemoi instances achieve a 4–6× reduction in signature size and a 5–8× reduction in R1CS constraints compared to Loquat. Another advantage of our approach compared to Loquat is that the security of our schemes solely relies on the underlying family of permutations, whereas Loquat further relies on the security of the Legendre PRF. The second scheme is from a recent independent work [AdSGK24b] which applies a STARK proof system to the RescuePrime permutation. While this methodology is similar to ours (hash-based proof system applied to an arithmetization-oriented permutation), our work goes further in the optimization of this approach, notably by introducing the SmallWood-ARK proof system, which is specifically designed to obtain small proofs. For this reason, we obtain much shorter signatures than [AdSGK24b].

[8] According to [ZSE$^+$24], a SPHINCS$^+$ signature requires more than 100 K hash computations for its fast instance and more than 2000 K hash computations for its short instance. This is greater than the number of hash computations in our fast and short instances by two order of magnitudes.

Table 5: Comparison of post-quantum signatures based on symmetric-key primitives. Except for the CAPSS instances, the numbers of R1CS constraints are from [ZSE+24]. Asterisks indicate estimated timings.

Signature Scheme	**Sig. Size**	**#R1CS**	**Signing Time**	**Verif. Time**	**Assumptions**
SPHINCS+s	8 KB	≈ 460 K	–	–	Hash
SPHINCS+f	16 KB	≈ 1 400 K	–	–	Hash
Picnic1	32 KB	≈ 3 500 K	–	–	LowMC + Hash
Picnic3	12 KB	≈ 21 600 K	–	–	LowMC + Hash
LegRoast	16 KB	≈ 1 100 K	–	–	Leg. PRF + Hash
Banquet	12 KB	≈ 11 800 K	–	–	AES + Hash
Rainer	8 KB	≈ 26 100 K	–	–	Rain + Hash
FAEST	5 KB	–	–	–	AES + Hash
Loquat-128 (Keccak) [ZSE+24]	57 KB	–	5.0 s	0.2 s	Legendre PRF + Keccak
Loquat-128 (Griffin) [ZSE+24]	57 KB	≈ 150 K	105 s	11 s	Legendre PRF + Griffin
Loquat*-128 (Keccak) [ZSE+24]	114 KB	–	5.0 s	0.2 s	Legendre PRF + Keccak
Loquat*-128 (Griffin) [ZSE+24]	114 KB	≈ 300 K	214 s	25 s	Legendre PRF + Griffin
RescuePrime + STARKs [AdSGK24b]	80–100 KB	–	9–23 ms	1 ms	RescuePrime (+ Blake3)
RescuePrime + STARKs [AdSGK24b]	80–100 KB	–	94–370 ms	21–27 ms	RescuePrime
CAPSS-Anemoi	9–14 KB	19 K – 30 K	400 ms – 6 s *	22–28 ms *	Anemoi
CAPSS-Griffin	10–14 KB	17 K – 29 K	200 ms – 6 s *	12–17 ms *	Griffin
CAPSS-Poseidon	16–24 KB	34 K – 60 K	100 ms – 3 s *	4–7 ms *	Poseidon
CAPSS-RescuePrime	10–14 KB	32 K – 59 K	500 ms – 14 s *	31–47 ms *	RescuePrime

Acknowledgements

This research was supported by the Ethereum Foundation, under the 2024 Academic Grants Round. The authors would like to thank Léo Perrin for insightful discussions on arithmetization-friendly permutations.

References

AAB+20. Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symm. Cryptol.*, 2020(3):1–45, 2020.

AdSGK24a. Shahla Atapoor, Cyprien Delpech de Saint Guilhem, and Al Kindi. STARK-based signatures from the RPO permutation. Cryptology ePrint Archive, Paper 2024/1553, 2024.

AdSGK24b. Shahla Atapoor, Cyprien Delpech de Saint Guilhem, and Al Kindi. STARK-based signatures from the RPO permutation. Cryptology ePrint Archive, Paper 2024/1553, 2024.

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

AHIV23. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: lightweight sublinear arguments without a trusted setup. *DCC*, 91(11):3379–3424, 2023.

AKM+22. Tomer Ashur, Al Kindi, Willi Meier, Alan Szepieniec, and Bobbin Threadbare. Rescue-prime optimized. Cryptology ePrint Archive, Report 2022/1577, 2022.

BBC+23. Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Léo Perrin, Robin Salen, Vesselin Velichkov, and Danny Willems. New design techniques for efficient arithmetization-oriented hash functions: Anemoi permutations and Jive compression mode. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 507–539. Springer, Cham, August 2023.

BBD+23a. Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023.

BBD+23b. Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. FAEST: Algorithm Specifications – Version 1.1, 2023. `https://faest.info/faest-spec-v1.1.pdf`.

BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018.

BBHR19. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, Cham, August 2019.

BBL+24. Augustin Bariant, Aurélien Boeuf, Axel Lemoine, Irati Manterola Ayala, Morten Øygarden, Léo Perrin, and Håvard Raddum. The algebraic FreeLunch: Efficient Gröbner basis attacks against arithmetization-oriented primitives. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IV*, volume 14923 of *LNCS*, pages 139–173. Springer, Cham, August 2024.

BCG20. Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 19–46. Springer, Cham, November 2020.

BCR+19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Cham, May 2019.

BDK+21. Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 266–297. Springer, Cham, May 2021.

BDPA11. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011.

BDPV08. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, Berlin, Heidelberg, April 2008.

BFG+24. Loïc Bidoux, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, and Matthieu Rivain. Dual support decomposition in the head: Shorter signatures from rank SD and MinRank. Cryptology ePrint Archive, Report 2024/541, 2024.

BGKS20. Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 5:1–5:32. LIPIcs, January 2020.

BGLS03. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Berlin, Heidelberg, May 2003.

Cha82. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Berlin, Heidelberg, May 2001.

CNR+22. Matteo Campanelli, Anca Nitulescu, Carla Ràfols, Alexandros Zacharakis, and Arantxa Zapico. Linear-map vector commitments and their practical applications. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 189–219. Springer, Cham, December 2022.

Fis06. Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77. Springer, Berlin, Heidelberg, August 2006.

FR23. Thibauld Feneuil and Matthieu Rivain. Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments. Cryptology ePrint Archive, Report 2023/1573, 2023.

GHR+23. Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. Horst meets fluid-SPN: Griffin for zero-knowledge applications. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 573–606. Springer, Cham, August 2023.

GKR+21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021.

GKS23. Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the poseidon hash function. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 177–203. Springer, Cham, July 2023.

GLS+23. Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 193–226. Springer, Cham, August 2023.

Hir18. Shoichi Hirose. Sequential hashing with minimum padding. *Cryptography*, 2:11, 06 2018.

HJ24. Janik Huth and Antoine Joux. VOLE-in-the-head signatures from subfield bilinear collisions. Cryptology ePrint Archive, Paper 2024/1537, 2024.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.

Lee21. Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, Cham, November 2021.

OTX24. Ying Ouyang, Deng Tang, and Yanhong Xu. Code-based zero-knowledge from VOLE-in-the-head and their applications: Simpler, faster, and smaller. Cryptology ePrint Archive, Paper 2024/1414, 2024.

SAD20. Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. Rescue-prime: a standard specification (SoK). Cryptology ePrint Archive, Report 2020/1143, 2020.

Sta21. StarkWare. ethSTARK documentation. Cryptology ePrint Archive, Report 2021/582, 2021.

VP19. Alexander Vlasov and Konstantin Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. Cryptology ePrint Archive, Report 2019/1020, 2019.

ZSE+24. Xinyu Zhang, Ron Steinfeld, Muhammed F. Esgin, Joseph K. Liu, Dongxi Liu, and Sushmita Ruj. Loquat: A SNARK-friendly post-quantum signature based on the legendre PRF with applications in ring and aggregate signatures. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 3–38. Springer, Cham, August 2024.

A Summary of the Hiding Tweaks

Three leakage sources are tackled by our tweaks:

- *In the PCS construction.* Opening a PCS into an evaluation point e consists of opening the underlying LVCS for the coefficient tuple $(1, e, \ldots, e^{n_{\text{rows}}})$. By denoting $\boldsymbol{v}^{(e)}$ the underlying linear combination opended through the LVCS, the polynomial evaluation is deduced as $P(e) = \langle \boldsymbol{v}, (1, e^{\mu}, \ldots, e^{(\nu-1)\mu}) \rangle$. However, $\boldsymbol{v}$ should not leak information about P and to proceed, one needs to commit few additional rows in the LVCS that contains randomness (see the coefficients $\{r_{j,i,k}\}_{j,i,k}$ in Figure 3).
- *In the LVCS construction.* Each row is interpolated as a polynomial which is committed using the DECS protocol (or Merkle trees with Ligero's proximity test for Ligero-PCS). Then the LVCS opens those polynomials into ℓ random points. To avoid leakage, the polynomials should be interpolated with enough randomness (see the coefficients $\bar{\boldsymbol{r}}_1, \ldots, \bar{\boldsymbol{r}}_{n_{\text{rows}}}$ in Figure 2).
- *In the DECS (or Merkle trees with Ligero's proximity test for Ligero-PCS).* To ensure that the committed polynomials are of the right degree, one reveals random linear combination of those polynomials. We mask the result of these linear combinations by additional random polynomials (see the polynomials $M_1, \ldots, M_\eta$ in Figure 1). Moreover, we add random tapes in hash computation of the Merkle leaves (see the tapes $\rho_1, \ldots, \rho_N$ in Figure 1) and, for each revealed leaf, we need to reveal the corresponding tape.

B Grinding

In what follows, we formally describe how we incorporate a κ-bit proof of work into the Fiat-Shamir hash computation for the DECS opening challenge in SmallWood-ARK.

Let us denote $\mathcal{H}$ the set of all the hash digests. In a textbook implementation of the DECS, the prover hashes some materials $\texttt{mat}$ to obtain a hash digest $h := \text{Hash}(\texttt{mat}) \in \mathcal{H}$ and derives the opening challenge from h. In the SmallWood-ARK, after hashing, the prover checks that h belongs to a fix public subset $\mathcal{S} \subset \mathcal{H}$ of size $|\mathcal{S}| = \frac{1}{2^\kappa} \cdot |\mathcal{H}|$. If it is not the case, they increase a counter and repeat the hash operation until the belonging condition is satisfied. For example, the challenge could be derived as

1. Set a counter c as 0.
2. Compute $h = \text{Hash}(c \,\|\, \texttt{mat}) \in \mathcal{H}$
3. If $h \notin \mathcal{S}$,
   Increase the counter $c = c + 1$
   Go to Instruction 2.
4. Derive the opening challenge from h.

This strategy increases the cost of hashing the opening challenge by a factor 2^κ and hence increases the security of κ bits (for this challenge). This thus allows us to take smaller N and ℓ: to achieve a λ-bit security, we select the parameters N, ℓ and κ such that

$$\frac{\binom{n_{\text{cols}}+\ell-1}{\ell}}{\binom{N}{\ell}} \cdot 2^{-\kappa} \leq 2^{-\lambda} \ .$$

Most of the time (in Section 4, it will not be the case), the set of hash digests are $\mathcal{H} := \{0,1\}^{2\lambda}$, and so the set $\mathcal{S}$ of accepting digests can be all the hash digests for which the κ last bits are all zeros.

C Fiat-Shamir Transformation of SmallWood-ARK

Figure 15 gives a sketch of the non-interactive version of the proof system obtained by applying the Fiat-Shamir transformation to the composition of the LPPC PIOP and SmallWood-PCS, while optimizing the communication. Specifically, we outline the computation order and the routine interfaces needed to generate a proof that adheres to the sizes specified in Section 3.4.

SmallWood-ARK.Run():

1. Compute $\boldsymbol{P}$, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, together with their commitment $\mathsf{com}_{\mathrm{PCS}}$.
2. Compute $\Gamma' = \mathrm{PRG}(h_{\Gamma'})$, with $h_{\Gamma'} = \mathrm{Hash}(\mathsf{com}_{\mathrm{PCS}})$.
3. Compute $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$.
4. Compute $E' = \mathrm{PRG}(h_{E'})$, with $h_{E'} = \mathrm{Hash}(\boldsymbol{Q}_1, \boldsymbol{Q}_2)$.
5. Compute $(\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}$ and π_{pcs}.
6. Set $\pi = (h_{E'}, (\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}, \pi_{\mathsf{pcs}}, \mathrm{Tr}(\boldsymbol{Q}_1), \mathrm{Tr}(\boldsymbol{Q}_2))$.

(a) PIOP (with PCS) – Non-interative Prover.

Input: $\pi = (h_{E'}, (\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}, \pi_{\mathsf{pcs}}, \mathrm{Tr}(\boldsymbol{Q}_1), \mathrm{Tr}(\boldsymbol{Q}_2))$

1. Compute $E' = \mathrm{PRG}(h_{E'})$.
2. Deduce $\mathsf{com}_{\mathrm{PCS}}$ from E', $(\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}$ and π_{pcs}.
3. Compute $\Gamma' = \mathrm{PRG}(h_{\Gamma'})$, with $h_{\Gamma'} = \mathrm{Hash}(\mathsf{com}_{\mathrm{PCS}})$.
4. Compute $(\boldsymbol{Q}_1, \boldsymbol{Q}_2)|_{E'}$ from E', $(\boldsymbol{P}, \boldsymbol{M}_1, \boldsymbol{M}_2)|_{E'}$ and Γ', then deduce $\boldsymbol{Q}_1, \boldsymbol{Q}_2$ using $\mathrm{Tr}(\boldsymbol{Q}_1), \mathrm{Tr}(\boldsymbol{Q}_2)$.
5. Compute $h'_{E'} = \mathrm{Hash}(\boldsymbol{Q}_1, \boldsymbol{Q}_2)$.
6. Check that $h_{E'} =_? h'_{E'}$.

(b) PIOP (with PCS) – Non-interative Verifier.

PCS.Commit$(P_1, \ldots, P_{n_{\mathrm{pcs}}})$:

1. Compute $\mathsf{com}_{\mathrm{PCS}} := \mathsf{com}_{\mathsf{lvcs}}$ from $P_1, \ldots, P_{n_{\mathrm{pcs}}}$, and return it.

PCS.Open$(E', (P_1, \ldots, P_{n_{\mathrm{pcs}}})|_{E'})$:

1. Compute $\mathcal{C}$ from E'.
2. Compute all $\boldsymbol{v}_{\boldsymbol{k}}^{(\boldsymbol{e})}$'s and π_{lvcs}.
3. Set π_{pcs} as $(\pi_{\mathsf{lvcs}}, \{\mathrm{Tr}(\boldsymbol{v}_{\boldsymbol{k}}^{(\boldsymbol{e})})\}_{e,k})$.

(c) PCS – Non-interative Prover/Committer.

Input: E', $(P_1, \ldots, P_{n_{\mathrm{pcs}}})|_{E'}$, and $\pi_{\mathsf{pcs}} = (\pi_{\mathsf{lvcs}}, \{\mathrm{Tr}(\boldsymbol{v}_{\boldsymbol{k}}^{(\boldsymbol{e})})\}_{e,k})$

1. Compute $\mathcal{C}$ from E'.
2. Deduce all $\boldsymbol{v}_{\boldsymbol{k}}^{(\boldsymbol{e})}$'s using $\{\mathrm{Tr}(\boldsymbol{v}_{\boldsymbol{k}}^{(\boldsymbol{e})})\}_{e,k}$ and $(P_1, \ldots, P_{n_{\mathrm{pcs}}})|_{E'}$ and E'.
3. Compute $\mathsf{com}_{\mathrm{PCS}} := \mathsf{com}_{\mathsf{lvcs}}$ from $\mathcal{C}$, $\{\boldsymbol{v}_{\boldsymbol{k}}^{(\boldsymbol{e})}\}_{e,k}$ and π_{lvcs}, and return it.

(d) PCS – Non-interative Verifier.

LVCS.Commit$(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathrm{rows}}})$:

1. Compute $\boldsymbol{P} := (P_1, \ldots, P_{n_{\mathrm{decs}}})$ from $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_{\mathrm{rows}}}$.
2. Compute $\mathsf{com}_{\mathsf{lvcs}} := \mathsf{com}_{\mathsf{decs}}$ from $\boldsymbol{P}$, and return it.

LVCS.Open$(\mathcal{C}, \{\boldsymbol{v_k}\}_k)$:

1. Compute all $\bar{\boldsymbol{v}}_k$'s.
2. Compute $E = \mathrm{PRG}(h_E)$, with $h_E = \mathrm{Hash}(\{\boldsymbol{v_k}, \bar{\boldsymbol{v}}_k\}_k)$.
3. Compute $\boldsymbol{P}|_E$ and π_{decs}.
4. Set π_{lvcs} as $(\pi_{\mathsf{decs}}, \{\bar{\boldsymbol{v}}_k\}_k, \mathrm{Tr}(\boldsymbol{P}|_E))$.

(e) LVCS – Non-interative Prover/Committer.

Input: $\mathcal{C}$, $\{\boldsymbol{v_k}\}_k$, and $\pi_{\mathsf{lvcs}} = (\pi_{\mathsf{decs}}, \{\bar{\boldsymbol{v}}_k\}_k, \mathrm{Tr}(\boldsymbol{P}|_E))$

1. Compute $E = \mathrm{PRG}(h_E)$, with $h_E = \mathrm{Hash}(\{\boldsymbol{v_k}, \bar{\boldsymbol{v}}_k\}_k)$.
2. Deduce $\boldsymbol{P}|_E$ from E, $\mathrm{Tr}(\boldsymbol{P}|_E)$ and $\{\boldsymbol{v_k}, \bar{\boldsymbol{v}}_k\}_k$.
3. Compute $\mathsf{com}_{\mathsf{lvcs}} := \mathsf{com}_{\mathsf{decs}}$ from E, $\boldsymbol{P}|_E$ and π_{decs}, and return it.

(f) LVCS – Non-interative Verifier.

DECS.Commit$(\boldsymbol{P})$:

1. Sample the polynomials $\boldsymbol{M}$.
2. Compute the Merkle root root.
3. Compute $\Gamma = \mathrm{PRG}(h_\Gamma)$, with $h_\Gamma = \mathrm{Hash}(\mathsf{root})$.
4. Compute $\boldsymbol{R}$ from $\boldsymbol{P}$, $\boldsymbol{M}$ and Γ.
5. Return $(h_\Gamma, \boldsymbol{R})$.

DECS.Open$(E, \boldsymbol{P}|_E)$:

1. Compute the authentication paths auth.
2. Compute $\boldsymbol{M}|_E$.
3. Set π_{decs} as $(\mathsf{auth}, \boldsymbol{M}|_E, \mathrm{Tr}(\boldsymbol{R}))$.

(g) DECS – Non-interative Prover/Committer.

Input: E, $\boldsymbol{P}|_E$, and $\pi_{\mathsf{decs}} = (\mathsf{auth}, \boldsymbol{M}|_E, \mathrm{Tr}(\boldsymbol{R}))$

1. Compute root from E, $\boldsymbol{P}|_E$, $\boldsymbol{M}|_E$ and auth.
2. Compute $\Gamma = \mathrm{PRG}(h_\Gamma)$, with $h_\Gamma = \mathrm{Hash}(\mathsf{root})$.
3. Compute $\boldsymbol{R}|_E$ from $\boldsymbol{P}|_E$, $\boldsymbol{M}|_E$ and Γ, then deduce $\boldsymbol{R}$ using $\mathrm{Tr}(\boldsymbol{R})$.
4. Return $(h_\Gamma, \boldsymbol{R})$.

(h) DECS – Non-interative Verifier.

Fig. 15: Sketch of SmallWood-ARK, the non-iteractive version of the "LPPC PIOP + SmallWood-PCS" proof system with optimized communication. $\mathrm{Tr}(\cdot)$ is a routine that truncates the input data (the amount of truncation is left implicit).