Verification-efficient Homomorphic Signatures for Verifiable Computation over Data Streams

Gaspard Anthoine[1,2], Daniele Cozzo[1], and Dario Fiore[1]

[1] IMDEA Software Institute, Spain.
firstname.lastname@imdea.org
[2] Universidad Politécnica de Madrid, Madrid, Spain.

Abstract. Homomorphic signatures for NP (HSNP) allow proving that a signed value is the result of a non-deterministic computation on signed inputs. At CCS'22, Fiore and Tucker introduced HSNP, showed how to use them for verifying arbitrary computations on data streams, and proposed a generic HSNP construction obtained by efficiently combining zkSNARKs with linearly homomorphic signatures (LHS), namely those supporting linear functions. Their proposed LHS however suffered from an high verification cost.
In this work we propose an efficient LHS that significantly improves on previous work in terms of verification time. Using the modular approach of Fiore and Tucker, this yields a verifier-efficient HSNP. We show that the HSNP instantiated with our LHS is particularly suited to the case when the data is taken from consecutive samples, which captures important use cases including sliding window statistics such as variances, histograms and stock market predictions.

1 Introduction

We consider the problem of verifiable computation on data streams (VCS) in which a data provider $\mathcal{D}$ streams a large amount of data to a server $\mathcal{S}$, which later computes on portions of this stream in order to reply to the queries of a client $\mathcal{C}$. This problem emerges in a variety of applications that require the continuous monitoring and analysis of vast, dynamically generated, data. Notable examples include financial data (e.g., stock-market, blockchains transactions), health or environmental sensors, network traffic, and smart metering – all fields that demand efficient and reliable solutions.

More in detail, cryptographic solutions for VCS aim to achieve five main properties: (1) Workflow: the communication from $\mathcal{D}$ to $\mathcal{S}$ is unidirectional and the stream is ordered, and $\mathcal{C}$ can verify the queries without having to follow the stream (i.e., it does not need to stay online). (2) Security: the client accepts the correct result while trusting the data provider but not trusting the server. (3) Efficiency: the communication from $\mathcal{S}$ to $\mathcal{C}$ is at most logarithmic in the size of the stream, and $\mathcal{C}$'s cost to verify the queries is smaller than the cost of running the computation. (4) Privacy-preserving: the verifier does not learn any information about the input stream beyond the received output. (5) Non-deterministic: the

server can execute and prove non-deterministic computations of the form $\exists w : y = f(x, w)$.

In a recent work [9], Fiore and Tucker proposed a solution to VCS through a new primitive that they called *homomorphic signatures for NP* (HSNP). HSNP essentially generalize classical homomorphic signatures [2,12], mainly to support non-deterministic computation (i.e., statements in NP as opposed to P) and efficient verification. The application of HSNP to VCS is natural and proceeds as follows. The data provider $\mathcal{D}$ uses a secret key to sign each element x_i of the stream and gives (x_i, σ_i) to the server. Upon a client's request of computing a function f on a portion Q of the stream, the HSNP scheme enables the server to derive a short signature σ_y that vouches for the correctness of statements of the form $\exists w$ such that $y = f(\{x_j\}_{j \in Q}, w)$ and for the fact that each input $\{x_j\}_{j \in Q}$ is legitimately signed by $\mathcal{D}$. Then any client can publicly check the correctness of y by using (f, Q, y, σ_y) and $\mathcal{D}$'s public key. Notably, this configuration satisfies the workflow property of VCS. Also, HSNP signatures are guaranteed to be short and can be efficiently verified without knowledge of the inputs, which satisfies the efficiency requirement of VCS.

Fiore and Tucker [9] proposed a generic HSNP construction based on a combination of commit-and-prove (CaP) SNARKs and linearly-homomorphic signatures for committed outputs (ComLHS)[3], and then proposed and implemented an efficient pairing-based instantiation, dubbed SPHinx. In their experiments, they confirm that SPHinx features a proof generation that, depending on the application, is between 15× and 1 300× faster than a straw-man approach in which one uses a general-purpose SNARK to prove the correctness of the computation and the validity of the signatures. However, the fast prover of SPHinx comes at the price of high verifier's costs. For a computation on a stream's portion of t values, verification is largely dominated by a multi-exponentiation of length t. Concretely, for $t \approx 1,000,000$ verification approaches 20 seconds whereas the SNARK solution enjoys a cheap verification of 11ms.

1.1 Our Results

In this work we continue this line of research on efficient HSNP for verifiable computation on data streams. Our main result is the construction of two new HSNP solutions that achieve fast proof generation without the need of sacrificing the verification. Our new schemes feature a verification time that is up to 88× faster than SPHinx. Therefore our HSNP solutions improve over the SNARK-based one on all fronts, without any tradeoff.

More in detail, our contributions are the following. Our first (and main technical contribution) is a new ComLHS scheme that can be replaced in the SPHinx construction. Indeed, it turns out that the slow verification process of SPHinx is entirely due to slow verification in their ComLHS scheme which requires the

[3] ComLHS are LHS for the non-deterministic computation that checks that a commitment opens to the result of a linear function $\boldsymbol{s}$ on signed messages $\boldsymbol{x}$, i.e., $\exists r : c = \mathsf{Commit}(\langle \boldsymbol{x}, \boldsymbol{s} \rangle; r)$.

verifier to perform a multi-exponentiation of length t. We solve this issue by proposing the first pairing-based (Com)LHS in which the verifier needs to perform only $O(t)$ *field operations*, which are way faster than exponentiations. More precisely, our scheme supports two verification profiles: (A) when verifying the output of an arbitrary linear function verification costs $O(t)$ field operations and constant group operations and pairings; (B) for structured linear functions described by a vector of powers $(1, s, s^2, \ldots, s^{t-1})$ verification costs $O(\log t)$. Our technique to enable verification based on field operations requires the prover to hold an auxiliary information with which it can help the verifier. Interestingly, this condition comes for free in the streaming setting where the signer streams signed elements in order to the prover. We formally call this model *streaming-friendly correctness*. We refer to Section 3 for more details.

An immediate application of our ComLHS (with (A)-type verification) is to replace the ComLHS in SPHinx. We dub the resulting scheme SPHinx$^+$ and show that it has 2 to 88 times faster verification than SPHinx, at the small price of 7% more expensive prover.

Next, we revisit the generic HSNP construction of [9] in order to be instantiated with our new ComLHS with (B)-type verification and thus to achieve even an asymptotic exponential speedup in verification, from $O(t)$ to $O(\log t)$. Their generic HSNP requires a CaP SNARK in which committed vectors are encoded with polynomials based on (univariate or multivariate) interpolation techniques. In Section 4 we generalize this construction to work with any vector encoding. This generalization allows us to capture the case where the vectors is encoded in the coefficients of a polynomial (so-called monomial-base encoding). Thanks to this change we can instantiate the HSNP generic construction with (monomial-base) (CaP) SNARKs and our new LHS with (B)-type verification. We dub this scheme SPHinx^{++}. On the other hand, for the techniques underlying both verification (A) and (B) to work, we need to assume that the portion of inputs $\{x_i\}_{i \in Q}$ to compute over is consecutive, i.e. $Q = \{\delta + 1, \ldots, \delta + t\}$ for a fixed δ. Even though this seems a restriction at the application level, it captures important use cases like sliding window statistics over data streams (histograms, variance, stock prices predictions), where one is interested in analyzing the "window" of the last t items of the stream. Importantly, these use cases with consecutive labels arise in several application scenarios where a faster verification can be of crucial importance, e.g., whenever signatures must be verified in real time. Concretely, consider for example a scenario where a medical device receives statistics computed from values streamed by health sensors which can be labeled using consecutive time instants, hence consecutive labels. In such cases, verifying the authenticity of the data and computation results is essential for security, while fast and efficient verification ensures that critical decisions can be made on-the-fly.

Finally, as the last contribution, we implement our schemes and evaluate them experimentally on the set of applications of HSNP for sliding window statistics.

2 Preliminaries

2.1 Notation

For a $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. Vectors are in bold. We denote by v_i the i-th coefficient of the vector $\mathbf{v}$. The inner product between two vectors $\mathbf{x}$ and $\mathbf{y}$ is denoted by $\langle \mathbf{x}, \mathbf{y} \rangle$. Let D be a distribution over a set X. For a $x \in X$, we say $x \leftarrow D$ to say that the element x is sampled from X according to D. All algorithms are assumed to be PPT machines.

A universal relation $\mathcal{R}$ is a set of triples (R, y, w) where R is a relation over $\mathcal{Y} \times \mathcal{W}$, y is called the instance and w the witness. We write $(y, w) \in \mathsf{R}$ to denote that R holds on the pairs (y, w), otherwise $(y, w) \notin \mathsf{R}$. When discussing schemes that prove statements on committed values, the witness can be a pair $(x, w) \in \mathcal{X} \times \mathcal{W}$, with x being the value committed into y. We sometimes use a finer grained specification of $\mathcal{X}$, assuming it splits over ℓ domains $\mathcal{X}_1 \times \cdots \times \mathcal{X}_\ell$ for some arity ℓ.

2.2 Bilinear groups

We denote by $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ cyclic groups of prime order and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map that is non-degenerate and efficiently computable. For the formal definition and the d-Power-DLog assumption we refer the reader to Sec. A.2.

Assumption 2.1 *Given a group generator $\mathcal{G}$, any polynomial p, and any PTT adversary $\mathcal{A}$ making $t = p(\lambda)$ queries, and for large enough λ:*

$$\Pr\left[g_1^a \leftarrow \mathcal{A}\left(\begin{array}{c} \mathsf{pp}_{\mathcal{G}}, \boldsymbol{\omega}, g_1, \{g_1^{c^i}\}_{i\in[t]}, \{g_1^{a(c^i+\omega_i)}\}_{i\in[t]}, \\ h, h^a, g_2, \{g_2^{c^i}\}_{i\in[t]}, g_2^{1/a} \end{array} \right) \middle| \begin{array}{c} \mathsf{pp}_{\mathcal{G}} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p) \leftarrow \mathcal{G}(1^\lambda), \\ g_1 \leftarrow \mathbb{G}_1, h \leftarrow \mathbb{G}_1, g_2 \leftarrow \mathbb{G}_2, \\ \boldsymbol{\omega} \leftarrow \mathbb{Z}_p^t, c \leftarrow \mathbb{Z}_p, a \leftarrow \mathbb{Z}_p \end{array} \right] = \mathsf{negl}(\lambda).$$

In Sec. A.2 we prove that assumption 2.1 holds in the Algebraic group model (AGM) [10].

2.3 Commitment schemes

A commitment scheme Com is a tuple $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ where $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ generates a commitment key; $(\mathsf{cm}, \mathsf{o}) \leftarrow \mathsf{Commit}(\mathsf{ck}, x)$ generates a commitment cm and an opening o, given a input message x; $b \leftarrow \mathsf{VerCom}(\mathsf{ck}, \mathsf{cm}, x, \mathsf{o})$ checks if o is a valid opening for cm to x. In this work, unless otherwise specified, we use commitments that are computationally binding - it is hard to open the same cm to two distinct messages - and statistically hiding - cm leaks no information about the message x.

A succinct argument of knowledge (SNARK) with specializable universal SRS for a universal relation $\mathcal{R}$ is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Derive}, \mathsf{Prove}, \mathsf{Verify})$, where $\mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$ outputs a universal structured reference string srs;

$(\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R}) \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R})$ takes a universal srs and a relation $\mathsf{R} \in \mathcal{R}$, and outputs a specialized SRS consisting of an evaluation key and a verification key; $\pi \leftarrow \mathsf{Prove}(\mathsf{ek}_\mathsf{R}, \mathsf{R}, y, w)$ takes an evaluation key for a relation R, a relation R, an instance y, and a witness w such that $(y, w) \in \mathsf{R}$, and returns a proof π; $b \leftarrow \mathsf{VerProof}(\mathsf{vk}_\mathsf{R}, y, \pi)$ takes a specialized verification key, an instance y, and a proof π, and accepts ($b = 1$) or rejects ($b = 0$). A SNARK satisfies the properties of completeness, succinctness, knowledge soundness and composable zero-knowledge, that we define in Sec.A.4.

2.4 Homomorphic signature for NP relations

HSNP extend classical homomorphic signatures (HS) to support the evaluation of non-deterministic computations on signed data. Informally, this means that anyone who knows inputs $x_1, \ldots, x_n$ and signatures $\sigma_1, \ldots, \sigma_n$ on them, can compute $y = f(x_1, \ldots, x_n, w)$ where w is a non-deterministic input, and then derive a signature $\hat{\sigma}$ on y which guarantees that there exists a w and validly authenticated $x_1, \ldots, x_n$ such that $y = f(x_1, \ldots, x_n)$. For an HSNP we consider a privacy notion which guarantees that (σ, y) leaks no information about $(x_1, \ldots, x_n)$ beyond what can be inferred from the result of the computation y.

An interesting feature of HSNP is that they can prove statements about signed inputs that are committed to. For example, one can authenticate commitments to outputs of functions on signed data, i.e., authenticate cm_y which commits to $y = f(x, w)$. This property plays an important role for composing building blocks in large protocols, see for example Sec. 2.4 and Sec. 3.

An important class of HSNP are those for linear relations, namely when the function f is a linear function of the inputs $\mathbf{x}$. To emphasize the commit-and-proof feature, we will refer to these as ComLHS.

Labelled relations . In the context of homomorphic signatures it is important that inputs are labelled. Such a technicality was introduced in [6] for the sake of verifying computations on datasets. Concretely, any data element x_i is signed with respect to the position τ_i it occupies within the stream or dataset. The verifier knows the labels $(\tau_1, \ldots, \tau_n)$ (e.g. the locations of the data in the stream or database) corresponding to the actual data $(x_1, \ldots, x_n)$ which are unknown to it, yet it wants to be assured that a computation $y = f(x_1, \ldots, x_n, w)$ was done on the inputs associated with the labels $(\tau_1, \ldots, \tau_n)$. To give an intuition on why when computing on authenticated data just signing inputs is not enough, suppose that one needs to compute a linear function $\mathbf{a} = (a_1, \ldots, a_n)$ on signed inputs $x_1, \ldots, x_n$ as $\sum_{i=1}^{n} a_i x_i$. If it happens that $x_i = x_j$ for some $i \neq j$ then this might lead to ambiguity.

Let $\mathcal{R} = \{\mathsf{R} \ : \ \mathsf{R} \subset \mathcal{Y} \times \mathcal{M}^t \times \mathcal{W}\}$ be a universal relation, for some $t \in \mathbb{N}$. A *labelled relation* is a tuple $(\mathsf{R}, (\tau_1, \ldots, \tau_t))$, where $\mathsf{R} \in \mathcal{R}$ and τ_i is a label for the i-th slot of $\mathcal{M}^t$. We will denote by $\mathsf{R}_{\mathrm{id}} := (x, x, \varnothing)$ the identity relation on $\mathcal{Y} \times \mathcal{M}^t \times \mathcal{W}$.

Definition 2.1 (HSNP). *A homomorphic signature scheme for NP consists of the following PPT algorithms:*

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L}, \mathcal{R})$**:** *on input $\lambda \in \mathbb{N}$, a set of labels*[4] *$\mathcal{L}$, and a universal relation $\mathcal{R}$, outputs a secret signing key* sk*, and a public key* vk*.*

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, m)$ *: on input the signing key* sk*, a label τ, and a message m, outputs a signature σ.*

$\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \sigma_1, \ldots, \sigma_t, w)$ *: on input the verification key* vk*, a relation $\mathsf{R} \in \mathcal{R}$ over $\mathcal{Y} \times \mathcal{M}^t \times \mathcal{W}$ for some $t \leqslant |\mathcal{L}|$, a statement $y \in \mathcal{Y}$, signatures $\{\sigma_1, \ldots, \sigma_t\}$, and a witness $w \in \mathcal{W}$, outputs a new signature σ.*

$b \leftarrow \mathsf{Verify}(\mathsf{vk}, (\mathsf{R}, \tau_1, \ldots, \tau_t), y, \sigma)$ *: on input a verification key* vk*, a labelled relation $(\mathsf{R}, \tau_1, \ldots \tau_t)$ where $\mathsf{R} \in \mathcal{R}$ is over $\mathcal{Y} \times \mathcal{M}^t \times \mathcal{W}$, a statement $y \in \mathcal{Y}$, and a signature σ, outputs 0 (reject) or 1 (accept).*

These algorithms must satisfy the following properties (see also Sec. A.6 for the formal definitions):

Definition 2.2 (Authentication correctness). *For any $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L}, \mathcal{R})$, $\tau \in \mathcal{L}, x \in \mathcal{M}$, if $\sigma_\tau \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, x)$ then $\mathsf{Verify}(\mathsf{vk}, (\mathsf{R}_{id}, \tau), x, \sigma_\tau) = 1$.*

Definition 2.3 (Evaluation correctness). *Consider any $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L}, \mathcal{R})$, any $\mathsf{R} \in \mathcal{R}$, and any set of label/message/signature triples $\{\tau_i, x_i, \sigma_{\tau_i}\}_{i=1}^t$ satisfying $\mathsf{Verify}(\mathsf{vk}, (\mathsf{R}_{id}, \tau_i), x_i, \sigma_{\tau_i}) = 1$. If $(y, (x_{\tau_i}, \ldots, x_{\tau_i}), w) \in \mathsf{R}$, and $\sigma = \mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \sigma_{\tau_1}, \ldots, \sigma_{\tau_t}, w)$ then $\mathsf{Verify}(\mathsf{vk}, (\mathsf{R}, \tau_1, \ldots, \tau_t), y, \sigma) = 1$.*

Definition 2.4 (Succinctness). *The size of the evaluated signatures depends at most logarithmically on the size of the signed and non-deterministic inputs.*

Definition 2.5 (Efficient verification). *After a preprocessing step that depends on the size of the relation being proved, the verification algorithm can be efficiently run independently on the input size.*

Definition 2.6 (Adaptive security). *This definition can be seen as an adaptation of the standard notion of unforgeability for HSNP. Intuitively, the evaluator should only be able to compute valid signatures for statements y for which it received signatures for data items $x_1, \ldots, x_t$ and knows a witness w satisfying $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$. The idea is to have an adversary $\mathcal{A}$ which can adaptively query signatures for labelled messages of its choice. Now assume that $\mathcal{A}$ outputs a valid signature σ for a statement y as output of some labelled relation $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ for some $t \geqslant 1$. Then with overwhelming probability (i) $\mathcal{A}$ must have queried signatures for each label τ_i; and, denoting these queries $\{\tau_i, x_i\}_{i \in [t]}$ (ii) the adversary $\mathcal{A}$ must know a witness w such that $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$.*

Definition 2.7 (Zero-knowledge). *The zero-knowledge property guarantees that evaluated signatures do not reveal anything about the signed inputs and the non-deterministic input beyond the fact that the signed statement satisfies the relation. The intuition is the following. The adversary queries the* Sign *algorithm on labelled messages on its choice. Then it chooses a statement y, a labelled relation $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ (where each τ_i must have been queried along with an input*

[4] In our construction in Sec 3, we do not need to specify in advance the labels

x_i resulting in the signature σ_{τ_i}) and a witness w satisfying $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$. Then $\mathcal{A}$ is given a signature σ for the statement y, which is either the result of the computation of the relation R on the signatures $\sigma_{\tau_1}, \ldots, \sigma_{\tau_t}$ or the output of the simulator that receives the secret key sk and $y, (\mathsf{R}, \boldsymbol{\tau})$ but does not have access to the signed inputs. The scheme is zero-knowledge if it is hard for $\mathcal{A}$ to tell which is the case.

A generic HSNP from ComLHS In [9] Fiore and Tucker construct a generic HSNP using a commit-and-prove zkSNARK, a polynomial commitment and a ComLHS as building blocks. Here we sketch the idea behind their construction (with a simple generalization of it that enables more instantiations). For a complete description, we refer the reader to Sec. B.

Let $\mathbb{T} = \{\tau_1, \ldots, \tau_t\}$ be a set of labels. Let $\mathsf{Dec} : \mathbb{F}[X_1, \ldots, X_n] \to \mathbb{F}^t$ be a decoding function and $\boldsymbol{\chi}(X_1, \ldots, X_n)$ be a list of t n-variate polynomials satisfying the following properties:

- Dec is $\mathbb{F}$-linear and $\mathsf{Dec}(\chi_i) = \mathbf{e}_i$, where $\mathbf{e}_i$ is the ith vector of the canonical basis of $\mathbb{F}^t$;
- $\deg(\chi_i(X_1, \ldots, X_n))/|\mathbb{F}|$ is negligible for all $i = 1, \ldots, t$;
- $\forall \mathbf{r} \in \mathbb{F}^n$ then $\chi_i(\mathbf{r})$ is computable in $O(t)$ $\mathbb{F}$-operations.

We define $\tilde{x}(X_1, \ldots, X_t) = \langle \mathbf{x}, \boldsymbol{\chi}(X_1, \ldots, X_n) \rangle$ as the polynomial encoding of a vector $\mathbf{x}$.

The first condition, the existence of a decoding function Dec, is our generalization. In SPHinx, it is instead assumed the existence of a public set $\mathbb{H} = \{h_1, \ldots, h_t\}$ such that $\chi_i(h_j) = 1$ if $i = j$ and 0 otherwise. Namely, the polynomials χ_i are the Lagrange polynomials corresponding to the set $\mathbb{H}$. For instance, in the univariate case $\chi_i(X) = \frac{h_i(X^{|\mathbb{H}|}-1)}{|\mathbb{H}|(X-h_i)}$, and thus $\mathbf{x} = \mathsf{Dec}(\tilde{x}(X))$.

The building blocks for their generic HSNP are:

1. A commitment scheme $\mathsf{Com} = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerCom})$ for committing to n-variate polynomials with coefficients in $\mathbb{F}$.
2. A universal commit-and-prove (CaP) zkSNARK $\mathsf{CP}_{\mathcal{R}}$ for Com and relation $\mathcal{R}$. Specifically, given a statement y and a commitment $\mathsf{cm}_{\tilde{x}}$, $\mathsf{CP}_{\mathcal{R}}$ proves the existence of witnesses $\tilde{x}(X_1, \ldots, X_n) \in \mathbb{F}[X_1, \ldots, X_n]$, $\mathsf{o}_{\tilde{x}}$ and w such that
$$(y, \mathsf{Dec}(\tilde{x}), w) \in \mathsf{R} \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{cm}_{\tilde{x}}, \tilde{x}, \mathsf{o}_{\tilde{x}}) = 1.$$
3. A universal CP zkSNARK $\mathsf{CP}_{\mathrm{ev}}$ for committed polynomial evaluations. Specifically, given a statement $\mathbf{r} \in \mathbb{F}^n$ and commitments $\mathsf{cm}_{\tilde{x}}, \mathsf{cm}_z$ to $\tilde{x} \in \mathbb{F}[X_1, \ldots, X_n]$ and $z \in \mathbb{F}$, it proves that
$$z = \tilde{x}(\mathbf{r}) \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{cm}_{\tilde{x}}, \tilde{x}, \mathsf{o}_{\tilde{x}}) = 1 \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{cm}_z, z, \mathsf{o}_z) = 1$$
4. A ComLHS scheme $\mathsf{ComLHS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Verify})$ for the relation
$$\mathcal{R}_{\text{com-ip}} = \{\mathsf{R}^{t,\mathbf{s}}_{\text{com-ip}} := \{(\mathsf{cm}, \mathbf{x}, \mathsf{o}) \ : \ \mathsf{VerCom}(\mathsf{ck}, \mathsf{cm}, \langle \mathbf{x}, \mathbf{s} \rangle, \mathsf{o}) = 1\} \ : \ t \leqslant N, \mathbf{s} \in \mathbb{F}^t\}$$

For a relation R that can be encoded as a circuit $F : \mathbb{F}^{t+|w|} \to \mathbb{F}$, given t signed inputs $x_1, \ldots, x_t$ with labels $\tau_1, \ldots, \tau_t$, to compute a signature on the output $y = F(x_1, \ldots, x_t, w)$, where w is a witness, the evaluator proceeds as follows. It commits to the polynomial $\tilde{x}$, say $\mathsf{cm}_{\tilde{x}}$. It then computes a proof π using $\mathsf{CP}_{\mathcal{R}}$ for the relation $\mathcal{R}$ to prove that the computation was performed correctly on the inputs $\mathbf{x} = \mathsf{Dec}(\tilde{x}(X))$ with witness w. Then it proves that these inputs are signed. For this, it gets a random challenge $\mathbf{r}$, computes $z = \tilde{x}(\mathbf{r}) = \langle \mathbf{x}, \boldsymbol{\chi}(\mathbf{r}) \rangle$, commits to z in, say, cm_z, and uses CP_{ev} to prove this committed evaluation. Finally, the ComLHS is used to prove that cm_z opens to the signed result of the linear function $\langle \mathbf{x}, \boldsymbol{\chi}(\mathbf{r}) \rangle$ of the signed inputs $\mathbf{x}$.

3 Linearly homomorphic signature

We present our construction of a linearly-homomorphic signature for committed outputs (ComLHS), which is an HSNP for the family of relations

$$\mathcal{R}_{\text{com-ip}} := \{\mathsf{R}^{(t,\mathbf{s})}_{\text{com-ip}} := (c, \mathbf{x}, o) \ : \ \mathsf{VerCom}(\mathsf{ck}, c, \langle \mathbf{x}, \mathbf{s} \rangle, o) = 1, t \leqslant N, \mathbf{s} \in \mathbb{F}^t\}$$

where $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{VerCom})$ is a commitment scheme.

Our scheme supports labels that are integers, without an a-priori fixed upper bound, and has streaming-friendly correctness. Namely, when evaluating on a set of labels $\mathbb{T} = \{\tau_1, \ldots, \tau_t\} \in \mathbb{N}^t$, it achieves correctness if the evaluator has received (a stream of) signatures with labels $1, 2, \ldots, \tau_t$.

We construct our scheme in bilinear groups, and we additionally make use of a secure (EUF-CMA) signature $\Sigma = (\Sigma.\mathsf{KG}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Verify})$, and a proof system $\mathsf{NIZK} = (\mathsf{K}, P, V)$ for the relation $\mathsf{R}_{\text{c}} = \{(y, (z, w)) \ : \ y = g_1^z h^w\}$. Concretely, we will instantiate Σ with Schnorr's signature and NIZK with Okamoto's sigma protocol [15].

We describe our ComLHS in Fig. 1. Below we provide an intuition of the construction, comparing it to the scheme proposed in [9].

Previous LHS. First, we will sketch the LHS proposed in [9]. We will only include the parts that are important to understand our improvement. A key generation phase produces secret keys a, b along with a public key consisting among other things of $\Gamma_1 = h_1^a, \Gamma_2 = g_2^{1/a}, B = g_1^b$. To sign an input x with label τ, the signer first hashes the label τ to obtain $C_\tau \leftarrow H(\tau)$. It computes $\Lambda_\tau = (C_\tau g_1^{br_\tau + x})^a$, where r_τ is obtained from applying a PRF to τ. The signature on x, τ is then (Λ_τ, r_τ). To compute a linear function with public coefficients $\mathbf{s} \in \mathbb{Z}_p^t$ on signed inputs $(x_1, \ldots, x_t)$ with labels $\tau_1, \ldots, \tau_t$, given a commitment $y = g_1^{\langle \mathbf{x}, \mathbf{s} \rangle} h^w$, the evaluator first applies the linear function to the pseudo-randomness $(r_{\tau_1}, \ldots, r_{\tau_t})$ resulting in $r = \langle \mathbf{r}, \mathbf{s} \rangle$. It then computes the product $\Lambda \leftarrow \Gamma_1^w \prod_{i=1}^t (\Lambda_i)^{s_i}$. The evaluated signature then includes (y, Λ, r). To verify the above signature with the public key, the verifier computes $C \leftarrow \prod_{i=1}^t (H(\tau_i))^{s_i}$ and then checks that $e(yB^r C, g_2) = e(\Lambda, \Gamma_2)$.

From the above description, it appears that the cost for both evaluating and verifying an evaluated signature is dominated by the computation of $C =$

$\prod_{i=1}^{t} H(\tau_i)^{s_i}$ which is a multi-exponentiation of t factors, namely t $\mathbb{G}_1$-operations. This is the main bottleneck for SPHinx, making its verification orders of magnitude slower than that from a solution using general purpose SNARKs due to both hashing into the group t times and then computing the multi-exponentiation.

Our algebraic LHS Our idea to speed up verification is to delegate the computation of the term C to the prover and let it provide a proof of correct computation which is fast to verify. Notably, the challenge of achieving this is that we simultaneously want to keep the prover's cost in the same order of magnitude, that is we cannot use a general purpose delegation scheme which would likely need to encode the multi-exponentiation in a circuit, with prohibitive overhead.

Therefore, to achieve our goal we change the ComLHS scheme in such a way that the terms C_τ have a convenient algebraic form which leads more naturally to an efficient and succinct proof of correct computation. Our starting point is to replace the random oracle in the terms C_τ with $g_1^{c^\tau}$ for a secret c that is part of the secret key. This way, the term $C = \prod_{i=1}^{t} (g_1^{c^{\tau_i}})^{s_i}$ can be interpreted as a commitment to the polynomial $\sum_{i=1}^{t} s_i X^{\tau_i}$, for which the prover can provide a succinct proof using a KZG-style proof (see Sec. A.5). The issue is that the terms C_{τ_i} in the product C depend on the size of the whole stream, while we want the computation as well as the verification to depend only on the size of the sample $\mathbb{T}$. To avoid this we need to make some restriction on the sample set $\mathbb{T}$. For simplicity, we focus on the case when the label set $\mathbb{T} = \{\tau_1, \ldots, \tau_t\}$ can be represented a as consecutive labels[5], namely $\{\delta + 1, \ldots, \delta + t\}$ for a $\delta \in \mathbb{N}$ which is possibly $O(N)$, where N is the size of the whole stream.

The key generation produces secret keys a, b, c along with a public key consisting among other things of $\Gamma_1 = h^a, \Gamma_2 = g_2^{1/a}, B = g_1^b$, with $\mathbb{G}_1 = (g_1), \mathbb{G}_2 = (g_2)$ and h being a random element from $\mathbb{G}_1$, and a PRF key κ.

To sign a data element x with label τ, the signer now computes $C_\tau^{(1)} := g_1^{c^\tau}$. It then computes r_τ from a PRF F_κ applied to τ and then computes the element $\Lambda_\tau = (C_\tau^{(1)} g_1^{br_\tau + x})^a$ as before. Note that this uniquely binds the Λ_τ to the label τ as long as c is secret. The signature σ_τ then includes $C_\tau^{(1)}$, $C_\tau^{(2)} := g_2^{c^\tau}, r_\tau, \Lambda_\tau$.

To compute a linear function with public coefficients $\mathbf{s} \in \mathbb{Z}_p^t$ on signed inputs $(x_{\delta+1}, \ldots, x_{\delta+t})$ with labels $\delta+1, \ldots, \delta+t$, given a commitment $y = g_1^{\langle \mathbf{x}, \mathbf{s} \rangle} h^w$, the evaluator proceeds as follows. As before, the evaluator applies the linear function on the pseudo-randomness $\mathbf{r}$, resulting in $r = \langle \mathbf{r}, \mathbf{s} \rangle$. The goal is to compute $g_1^{\bar{S}(c)}$ where $\bar{S}(X) = \sum_{i=1}^{t} s_i X^{\delta+i}$ and provide a succinct proof of this. Define the polynomial $S(X) := \sum_{i=1}^{t} s_i X^{i-1}$. Note that $\bar{S}(X) = X^{\delta+1} S(X)$ and that $S(X)$ does not depend on δ. The evaluator first computes $C_S := \prod_{i=0}^{t-1} (C_i^{(1)})^{s_{i+1}}$ and $C_{\bar{S}} := \prod_{i=1}^{t} (C_{\delta+i}^{(1)})^{s_i}$. It then computes a KZG-like proof $\pi_S := g_1^{q(u)}$ where $q(X)$

[5] In fact, in order to make the prover depend only on the size of $\mathbb{T}$ one can even allow gaps between labels provided they are of constant size. We will argue that both cases are enough to capture realistic use cases, for example the important case of sliding-window statistics.

is the polynomial defined by $q(X) = \frac{S(X)-S(u)}{X-u}$ for a random point u, derived e.g. from a random oracle. Then it computes the product $\Lambda \leftarrow \Gamma_1^w \prod_{i=1}^{t} (\Lambda_i)^{s_i}$. The outputed evaluated signature then includes $C_S, C_{\bar{S}}, \pi_S, r, \Lambda$.

To verify the above signature, the verifier first derives the random point u from the RO. It then computes $S(u) = \sum_{i=1}^{t} s_i u^{i-1}$ and checks that $e(yB^r C_{\bar{S}}, g_2) = e(\Lambda, \Gamma_2)$ and that $e(\pi_S, g_2) = e(C_S g_1^{-S(u)}, g_2^{c-u})$ and $e(C_{\bar{S}}, g_2) = e(C_S, C^{(2)}_{\delta+1})$, the latter check testing that $\bar{S}(X) = X^{\delta+1} S(X)$. Note that the verifier cannot, in principle, check that the term $C^{(2)}_{\delta+1}$ is indeed $g_2^{c^{\delta+1}}$. To ensure this, we let the signer include as part of the output a signature, e.g. Schnorr's, that the value $C^{(2)}_{\tau}$ correspond to the label τ. The signature corresponding to the label $\delta + 1$ is then appended to the evaluated signature from the evaluator so the verifier just needs to check the validity of the signature on the element $C^{(2)}_{\delta+1}$ under the signer's public key, which it trusts.

It is clear that in our LHS now the verification only involves $O(t)$ field operations. We will later introduce further optimizations that allow reducing the verification cost to $O(\log t)$ field operations.

Further comments Here we explain some technical details in our LHS that we omitted in the above intuitive description.

In both [9] and our ComLHS, the evaluator uses a succinct NIZK for proving that $y = g_1^z h^w$, where $z = \langle \mathbf{x}, \mathbf{s} \rangle$. This is needed in the proof of adaptive security to extract the witness w.

In our ComLHS, the evaluator needs to compute the element $C'_{\bar{S}} = g_1^{\alpha \bar{S}(c)}$ so as to allow the extraction of the polynomial $\bar{S}(X)$ in the security proof. To do this, we include the elements $g_1^{\alpha}, g_2^{\alpha}$ in the verification key, and let the signer produce and send the element $g_1^{\alpha c^{\tau}}$ for each label τ. A final pairing is needed to check that $C'_{\bar{S}} = C^{\alpha}_{\bar{S}}$.

To sign labelled inputs, we make use of a PRF $F_\kappa : \mathbb{N} \to \mathbb{Z}_p$ to generate the randomness r_τ. This is needed for proving the zero-knowledge property of the ComLHS. This makes the value $r = \langle \mathbf{r}, \mathbf{s} \rangle$ a deterministic function of the labels, the relation $\mathsf{R}^{t,\mathbf{s}}_{\text{com-ip}}$ and the secret key. All this information being known to the simulator, it can simulate r.

Theorem 3.1. *If* NIZK *is a proof of knowledge for* R_c*,* Σ *is a EUF-CMA signature, RO is modelled as a random oracle and* F *is a secure PRF, then the ComLHS scheme in Fig. 1 satisfies authentication correctness, evaluation correctness and adaptive security under assumption 2.1. Furthermore, if* NIZK *is zero-knowledge, then the LHS is zero-knowledge.*

The proof of Theorem 3.1 is in Sec. C.

3.1 Cost of our ComLHS

Here we give the costs for our ComLHS in Fig. 1 and compare it with the one in Table 1 of Fiore and Tucker [9].

$\mathsf{KeyGen}(1^\lambda, \mathcal{R}_{\mathbf{com\text{-}ip}})$:

1 $\mathsf{pp}_\mathcal{G} := (\mathbb{G}_1 = (g_1), \mathbb{G}_2 = (g_2), \mathbb{G}_T, e, p) \leftarrow \mathcal{G}(1^\lambda), h \leftarrow \mathbb{G}_1$;
2 $\mathsf{ck} := (\mathsf{pp}_\mathcal{G}, h)$;
3 $\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$;
4 $\mathsf{crs} \leftarrow \mathsf{NIZK.K}(1^\lambda, \mathsf{ck})$;
5 $\kappa \leftarrow \mathcal{K}$;
6 $a, b, c \leftarrow \mathbb{Z}_p, \alpha \leftarrow \mathbb{Z}_p$;
7 $\mathit{\Gamma}_1 \leftarrow h^a, \mathit{\Gamma}_2 \leftarrow g_2^{1/a}, B \leftarrow g_1^b$;
8 Return $\mathsf{sk} := ((a, b, c), \alpha, \kappa, \mathsf{sk}_\Sigma), \mathsf{vk} := (\mathsf{ck}, \mathsf{crs}, \mathit{\Gamma}_1, \mathit{\Gamma}_2, B, g_1^c, g_2^c, g_1^\alpha, g_2^\alpha, \mathsf{pk}_\Sigma)$;

$\mathsf{Sign}(\mathsf{sk}, \tau, x)$

1 $C_\tau^{(1)} \leftarrow g_1^{c^\tau}, C'_\tau \leftarrow g_1^{\alpha c^\tau}, C_\tau^{(2)} \leftarrow g_2^{c^\tau}$;
2 $r_\tau \leftarrow F_\kappa(\tau)$;
3 $\mathit{\Lambda}_\tau \leftarrow \left(C_\tau^{(1)} g_1^{x + b r_\tau}\right)^a$;
4 $sig_\tau \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_\Sigma, C_\tau^{(2)} \| \tau)$;
5 Return $\sigma_\tau = (x_\tau, \mathit{\Lambda}_\tau, r_\tau, C_\tau^{(1)}, C'_\tau, C_\tau^{(2)}, sig_\tau, \varnothing)$;

$\mathsf{Eval}(\mathsf{vk}, \mathcal{R}^{t,\mathbf{s}}_{\mathbf{com\text{-}ip}}, \delta, y, \{\sigma_\tau\}_{\tau \in [N]}, w)$:

1 Parse $\boldsymbol{s} \in \mathbb{Z}_p^t$ from $\mathcal{R}^{(t,\mathbf{s})}_{\text{com-ip}}$;
2 For $i \in [t]$ do:
  2.1 Parse $(x_{\delta+i}, \mathit{\Lambda}_{\delta+i}, r_{\delta+i}, C^{(1)}_{\delta+i}, C'_{\delta+i}, C^{(2)}_{\delta+i}, sig_{\delta+i}, \varnothing) = \sigma_{\delta+i}$;
  2.2 Parse $(x_i, \mathit{\Lambda}_i, r_i, C_i^{(1)}, C'_i, C_i^{(2)}, sig_i, \varnothing) = \sigma_i$;
3 Set $\mathbf{x} = (x_{\delta+1}, \ldots, x_{\delta+t}), \mathbf{r} = (r_{\delta+1}, \ldots, r_{\delta+t})$;
4 $z \leftarrow \langle \mathbf{x}, \mathbf{s} \rangle, r \leftarrow \langle \boldsymbol{r}, \boldsymbol{s} \rangle$;
5 $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK}.P(\mathcal{R}_\mathrm{c}, \mathsf{crs}, y, (z, w))$;
6 $\mathit{\Lambda} \leftarrow \mathit{\Gamma}_1^w \cdot \prod_{i \in [t]} (\mathit{\Lambda}_{\delta+i})^{s_i}$;
7 Set $S(X) := \sum_{i=0}^{t-1} s_{i+1} X^i, \bar{S}(X) := \sum_{i=1}^{t} s_i X^{\delta+i}$;
8 $C_S \leftarrow g_1^{s_1} \cdot \prod_{i \in [t]} (C_{i-1}^{(1)})^{s_{i+1}}, C_{\bar{S}} \leftarrow \prod_{i \in [t]} (C_{\delta+i}^{(1)})^{s_i}, C'_{\bar{S}} \leftarrow \prod_{i \in [t]} (C'_{\delta+i})^{s_i}$;
9 $u \leftarrow RO(\mathsf{vk}, \mathcal{R}^{(t,\mathbf{s})}_{\text{com-ip}}, \delta, sig_{\delta+1}, y, \mathit{\Lambda}, r, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, \pi_{\mathsf{NIZK}})$;
10 Compute $q(X) = \sum_{i=0}^{t-2} q_i X^i := \frac{S(X) - S(u)}{X - u}$;
11 $\pi_S \leftarrow \prod_{i=0}^{t-2} \left(C_i^{(1)}\right)^{q_i}$;
12 Return $\sigma := (y, \mathit{\Lambda}, r, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, sig_{\delta+1}, \pi_{\mathsf{NIZK}}, \pi_S)$

$\mathsf{Verify}(\mathsf{vk}, \mathcal{R}^{(t,\mathbf{s})}_{\mathbf{com\text{-}ip}}, \delta, \sigma)$:

1 If $\sigma = \sigma_\tau = (x, \mathit{\Lambda}_\tau, r_\tau, C_\tau^{(1)}, C'_\tau, C_\tau^{(2)}, sig_\tau, \varnothing)$:
  1.1 If $\mathit{\Sigma}.\mathsf{Verify}(\mathsf{pk}_\Sigma, sig_\tau, C_\tau^{(2)} \| \tau) = 1 \wedge (e(g_1^x B^{r_\tau}, g_2) = e(\mathit{\Lambda}_\tau, \mathit{\Gamma}_2))$: return 1;
  1.2 Else: return 0;
2 Else if $\sigma = (y, \mathit{\Lambda}, r, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, sig_{\delta+1}, \pi_{\mathsf{NIZK}}, \pi_S)$:
  2.1 $u \leftarrow RO(\mathsf{vk}, \mathcal{R}^{(t,\mathbf{s})}_{\text{com-ip}}, \delta, y, \mathit{\Lambda}, r, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, \pi_{\mathsf{NIZK}}, \pi_S)$;
  2.2 $\rho \leftarrow \sum_{i=0}^{t-1} s_{i+1} u^i$;
  2.3 If $\mathit{\Sigma}.\mathsf{Verify}(\mathsf{pk}_\Sigma, sig_{\delta+1}, C^{(2)}_{\delta+1} \| \delta + 1) \neq 1$: return 0;
  2.4 If $e(\pi_S, g_2^c g_2^{-u}) \neq e(C_S g_1^{-\rho}, g_2) \vee e(C_S, C^{(2)}_{\delta+1}) \neq e(C_{\bar{S}}, g_2)$: return 0;
  2.5 If $e(C'_{\bar{S}}, g_2) \neq e(C_{\bar{S}}, g_2^\alpha)$: return 0;
  2.6 If $(\mathsf{NIZK}.V(\mathcal{R}_\mathrm{c}, \mathsf{crs}, y, \pi_{\mathsf{NIZK}}) = 1) \wedge e(y B^r C_{\bar{S}}, g_2) = e(\mathit{\Lambda}, \mathit{\Gamma}_2)$: return 1;
  2.7 Else: Return 0;

Fig. 1. Our new ComLHS

Table 1. Comparison of our LHS with the one in [9]. Our construction improves on the verification cost by removing the expensive group operations that were the main bottleneck of previous work. At the cost of adding a constant overhead in the proving time and signature size. The values of the last column refer to our instantiation with the BLS12-381 curve. For evaluation and verification we do not include the costs that do not depend on t. Verification requires performing a constant number of pairings. These do not affect the efficiency of the scheme, as shown our experiments, see Sec. 5.

Scheme	Evaluation	Verifier	Evaluated sig. size	Evaluated sig. size (in Bytes)
[9]	t $\mathbb{G}_1$-operations $2t$ $\mathbb{F}$-operations	t $\mathbb{G}_1$-operations	$2\mathbb{G}_1$ $4\mathbb{F}$	255
This work arbitrary linear functions	$5t$ $\mathbb{G}_1$-operations $3t$ $\mathbb{F}$-operations	t $\mathbb{F}$-operations	$8\mathbb{G}_1, 1\mathbb{G}_2$ $4\mathbb{F}$	478
This work structured linear functions	$4t$ $\mathbb{G}_1$-operations $2t$ $\mathbb{F}$-operations	$\log t$ $\mathbb{F}$-operations	$8\mathbb{G}_1, 2\mathbb{G}_2$ $5\mathbb{F}$	557

(A)-type verification: arbitrary linear functions. This is the case where the vector $\mathbf{s}$ is arbitrary. The cost for the evaluator is dominated by the computation of the elements $\Lambda, C_S, C_{\bar{S}}, C'_{\bar{S}}, \pi_S$, each one being a multi-exponentiation of size t. Using Pippenger's algorithm this takes around $t\log(q)/\log(t\log(q))$ group operations.

The cost for the verifier is dominated by the computation of the term ρ, which is an inner product of two vectors of size t. This requires $O(t)$ field operations.

An evaluated signature consists of the proof π_{NIZK} which counts for one group element and two field elements, the field elements r, and the group elements $y, \Lambda, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, \pi_S$ and the signature $sig_{\delta+1}$.

Faster prover It is possible to verify π_S without the need of communicating C_S. Notice that line 2.4 of Verify checking that $q(X) = \frac{S(X)-S(u)}{X-u}$ and $\bar{S}(X) = X^{\delta+1}S(X)$ can be replaced by checking that $\bar{S}(X) - S(u)X^{\delta+1} = q(X)(X-u)X^{\delta+1} = \frac{S(X)-S(u)}{X-u}(X-u)X^{\delta+1}$. This can be checked by replacing line 2.4 with the check $e(C_{\bar{S}}, g_2)e(g_1^{-S(u)}, C^{(2)}_{\delta+1}) = e(\pi_S, C^{(2)}_{\delta+2}(C^{(2)}_{\delta+1})^{-u})$. For this, the Eval algorithm should also send the element $C^{(2)}_{\delta+2}$ along with a signature. This saves one multi-exponentiation for the prover (since C_S is not needed) and one pairing for the verifier.

(B)-type verification: structured linear functions. If we assume now that the vector $\mathbf{s}$ has additional structure, for example $\mathbf{s} = (1, s, s^2, \ldots, s^{t-1})$, then we can further reduce the cost for both the prover and the verifier.

By exploiting the structure of $\mathbf{s}$, the evaluator does not have to produce the proof π_S. The only two modifications in Fig. 1 are the following. In Eval the evaluator sends $(g_2^{c^t}, sig_t)$. In Verify the verifier needs to additionally check the signature sig_t and that $e(C_S, g_2(g_2^c)^{-s}) = e(g_1, g_2(g_2^{c^t})^{-s^t})$ which tests $S(c) = \frac{1-(sc)^t}{1-sc}$. This saves the computation of $u, q(X), \pi_S$ in Eval (lines 9, 10, 11) and of u, ρ and the check $e(\pi_S, g_2^c g_2^{-u}) \neq e(C_S g_1^{-\rho}, g_2)$ (lines 2.1, 2.2 and first part of line 2.4) in Verify.

In particular, the cost for the verifier is then dominated by computing the exponent s^t, which can be done in $O(\log t)$ field operations. Therefore, in this special case the verifier is succinct in the size of the sample set $\mathbb{T}$.

4 Application to efficient HSNP

Here we use our ComLHS to obtain efficient HSNP schemes.

4.1 Our SPHinx$^+$ instantiation with our ComLHS and Marlin

Fiore and Tucker [9] give an efficient instantiation of their generic HSNP (see also sec. 2.4, called SPHinx, obtained by instantiating the building blocks as follows: $\chi_i(X)$ are univariate Lagrange polynomials; Com is KZG (see Sec. A.5); $\mathsf{CP}_{\mathcal{R}}$ is a commit-and-prove variant of Marlin that they propose; $\mathsf{CP}_{\mathrm{ev}}$ is a version of KZG with committed outputs, optmized from [8]; ComLHS is instantiated with their own scheme.

In our paper we exploit the modularity of the SPHinx scheme, by taking the same instantiation as above except that we replace their ComLHS with our new, verifier-efficient ComLHS scheme in Fig. 1. The rest of the building blocks remains the same. We dub this instantiation SPHinx$^+$. As we confirm in Section 5, the faster verification algorithm of our ComLHS based on field operations makes the SPHinx$^+$ HSNP very efficient for the verifier, removing the drawback of SPHinx.

4.2 Our SPHinx^{++} instantiation with our ComLHS and VOMarlin

We show a second HSNP instantiation, dubbed SPHinx^{++}, obtained using our ComLHS with the (B)-type verification. However, this requires the public linear function to be structured. To this end, to build SPHinx^{++} we set $\boldsymbol{\chi}(X_1, \ldots, X_n) = \boldsymbol{\chi}(X)$ univariate with $\chi_i(X) = X^{i-1}$. The polynomial $\tilde{x}(X)$ encoding the inputs $\mathbf{x}$ is then just the polynomial having $\mathbf{x}$ as coefficients, namely $\tilde{x}(X) = f_{\mathbf{x}}(X) := \sum_{i=1}^{t} x_i X^{i-1}$.

To support this choice, though, we should replace CP.Marlin with a zkSNARK that is commit-and-prove w.r.t. a polynomial commitment with such monomial-basis encoding of vectors. Recently, several PIOPs have been proposed that can be compiled with polynomial commitments in the monomial basis, for example Claymore [16] and the ones from vector oracles [17]. We focus on the latter, specifically those for R1CS relations (see Sec. D), that we name VOMarlin for the sake of clarity. To use this scheme in the HSNP construction, we make it commit-and-prove, following an approach similar to the one used in [9] for Marlin. We give the construction in the Appendix D.

5 Applications and experiments

HSNPs yield a natural application to verifiable computation on data streams. In this domain, most concrete use cases fall into so called sliding-window statistics,

a method for continuously updating statistical measures over a subset of data, focusing on the t most recent items. They are essential for real-time data analysis across various applications as they provide dynamic and efficient insights into evolving data streams with limited computational resources. They are particularly valuable for continuous monitoring in fields such as finance, network security, and sensor data management.

Sliding-window statistics fit the requirements of our HSNP realizations in which the data owner can stream values with consecutive labels, and computation occurs on a portion of consecutive data. In this section, we evaluate the performance of our HSNP realizations using three benchmarks, that are the following three sliding-window statistics considered in [9]:

Variance: Measures the spread of a set of points from their average value within a given window. For example, this is useful in financial data analysis or sensor data monitoring, where sudden changes in variance could indicate important shifts or anomalies. Variance can be expressed by an R1CS circuit of $t + 2$ variables and $2t + 2$ constraints.

Histogram: Is useful for visualizing and understanding the distribution of data within each window. For example by plotting the frequency of data points in predefined intervals, histograms can reveal patterns or trends that evolve over time in the data stream. Denoting by k the number of intervals, histograms can be expressed by an R1CS circuit with $36tk$ constraints and $96tk$ variables.

Multi-linear regression: Is a generalisation of linear regression that allows to take into account multiple factors before outputting a prediction. This is particularly relevant in scenarios where relationships between variables need to be analyzed dynamically as new data arrives. Previous work [9] showed that for n days, computing both k additional features and the prediction of the model can be represented by an $R1CS$ with $n(2k^2+8k+4)+k^3+5k^2+9k+6$ constraints, and $n\left(\frac{3}{2}k^2+\frac{15}{2}k+4\right)+k^3+4k^2+7k+4$ variables.

5.1 Implementation and experimental setup

The implementation of our scheme extends the previous SPHinx library. It is done in Rust and based on the arkworks[6] libraries. Pairing-friendly curves are instantiated with BLS12-381 [3]. The experiments were run on a Debian GNU/Linux virtual machine running with 8 cores Xeon-Gold-6154 clocked at 3GHz and with 128 GB of RAM. All the reported timings correspond to the median of measurements over 10 executions.

We evaluate the performance of our implementation of SPHinx$^+$ on the three benchmarks above, comparing it with SPHinx and SigMarlin. The latter is a generic HSNP that uses Marlin to prove both the computation and that the corresponding inputs were signed. We simulate SigMarlin by running Marlin on R1CS with an additional 5,000 constraints per signed input to include the cost of verifying signatures. As explained in [9] it is important to note that the addition of 5,000 constraints per signed input is a lowered estimation, with state-of-the-art suggesting that the actual costs could be double this amount.

[6] https://github.com/arkworks-rs

To benchmarks SPHinx and SPHinx$^+$ we executed all components of our HSNP protocol on the specified R1CS instance and include the associated costs.

5.2 Evaluation

Signing For SPHinx$^+$ signing takes 1.7ms which is around 3 times slower than SPHinx signing time as a result of few more group operations. However we argue that this trade off is worth in order to improve the verification time and unnoticeable in practice. The signature still remains of constant size.

Proving and verification time Figure 2 shows the verification time for histograms with 4 buckets and variance computation over varying input sizes. Verification time for MLR with 30 additional features is similar to the one of histograms and for lack of space is not included as a plot.

Our experiments show that SPHinx$^+$ improves significantly on SPHinx performances from 2 times for histograms and MLR with $k = 4$ and $t = 2^{12}$ to 88 times faster for the biggest instance we tested (variance with $t = 2^{20}$), staying under a quarter of a second. The additional proving cost stays constant through any input size and is of around 7%, which is negligible compared to the cost of running Marlin. It is worth mentioning that the implementation does not currently use any batching or preprocessing of the pairings. We argue that this would further reduce the verification time bringing it closer to SigMarlin without inheriting its limitations on proving time and RAM usage. Compared to SPHinx$^+$, the advantage of SPHinx^{++} in terms of verification time is in the order of milliseconds, and t had to be increased significantly to see a noticeable difference. For this reason and for lack of space, we do not include it here.

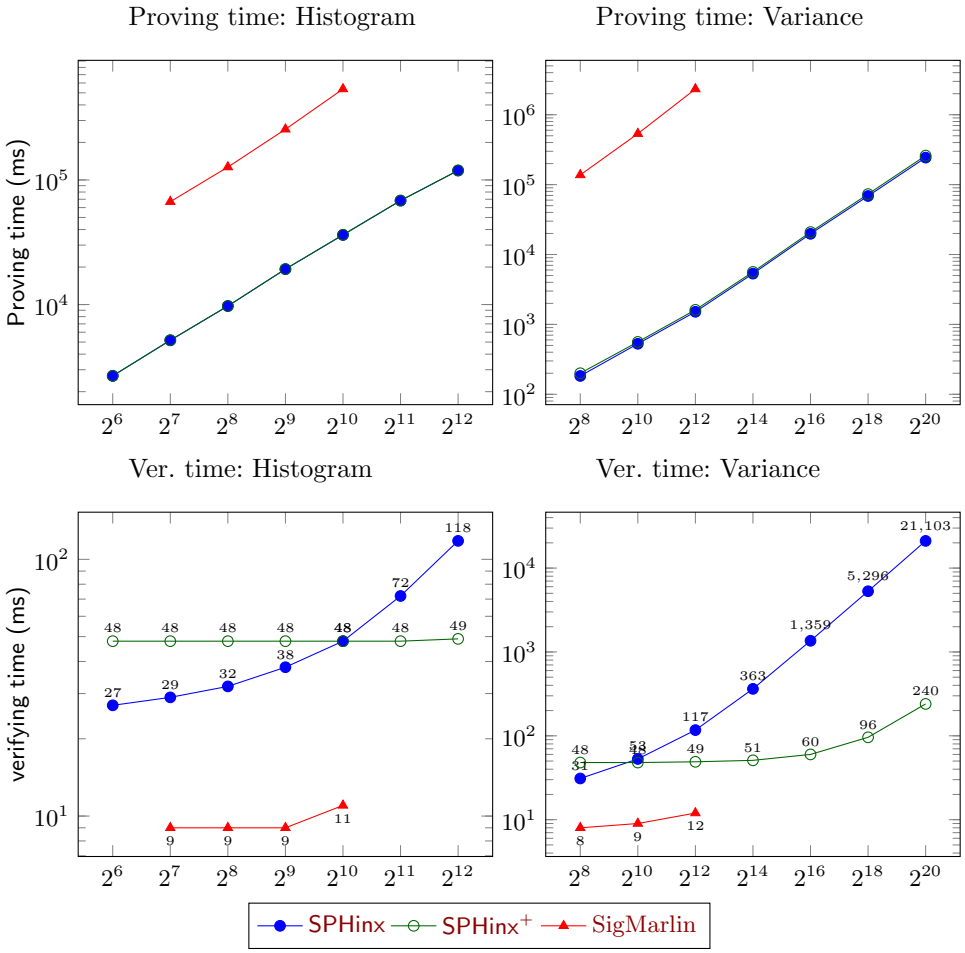

Fig. 2. Comparison of SPHinx, SPHinx$^+$ and SigMarlin on: Histogram with 4 buckets and varying input (left), and variance (right). The x-axis displays the number of signed inputs. Plots in log-scale.

Acknowledgments

We would like to thank Yuncong Zhang for clarifying some points in the VO-Proof paper and the anonymous reviewers for their comments. This work is supported by the PICOCRYPT project that has received funding from the European Re- search Council (ERC) under the European Unions Horizon 2020 research and innovation programme (Grant agreement No. 101001283), partially supported by projects PRODIGY (TED2021-132464B- I00) and ESPADA (PID2022-142290OB-I00) funded by MCIN/AEI/10.13039/501100011033/. This work is part of the grant JDC2023-050791-I, funded by MCIN/AEI/10.13039/501100011033 and the ESF+.

References

1. Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 121–151. Springer, Heidelberg, August 2020.
2. Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, Heidelberg, May 2011.
3. Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. https://eprint.iacr.org/2017/1050.
4. Xavier Boyen. The uber-assumption family (invited talk). In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 39–56. Springer, Heidelberg, September 2008.
5. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.
6. Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, Heidelberg, May 2013.
7. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.
8. Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 124–154. Springer, Heidelberg, May 2020.
9. Dario Fiore and Ida Tucker. Efficient zero-knowledge proofs on signed data with applications to verifiable computation on data streams. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1067–1080. ACM Press, November 2022.
10. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.
11. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018.
12. Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Heidelberg, February 2002.
13. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

14. Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.
15. Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Heidelberg, August 1993.
16. Alan Szepieniec and Yuncong Zhang. Polynomial IOPs for linear algebra relations. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 523–552. Springer, Heidelberg, March 2022.
17. Yuncong Zhang, Alan Szepieniec, Ren Zhang, Shi-Feng Sun, Geng Wang, and Dawu Gu. VOProof: Efficient zkSNARKs from vector oracle compilers. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 3195–3208. ACM Press, November 2022.

A Preliminary material

A.1 Schwartz-Zippel Lemma

Lemma A.1 (Schwartz-Zippel). *Let f be a non-zero polynomial in n variables of total degree d over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$, and let $r_1, r_2, \ldots, r_n$ be selected uniformly and independently at random from S. Then the probability that f evaluates to zero at the point $(r_1, r_2, \ldots, r_n)$ is bounded by:*

$$Pr[f(r_1, r_2, \ldots, r_n) = 0] \leqslant \frac{d}{|S|}.$$

A.2 Bilinear groups

Let $\mathcal{G}$ be a group generator which takes as input a security parameter λ. A bilinear group is a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p) \leftarrow \mathcal{G}(1^\lambda)$ where:

- $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic groups of prime order p.
- $\mathbb{G}_T$ is a cyclic group of the same order p.
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map, which satisfies the following properties:
  1. Bilinearity: For all $g_1, g_2 \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
  2. Non-degeneracy: There exist $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ such that $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$, where $1_{\mathbb{G}_T}$ is the identity element of $\mathbb{G}_T$.
  3. Computability: There is an efficient algorithm to compute $e(g_1, g_2)$ for any $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$.

A.3 Computational assumptions

Assumption A.1 (d-Power-DLog) *Given a degree bound $d \in \mathbb{N}$, the d-Power Discrete Logarithm assumption [14] holds for a bilinear group generator $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$:*

$$\Pr\left[c' = c \quad : \begin{array}{l} \mathsf{pp}_\mathcal{G} \leftarrow \mathcal{G}(1^\lambda), c \leftarrow \mathbb{F} \\ c' \leftarrow \mathcal{A}(\mathsf{pp}_\mathcal{G}, \{g_1^{c^i}\}_{i\in[t]}, \{g_2^{c^i}\}_{i\in[t]}) \end{array}\right] = \mathsf{negl}(\lambda). \tag{1}$$

Security of assumption 2.1 To show that assumption 2.1 holds in the AGM. we will prove that it falls into the Über assumption family of [1, 4] for a given algebraic adversary $\mathcal{A}$ and a bilinear group generator $\mathcal{G}$. Note that as the target element in our assumption is g_1^a and that we are using type 3 bilinear group, the adversary will not be able to use any elements from $\mathbb{G}_2$ or $\mathbb{G}_T$ and therefore we define the simplified Game 3.

Definition A.2. *Let $\boldsymbol{R} \in \mathbb{Z}_p[X_1, \ldots, X_m]^r$ be a vector of r m-variate polynomials over $\mathbb{Z}_p$ and let $W \in \mathbb{Z}_p[X_1, \ldots, X_m]$ be an m-variate polynomial over $\mathbb{Z}_p$. We say that W is linearly independant on $\boldsymbol{R}$ if there exist no coefficients $\{a_i\}_{i\in[r]} \in \mathbb{Z}_p^r$ such that:*

$$W = \sum_{i=1}^{r} a_i R_i$$

Here we will define an adapted algebraic game.

Input: $\boldsymbol{R} \in \mathbb{Z}_p[X_1, \ldots, X_m]^r$ a vector of m-variate polynomials, $R' \in \mathbb{Z}_p[X_1, \ldots, X_m]$ the targeted polynomial.
- $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$
- $(g_1) \leftarrow \mathbb{G}_1$
- $\boldsymbol{x} \leftarrow \mathbb{Z}_p^m$
- $\boldsymbol{U} \leftarrow (g_1^{R_1(\boldsymbol{x})}, \ldots, g_1^{R_r(\boldsymbol{x})})$
- $A \leftarrow \mathcal{A}(\boldsymbol{U})$

Output: $A = g_1^{R'(\boldsymbol{x})}$

Fig. 3. Game $(R, \mathbf{R}) - \mathsf{Uber}_{\mathcal{G}}^{\mathcal{A}}(1^\lambda)$

Definition A.3. *(Non-triviality). For a type 3 billinear group, a vector of polynomials $\boldsymbol{R} \in \mathbb{Z}_p[X]^r$ and a target element $R' \in \mathbb{Z}_p[X]$, we will say that the tuple $(\boldsymbol{R}, R')$ is non-trivial if R' is linearly independent from $\boldsymbol{R}$.*

As shown by Bauer and al. [1], for a type 3 pairing, if the tuple $(\boldsymbol{R}, R')$ is non trivial and as long as $(q_1, q_2) - \mathbf{dlog}_{\mathcal{G}}$ assumption holds, then the corresponding Uber assumption holds for any algebraic adversary. Furthermore the security loss of the reduction will only depend on the degree of the involved polynomials, i.e. the maximum degree of R' and the polynomials of $\boldsymbol{R}$ and the components of $\boldsymbol{R}$.

Lemma A.4. *Assumption 2.1, is secure under the uber assumption.*

Proof. Let us suppose there is an algorithm $\mathcal{A}$ that solves assumption 2.1 outputting $A = g^a$ for a random integer a. We will show that $\mathcal{A}$ also outputs a

non trivial tuple for a family $(\boldsymbol{R}, R') - \text{uber}^{\mathcal{A}}_{\mathcal{G}}(1^\lambda)$ game. Let's first fix a random $\boldsymbol{\omega} \leftarrow \mathbb{Z}_p^t$ for an integer t. We will call $\boldsymbol{R}_{\boldsymbol{\omega}} \in \mathbb{Z}[X]_p^{2t}$ a vector distribution parametrize with $\boldsymbol{\omega}$ as follow:

$$\boldsymbol{R}_{\boldsymbol{\omega}} = (X, \dots, X^t, Y(X+\omega_1), \dots, Y(X^t+\omega_t))$$

Then it follows that we can use this vector of bivariate polynomials in an $(\boldsymbol{R}, R') - \text{uber}^{\mathcal{A}}_{\mathcal{G}}(1^\lambda)$ game with the polynomial $R'(X,Y) = Y$ which result in a non trival tuple as any monomial of a polynomial $R \in \boldsymbol{R}_{\boldsymbol{\omega}}$ is of degree at least 1 in X. To finish, it is easy to notice that this observation is true for any value taken by ω with overwhelming probability as long as $t \ll p$, i.e. $\Pr\left[\bigcup_{i=1}^t x^i + \omega = 1 | X = x\right] \leqslant \frac{t}{p}$ by the union bound.

According to lemma A.4, assumption 2.1 holds in the Algebraic group model [10].

A.4 Universal zkSNARKs

We recall the definition of (pre-processing) zero-knowledge succinct non-interactive arguments of knowledge (zkSNARK) with specializable universal structured reference string (SRS) [11]. In this work, we use the term zkSNARK to refer to a universal zkSNARK.

Definition A.5. *A SNARK with specializable universal SRS for a universal relation $\mathcal{R}$ is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Derive}, \mathsf{Prove}, \mathsf{Verify})$ satisfying completeness, succinctness, knowledge soundness and composable zero-knowledge and defined as follows:*
$\mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$*: takes security parameter λ and a universal relation $\mathcal{R}$, and outputs a universal structured reference string* srs.
$(\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R}) \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R})$*: a deterministic algorithm that takes a universal SRS* srs *and a relation* $\mathsf{R} \in \mathcal{R}$*, and outputs a specialized SRS consisting of an evaluation key and a verification key.*
$\pi \leftarrow \mathsf{Prove}(\mathsf{ek}_\mathsf{R}, \mathsf{R}, y, w)$*: takes an evaluation key for a relation* R*, a relation* R*, an instance y, and a witness w such that $(y,w) \in \mathsf{R}$, and returns a proof π.*
$b \leftarrow \mathsf{VerProof}(\mathsf{vk}_\mathsf{R}, y, \pi)$*: takes a specialized verification key, an instance y, and a proof π, and accepts $(b=1)$ or rejects $(b=0)$.*

Completeness. *For all $\lambda \in \mathbb{N}$, $\mathsf{R} \in \mathcal{R}$ and $(y,w) \in \mathsf{R}$, if* $\mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$ $(\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R}) \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R})$ *and* $\pi \leftarrow \mathsf{Prove}(\mathsf{ek}_\mathsf{R}, y, \pi)$ *then* $\mathsf{Verify}(\mathsf{vk}_\mathsf{R}, y, \pi) = 1$.
Succinctness. *Verification runs in time* $\mathsf{poly}(\lambda)(\lambda + |y| + \log(|w|))$ *and* $|\pi| = \mathsf{poly}(\lambda)(\lambda + \log(|w|))$.
Knowledge soundness. *Let* $\mathsf{Rg}(1^\lambda)$ *be a relation generator outputting the description of a universal relation that contains $\mathcal{R}$. A SNARK is knowledge sound for* Rg*, if for any prover producing a valid proof π for some (R, y), there is an efficient extractor* Ext *that can extract a valid witness w, i.e. satisfying $(\mathsf{R}, y, w) \in \mathcal{R}$.*
Composable zero-knowledge. *A SNARK satisfies composable zero-knowledge for relation generator* Rg *if there exists a simulator* Sim *that without access to*

the witness can generate a SRS and a proof (using a trapdoor included in the trapdoor) such that no PPT adversary can tell if it is given an honest SRS and honest proofs, or a simulated SRS and simulated proofs. If a SNARK satisfies this property is referred to as zkSNARK.

A.5 KZG polynomial commitment

The Kate-Zaverucha-Goldberg (KZG) polynomial commitment scheme [13] allows committing to a polynomial of a fixed degree and later provide a proof of an evaluation of the polynomial in a given point. Although we do not make explicit use of KZG in this work, nevertheless we include it here as we will use techniques that are similar. The KZG polynomial commitment is defined for a type 3 bilinear group as a tuple of algorithms $\mathsf{KZG} = (\mathsf{KZG.Setup}, \mathsf{KZG.Com}, \mathsf{KZG.Open}, \mathsf{KZG.Verify})$:

- $\mathsf{KZG.Setup}(1^\lambda, d)$: Given a security parameter λ and a maximum degree bound d, for some secret $s \in \mathbb{Z}_p$ and $g_1 \in \mathbb{G}_1$ it outputs a structured reference string as commitment key $\mathsf{ck} = \{g_1^{s^i}\}_{i\in[d]}$ and verification key $\mathsf{vk} = g_2^s$.
- $\mathsf{KZG.Commit}(\mathsf{ck}, p(X))$: given ck and a polynomial $p(X)$ of degree d, it outputs a commitment $C = g^{P(s)}$.
- $\mathsf{KZG.Open}(\mathsf{ck}, \mathsf{cm}_p, z, y; p(X))$: for an evaluation point z and value y, compute the quotient polynomial $q(X) = \frac{p(X)-y}{X-z}$ and output the proof $\pi = g_1^{q(s)}$.
- $\mathsf{KZG.Verify}(\mathsf{vk}, \mathsf{cm}, d, z, y, \pi)$: Given a verification key vk, a commitment cm it outputs 1 if and only if $e(\pi, g_2) = e(\mathsf{cm} g_1^y, g_2^s g_2^{-z})$.

On the extractability of KZG It is possible to make KZG extractable by including in the SRS elements $\{g_2^\alpha, \{g_1^{\alpha s^i}\}_{i\in[d]}\}$ for a secret α known to the verifier. The algorithm Commit then must return the element $C_\alpha := g_1^{\alpha p(s)}$. The verification algorithm must check that $C^\alpha = C_\alpha$ using a pairing operation. Extractability follows by the power knowledge of exponent assumption, see [7].

A.6 Security properties of HSNP

Here we provide the formal definitions of the security properties of a HSNP.

Definition A.6 (Authentication correctness). *For any* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L}, \mathcal{R})$, $\tau \in \mathcal{L}, x \in \mathcal{M}$, *if* $\sigma_\tau \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, x)$ *then* $\mathsf{Verify}(\mathsf{vk}, (\mathsf{R}_{id}, \tau), x, \sigma_\tau) = 1$.

Definition A.7 (Evaluation correctness). *Consider any* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L}, \mathcal{R})$, *any* $\mathsf{R} \in \mathcal{R}$, *and any set of label/message/signature triples* $\{\tau_i, x_i, \sigma_{\tau_i}\}_{i=1}^t$ *satisfying* $\mathsf{Verify}(\mathsf{vk}, (\mathsf{R}_{id}, \tau_i), x_i, \sigma_{\tau_i}) = 1$. *If* $(y, (x_{\tau_i}, \ldots, x_{\tau_i}), w) \in \mathsf{R}$, *and* $\sigma = \mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \sigma_{\tau_1}, \ldots, \sigma_{\tau_t}, w)$ *then* $\mathsf{Verify}(\mathsf{vk}, (\mathsf{R}, \tau_1, \ldots, \tau_t), y, \sigma) = 1$.

Definition A.8 (Succinctness). *Consider any* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L}, \mathcal{R})$, *any* $\tau_i \in L$, $x_{\tau_i} \in \mathcal{M}$ $\forall i \in [t]$, *any relation* $\mathsf{R} \in \mathcal{R}$ *over* $\mathcal{Y} \times \mathcal{M}^t \times \mathcal{W}$ *and any* $w \in \mathcal{W}$. *If for all* $i \in [t]$ $\sigma_{\tau_i} \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau_i, x_{\tau_i})$ *and* $\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \sigma_{\tau_1}, \ldots, \sigma_{\tau_t}, w)$, *then it holds* $|\sigma| \leqslant \mathsf{poly}(\lambda) \cdot \log(t + |w|)$.

Definition A.9 (Amortized efficiency). *There is a pair of algorithms* (VerPrep, EffVer) *such that for any relation* $\mathsf{R} \in \mathcal{R}$ *and any tuple of labels* $\boldsymbol{\tau}$ *we have:*

- *for any* y, σ *such that* $\mathsf{Verify}(\mathsf{vk}, (\mathsf{R}, \boldsymbol{\tau}), y, \sigma) = 1$*, it holds that* $\mathsf{EffVer}(\mathsf{VerPrep}(\mathsf{vk}, \mathsf{R}), \boldsymbol{\tau}, y, \sigma) = 1$*;*
- *given* $\mathsf{vk}_\mathsf{R} \leftarrow \mathsf{VerPrep}(\mathsf{vk}, \mathsf{R})$*, the running time of* $\mathsf{EffVer}(\mathsf{vk}_\mathsf{R}, \boldsymbol{\tau}, y, \sigma) = 1$ *does not depend on* $|\mathsf{R}|$*.*

Definition A.10 (Adaptive security). *This definition can be seen as an adaptation of the standard notion of unforgeability for HSNP. Intuitively, the evaluator should only be able to compute valid signatures for statements* y *for which it received signatures for data items* $x_1, \ldots, x_t$ *and knows a witness* w *satisfying* $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$*. The idea is to have an adversary* $\mathcal{A}$ *which can adaptively query signatures for labelled messages of its choice. Now assume that* $\mathcal{A}$ *outputs a valid signature* σ *for a statement* y *as output of some labelled relation* $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ *for some* $t \geqslant 1$*. Then with overwhelming probability (i)* $\mathcal{A}$ *must have queried signatures for each label* τ_i*; and, denoting these queries* $\{\tau_i, x_i\}_{i \in [t]}$ *(ii) the adversary* $\mathcal{A}$ *must know a witness* w *such that* $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$*. We formalize the knowledge of* w *similarly to the knowledge soundness of SNARKs, namely via an efficient extractor* Ext *which, given* $\mathcal{A}$*'s* s *view, can output such a witness* w*. Precisely, since* $\mathcal{A}$ *is an adversary that interacts with the signing oracle of the HSNP scheme, we also give to* Ext *the transcript (i.e., inputs and outputs) of the signing queries made by A.*

More formally, let Rg *be a relation generator and let* $\mathcal{Z}$ *be an auxiliary input distribution. Let* $Adv^{UF}_{\mathcal{A},\mathsf{Ext}}$ *be the advantage of* $\mathcal{A}$ *on winning the experiment in Fig. 4. An HSNP scheme is adaptively secure for* Rg *and* $\mathcal{Z}$ *if for all PPT adversaries* $\mathcal{A}$*, there is a PPT extractor* Ext *such that*

$$Adv^{UF}_{\mathcal{A},\mathsf{Ext}}(\lambda) = negl(\lambda).$$

We say that an HSNP is adaptively secure if there are benign Rg *and* $\mathcal{Z}$ *for which the HSNP is adaptively secure.*

Definition A.11 (Zero-knowledge). *An HSNP scheme is zero-knowledge if for any large enough* λ*, any label space* $\mathcal{L}$ *and PPT adversary* $\mathcal{A}$ *there is a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that*

$$Adv^{zk}_{\mathcal{A}} := |Pr\left[\mathsf{ZK}^{real}_{\mathcal{A}}(\lambda)\right] - Pr\left[\mathsf{ZK}^{sim}_{\mathcal{A},\mathsf{Sim}}(\lambda)\right]| = negl(\lambda),$$

where $\mathsf{ZK}^{real}_{\mathcal{A}}(\lambda), \mathsf{ZK}^{sim}_{\mathcal{A},\mathsf{Sim}}(\lambda)$ *are the experiments in Fig. 5 and Fig. 6 respectively.*

Key generation The challenger proceeds as follows:
- Initialize an empty list $T = \{\}$.
- $(\mathcal{R}, \mathsf{aux}_{\mathcal{R}}) \leftarrow \mathsf{RG}(1^\lambda)$; $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L}, \mathcal{R})$
- $\mathsf{aux}_{\mathsf{Z}} \leftarrow \mathcal{Z}(\mathcal{R}, \mathsf{aux}_{\mathcal{R}}, \mathsf{crs})$
- $((\mathsf{R}', \boldsymbol{\tau}'), \sigma', x') \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot,\cdot)}(\mathcal{R}, \mathsf{vk}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}})$
- $w \leftarrow \mathsf{Ext}(\mathcal{R}, \mathsf{vk}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}}, T)$

Signing queries $\mathcal{A}$ adaptively submits queries (τ, x), $\tau \in \mathcal{L}$, and $x \in \mathcal{M}$. The challenger proceeds as follows:
- If $\exists \sigma$, $(\tau, x, \sigma) \in T$ (i.e., $\mathcal{A}$ has previously queried (τ, x)), then return σ to $\mathcal{A}$.
- If $(\tau, x', \cdot) \in T$, but $x \neq x'$ (i.e., $\mathcal{A}$ has previously queried (τ, x')), then ignore the query.
- Else compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, x)$, update $T \leftarrow T \cup (\tau, x, \sigma)$ and return σ to $\mathcal{A}$.

Experiment output Return

$$\mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}', \boldsymbol{\tau}'), x', \sigma') \wedge \left((\exists j \in [t] : (\tau'_j, \cdot, \cdot) \notin T) \vee (y', (x_1, \ldots, x_t), w) \notin \mathsf{R}'\right)$$

where $(x_1, \ldots, x_t)$ are such that $\forall j \in [t] : (\tau'_j, x_j, \cdot) \in T$.

Fig. 4. Experiment $\mathsf{UF}_{\mathsf{Rg},\mathcal{Z},\mathcal{A},\mathsf{Ext}}(1^\lambda)$

Key generation:
- Initialize an empty list $T = \{\}$
- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L}, \mathcal{R})$
- Send vk to $\mathcal{A}$

Signing queries: $\mathcal{A}$ adaptively submits $\text{poly}(\lambda)$ many queries (τ, x), $\tau \in \mathcal{L}$, and $x \in \mathcal{M}$. The challenger proceeds as follows:
- If $\exists \sigma$, $(\tau, x, \sigma) \in T$ (i.e., $\mathcal{A}$ has previously queried (τ, x)), then return σ to $\mathcal{A}$.
- If $(\tau, x', \cdot) \in T$, but $x \neq x'$ (i.e., $\mathcal{A}$ has previously queried (τ, x')), then ignore the query.
- Else compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, x)$, update $T \leftarrow T \cup (\tau, x, \sigma)$ and return σ to $\mathcal{A}$.

Choose:
- $\mathcal{A}$ outputs $y \in \mathcal{Y}$, $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ and $w \in \mathcal{W}$
- If $\exists j \in [t]$ such that $(\tau_j, \bullet, \bullet) \notin T^*$ return $\perp$
- For $i \in [t]$ denote by (x_i, σ_i) the pairs such that $(\tau_i, x_i, \sigma_i) \in T^*$
- If $(y, \tau_1, \ldots, \tau_t, w) \notin \mathsf{R}$ return $\perp$

Challenge:
- $\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, (\mathsf{R}, \tau_1, \ldots, \tau_t), y)$
- Send σ to $\mathcal{A}$

Experiment output: $\mathcal{A}$ outputs a bit b

Fig. 5. Experiment $\mathsf{ZK}^{\text{real}}_{\mathcal{A}}(1^\lambda)$

Key generation:
- Initialize an empty list $T = \{\}$
- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sim}_1(1^\lambda, \mathcal{L}, \mathcal{R})$
- Send vk to $\mathcal{A}$

Signing queries: as in Fig. 5

Choose: as in Fig. 5

Challenge:
- $\sigma \leftarrow \mathsf{Sim}_2(\mathsf{sk}, \mathsf{vk}, (\mathsf{R}, \tau_1, \ldots, \tau_t), y)$
- Send σ to $\mathcal{A}$

Experiment output: $\mathcal{A}$ outputs a bit b

Fig. 6. Experiment $\mathsf{ZK}^{\text{sim}}_{\mathcal{A},\mathsf{Sim}}(1^\lambda)$

B A generic HSNP scheme

Our construction is adapted from the one of Fiore and Tucker [9]. Let x_i be data items in a finite field $\mathbb{F}$, and let $N \in \mathbb{N}$ be a positive integer specifying the maximum number of data items to be processed. In this section, we introduce a generic HSNP scheme for the universal relation $\mathcal{R}$, where each relation $\mathsf{R} \subseteq \mathcal{D}_y \times \mathbb{F}^t \times \mathcal{D}_w$ for some positive integer $t \leqslant N$.

Our construction builds upon the following components:

- A Commitment Scheme $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$: This scheme allows committing to vector of size N by using a polynomial basis $\boldsymbol{\chi} = (\chi_i(X_1, \ldots, X_n))_{i \in [t]}$ of total degree less than N with coefficients in $\mathbb{F}$. To be more specific, it can be used to commit to a vector $\mathbf{x} = (x_1, \ldots, x_t) \in \mathbb{F}^t$, the corresponding commited polynomial will be $\tilde{x}(X) = \sum_{i=1}^{t} x_i \chi_i(X_1, \ldots, X_n)$. Com as well allows committing to scalars in $\mathbb{F}$.
- A Universal CP-SNARK $\mathsf{CP}_{\mathcal{R}}$ for the Commitment Com and the universal relation $\mathcal{R}$: Specifically, since Com commits to polynomials, given a statement y and a commitment $\mathsf{cm}_{\mathbf{x}}$, $\mathsf{CP}_{\mathcal{R}}$ proves the existence of witnesses $\tilde{x} \in \mathbb{F}[X_1, \ldots, X_n]$, an opening $o_{\mathbf{x}}$, and $w \in \mathcal{D}_w$, such that:
  $(y, \tilde{x}, w) \in \mathsf{R} \wedge \mathsf{Com.Verify}(\mathsf{ck}, \mathsf{cm}_{\mathbf{x}}, \tilde{x}, o_{\boldsymbol{x}}) = 1$. For our construction, we assume that one can define a decoding function $\mathsf{Dec} : \mathbb{F}[X_1, \ldots, X_n] \to \mathbb{F}^t$ and given $\boldsymbol{\chi}(X_1, \ldots, X_n)$, a list of t n-variate polynomials, they both satisfy the following properties:
  - Dec is $\mathbb{F}$-linear and $\mathsf{Dec}(\chi_i) = \mathbf{e}_i$, where $\mathbf{e}_i$ is the ith vector of the canonical basis of $\mathbb{F}^t$;
  - $\deg(\chi_i(X_1, \ldots, X_n))/|\mathbb{F}|$ is negligible for all $i = 1, \ldots, t$;
  - $\forall \mathbf{r} \in \mathbb{F}^n$ then $\chi_i(\mathbf{r})$ is computable in $O(t)$ $\mathbb{F}$-operations.
- A universal CP-SNARK $\mathcal{CP}_{\text{ev}}$ for the commitment Com and the universal relation $\mathcal{R}_{\text{ev}}$ such that each $\mathsf{R}_{\text{ev}} \in \mathcal{R}_{\text{ev}}$ is parametrized by an integer $t \leqslant N$ and $\mathsf{R}_{\text{ev}} \subset \mathbb{F}^n \times \mathbb{F}[X_1, \ldots, X_n] \times \mathbb{F}$ is such that $(\boldsymbol{r}, \tilde{x}(X_1, \ldots, X_n), z) \in \mathsf{R}_{\text{ev}}$ if and only if $z = \tilde{x}(\boldsymbol{r})$. Given a public statement $\boldsymbol{r} \in \mathbb{F}^n$, and commitments $\mathsf{cm}_{\mathbf{x}}$ and cm_z, CP_{ev} proves the existence of committed values $\tilde{x} \in \mathbb{F}[X_1, \ldots, X_n]$ and $z \in \mathbb{F}$ (and opening values $o_{\mathbf{x}}, o_z$) such that:

  $$z = \tilde{x}(\boldsymbol{r}) \wedge \mathsf{Com.Verify}(\mathsf{ck}, \mathsf{cm}_{\mathbf{x}}, \tilde{x}(X_1, \ldots, X_n), o_{\mathbf{x}}) = 1 \wedge \mathsf{Com.Verify}(\mathsf{ck}, \mathsf{cm}_z, z, o_z) = 1.$$

- A random oracle $RO : \{0,1\}^* \leftarrow \mathbb{F}$ that is used to generate randomness for the proofs.
- An HSNP scheme $\mathsf{ComLHS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Verify})$ for:

  $$\mathsf{R} := \left\{ \mathsf{R}^{(t,\boldsymbol{s})} := \{(\mathsf{cm}, (x_1, \ldots, x_t), o) : \mathsf{Com.Verify}(\mathsf{ck}, \mathsf{cm}, \langle \mathbf{x}, \boldsymbol{s} \rangle, o)\} : t \leqslant N, \boldsymbol{s} \in \mathbb{F}^t \right\}$$

$\underline{\mathsf{KeyGen}(1^\lambda, \mathcal{R})}$**:**

$\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$;
$(\mathsf{vk}_{\text{com-ip}}, \mathsf{sk}_{\text{com-ip}}) \leftarrow \mathsf{ComLHS.KeyGen}(1^\lambda, \mathcal{R}_{\text{com-ip}})$;
$\mathsf{srs}_{\text{ev}} \leftarrow \mathsf{KeyGen}^{\text{ev}}(\mathsf{ck}, \mathcal{R}_{\text{ev}})$;
$\mathsf{srs}_{\mathcal{R}} \leftarrow \mathsf{KeyGen}^{\mathcal{R}}(\mathsf{ck}, \mathcal{R})$;
Return $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs}_{\text{ev}}, \mathsf{srs}_{\mathcal{R}}, \mathsf{vk}_{\text{com-ip}}, RO)$ and $\mathsf{sk} := \mathsf{sk}_{\text{com-ip}}$;

$\underline{\mathsf{Sign}(\mathsf{sk}, \tau, x)}$

$\sigma \leftarrow \mathsf{ComLHS.Sign}(\mathsf{sk}_{\text{com-ip}}, \tau, x)$;
Return $\hat{\sigma} := (x, \sigma)$;

$\underline{\mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \delta, \{\hat{\sigma}_\tau\}_{\tau \in [N]}, w)}$**:**

Parse $(x_i, \sigma_i) = \hat{\sigma}_{\delta+i}$ For all $i \in [t]$; Let $\boldsymbol{x} := (x_1, \ldots, x_t)$;
$(\mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}) \leftarrow \mathsf{Derive}^{\mathcal{R}}(\mathsf{srs}_{\mathcal{R}}, \mathsf{R})$;
$(\mathsf{ek}_{\text{ev}}, \mathsf{vk}_{\text{ev}}) \leftarrow \mathsf{Derive}^{\text{ev}}(\mathsf{srs}_{\text{ev}}, \mathsf{R}_{\text{ev}})$;
$\tilde{x}(X_1, \ldots, X_n) \leftarrow \langle \mathbf{x}, \boldsymbol{\chi}(X_1, \ldots, X_n) \rangle$;
$(\mathsf{cm}_{\mathbf{x}}, o_{\mathbf{x}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \tilde{x}(\mathbf{x}))$;
$\pi_y \leftarrow \mathsf{Prove}^{\mathcal{R}}(\mathsf{ek}_{\mathsf{R}}, \mathsf{R}, y, \mathsf{cm}_{\mathbf{x}}, \tilde{x}, o_{\mathbf{x}}, w)$;
$(\mathsf{cm}_{\text{sum}}, o_{\text{sum}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{1}^\intercal \cdot \mathbf{x})$
$\sigma^{\text{com-ip}}_{\text{sum}} \leftarrow \mathsf{ComLHS.Eval}(\mathsf{vk}_{\text{com-ip}}, \mathsf{R}^{(t,1)}_{\text{com-ip}}, \mathsf{cm}_{\text{sum}}, \{\sigma_i\}_{i \in [t]}, o_{\text{sum}})$;
$\boldsymbol{r} \leftarrow RO(\mathsf{vk}, \mathsf{R}, y, \mathsf{cm}_{\mathbf{x}}, \pi_y, \delta, \mathsf{cm}_{\text{sum}}, \sigma^{\text{com-ip}}_{\text{sum}})$;
$z \leftarrow \langle \mathbf{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle$;
$(\mathsf{cm}_z, o_z) \leftarrow \mathsf{Commit}(\mathsf{ck}, z)$;
$\pi_z \leftarrow \mathsf{Prove}^{\text{ev}}(\mathsf{ek}_{\text{ev}}, r, (\mathsf{cm}_{\mathbf{x}}, c_z), (\tilde{x}, z), (o_x, o_z))$;
$\sigma^{\text{com-ip}} \leftarrow \mathsf{ComLHS.Eval}(\mathsf{vk}_{\text{com-ip}}, \mathsf{R}^{(t, \boldsymbol{B}(r))}_{\text{com-ip}}, \mathsf{cm}_z, \{\sigma_i\}_{i \in [t]}, o_z)$;
Return $\hat{\sigma} := (\mathsf{cm}_{\mathbf{x}}, \pi_y, \mathsf{cm}_{\text{sum}}, \sigma^{\text{com-ip}}_{\text{sum}}, \mathsf{cm}_z, \pi_z, \sigma^{\text{com-ip}})$;

$\underline{\mathsf{VerPrep}(\mathsf{vk}, \mathsf{R})}$**:**

$(\mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}) \leftarrow \mathsf{Derive}^{\mathcal{R}}(\mathsf{srs}_{\mathcal{R}}, \mathsf{R})$;
$(\mathsf{ek}_{\text{ev}}, \mathsf{vk}_{\text{ev}}) \leftarrow \mathsf{Derive}^{\text{ev}}(\mathsf{srs}_{\text{ev}}, \mathsf{R}_{\text{ev}})$;
Return $\mathsf{vk}_{\mathsf{R}} := (\mathsf{ck}, \mathsf{vk}_{\text{ev}}, \mathsf{vk}_{\mathsf{R}}, \mathsf{vk}_{\text{com-ip}}, RO)$;

$\underline{\mathsf{EffVer}(\mathsf{vk}, \delta, y, \hat{\sigma})}$**:**

Parse $(\mathsf{cm}_{\mathbf{x}}, \pi_y, \mathsf{cm}_{\text{sum}}, \sigma^{\text{com-ip}}_{\text{sum}}, \mathsf{cm}_z, \pi_z, \sigma^{\text{com-ip}}) = \hat{\sigma}$;
$\boldsymbol{r} \leftarrow RO(\mathsf{vk}, \mathsf{R}, y, \mathsf{cm}_{\mathbf{x}}, \pi_y, \delta, \mathsf{cm}_{\text{sum}}, \sigma^{com-ip}_{\text{sum}})$;
$\boldsymbol{s} \leftarrow \chi(\boldsymbol{r})$
If $(\mathsf{Verify}^{\mathcal{R}}(\mathsf{vk}_{\mathsf{R}}, y, \mathsf{cm}_{\mathbf{x}}, \pi_y) = 0) \vee (\mathsf{Verify}^{\text{ev}}(\mathsf{vk}_{\text{ev}}, r, (\mathsf{cm}_{\mathbf{x}}, \mathsf{cm}_z), \pi_z) = 0)$:
  Return 0;
If $(\mathsf{ComLHS.Verify}(\mathsf{vk}_{\text{com-ip}}, \mathsf{R}^{(t,1)}_{\text{com-ip}}, \delta, \sigma^{\text{com-ip}}_{\text{sum}}) = 0) \vee (\mathsf{ComLHS.Verify}(\mathsf{vk}_{\text{com-ip}}, \mathsf{R}^{(t,\boldsymbol{s})}_{\text{com-ip}}, \delta, \sigma^{\text{com-ip}}) = 0)$:
  Return 0;
Return 1;

Fig. 7. Our Generic HSNP

C Proof of Theorem 3.1

Let $\mathsf{sk}, \mathsf{vk} \leftarrow \mathsf{Kg}(1^\lambda)$, where

$$\mathsf{sk} = ((a,b,c), \alpha, \mathsf{sk}_\Sigma), \qquad \mathsf{vk} := (\mathsf{ck}, \mathsf{crs}, \Gamma_1, \Gamma_2, B, g_1^c, g_2^c, g_1^\alpha, g_2^\alpha, \mathsf{pk}_\Sigma)$$

with $h \in \mathbb{G}_1$, $\Gamma_1 := h^a$, $\Gamma_2 := g_2^{1/a}$ and $B := g_1^b$.

Perfect authentication correctness For any $\tau \in \mathbb{N}$, and $x \in \mathbb{Z}_p$, let $\sigma_\tau = (x_\tau, \Lambda_\tau, r_\tau, C_\tau^{(1)}, C'_\tau, C_\tau^{(2)}, sig_\tau, \varnothing)$ be the output of $\mathsf{Sign}(\mathsf{sk}, \tau, x)$. By the correctness of the signature Σ, we have $\Sigma.\mathsf{Verify}(\mathsf{sk}_\Sigma, C_\tau^{(2)} \| \tau, sig_\tau) = 1$. And because $\Lambda_\tau = (g_1^{c^\tau + x + br_\tau})^a = (C_\tau^{(1)} g_1^x B^{r_\tau})^a$, thus $e(g_1^x B^{r_\tau} C^{(1)}, g_2) = e((g_1^{x+br_\tau+c^\tau})^a, g_2^{1/a}) = e(\Lambda_\tau, \Gamma_2)$, and $\mathsf{Verify}(\mathsf{vk}, \tau, \mathcal{R}_{\text{com-ip}}^{(t,\mathbf{s})}, \sigma_\tau) = 1$.

Perfect evaluation correctness Consider any $t \in [N]$ and $\mathbf{s} \in \mathbb{Z}_p^t$, and any set of label/message/signature triples $\{\delta+i, x_{\delta+i}, \sigma_{\delta+i}\}_{i=1}^t$, where $\sigma_{\delta+i} = (x_{\delta+i}, \Lambda_{\delta+i}, r_{\delta+i}, C_{\delta+i}^{(1)}, C_{\delta+i}^{(2)}, s_{\delta+i}, \varnothing)$. For these to be valid signatures, it must hold that for $i \in [t]$:

$$e(g_1^{x_{\delta+i}} B^{r_{\delta+i}} C_{\delta+i}^{(1)}, g_2) = e(\Lambda_{\delta+i}, g_2^{1/a}).$$

Let $z = \sum_{i=1}^t x_{\delta+i} s_i$, $r = \sum_{i=1}^t r_{\delta+i} s_i$, and, for any $w \in \mathbb{Z}_p$ let:

$$\sigma := (y, \Lambda, r, C_S, C_{\bar{S}}, C'_{\bar{S}}, C_{\delta+1}^{(2)}, sig_{\delta+1}, \pi_{\mathsf{NIZK}}, \pi_S) \leftarrow \mathsf{Eval}(\mathsf{vk}, \mathcal{R}_{\text{com-ip}}^{t,\mathbf{s}}, y, \delta, \{\sigma_\tau\}_{\tau\in[N]}, w).$$

In particular, $y = g_1^z h^w$, $\Lambda = (h^a)^w \prod_{i=1}^t (\Lambda_{\delta+i})^{s_i}$, and $\pi_{\mathsf{NIZK}} = \mathsf{NIZK}.P(\mathsf{R}_c, \mathsf{crs}, y, (z, w))$. Let $\pi_S = g_1^{q(c)}$ where $q(X) = \frac{S(X)-S(u)}{X-u}$ where $S(X) = \sum_{i=1}^t s_i X^{i-1}$ and let $C_{\delta+1}^{(2)} = g_2^{c^{\delta+1}}, C_{\bar{S}} = \prod_{i=1}^t (g_1^{c^{\delta+i}})^{s_i}, C_S = g_1^{S(c)}$.

This implies that $e(\pi_S, g_2^c \cdot g_2^{-u}) = e(C_S \cdot g_1^{-S(u)}, g_2)$ and that $e(C_S, C_{\delta+1}^{(2)}) = e(C_{\bar{S}}, g_2)$. By correctness of the signature Σ, one has that $\Sigma.\mathsf{Verify}(\mathsf{sk}_\Sigma, C_{\delta+1}^{(2)} \| \delta + 1, s_{\delta+1}) = 1$. By correctness of the proof system, $\mathsf{NIZK}.V(\mathsf{R}_c, \mathsf{crs}, y, \pi_{\mathsf{NIZK}}) = 1$, and

$$\begin{aligned} e\left(yB^r \prod_{i=1}^t C_{\delta+i}^{s_i}, g_2\right) &= e\left(h^w \prod_{i=1}^t \left(g_1^{x_{\delta+i}} B^{r_{\delta+i}} C_{\delta+i}\right)^{s_i}, g_2\right) \\ &= e\left((h^a)^w \prod_{i\in[t]} \Lambda_{\delta+i}^{s_i}, g_2^{1/a}\right) \\ &= e\left(\Lambda, \Gamma_2\right). \end{aligned}$$

It follows that $\mathsf{Verify}(\mathsf{vk}, \mathcal{R}_{\text{com-ip}}^{(t,\mathbf{s})}, \delta, \sigma) = 1$.

Adaptive security The scheme's adaptive security relies on assumption 2.1 and assumption A.1. Let's consider a PPT adversary $\mathcal{A}$ for the adaptive security of the LHS that will make $t = \mathsf{poly}(\lambda)$ signing queries. $\mathcal{A}$ can forge signatures with probability ϵ. We will construct an algorithm $\mathcal{B}$ that will use $\mathcal{A}$ to break assumption 2.1 with probability at most ϵ.

The main idea of the security proof is that any $\mathcal{A}$ breaking our scheme can be reduced to an adversary $\mathcal{B}$ against assumption 2.1. However, for this reduction to go through we have to rule out the occurrence of some bad events, which can in turn be reduced to the knowledge soundness of the NIZK, the use of AGM and ROM, and the power dlog assumption. In what follows we focus on presenting the reduction to Assumption 2.1, and along its presentation we describe how to handle these cases.

Key generation: $\mathcal{B}$ is given $(\boldsymbol{\omega}, g_1, \{g_1^{c^i}\}_{i\in[t]}, \{g_1^{a(c^i+\omega_i)}\}_{i\in[t]}, h, h^a, g_2, \{g_2^{c^i}\}_{i\in[t]}, g_2^{1/a}) \in \mathbb{Z}_p^t \times \mathbb{G}_1^{2t+3} \times \mathbb{G}_2^{t+2}$ with $a, c \leftarrow \mathbb{Z}_p^*$ as input. We will simulate Kg as follow:

- $\alpha, b \leftarrow \mathbb{Z}_p$
- $\Gamma_1 \leftarrow h^a; \Gamma_2 \leftarrow g_2^{1/a}; B \leftarrow g_1^b$
- $\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma \leftarrow \Sigma.\mathsf{KG}(1^\lambda)$
- $(\mathsf{crs}, \xi) \leftarrow \mathsf{Ext}_1(1^\lambda)$
- $\mathsf{ck} = (\mathsf{pp}_\mathcal{G}, h)$

$\mathcal{B}$ will then send the simulated verification key $\mathsf{vk} := (\mathsf{ck}, \mathsf{crs}, \Gamma_1, \Gamma_2, B, g_1^c, g_2^c, g_1^\alpha, g_2^\alpha, \mathsf{pk}_\Sigma)$ to $\mathcal{A}$. Considering that our proof of knowledge scheme NIZK is *perfectly knowledge extractable*, vk follows the same distribution as if it would have been generated by Kg, making it indistinguishable from a real execution from $\mathcal{A}$'s point of view.

Signing queries. To simulate the signing query for a given label/message pair (i, x_i), $\mathcal{B}$ will compute:

- $r \leftarrow (\omega_i - x_i)b^{-1} \bmod p$
- $\Lambda_i \leftarrow g_1^{(c^i+\omega_i)a}$
- $sig_i \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_\Sigma, C_i^{(2)} \| i)$

The simulated signature $\sigma_i = (x_i, \Lambda_i, r, g_1^{c^i}, g_1^{\alpha c^i}, g_2^{c^i}, sig_i, \varnothing)$ is then sent to $\mathcal{A}$. Notice that $\Lambda_i = g_1^{c^i + x_i + br}$ making it indistinguishable for $\mathcal{A}$ from a signature coming from a real execution.

Forgery. After $t = \mathsf{poly}(\lambda)$ queries, $\mathcal{A}$ outputs $(\mathsf{R}, t, \bar{\sigma})$ where R specifies t and $\mathbf{s} \in (\mathbb{Z}_p^*)^t$.

From the double commitments $C_{\bar{S}}, C'_{\bar{S}}$ and C_S, C'_S the extractor is able to extract polynomials $\bar{S}^*(X)$ satisfying $\bar{S}^*(X) = X^\delta S^*(X)$ (checked through the pairing equation). The KZG proof π_S also guarantees that $\bar{S}^*(X) = \bar{S}(X)$ and $S^*(X) = S(X)$ otherwise it would be possible to construct an efficient algorithm that solves the t-Power-DLog assumption A.1. The reduction works as follows. Suppose that we have an algorithm $\mathcal{C}$ that takes as input the string $\{g_1^{c^i}, g_2^{c^i}\}_{i=0}^t$ and returns two polynomials $p_1(X), p_2(X)$ with the property that $p_1(c) = p_2(c)$ but $p_1(X) \neq p_2(X)$. Then an adversary $\mathcal{D}$ trying to break the t-Power-DLog assumption would simply forward the challenge to $\mathcal{C}$, set $p(X) := p_1(X) - p_2(X)$,

where p_1, p_2 are the polynomials output by $\mathcal{D}$, and find c by factoring $p(X)$, which can be done efficiently. Therefore we can assume that $C_{\bar{S}} = g_1^{\bar{S}(c)}$ and $C_S = g_1^{S(c)}$.

As in definition A.10 there are two cases. Case 1) is when $(\bar{y}, \mathsf{R}^{t,\mathbf{s}}, \bar{\sigma}, \mathbb{T})$ is a signature where one of the indexes was not submitted to the Sign algorithm. Case 2) is when one cannot extract $(\bar{z}, \bar{w})$ such that $(\bar{y}, \mathbf{x}, \bar{w}) \in \mathsf{R}^{t,\mathbf{s}}$.

Case 1). Notice that $\pi_{\mathsf{NIZK}} \neq \varnothing$, because otherwise this would correspond to the case $t = 1$, which can be ruled out thanks to the unforgeability of Σ. This guarantees that $\overline{C_\tau^{(2)}}$ must be equal to $g_2^{c^\tau}$ and that Eval must have been queried on label τ. Therefore we can assume $\pi_{\mathsf{NIZK}} \neq \varnothing$. Now we argue that this case can be reduced to the case 2) forgery. Assume that there is an index j such that label τ_j was not signed. We can assume that this is the last label, namely $j = t$.

Assume that $\mathcal{A}_1$ returns a forgery $(\bar{y}, \mathsf{R}^{t,\mathbf{s}}, \bar{\sigma}, \delta+1, \ldots, \delta+t)$ for case 1). The extractor extracts $(\bar{z}, \bar{w})$. It samples a uniformly random $x^*_{\delta+t} \leftarrow \mathbb{F}$. Set $\mathbf{x}^* := (x_1, \ldots, x_{t-1}, x_t^*)$. It then queries the Sign oracle on label $\delta + t$, and gets $\sigma_{\delta+t}$. Then $(\bar{y}, \mathsf{R}^{t,\mathbf{s}^*}, \bar{\sigma}, \delta+1, \ldots, \delta+t)$ is a forgery of type 2), since all labels have been queried and $\bar{z} \neq \langle \mathbf{s}, \mathbf{x}^* \rangle$.

Case 2). Then $\mathcal{B}$ can use Ext to extract $(\bar{z}, \bar{w})$ but for which $\bar{y} \neq g_1^{\langle \mathbf{x}, \mathbf{s} \rangle} h^{\bar{w}}$. Then $\mathcal{B}$ computes an honest signature σ on $\sigma_{\delta+1} \ldots, \sigma_{\delta+t}$ using $\bar{w}$ as witness. Therefore, it can compute:

$$\left.\begin{aligned} e\left(yB^r C_{\bar{S}}, g_2\right) &= e\left(\Lambda, g_2^{1/a}\right) \\ e\left(\bar{y}B^{\bar{r}} C_{\bar{S}}, g_2\right) &= e\left(\bar{\Lambda}, g_2^{1/a}\right) \end{aligned}\right\} \implies \frac{g_1^z h^w B^r C_{\bar{S}}}{g_1^{\bar{z}} h^{\bar{w}} B^{\bar{r}} C_{\bar{S}}} = \left(\frac{\Lambda}{\bar{\Lambda}}\right)^{1/a}$$

Assuming that $\Lambda / \bar{\Lambda} \neq 1_{\mathbb{G}_1}$:

$$\begin{aligned} g_1^{z-\bar{z}} B^{r-\bar{r}} &= \left(\frac{\Lambda}{\bar{\Lambda}}\right)^{1/a} \\ \Leftrightarrow \left(g_1^{z-\bar{z}+b(r-\bar{r})}\right)^a &= \frac{\Lambda}{\bar{\Lambda}} \end{aligned}$$

With respect to the hardness of computing the discrete logarithm in $\mathbb{G}_1$, $z - \bar{z} - b(r - \bar{r}) \neq 0 \mod p$ otherwise it would be easy for $\mathcal{A}$ to compute $b = \frac{\bar{z}-z}{r-\bar{r}} \mod p$. From the above $\mathcal{B}$ can compute:

$$g_1^a = \left(\frac{\Lambda}{\bar{\Lambda}}\right)^{(z-\bar{z}+b(r-\bar{r}))^{-1}}$$

Breaking assumption 2.1.

Zero-knowledge Let $\mathcal{A}$ be a PPT adversary for the zero-knowledge property of the LHS. In the experiment $\mathsf{ZK}_{\mathcal{A}}^{real}$ the challenger runs $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{R}_{\text{com-ip}})$ and gives vk to $\mathcal{A}$. During the signing queries $\mathcal{A}$ adaptively sends queries (τ, x) to the challenger, that honestly computes a signature σ_τ on them and sends σ_τ to $\mathcal{A}$.

In the choose phase, the adversary chooses and outputs a tuple $(y, (\mathsf{R}^{t,\mathbf{s}}_{com-ip}, \delta + 1, \ldots, \delta + t), w)$. If such a tuple causes the challenger to abort, then this happens with the same probability in both the real and simulated case. Thus we assume that for each $i = 1, \ldots, t$ there exists a $\sigma_{\delta+i} = (x_{\delta+i}, \Lambda_{\delta+i}, r_{\delta+i}, C^{(1)}_{\delta+i}, C'_{\delta+i}, C^{(2)}_{\delta+i}, sig_{\delta+i}, \varnothing)$ such that $(\delta + i, x_{\delta+i}, \sigma_{\delta+i}) \in T$ and $(y, (\mathsf{R}^{t,\mathbf{s}}_{com-ip}, \delta + 1, \ldots, \delta + t), w) \in \mathsf{R}^{t,\mathbf{s}}_{\text{com-ip}}$.

We prove the zero-knowledge property by a sequence of games. G_0 is the experiment $\mathsf{ZK}^{\text{real}}_{\mathcal{A}}$ in Fig. 5 and G_2 is the experiment $\mathsf{ZK}^{\text{sim}}_{\mathsf{Sim},\mathcal{A}}$ in Fig. 6. We denote by $\mathsf{E_i}$ the event that the adversary $\mathcal{A}$ outputs 1 in G_i. Thus, $\Pr[\mathsf{E}_0] = \Pr\left[\mathsf{ZK}^{\text{real}}_{\mathcal{A}}\right] = 1$ and $\Pr[\mathsf{E}_2] = \Pr\left[\mathsf{ZK}^{\text{sim}}_{\mathcal{A},\mathsf{Sim}}\right] = 1$. By proving that the chain of games is indistinguishable from $\mathcal{A}$'s point of view we prove that the LHS is zero-knowledge. The changes in going from one game to the next one are highlighted.

From game 0 to game 1. Since the NIZK scheme is ZK, there exists an efficient simulator $\mathsf{NIZK.Sim} = (\mathsf{NIZK.Sim}_1, \mathsf{NIZK.Sim}_2)$. We thus replace in G_0 the set-up and proving algorithm of NIZK with $\mathsf{NIZK.Sim}_1$ and $\mathsf{NIZK.Sim}_2$. The ZK property in particular says that these changes are indistinguishable from $\mathcal{A}$'s point of view. Thus we have

$$|\Pr[\mathsf{E}_0] - \Pr[\mathsf{E}_1]| = \mathsf{negl}(\lambda).$$

Note that in G_1 the proof does not depend on the witness anymore.

From game 1 to game 2 In Game 2 we do not provide the signatures σ_i. To compute the signature σ, Game 2 either uses the public information or it uses the secret key. In particular, it computes the values $\Gamma, C_S, C_{\bar{S}}, C'_{\bar{S}}, sig_{\delta+1}$ using $a, b, c, \alpha, \kappa, \mathsf{sk}_\Sigma$. By construction, these values are identical to those in the previous game G_1. Thus

$$\Pr[E_2] = \Pr[E_1].$$

This concludes the proof as

$$\left|\Pr\left[\mathsf{ZK}^{\text{sim}}_{\mathsf{Sim},\mathcal{A}} = 1\right] - \Pr\left[\mathsf{ZK}^{\text{real}}_{\mathcal{A}} = 1\right]\right| = |\Pr[E_2] - \Pr[E_0]| = |\Pr[E_1] - \Pr[E_0]| = \mathsf{negl}(\lambda).$$

Key generation:

- $\mathsf{pp}_{\mathcal{G}} \leftarrow \mathcal{G}(1^\lambda), h \leftarrow \mathbb{G}_1$
- $\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$
- $\mathsf{ck} \leftarrow (\mathsf{pp}_{\mathcal{G}}, h)$
- $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathsf{ck})$
- $a, b, c \leftarrow \mathbb{Z}_p, \alpha \leftarrow \mathbb{Z}_p$
- $\kappa \leftarrow \mathcal{K}$
- $\Gamma_1 \leftarrow h^a, \Gamma_2 \leftarrow g_2^{1/a}, B \leftarrow g_1^b$
- $\mathsf{sk} \leftarrow ((a, b, c), \alpha, \kappa, \mathsf{sk}_\Sigma)$
- $\mathsf{vk} \leftarrow (\mathsf{ck}, \mathsf{crs}, \Gamma_1, \Gamma_2, B, g_1^c, g_2^c, g_1^\alpha, g_2^\alpha, \mathsf{pk}_\Sigma)$
- Send vk to $\mathcal{A}$.

Challenge:

- $\sigma_{\delta+i} := (x_{\delta+i}, \Lambda_{\delta+i}, r_{\delta+i}, C^{(1)}_{\delta+i}, C'_{\delta+i}, C^{(2)}_{\delta+i}, sig_{\delta+i}, \varnothing) \leftarrow \mathsf{Sign}(\mathsf{sk}, \delta + i, x_{\delta+i})$, $i \in [t]$;
- $\sigma_i := (x_i, \Lambda_i, r_i, C^{(1)}_i, C'_i, C^{(2)}_i, \mathsf{Sig}_i, \varnothing) \leftarrow \mathsf{Sign}(\mathsf{sk}, i, x_i)$, $i \in [t]$;
- $z = \langle \mathbf{x}, \mathbf{s} \rangle, r = \langle \mathbf{r}, \mathbf{s} \rangle$;
- $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK}.P(\mathsf{R_c}, \mathsf{crs}, y, (z, w))$
- $\Lambda \leftarrow \Gamma_1^w \cdot \prod_{i \in [t]} (\Lambda_{\delta+i})^{s_i}$
- Compute $S(X) = \sum_{i=0}^{t-1} s_{i+1} X^i, \bar{S}(X) = \sum_{i=1}^{t} s_i X^{\delta+1}$
- $C_S = g_1^{s_1} \prod_{i=1}^{t} (C^{(1)}_{i-1})^{s_i}, C_{\bar{S}} = \prod_{i=1}^{t} (C^{(1)}_{\delta+i})^{s_i}, C'_{\bar{S}} = \prod_{i=1}^{t} (C'_{\delta+i})^{s_i}$
- $u \leftarrow RO(\mathsf{vk}, \mathcal{R}^{(t,\mathbf{s})}_{\text{com-ip}}, \delta, sig_{\delta+1}, y, \Gamma, r, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, sig_{\delta+1}, \pi_{\mathsf{NIZK}})$
- Compute $q(X) = \frac{S(X) - S(u)}{X - u}$, $\pi_S = g_1^{q(c)}$
- $\sigma \leftarrow (y, r, \pi_{\mathsf{NIZK}}, \Lambda, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, sig_{\delta+1}, \pi_S)$
- Send σ to $\mathcal{A}$
- $\mathcal{A}$ outputs b'

Fig. 8. Game 0

Key generation:

- $\mathsf{pp}_{\mathcal{G}} \leftarrow \mathcal{G}(1^\lambda), h \leftarrow \mathbb{G}_1$
- $\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$
- $\mathsf{ck} \leftarrow (\mathsf{pp}_{\mathcal{G}}, h)$
- $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda, \mathsf{ck})$
- $a, b, c \leftarrow \mathbb{Z}_p, \alpha \leftarrow \mathbb{Z}_p$
- $\kappa \leftarrow \mathcal{K}$
- $\Gamma_1 \leftarrow h^a, \Gamma_2 \leftarrow g_2^{1/a}, B \leftarrow g_1^b$
- $\mathsf{sk} \leftarrow ((a, b, c), \alpha, \kappa, \mathsf{sk}_\Sigma)$
- $\mathsf{vk} \leftarrow (\mathsf{ck}, \mathsf{crs}, \Gamma_1, \Gamma_2, B, g_1^c, g_2^c, g_1^\alpha, g_2^\alpha, \mathsf{pk}_\Sigma)$
- Send vk to $\mathcal{A}$.

Challenge:

- $\sigma_{\delta+i} := (x_{\delta+i}, \Lambda_{\delta+i}, r_{\delta+i}, C^{(1)}_{\delta+i}, C'_{\delta+i}, C^{(2)}_{\delta+i}, \mathsf{Sig}_{\delta+i}, \varnothing) \leftarrow \mathsf{Sign}(\mathsf{sk}, \delta + i, x_{\delta+i})$, $i \in [t]$;
- $\sigma_i := (x_i, \Lambda_i, r_i, C^{(1)}_i, C'_i, C^{(2)}_i, \mathsf{Sig}_i, \varnothing) \leftarrow \mathsf{Sign}(\mathsf{sk}, i, x_i)$, $i \in [t]$;
- $z = \langle \mathbf{x}, \mathbf{s} \rangle, r = \langle \mathbf{r}, \mathbf{s} \rangle$;
- $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Sim}_2(\mathsf{R_c}, \mathsf{crs}, y, \mathsf{td})$
- $\Lambda \leftarrow \Gamma_1^w \cdot \prod_{i \in [t]} (\Lambda_{\delta+i})^{s_i}$
- Compute $S(X) = \sum_{i=0}^{t-1} s_{i+1} X^i, \bar{S}(X) = \sum_{i=1}^{t} s_i X^{\delta+1}$
- $C_S = g_1^{s_1} \prod_{i=1}^{t} (C^{(1)}_{i-1})^{s_i}, C_{\bar{S}} = \prod_{i=1}^{t} (C^{(1)}_{\delta+i})^{s_i}, C'_{\bar{S}} = \prod_{i=1}^{t} (C'_{\delta+i})^{s_i}$
- $u \leftarrow RO(\mathsf{vk}, \mathcal{R}^{(t,\mathbf{s})}_{\text{com-ip}}, \delta, sig_{\delta+1}, y, \Gamma, r, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, sig_{\delta+1}, \pi_{\mathsf{NIZK}})$
- Compute $q(X) = \frac{S(X) - S(u)}{X - u}$, $\pi_S = g_1^{q(c)}$
- $\sigma \leftarrow (y, r, \pi_{\mathsf{NIZK}}, \Lambda, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, sig_{\delta+1}, \pi_S)$
- Send σ to $\mathcal{A}$
- $\mathcal{A}$ outputs b'

Fig. 9. Game 1

Key generation:

- $\mathsf{pp}_{\mathcal{G}} \leftarrow \mathcal{G}(1^\lambda), h \leftarrow \mathbb{G}_1$
- $\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$
- $\mathsf{ck} \leftarrow (\mathsf{pp}_{\mathcal{G}}, h)$
- $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda, \mathsf{ck})$
- $a, b, c \leftarrow \mathbb{Z}_p, \alpha \leftarrow \mathbb{Z}_p$
- $\kappa \leftarrow \mathcal{K}$
- $\Gamma_1 \leftarrow h^a, \Gamma_2 \leftarrow g_2^{1/a}, B \leftarrow g_1^b$
- $\mathsf{sk} \leftarrow ((a, b, c), \alpha, \kappa, \mathsf{sk}_\Sigma)$
- $\mathsf{vk} \leftarrow (\mathsf{ck}, \mathsf{crs}, \Gamma_1, \Gamma_2, B, g_1^c, g_2^c, g_1^\alpha, g_2^\alpha, \mathsf{pk}_\Sigma)$
- Send vk to $\mathcal{A}$.

Challenge:

- $r_{\delta+i} \leftarrow F_\kappa(\delta + i)\ \forall i \in [t]$
- $sig_{\delta+1} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_\Sigma, g_2^{c^{\delta+1}} \| \delta + 1)$
- $C_{\delta+1}^{(2)} = g_2^{c^{\delta+1}}$
- $r = \langle \mathbf{r}, \mathbf{s} \rangle$;
- $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Sim}_2(\mathsf{R_c}, \mathsf{crs}, y, \mathsf{td})$
- $\Lambda = y^a B^{ar} \prod_{i=1}^t (g_1^{c^{\delta+i}})^{as_i}$
- Compute $S(X) = \sum_{i=0}^{t-1} s_{i+1} X^i, \bar{S}(X) = \sum_{i=1}^t s_i X^{\delta+1}$
- $C_S = g_1^{s_1} \prod_{i=1}^t (g_1^{c^{i-1}})^{s_i}, C_{\bar{S}} = \prod_{i=1}^t (g_1^{c^{\delta+i}})^{s_i}, C'_{\bar{S}} = \prod_{i=1}^t (g_1^{\alpha c^{\delta+i}})^{s_i}$
- $u \leftarrow RO(\mathsf{vk}, \mathcal{R}_{\text{com-ip}}^{(t,\mathbf{s})}, \delta, sig_{\delta+1}, y, \Gamma, r, C_S, C_{\bar{S}}, C'_{\bar{S}}, C_{\delta+1}^{(2)}, sig_{\delta+1}, \pi_{\mathsf{NIZK}})$
- Compute $q(X) = \frac{S(X) - S(u)}{X - u}$, $\pi_S = g_1^{q(c)}$
- $\sigma \leftarrow (y, r, \pi_{\mathsf{NIZK}}, \Lambda, C_S, C_{\bar{S}}, C'_{\bar{S}}, C_{\delta+1}^{(2)}, sig_{\delta+1}, \pi_S)$
- Send σ to $\mathcal{A}$
- $\mathcal{A}$ outputs b'

Fig. 10. Game 2

$\mathsf{Sim}_1(1^\lambda, \mathcal{R}_{\mathbf{com\text{-}ip}})$:
- $\mathsf{pp}_\mathcal{G} \leftarrow \mathcal{G}(1^\lambda), h \leftarrow \mathbb{G}_1$
- $\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$
- $\mathsf{ck} \leftarrow (\mathsf{pp}_\mathcal{G}, h)$
- $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda, \mathsf{ck})$
- $a, b, c \leftarrow \mathbb{Z}_p, \alpha \leftarrow \mathbb{Z}_p$
- $\Gamma_1 \leftarrow h^a, \Gamma_2 \leftarrow g_2^{1/a}, B \leftarrow g_1^b$
- $\mathsf{sk} \leftarrow ((a, b, c), \alpha, \kappa, \mathsf{sk}_\Sigma)$
- $\mathsf{vk} \leftarrow (\mathsf{ck}, \mathsf{crs}, \Gamma_1, \Gamma_2, B, g_1^c, g_2^c, g_1^\alpha, g_2^\alpha, \mathsf{pk}_\Sigma)$
- Send vk to $\mathcal{A}$.

$\mathsf{Sim}_2(\mathsf{sk}, \mathsf{vk}, \mathsf{R}^{t,\mathbf{s}}_{\mathbf{com\text{-}ip}}, (\delta+1, \ldots, \delta+t), y)$:
- $r_{\delta+i} \leftarrow F_\kappa(\delta+i)\ \forall i \in [t]$
- $sig_{\delta+1} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_\Sigma, g_2^{c^{\delta+1}} \| \delta+1)$
- $C^{(2)}_{\delta+1} = g_2^{c^{\delta+1}}$
- $r = \langle \mathbf{r}, \mathbf{s} \rangle$;
- $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Sim}_2(\mathsf{R_c}, \mathsf{crs}, y, \mathsf{td})$
- $\Lambda = y^a B^{ar} \prod_{i=1}^t (g_1^{c^{\delta+i}})^{a s_i}$
- Compute $S(X) = \sum_{i=0}^{t-1} s_{i+1} X^i, \bar{S}(X) = \sum_{i=1}^t s_i X^{\delta+1}$
- $C_S = g_1^{s_1} \prod_{i=1}^t (g_1^{c^{i-1}})^{s_i}, C_{\bar{S}} = \prod_{i=1}^t (g_1^{c^{\delta+i}})^{s_i}, C'_{\bar{S}} = \prod_{i=1}^t (g_1^{\alpha c^{\delta+i}})^{s_i}$
- $u \leftarrow RO(\mathsf{vk}, \mathcal{R}^{(t,\mathbf{s})}_{\text{com-ip}}, \delta, sig_{\delta+1}, y, \Gamma, r, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, sig_{\delta+1}, \pi_{\mathsf{NIZK}})$
- Compute $q(X) = \frac{S(X)-S(u)}{X-u}$, $\pi_S = g_1^{q(c)}$
- $\sigma \leftarrow (y, r, \pi_{\mathsf{NIZK}}, \Lambda, C_S, C_{\bar{S}}, C'_{\bar{S}}, C^{(2)}_{\delta+1}, sig_{\delta+1}, \pi_S)$
- Return σ

Fig. 11. The simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ for the ZK of the LHS

D A commit-and-prove version of VOMarlin

We start from the observation that VOMarlin in [17] is a commit-carrying zk-SNARK for the following relation

$$\mathcal{R}_{R1CS} = \left\{ \begin{array}{lll} & \mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{n \times m}, \mathbf{y} \in \mathbb{F}^{\ell}, \mathbf{w} \in \mathbb{F}^{m-\ell-1} & \\ n, m, \left(\mathbf{A}\ \mathbf{B}\ \mathbf{C}\right) & \mathbf{Az} \circ \mathbf{Bz} = \mathbf{Cz} & \\ \mathbf{y}, \mathsf{cm}_{\mathbf{w}} & : \ \mathsf{Com.Verify}(\mathsf{ck}, \mathsf{cm}_{\mathbf{w}}, f_{\mathbf{w}}, \mathsf{o}_{\mathbf{w}}) = 1 & \\ \mathbf{w} & \text{where } \mathbf{z} = (1 \| \mathbf{y} \| \mathbf{w}), f_{\mathbf{w}}(X) = \sum_{i \in [m-\ell-1]} w_i X^{i-1} & \end{array} \right\}.$$

As defined in [5], commit-carrying means that the scheme is like a commit-and-prove except that the commitment to the witness is generated by the prover algorithm and returned as part of the proof, instead of being taken as an input.

Below we describe a method for turning VOMarlin into a (full fledged) commit-and-prove zkSNARK, dubbed $\mathsf{CP}_{\mathcal{R}}$.VOMarlin, for the following relation

$$\mathcal{R}^{CP}_{R1CS} = \left\{ \begin{array}{lll} & \mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{n \times m}, \mathbf{y} \in \mathbb{F}^{\ell}, \mathbf{w} \in \mathbb{F}^{m-\ell-1} & \\ n, m, \left(\mathbf{A}\ \mathbf{B}\ \mathbf{C}\right) & \mathbf{Az} \circ \mathbf{Bz} = \mathbf{Cz} & \\ \mathbf{y}, \mathsf{cm}_{\mathbf{x}} & : \ \mathsf{Com.Verify}(\mathsf{ck}, \mathsf{cm}_{\mathbf{x}}, f_{\mathbf{x}}, \mathsf{o}_{\mathbf{x}}) = 1 & \\ \mathbf{w} = (\mathbf{x}, \mathbf{w}') & \text{where } \mathbf{z} = (1 \| \mathbf{y} \| \mathbf{x} \| \mathbf{w}'), f_{\mathbf{x}}(X) = \sum_{i \in [t-1]} x_i X^{i-1} & \end{array} \right\}.$$

VOMarlin is the result of compiling a HVZK PIOP via the KZG polynomial commitment that encodes vectors using the monomial basis encoding. Specifically, VOMarlin's proof consists, among other things, of a KZG commitment $\mathsf{cm}_{\mathbf{w}} = g_1^{f_{\mathbf{w}}(s)}$ to the polynomial $f_{\mathbf{w}}(X)$ defined in $\mathcal{R}_{R1CS}$, i.e., whose coefficients are the elements of the witness $\mathbf{w}$ of the R1CS relation. Note that this commitment is not hiding, as the underlying PIOP is already HVZK even if $f_{\mathbf{w}(s)}$ is revealed. In order to achieve knowledge soundness, the commitment should be extractable which can be achieved by adding a "double commitment" $g_1^{\alpha f_{\mathbf{w}}(s)}$ to the proof [7], where α is a secret value in the CRS.

The idea is to make VOMarlin a CaP SNARK for subvector relations. Specifically, given the commitment $\mathsf{cm}_{\mathbf{w}}$ to the witness produced by VOMarlin's prover, we additionally prove that $\mathsf{cm}_{\mathbf{w}}$ is a commitment to a vector of the form $\mathbf{w} = (\mathbf{x}, \mathbf{w}')$, given a commitment $\mathsf{cm}_{\mathbf{x}}$ to the polynomial $f_{\mathbf{x}}(X)$ representing the subvector $\mathbf{x}$.

A commit-and-proof SNARK for subvector relations Let $t < n$ and let

$$\mathcal{R}_{\text{subvec}} : \{(\mathbf{w}, \mathbf{v}) \in \mathbb{F}^t \times \mathbb{F}^n \ : \ \mathbf{v} = \mathbf{w} \| \mathbf{z}, \mathbf{z} \in \mathbb{F}^{n-t}\} \tag{2}$$

We present a CaP SNARK for the relation above, when the vectors are committed using KZG, as polynomials with a monomial-base encoding. Namely, we

give a zkSNARK $\mathsf{CP}_{\text{subvec}}$ for the relation

$$\mathcal{R}_{\text{subvec}} = \left\{ \begin{array}{l} \mathsf{cm}_g, \mathsf{cm}_f \\ g(X), f(X), h(X), r_g \end{array} : \begin{array}{l} g(X) \in \mathbb{F}[X]_{t-1}, f(X) \in \mathbb{F}[X]_{n-1}, \\ h(X) \in \mathbb{F}[X]_{n-t-1} \\ f(X) = g(X) + X^t h(X), \\ \mathsf{cm}_g = g_1^{g(s)} h_1^{r_g}, \mathsf{cm}_f = g_1^{f(s)} \end{array} \right\}. \quad (3)$$

Note that, for the sake of our application, we assume cm_g to be a hiding commitment. We present the scheme $\mathsf{CP}_{\text{subvec}}$ in Fig. 12.

$\mathsf{Setup}(\mathsf{ck})$:
- Output $(\{g_1^{s^i}, g_1^{\alpha s^i}, h_1^{s^i}, h_1^{\alpha s^i}\}_{i=0}^{n-1}, g_2^{\alpha})$ as universal SRS.

$\mathsf{Derive}(\mathsf{ck}, t)$:
- Set $\mathsf{vk} := (g_1, h_1, g_2, g_2^{\alpha}, g_2^{\beta}, g_2^{s}, g_2^{s^t})$
- Output $\mathsf{ek} = \mathsf{ck}$ and vk.

$\mathsf{Prove}(\mathsf{ek}, (\mathsf{cm}_g, \mathsf{cm}'_g, \mathsf{cm}_f, \mathsf{cm}'_f), ((g(X), f(X), r_f(X))))$:
- Define $\bar{g}(X) := X^{n-t} g(X)$ and compute $\mathsf{cm}_{\bar{g}} = g_1^{\bar{g}(s)} h_1^{s^{n-t} r_g}$;
- Define $h(X) := \frac{f(X) - g(X)}{X^t}$
- Sample $r_h(X) \leftarrow \mathbb{F}[X]_2$
- Compute $\mathsf{cm}_h := g_1^{h(s)} h_1^{r_h(s)}$ and $\mathsf{cm}'_h := g_1^{\alpha h(s)} h_1^{\alpha r_h(s)}$;
- Set $r_0(X) := -r_f(X) - X^t r_h(X)$;
- Compute $\mathsf{cm}_0 := h_1^{r_0(s)}$ and $\mathsf{cm}'_0 := h_1^{\alpha r_0(s)}$;
- $\rho \leftarrow H(\mathsf{cm}_g, \mathsf{cm}'_g, \mathsf{cm}_f, \mathsf{cm}'_f, \mathsf{cm}_{\bar{g}}, \mathsf{cm}_h, \mathsf{cm}'_h, \mathsf{cm}_0, \mathsf{cm}'_0)$;
- Compute $q(X) := \frac{r_0(X) - r_0(\rho)}{X - \rho}$;
- Compute $\mathsf{cm}_q = h_1^{q(s)}$;
- Return $\pi := (\mathsf{cm}_{\bar{g}}, \mathsf{cm}_h, \mathsf{cm}'_h, \mathsf{cm}_0, \mathsf{cm}'_0, \mathsf{cm}_q, r_0(\rho))$

$\mathsf{Verify}(\mathsf{vk}, (\mathsf{cm}_{\bar{g}}, \mathsf{cm}_g, \mathsf{cm}'_g, \mathsf{cm}_f, \mathsf{cm}'_f), \pi)$:
- Compute $\rho \leftarrow H(\mathsf{cm}_g, \mathsf{cm}'_g, \mathsf{cm}_f, \mathsf{cm}'_f, \mathsf{cm}_{\bar{g}}, \mathsf{cm}_h, \mathsf{cm}'_h, \mathsf{cm}_0, \mathsf{cm}'_0)$;
- Return 1 if and only if all the following equations hold:

$$e(\mathsf{cm}_{\bar{g}}, g_2) = e(\mathsf{cm}_g, g_2^{s^{n-t}})$$

$$e(\mathsf{cm}'_f, g_2) = e(\mathsf{cm}_f, g_2^{\alpha}),\ e(\mathsf{cm}'_g, g_2) = e(\mathsf{cm}_g, g_2^{\alpha})$$

$$e(\mathsf{cm}'_h, g_2) = e(\mathsf{cm}_h, g_2^{\alpha}),\ e(\mathsf{cm}'_0, g_2) = e(\mathsf{cm}_0, g_2^{\alpha})$$

$$e\left(\frac{\mathsf{cm}_f}{\mathsf{cm}_g \mathsf{cm}_0}, g_2\right) = e\left(\mathsf{cm}_h, g_2^{s^t}\right)$$

$$e(\mathsf{cm}_0 h_1^{-r_0(\rho)}, g_2) = e(\mathsf{cm}_q, g_2^{s-\rho})$$

Fig. 12. $\mathsf{CP}_{\text{subvec}}$

The scheme is knowledge sound in the AGM and ROM, under the $(n-1)$-Power-DLog assumption. Here we provide an intuition of the security proof.

Consider an algebraic adversary who produces commitments and a valid proof that passes the verification above. By the algebraic property, the adversary also

returns vectors of coefficients explaining each group element that it produces. From the verification equations in the second and third line, we get polynomials

$$f^*(X), r_f^*(X), g^*(X), r_g^*(X), h^*(X), r_h^*(X), f_0^*(X), r_0^*(X)$$

such that

$$\mathsf{cm}_f = g_1^{f^*(s)} h_1^{r_f^*(s)}, \mathsf{cm}_g = g_1^{g^*(s)} h_1^{r_g^*(s)}, \mathsf{cm}_h = g_1^{h^*(s)} h_1^{r_h^*(s)}, \mathsf{cm}_0 = g_1^{f_0^*(s)} h_1^{r_0^*(s)}$$

By the validity of the last verification equation and the independence of the random point ρ we can conclude that $f^*(X) = 0$. The first equation instead gives us that $g^*(X)$ is of degree $< t$, while the fourth equation instead shows that $f^*(X) - g^*(X) = h^*(X)X^t$, which concludes the proof. Above every implication holds computationally, under the $(n-1)$-Power-DLog assumption.

Putting everything together We give a description of the CP version of VOMarlin in Fig. 13.

$\mathsf{Derive}(\mathsf{ck})$:
- $(\mathsf{ek}', \mathsf{vk}') \leftarrow \mathsf{VOMarlin.Derive}(\mathsf{ck})$;
- $(\mathsf{ek}_{\mathrm{svec}}, \mathsf{vk}_{\mathrm{svec}}) \leftarrow \mathsf{CP}_{\mathrm{svec}}.\mathsf{Derive}(\mathsf{ck})$;
- Return $\mathsf{ek} = (\mathsf{ek}', \mathsf{ek}_{\mathrm{svec}}), \mathsf{vk} = (\mathsf{vk}', \mathsf{vk}_{\mathrm{svec}})$;

$\mathsf{Prove}(\mathsf{ek}, \mathbf{y}, \mathsf{cm}_x, \mathsf{cm}'_x, \mathbf{x}, r_x, \mathbf{w})$:
- $\pi' \leftarrow \mathsf{VOMarlin.Prove}(\mathsf{ek}', \mathsf{R}, \mathbf{y}, (\mathbf{x}, \mathbf{w}))$;
- Parse $\pi' = (\mathsf{cm}_{f_{w'}}, \mathsf{cm}'_{f_{w'}}, \pi'')$, where $\mathsf{cm}_{f_{w'}} = g_1^{f_{w'}(s)}, \mathsf{cm}'_{f_{w'}} = g_1^{\alpha f_{w'}(s)}$;
- $\pi_{\mathrm{svec}} \leftarrow \mathsf{CP}_{\mathrm{svec}}.\mathsf{Prove}(\mathsf{ek}_{\mathrm{svec}}, (\mathsf{cm}_x, \mathsf{cm}'_x, \mathsf{cm}_{f_{w'}}, \mathsf{cm}'_{f_{w'}}), (\mathbf{x}, \mathbf{w}'), r_x)$;
- Return $\pi = (\pi', \pi_{\mathrm{svec}})$;

$\mathsf{Verify}(\mathsf{vk}, \mathbf{y}, \mathsf{cm}_x, \pi)$:
- Parse $\pi = (\pi', \pi_{\mathrm{svec}})$ and $\pi' = (\mathsf{cm}_{f_{w'}}, \pi'')$;
- Output 1 if and only if $\mathsf{VOMarlin.Verify}(\mathsf{vk}', \mathbf{y}, \pi') = 1$ and $\mathsf{CP}_{\mathrm{svec}}.\mathsf{Verify}(\mathsf{vk}_{\mathrm{svec}}, \mathsf{cm}_x, \mathsf{cm}'_x, \mathsf{cm}_{f_{w'}}, \mathsf{cm}'_{f_{w'}}, \pi_{\mathrm{svec}}) = 1$.

Fig. 13. $\mathsf{CP}_{\mathcal{R}}.\mathsf{VOMarlin}$