Efficient Zero-Knowledge Proofs on Signed Data with Applications to Verifiable Computation on Data Streams*

Dario Fiore
IMDEA Software Institute
Madrid, Spain
dario.fiore@imdea.org

Ida Tucker
IMDEA Software Institute
Madrid, Spain
idatucker91@gmail.com

ABSTRACT

We study the problem of privacy-preserving proofs on streamed authenticated data. In this setting, a server receives a continuous stream of data from a trusted data provider, and is requested to prove computations over the data to third parties in a correct and private way. In particular, the third party learns no information on the data beyond the validity of claimed results. A challenging requirement here, is that the third party verifies the validity with respect to the specific data authenticated by the provider, while communicating only with the server. This problem is motivated by various application areas, ranging from stock-market monitoring and prediction services; to the publication of government-ran statistics on large healthcare databases. All of these applications require a reliable and scalable solution, in order to see practical adoption.

In this paper, we identify and formalize a key primitive allowing one to achieve the above: homomorphic signatures which evaluate non-deterministic computations (HSNP). We provide a generic construction for an HSNP evaluating universal relations; instantiate the construction; and implement a library for HSNP. This in turn allows us to build SPHINX: a system for proving arbitrary computations over streamed authenticated data in a privacy-preserving manner. SPHINX improves significantly over alternative solutions for this model. For instance, compared to corresponding solutions based on Marlin (Eurocrypt'20), the proof generation of SPHINX is between 15× and 1 300× faster for various computations used in sliding-window statistics.

1 INTRODUCTION

1.1 Overview

We consider the problem of verifiable computation on delegated data streams, where a data provider $\mathcal{D}$ streams large amounts of data to a third party service provider $\mathcal{S}$. In turn, the latter answers some client $\mathcal{C}$'s queries on behalf of $\mathcal{D}$.

Such a configuration would apply, for example, to a stockbroker providing its clients with a real-time stock market monitoring service. Indeed, besides the raw data, which is generated by a trusted entity and which clients may access directly (real-time information about stock prices, indexes etc), the server may apply learning algorithms to large amounts of accumulated data and provide prediction services (e.g. predict the price in x days time) to paying customers. More generally, the third party server provider could perform analytics and expensive computations on the raw data which could provide more meaningful figures to non-expert clients. Of course such a model inherently raises trust issues, hence the importance of ensuring that server responses are correct.

Other applications of this configuration include a variety of scenarios – such as monitoring health sensors, financial data, network traffic, and smart metering – where clients are interested in sliding-window statistics or machine learning models and predictions on streams of data from sensors or government/healthcare/financial institutions (in Appendix C , we detail some fairly general applications as well as concrete examples).

A little more formally, our goal is to address the problem of verifiable computation on data streams (VCS) for which we target three key properties:

(1) Workflow. The communication between the data provider and the server is unidirectional and the stream is ordered: at each time unit, $\mathcal{D}$ sends a message to $\mathcal{S}$. Also, the client $\mathcal{C}$ does not follow the stream (it is not required to stay online); it is only assumed to know some fixed parameters provided by $\mathcal{D}$ and some metadata of the stream (e.g., indices to define queries).
(2) Security and trust model. The client does not trust the server to return the correct results but trusts the data provider about the data it streams.
(3) Efficiency. The client $\mathcal{C}$ is "efficient", in the sense that its work to verify is less than the cost of running the computation, e.g., it may depend at most logarithmically in the total size of the stream and linearly in the description of the query. Similarly, the communication between $\mathcal{C}$ and $\mathcal{S}$ should be at most logarithmic in the stream size.

Additionally, to increase the range of practical applications, we aim for a solution where the proof is (4) privacy-preserving, i.e., it does not reveal any information on the input beyond the truthfulness of the predicate, and (5) supports non-deterministic computations.

Goals (4) and (5) apply, for example, in situations where a data analyst buys datasets, which are arranged and selected in a creative manner, rendering them ideal for performing analyses, deriving insights and training machine learning algorithms. Such datasets may be protected by e.g. U.S. copyright law or by European Union Database rights, thereby disallowing data analysts from selling on their results to their own customers. With a privacy-preserving VCS – since no information is revealed on the original data, other than the result of the required function – the dataset remains protected, while the data analysts' clients have the guarantee that the information they buy is honestly computed from a reliable source.

Privacy of the stream also makes sense in scenarios analogue to the aforementioned stockbroker application, but where the streamed data are financial transactions happening in a bank or credit institution: while it may be required that the raw data remains secret,

[*] This is the full version of the paper that appears in the proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22) https://doi.org/10.1145/3548606.3560630

one may be able to reveal aggregated statistics. A similar setting could apply to medical data: individual data should be private but aggregated statistics can be revealed.

The problem described above is similar to that of verifiable computation (VC) [32], in which a client delegates a computation on a given input and wishes to verify the result more efficiently than running the computation locally. In spite of the similarities though, verifiable computation falls short of satisfying the workflow of the data streaming scenario, where *the verifier does not know the inputs*. See section 1.3 for a discussion on related work.

1.2 Our results

We propose a new solution to the problem of verifiable computation on data streams that achieves all 5 properties described earlier. Our approach is based on homomorphic signatures (HS) [10]. Using this cryptographic primitive, the data owner $\mathcal{D}$ uses a secret key to generate a signature σ_i on each element x_i of the stream, and sends σ_i to the server $\mathcal{S}$. When a client queries a function f on a subset Q of the stream, $\mathcal{S}$ uses $\mathcal{D}$'s public key and the signed inputs to derive a short signature σ_y certifying that y is indeed $f(\{x_j\}_{j \in Q})$, for inputs legitimately signed by $\mathcal{D}$. Finally $\mathcal{S}$ sends σ_y to the client who can publicly check it *without knowing the inputs*; that is, the client can be convinced that y is the correct result by using (f, Q, y, σ_y). This use of HS for VCS easily satisfies properties (1) and (2). The challenge though is that with existing HS schemes the verification cost is linear in the complexity of the function f (see Section 1.3 for more details). Also, existing HS only work for deterministic polynomial time computations and are concretely expensive.

In this paper, we propose a new HS scheme that overcomes the above shortcomings enabling the server to prove the correctness of non-deterministic computations on signed inputs in a privacy-preserving way. We call such a scheme an HS for NP.

As we discuss in Section 1.3, HS for NP can also be built by combining digital signatures and zkSNARKs: the server proves that $y = f(x_1, \ldots, x_t, w)$ and that there exists a valid signature σ_i for each input x_i. Although zkSNARKs have reached competitive performances today, this approach is very expensive due to the cost of encoding the verification of digital signatures in the zkSNARK constraint system (e.g., a circuit).

Our HS for NP construction instead follows a different approach based on a novel combination of linearly-homomorphic signatures for NP and commit-and-prove SNARKs [14]. We present an efficient instantiation of our construction based on pairings (called SPHinx).

In more detail, **our contributions are** as follows.

- We introduce and formalize the notion of HS for NP (HSNP), which extends HS to support statements of the form $\exists w : y = f(x, w)$ in such a way that the signatures on the results do not reveal any information about (x, w) beyond the output y, and the fact x was signed (Section 3).
- We propose a generic construction of HSNP for any NP computation (Section 4). In contrast to the expensive approach based on combining digital signatures and zkSNARKs, we propose a new methodology to build HSNP through the combination of commit-and-prove SNARKs [14] and a "basic" HSNP which only supports a specific NP computation, that is proving the correct evaluation of commitments to linear functions of signed data, i.e., that $\exists r : \mathsf{c} = \mathsf{Com}(f(\boldsymbol{x}); r)$. We call this primitive 'ComLHS'. The technical novelty of our generic HSNP construction lies in the way we combine ComLHS with the commit-and-prove SNARKs, whose main advantage is to not require any expensive circuit-encoding of cryptographic operations.
- We propose an efficient pairing-based instantiation of our generic HSNP construction, supporting computations expressed as rank-1 constraint systems (R1CS) [33] (Section 6). This scheme, that we call SPHinx,[1] is instantiated from a commit-and-prove variant of Marlin zkSNARK [21] and a ComLHS scheme for Pedersen commitments that we propose (Section 5). The latter scheme allows one, given a linear function f and signatures $\{\sigma_i\}_i$ for inputs $\{x_i\}_i$, to derive a signature σ_{c} for "$\exists r : \mathsf{c} = g^{<\boldsymbol{f},\boldsymbol{x}>} h^r$". Our ComLHS scheme is extremely efficient as the computation of σ_{c} essentially requires one multi-exponentiation of length $|\boldsymbol{x}|$. Compared to existing linearly homomorphic signatures, the novelty of ComLHS is to be the first one to support a non-deterministic statement, that is the computation of the Pedersen commitment.
- We implement SPHinx and evaluate it experimentally, comparing it to the generic solution, based on checking signatures, instantiated with Marlin [21]. We compare the two solutions on four benchmarks: three that model computations typical in data streams such as sliding window statistics, and one for generic circuits of fixed size and varying number of signed inputs.
  In our experiments we find that the high cost of encoding signature verifications in a circuit makes the generic solution rapidly impractical. We were not able to execute it on any benchmark with more than ≈ 2 000 inputs due to excessive memory requirements. In contrast, we ran SPHinx on benchmarks involving up to 1M inputs, and observed only a small cost overhead for proving signatures validity: SPHinx's proving time is only ≈ 10% more expensive than that of Marlin executed on the same computation *without* checking signatures. On the downside, SPHinx has slower verification time than the generic solution, yet it is concretely feasible. Verification is below 100ms for experiments with moderate input sizes where the generic solution could be executed, and below 4s in experiments with ≈ 1M inputs where the generic solution would require 5 billions constraints and be virtually infeasible. We refer to Section 7 for details.

1.3 Related Work

Verifiable Computation. Verifiable computation (VC) [32] allows a client to delegate a computation on a given input and to verify the result more efficiently than running the computation locally. In VC, the verifier must know the inputs of the computation. Hence using VC in our streaming scenario would not satisfy property (1), as the verifying client does not have access to the input, nor can it be sent as part of the proof as it is too large and would contradict property (3). A class of VC schemes, starting with Pantry [13], considers the setting where the client verifies proofs knowing only a digest of the input [14, 24, 25, 46]. The main reason for which these schemes do not apply to VCS is that in the streaming setting the digest should be updated and communicated to the verifier whenever a new element

[1] Sign and Prove through Homomorphic sIgNatures.

Solution	(1) Workflow	Efficiency (ver time)	Efficiency (comm.)	(4) Privacy	(5) NP	Concrete efficiency
VC	✗	✓	✓	✗	✗	●
HS	✓	✗	✓	✓	✗	○
HS + VC	✓	✓	✓	✗	✗	○
HS + zkSNARK	✓	✓	✓	✓	✓	○
Signatures + zkSNARK	✓	✓	✓	✓	✓	◑
ADSNARK	✓	✓	✓(sk) ✗(pk)	✓	✓	●
Ours	✓	✓	✓	✓	✓	●

Table 1: Comparison of candidate solutions for VCS. Here ✓ means that the property is satisfied whereas ✗ means it is not. For concrete efficiency, an empty circle ○ means that the possible instantiations have impractical computation/communication costs while an half-empty circle ◑ means that the instantiation is significantly expensive (see Section 6 for details).

is streamed. A few works [22, 23, 36] consider VC on data streams, but do not satisfy the workflow property (1), as they work in a model where the verifying client must have continuously received and lightly processed the stream, maintaining some short piece of information in local storage. This model would not only require the client to stay online all the time, but would also be unsuitable if one aims for privacy-preserving proofs in which clients should not learn information on the raw stream.

Homomorphic Signatures. This cryptographic primitive [10, 37, 39] allows one to certify the correctness of computations on signed data, and is the closest to solving our problem of verifying computations on data streams (as informally suggested in [17]). While HS can easily satisfy properties (1) and (2) of VCS, existing HS schemes incur three main drawbacks.

First, they fail to satisfy property (3). Any HS incurs a verification cost that is $\text{poly}(\lambda) \cdot |f|$ for every output (as the verifier must read the description of the computation f). The problem is that existing HS work for computations expressed as circuits, hence $|f|$ is as large as f's running time. Some HS constructions [19, 37] can amortize this cost over multiple verifications when the same f is executed on different "datasets". In the streaming setting, this would mean that the stream should be partitioned a priori in different portions $D_1, D_2, \ldots$ and a query could be performed only on one portion D_i. This preprocessing model does not apply to many useful queries for streams such as sliding window queries (e.g., average/variance on the last n elements of the stream).

Second, the current notion of HS only supports deterministic polynomial-time computations.

Third, existing HS for more than linear functions [10, 37] are based on lattices and are concretely expensive. For example, we estimate that instantiating the HS of Gorbunov et al. [37], for a boolean circuit of depth 2 and 128 bits of security, would result in a single signature being of size 17 GBytes.

A large body of works [2–4, 9, 11, 15, 16, 18, 20, 29, 34, 42] presents concretely efficient HS schemes that supports only linear functions. However, linear functions alone would be of little use to express client's queries in VCS. A technical twist of our paper is showing how to take advantage of the efficiency of linearly homomorphic signatures in order to build HS for arbitrary NP computations. As mentioned in the previous section, we expand linearly-HS techniques to build an efficient ComLHS scheme, that is an HSNP that can certify the correct computation of *commitments to linear functions* of signed data.

1.3.1 *Alternative solutions for VCS.*

We discuss a few more solutions for VCS that can be obtained by combining existing results, and for each solution we discuss its shortcomings. To keep the presentation simple, we only show these solutions informally; our main goal here is to expose the challenges of solving this problem and how our solution tackles them.

HS+VC A first idea to solve the verifier's efficiency problem in the HS-based solution is to combine HS with a (publicly verifiable) VC protocol as follows. Assuming that π is verified by an algorithm $\text{VC.Ver}(pk_f, (x_1, \ldots, x_n), y, \pi)$, then one can write this algorithm as a function $g(x_1, \ldots, x_n)$ and the server can use the HS evaluation to certify that $g(x_1, \ldots, x_n) = 1$ holds for the signed x_i. This solution satisfies the first three basic properties of VCS;[2] in particular by the VC efficiency property, g runs faster than f and thus $\text{poly}(\lambda) \cdot |g|$ is asymptotically faster than $|f|$. Besides the fact that it does not satisfy the extra properties (4)–(5), the main downside of this solution is its concrete efficiency. The VC verification algorithm is for sure more complex than a linear function (e.g., it can involve pairings and hash function computations) and thus this solution needs to be instantiated with a sufficiently expressive HS which, as discussed earlier, is prohibitively expensive.

HS+zkSNARK The previous solution can be adapted to work with a zkSNARK instead of a VC scheme, i.e., use the HS to certify that $\text{SNARK.Ver}(pk_f, (x_1, \ldots, x_n), y, \pi) = 1$. Compared to the previous solution, this one satisfies privacy (if the HS is context hiding and the SNARK is zero-knowledge) and supports NP computations since so does the zkSNARK. However, the same issue related to the prohibitive efficiency of its instantiation applies here.

Signatures+zkSNARK This generic construction uses a zkSNARK to prove the existence of messages and signatures $\{x_1, \sigma_1, \ldots, x_n, \sigma_n\}$ such that $y = f(x_1, \ldots, x_n)$ and each σ_i is a valid signature for x_i. This solution achieves all 5 desired properties and is most probably more efficient than that based on combining HS with a zkSNARK. Its bottleneck is the high cost for the prover due to having to prove the validity of the n signatures. In Section 7, we give a detailed

[2] We assume a VC where $|\pi| = polylog(n)$ otherwise it would not be efficient communication-wise since π must be sent to the verifier.

efficiency comparison between our solution and this one, and show that in ours the prover can be up to 1300 times faster.

ADSNARK Backes et al. [5] considered the problem of proving statements on authenticated data, efficiently, and in zero-knowledge. Their scheme, called ADSNARK, is concretely efficient (its proving time is faster than the above generic solution based on zkSNARKs and standard signatures) and satisfies the 5 properties of our VCDS problem. However, *it considers a weaker setting in which the data provider and the verifier share a common secret key* (i. e., they are the same entity). Backes et al. also provide a publicly verifiable variant of ADSNARK, but its proof size is linear in the size of the authenticated inputs, and thus fails to satisfy our third property relative to communication efficiency. Finally, the ADSNARK scheme requires a circuit-specific SRS and setup – ours works with a universal SRS.

2 PRELIMINARIES

2.1 Notations

We denote by $[n]$ the set of integers $\{1, \ldots, n\}$. Vectors are denoted in boldface. Given two vectors $\boldsymbol{a}, \boldsymbol{b}$ their inner product is denoted as $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$. By $\{u_j\}_{j \in [\ell]}$ we denote the tuple $(u_1, \ldots, u_\ell)$.

We use $\lambda \in \mathbb{N}$ to denote the security parameter, and 1^λ its unary representation. We assume all algorithms of a cryptographic scheme take input 1^λ, and thus omit it from the list of inputs. For a distribution D, we denote by $x \leftarrow\!\!\$\, D$ the fact that x is being sampled according to D. An ensemble $\mathcal{X} = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ is a family of probability distributions over a family of domains $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$. Two ensembles $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}' = \{D'_\lambda\}_{\lambda \in \mathbb{N}}$ are statistically indistinguishable (denoted by $\mathcal{D} \approx_s \mathcal{D}'$) if $\frac{1}{2} \sum_x |D_\lambda(x) - D'_\lambda(x)| < \mathsf{negl}(\lambda)$. If $\mathcal{A} = \{\mathcal{A}_\lambda\}$ is a (possibly non-uniform) family of circuits and $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ is an ensemble, then we denote by $\mathcal{A}(\mathcal{D})$ the ensemble of the outputs of $\mathcal{A}_\lambda(x)$ when $x \leftarrow\!\!\$\, D_\lambda$. We say two ensembles $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}' = \{D'_\lambda\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable (denoted by $\mathcal{D} \approx_c \mathcal{D}'$) if for every non-uniform polynomial time distinguisher $\mathcal{A}$ it holds that $\mathcal{A}(\mathcal{D}) \approx_s \mathcal{A}(\mathcal{D}')$.

We use the abbreviation (P)PT to refer to (probabilistic) polynomial time algorithms.

2.2 Bilinear groups

A bilinear group generator $\mathcal{G}$ takes input a security parameter λ, and returns $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathcal{G}(1^\lambda)$ with the following properties:

- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are cyclic groups of prime order q.
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map such that $\forall u \in \mathbb{G}_1$, $\forall v \in GG_2, \forall a, b, \in \mathbb{Z}$, it holds that $e(u^a, v^b) = e(u, v)^{ab}$.
- $g_1, g_2, e(g_1, g_2)$ generate $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ respectively.
- There are efficient algorithms for computing group operations, evaluating the bilinear map, comparing group elements and deciding membership of the groups.

We will work in what Galbraith et al. [31] call type III groups, where there are no efficiently computable isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.3 Universal Relations

A *universal relation* $\mathcal{R}$ is a set of triples (R, y, w), where R is a relation, y is called the *instance* and w the *witness.* We write $(y, w) \in \mathsf{R}$ to denote that R holds on (y, w), otherwise $(y, w) \notin \mathsf{R}$. Given $\mathcal{R}$, the corresponding *universal language* $\mathcal{L}(\mathcal{R})$ is the set $\{(\mathsf{R}, y) : \exists w : (\mathsf{R}, y, w) \in \mathcal{R}\}$.

When discussing schemes that prove statements on committed values, we assume the witness is a pair $(x, w) \in \mathcal{D}_x \times \mathcal{D}_w$. We sometimes use a finer grained specification of $\mathcal{D}_x$, assuming it splits over ℓ domains $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ for some arity ℓ.

2.4 Commitment schemes

We here recall the notion of non-interactive commitment schemes.

Definition 2.1. A (non-interactive) commitment scheme is a tuple of algorithms $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ with the following syntax, and satisfying the notions of correctness, binding and hiding defined below.

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{ck}$ takes the security parameter and outputs a commitment key ck. This key specifies the input space $\mathcal{D}$, commitment space $\mathcal{C}$ and opening space $\mathcal{O}$.

$\mathsf{Commit}(\mathsf{ck}, x) \rightarrow (\mathsf{c}, o)$ takes the commitment key ck, a value $x \in \mathcal{D}$, and outputs a commitment $\mathsf{c} \in \mathcal{C}$ and an opening $o \in \mathcal{O}$.

$\mathsf{VerCom}(\mathsf{ck}, \mathsf{c}, x, o) \rightarrow b$ on input a commitment c, a value x and an opening o, and accepts ($b = 1$) or rejects ($b = 0$).

Correctness. $\forall \lambda \in \mathbb{N}, \forall x \in \mathcal{D}$ it holds that if $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\mathsf{c}, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, x)$, then $\mathsf{VerCom}(\mathsf{ck}, \mathsf{c}, x, o) = 1$.

Computational Binding. For every PT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{c}, x, o, x', o') \leftarrow \mathcal{A}(\mathsf{ck}) \end{array} : \begin{array}{c} x \neq x' \\ \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{c}, x, o) = 1 \\ \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{c}, x', o') = 1 \end{array}\right] = \mathsf{negl}(\lambda).$$

Statistical Hiding. For $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and $\forall x, x' \in \mathcal{D}$, the following distributions are statistically close:

$$\{\mathsf{c} : (\mathsf{c}, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, x)\} \approx_s \{\mathsf{c}' : (\mathsf{c}', o') \leftarrow \mathsf{Commit}(\mathsf{ck}, x')\}.$$

The scheme is *perfectly* hiding if both distributions are identical.

2.5 Universal Zero-knowledge SNARKs

We recall the definition of (pre-processing) zero-knowledge succinct non-interactive arguments of knowledge (zkSNARK) [6, 7] with specializable universal structured reference string (SRS) [38]. In this work, we use the term zkSNARK to refer to a universal zkSNARK.

Definition 2.2. A SNARK with specializable universal SRS for a family of relations $\mathcal{R}$ is a tuple of algorithms $\Pi = (\mathsf{Kg}, \mathsf{Derive}, \mathsf{Prove}, \mathsf{VerProof})$ that work as follows and satisfy the notions of completeness, succinctness and knowledge soundness described below. If Π also satisfies zero-knowledge it is a zkSNARK.

$\mathsf{Kg}(1^\lambda, \mathcal{R}) \rightarrow \mathsf{srs}$ takes security parameter λ and a universal relation $\mathcal{R}$, and outputs a universal structured reference string srs.

$\mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \rightarrow (\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R})$ is 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.

$\mathsf{Prove}(\mathsf{ek}_\mathsf{R}, \mathsf{R}, y, w) \rightarrow \pi$ 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.

$\mathsf{VerProof}(\mathsf{vk}_\mathsf{R}, y, \pi) \to b$ 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 $\mathsf{srs} \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{R})$, $(\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R}) \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R})$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{ek}_\mathsf{R}, \mathsf{R}, y, w)$, then $\mathsf{VerProof}(\mathsf{vk}_\mathsf{R}, y, \pi) = 1$.

Succinctness. VerProof runs in time $\mathsf{poly}(\lambda + |y| + \log |w|)$ and the proof size is $\mathsf{poly}(\lambda + \log |w|)$.

Knowledge soundness. Let $\mathsf{Rg}(1^\lambda)$ be a relation generator that, on input a security parameter, returns the description of a universal relation that contains $\mathcal{R}$. Π is knowledge sound for Rg and auxiliary input distribution $\mathcal{Z}$, denoted $\mathsf{KSND}(\mathsf{Rg}, \mathcal{Z})$ for brevity, if for every (non-uniform) efficient adversary $\mathcal{A}$ there exists a (non-uniform) efficient extractor $\mathcal{E}$ such that

$$\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{Rg},\mathcal{Z},\mathcal{A},\mathcal{E}} = 1] = \mathsf{negl}(\lambda).$$

The scheme Π is knowledge sound if there exist benign Rg and $\mathcal{Z}$ such that Π is $\mathsf{KSND}(\mathsf{Rg}, \mathcal{Z})$.

$$\begin{array}{l}
\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{Rg},\mathcal{Z},\mathcal{A},\mathcal{E}} \to b \\
\hline
(\mathcal{R}, \mathsf{aux}_{\mathcal{R}}) \leftarrow \mathsf{Rg}(1^\lambda); \mathsf{srs} \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{R}) \\
\mathsf{aux}_{\mathcal{Z}} \leftarrow \mathcal{Z}(\mathcal{R}, \mathsf{aux}_{\mathcal{R}}, \mathsf{srs}) \\
(\mathsf{R}, y, \pi) \leftarrow \mathcal{A}(\mathcal{R}, \mathsf{srs}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}}) \\
w \leftarrow \mathcal{E}(\mathcal{R}, \mathsf{srs}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}}) \\
(\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R}) \leftarrow \mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \\
b \leftarrow \big(\mathsf{VerProof}(\mathsf{vk}_\mathsf{R}, y, \pi) \land (y, w) \notin \mathsf{R}\big)
\end{array}$$

Composable zero-knowledge. A SNARK satisfies composable zero-knowledge for relation generator Rg if there exists a simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{prv}})$ s.t. for all $\mathcal{A}$ the following hold:

Key indistinguishability

$$\Pr[(\mathcal{R}, \mathsf{aux}) \leftarrow \mathsf{Rg}(1^\lambda), \mathsf{srs} \leftarrow \mathsf{Kg}(\mathcal{R}) : \mathcal{A}(\mathsf{srs}, \mathsf{aux}) = 1] \approx$$
$$\Pr[(\mathcal{R}, \mathsf{aux}) \leftarrow \mathsf{Rg}(1^\lambda), (\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{S}_{\mathsf{kg}}(\mathcal{R}) : \mathcal{A}(\mathsf{srs}, \mathsf{aux}) = 1].$$

Proof indistinguishability $\forall (\mathsf{R}, y, w) \in \mathcal{R}$,

$$\begin{aligned}
&\Pr[(\mathcal{R}, \mathsf{aux}) \leftarrow \mathsf{Rg}(1^\lambda), (\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{S}_{\mathsf{kg}}(\mathcal{R}) : \\
&\qquad \pi \leftarrow \mathsf{Prove}(\mathsf{ek}_\mathsf{R}, \mathsf{R}, y, w), \mathcal{A}(\mathsf{srs}, \mathsf{aux}, \pi) = 1] \\
&\approx \Pr[(\mathcal{R}, \mathsf{aux}) \leftarrow \mathsf{Rg}(1^\lambda), (\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{S}_{\mathsf{kg}}(\mathcal{R}) : \\
&\qquad \pi \leftarrow \mathcal{S}_{\mathsf{prv}}(\mathsf{srs}, \mathsf{td}, \mathsf{R}, y), \mathcal{A}(\mathsf{srs}, \mathsf{aux}, \pi) = 1].
\end{aligned}$$

O-SNARKs. In our work we use the notion of O-SNARK introduced in [26], which assumes knowledge extraction in the presence of oracles. We refer to [26] for the formal definition. In brief, given an oracle $\mathcal{O}$, Π is an O-SNARK for $\mathcal{O}$ if the knowledge soundness definition holds for an experiment where $\mathcal{A}$ can make queries to $\mathcal{O}$ and the extractor $\mathcal{E}$ additionally takes as input the transcript (i.e., inputs and outputs) of $\mathcal{A}$'s queries.

2.6 Commit-and-prove SNARKs

A commit-and-prove SNARK (CP-SNARK) is a SNARK that, for a given commitment c_x, can prove knowledge of (y, w') such that $(y, w') \in \mathsf{R}$ for a witness $w' = (x, w)$, and x opens c_x. In our work we use the framework of definitions from [14] which allow explicitly handling relations where the input domain $\mathcal{D}_x$ splits over ℓ sub-domains, called commitment slots. We assume the description of the splitting is part of R's description.

Definition 2.3. Let $\mathcal{R}$ be a universal relation where each $\mathsf{R} \in \mathcal{R}$ is over $\mathcal{D}_y \times \mathcal{D}_x \times \mathcal{D}_w$, and $\mathcal{D}_x$ splits over ℓ domains $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ for some arity parameter $\ell \geqslant 1$. Let $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ be a commitment scheme with input space $\mathcal{D}$, where $\mathcal{D}_i \subset \mathcal{D}$ for $i \in [\ell]$; commitment space $\mathcal{C}$; and opening space $\mathcal{O}$. A universal commit and prove zkSNARK for Com and $\mathcal{R}$ is a universal zkSNARK for a universal relation $\mathcal{R}^{\mathsf{Com}}$ such that:

- every $\mathbf{R} \in \mathcal{R}^{\mathsf{Com}}$ is represented by a pair $(\mathsf{ck}, \mathsf{R})$ where $\mathsf{ck} \in \mathsf{Setup}(1^\lambda)$ and $\mathsf{R} \in \mathcal{R}$;
- relation $\mathbf{R}$ is over pairs $(\mathbf{y}, \mathbf{w})$ where the statement is $\mathbf{y} := (y, \{\mathsf{c}_j\}_{j \in [\ell]}) \in \mathcal{D}_y \times \mathcal{C}^\ell$, the witness is $\mathbf{w} := (\{x_j\}_{j \in [\ell]}, \{o_j\}_{j \in [\ell]}, w) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell \times \mathcal{O}^\ell \times \mathcal{D}_w$, and $(\mathbf{y}, \mathbf{w}) \in \mathbf{R}$ if and only if:

$$\bigwedge_{j \in [\ell]} \mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_j, x_j, o_j) = 1 \land \mathsf{R}(y, \{x_j\}_{j \in [\ell]}, w) = 1.$$

A universal CP-SNARK is a tuple of algorithms CP = (Kg, Derive, Prove, VerProof) with the following syntax:

$\mathsf{Kg}(\mathsf{ck}, \mathcal{R}) \to \mathsf{srs}$ outputs the universal SRS.
$\mathsf{Derive}(\mathsf{srs}, \mathsf{R}) \to (\mathsf{ek}_\mathsf{R}, \mathsf{vk}_\mathsf{R})$ outputs the specialized SRS.
$\mathsf{Prove}(\mathsf{ek}_\mathsf{R}, \mathsf{R}, y, \{\mathsf{c}_j\}_{j \in [\ell]}, \{x_j\}_{j \in [\ell]}, \{o_j\}_{j \in [\ell]}, w) \to \pi$.
$\mathsf{VerProof}(\mathsf{vk}_\mathsf{R}, y, \{\mathsf{c}_j\}_{j \in [\ell]}, \pi) \to b \in \{0, 1\}$.

Furthermore, CP is knowledge-sound for a relation generator Rg and auxiliary input generator $\mathcal{Z}$ (denoted $\mathsf{KSND}(\mathsf{Rg}, \mathcal{Z})$, for short) if it is a knowledge-sound SNARK for relation generator $\mathsf{Rg}_{\mathsf{Com}}(1^\lambda)$ that runs $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\mathcal{R}, \mathsf{aux}_{\mathcal{R}}) \leftarrow \mathsf{Rg}(1^\lambda)$, and returns $((\mathsf{ck}, \mathcal{R}), \mathsf{aux}_{\mathcal{R}})$.

3 HOMOMORPHIC SIGNATURES FOR NP

We introduce the notion of homomorphic signatures for NP relations (HSNP), which extends classical homomorphic signatures to support the evaluation of non-deterministic computations on signed data. At a very high level, this means that anyone who knows a value x and a signature σ_x, can compute $y = f(x, w)$ where w is a non-deterministic input, and then derive a signature σ_y which vouches for the fact that there exists a w and validly authenticated x such that $y = f(x, w)$. For an HSNP we consider a privacy notion which guarantees that σ_y leaks no information about (x, w) beyond what can be inferred from y.

An interesting application of HSNP is to authenticate commitments while ensuring data privacy (i.e. commitments that are cryptographically hiding). Consider the case of a party who generates a commitment $(\mathsf{c}_{\boldsymbol{x}}, o_{\boldsymbol{x}}) = \mathsf{Commit}(\mathsf{ck}, \boldsymbol{x})$ on a signed $\boldsymbol{x}$ and who wishes to prove that $\mathsf{c}_{\boldsymbol{x}}$ commits to an $\boldsymbol{x}$ for which there is a valid signature. An HSNP allows one to do so by generating a homomorphic signature on the bit 1 as output of $\mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_{\boldsymbol{x}}, \boldsymbol{x}, o_{\boldsymbol{x}})$, in which $o_{\boldsymbol{x}}$ is the witness. Taking this idea further, one can authenticate commitments to outputs of functions on signed data, i.e., authenticate c_y which commits to $y = f(\boldsymbol{x})$ (see Section 5).

Labelled relations A technicality in defining HS is the need of labelling signed inputs (see, e.g., [17, 35] for details on this problem). Briefly, this means that an input x_i is signed with respect to a public label τ_i (e.g., its position i in the stream), and the prover's goal is to convince a verifier that $y = f(x_1, \ldots, x_n, w)$ for inputs that have been correctly authenticated *with respect to labels* $(\tau_1, \ldots, \tau_n)$ known to the verifier. Labelling is a way to avoid ambiguity for proofs about inputs that are unknown to the verifier. As a simple example, assume one authenticates two measurements x_1 and x_2 at time instants 1 and 2 respectively, and consider a prover who wishes to certify a weighted sum $y = f_1 x_1 + f_2 x_2$. Clearly, the order of the measurements makes a difference. Also, labelling provides a mechanism to express queries that first filter a subset of the authenticated data and then execute a computation on the data after filtering. This property is useful to make the prover's complexity nearly independent on the total size of the stream (see Section 7). Below we give a formal definition of labelled relations.

Let $\mathcal{L}$ be a set of labels (e.g., integers in $[N]$ or strings in $\{0,1\}^*$) and let $\mathcal{R}$ be a universal relation such that each $\mathsf{R} \in \mathcal{R}$ is over $\mathcal{D}_y \times \mathcal{D}_x \times \mathcal{D}_w$, where $\mathcal{D}_x = \mathcal{M}^t$ for an arity parameter $t \geqslant 1$, and $\mathcal{M}$ is the signature scheme's message space. A labelled relation is a tuple $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ where $\mathsf{R} \in \mathcal{R}$, and $\tau_i \in \mathcal{L}$ is a label for the i-th slot of $\mathcal{D}_x$. We refer to $\mathsf{R}_{id} := \{(x, x, \emptyset)\}$ as the identity relation.

Homomorphic Signatures for NP relations From signatures for labelled data $\{(\tau_i, x_i)\}_{i \in [t]}$, and from any $y \in \mathcal{D}_y, w \in \mathcal{D}_w$ satisfying $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$, an HSNP scheme allows publicly computing a signature for y.

Definition 3.1 (HSNP). A homomorphic signature scheme for NP consists of the following PPT algorithms:

$\mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R})$ on input $\lambda \in \mathbb{N}$, a set of labels $\mathcal{L}$ (which fixes the maximum number N of inputs to be signed), and a universal relation $\mathcal{R}$, the key generation algorithm outputs a secret signing key sk, and a public key vk which contains a description of the message space $\mathcal{M}$.

$\mathsf{Sign}(\mathsf{sk}, \tau, x)$ on input signing key sk, a label $\tau \in \mathcal{L}$, and a message $x \in \mathcal{M}$, the signing algorithm outputs a signature σ.

$\mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \sigma_1, \ldots, \sigma_t, w)$ on input a verification key vk, a relation $\mathsf{R} \in \mathcal{R}$ over $\mathcal{D}_y \times \mathcal{M}^t \times \mathcal{D}_w$ for some $t \leqslant N$, a statement $y \in \mathcal{D}_y$, signatures $\{\sigma_1, \ldots, \sigma_t\}$, and a witness $w \in \mathcal{D}_w$, the evaluation algorithm outputs a new signature σ.

$\mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}, \boldsymbol{\tau}), y, \sigma)$ on input a verification key vk, a labelled relation $(\mathsf{R}, \tau_1, \ldots \tau_t)$ where $\mathsf{R} \in \mathcal{R}$ is over $\mathcal{D}_y \times \mathcal{M}^t \times \mathcal{D}_w$, a statement $y \in \mathcal{D}_y$, and a signature σ, the verification algorithm outputs 0 (reject) or 1 (accept).

An HSNP scheme is required to satisfy *authentication correctness*, *evaluation correctness* and *succinctness* described below. These are similar to those required to standard homomorphic signatures.

Authentication correctness For any $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R})$, any $\tau \in \mathcal{L}$, and $x \in \mathcal{M}$, if $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, x)$ then with overwhelming probability $\mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}_{id}, \tau), x, \sigma) = 1$.

Evaluation correctness Consider any $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R})$, any $\mathsf{R} \in \mathcal{R}$, and any set of label/message/signature triples $\{\tau_i, x_i, \sigma_i\}_{i=1}^t$ satisfying $\mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}_{id}, \tau_i), x_i, \sigma_i) = 1$. If $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$, and $\sigma = \mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \sigma_1, \ldots, \sigma_t, w)$ then with overwhelming probability $\mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}, \tau_1, \ldots \tau_t), y, \sigma) = 1$.

Succinctness For a fixed $\lambda \in \mathbb{N}$, the size of signatures depends at most logarithmically on the size of the signed and non-deterministic inputs. Formally, consider any $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R})$, any $\tau_i \in \mathcal{L}$, $x_i \in \mathcal{M}$ $\forall i \in [t]$, any relation $\mathsf{R} \in \mathcal{R}$ over $\mathcal{D}_y \times \mathcal{M}^t \times \mathcal{D}_w$ and any $w \in \mathcal{D}_w$. If for all $i \in [t]$ $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau_i, x_i)$ and $\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \sigma_1, \ldots, \sigma_t, w)$, then it holds $|\sigma| \leqslant \mathsf{poly}(\lambda) \cdot \log(t + |w|)$.

Key generation The challenger proceeds as follows:

- Initialise an empty list $T = \{\}$.
- $(\mathcal{R}, \mathsf{aux}_{\mathcal{R}}) \leftarrow \mathsf{Rg}(1^\lambda)$; $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R})$
- $\mathsf{aux}_{\mathcal{Z}} \leftarrow \mathcal{Z}(\mathcal{R}, \mathsf{aux}_{\mathcal{R}}, \mathsf{srs})$
- $((\mathsf{R}', \boldsymbol{\tau}'), \sigma', y') \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot, \cdot)}(\mathcal{R}, \mathsf{vk}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}})$
- $w \leftarrow \mathcal{E}(\mathcal{R}, \mathsf{vk}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}}, T)$

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 $\mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}', \boldsymbol{\tau}'), y', \sigma') \wedge \Big((\exists j \in [t] : (\tau'_j, \cdot, \cdot) \notin T) \vee (y', (x_1, \ldots, x_t), w) \notin \mathsf{R}' \Big)$ where $(x_1, \ldots, x_t)$ are such that $\forall j \in [t] : (\tau'_j, x_j, \cdot) \in T$.

Figure 1: Experiment $\mathsf{UF}_{\mathsf{Rg}, \mathcal{Z}, \mathcal{A}, \mathcal{E}}(\lambda)$

Amortized efficiency An HSNP scheme satisfies *amortized efficiency* if there is a pair of algorithms $(\mathsf{VerPrep}, \mathsf{EffVer})$ such that for any relation $\mathsf{R} \in \mathcal{R}$ and any tuple of labels $\boldsymbol{\tau} \in \mathcal{L}^t$ we have: (i) for any y, σ such that $\mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}, \boldsymbol{\tau}), y, \sigma) = 1$, it holds that $\mathsf{EffVer}(\mathsf{VerPrep}(\mathsf{vk}, \mathsf{R}), \boldsymbol{\tau}, y, \sigma) = 1$; (ii) 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 $|\mathsf{R}|$.

3.1 Security definitions

We here define meaningful security properties for HSNP. The first of which ensures unforgeability: an 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 value $w \in \mathcal{D}_w$ satisfying $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$. To formalise this, we consider 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 $t \in \mathbb{N}$. 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 $\mathcal{E}$ which, given $\mathcal{A}$'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, taking inspiration from [26], we also give to $\mathcal{E}$ the transcript (i.e., inputs and outputs) of the signing queries made by $\mathcal{A}$.

Definition 3.2 (Adaptive security). Let $\mathsf{Rg}(1^\lambda)$ be a relation generator and $\mathcal{Z}$ be an auxiliary input distribution. Consider the security experiment $\mathsf{UF}_{\mathsf{Rg}, \mathcal{Z}, \mathcal{A}, \mathcal{E}}(\lambda)$ depicted in Figure 1. An HSNP scheme is adaptively secure for Rg and $\mathcal{Z}$, denoted $\mathsf{UF}(\mathsf{Rg}, \mathcal{Z})$, if for every non-uniform efficient adversary $\mathcal{A}$ there exists a non-uniform efficient extractor $\mathcal{E}$ such that

$$\mathsf{Adv}^{\mathsf{uf}}_{\mathcal{A},\mathcal{E}}(\lambda) := \Pr[\mathsf{UF}_{\mathsf{Rg},\mathcal{Z},\mathcal{A},\mathcal{E}}(\lambda) = 1] = \mathsf{negl}(\lambda)$$

We say that an HSNP scheme is adaptively secure if there exist benign Rg and $\mathcal{Z}$ such that the scheme is $\mathsf{UF}(\mathsf{Rg}, \mathcal{Z})$.

Similarly to the O-SNARK notion [26] mentioned in Section 2.5, we say that Σ is adaptively secure with respect to an oracle $\mathcal{O}$ if adaptive security holds for a game where $\mathcal{A}$ has access to $\mathcal{O}$, and $\mathcal{E}$ additionally receives the transcript of $\mathcal{A}$'s queries to $\mathcal{O}$.

The second property we define is *zero-knowledge*, ensuring that evaluated signatures reveal nothing on the signed and non-deterministic inputs beyond the fact that the signed statement satisfies the relation. To formalise this, the adversary $\mathcal{A}$ first adaptively queries signatures for labelled messages of its choice. Then $\mathcal{A}$ chooses a challenge statement y, a labelled relation $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ (where each τ_i must have been queried with some message x_i, resulting in signature σ_i) and a witness w, satisfying $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$. Finally, $\mathcal{A}$ is given a signature for y, which is either honestly evaluated, or computed by a simulator, which has access to y, $(\mathsf{R}, \boldsymbol{\tau})$, and the secret signing key sk, but not to the signed and non-deterministic inputs. The scheme satisfies zero-knowledge if $\mathcal{A}$ cannot distinguish how σ was computed.

Definition 3.3 (Zero-knowledge). Consider the real and simulated experiments, $\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}}$ and $\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}}$ of Figures 2a and 2b. An HSNP scheme satisfies computational zero-knowledge if for any large enough $\lambda \in \mathbb{N}$, any label space $\mathcal{L}$, and any PPT adversary $\mathcal{A}$, there exists a PT simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{eval}})$, such that the following probability is negligible:

$$\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathcal{S}}(\lambda) := \left|\Pr[\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]\right|.$$

If $\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathcal{S}}(\lambda) = 0$ the scheme is perfect zero-knowledge.

3.2 HSNP for VCS

We briefly discuss how an HSNP yields a solution for verifiable computation on data streams, which achieves all five properties advocated earlier in the paper. In fact, our HSNP's properties closely match those of the VCS problem; hence we do not explicitly formalize VCS as a cryptographic primitive.

The use of HSNP for VCS is as follows. First, the data provider $\mathcal{D}$ generates a keypair $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R})$ in which $\mathcal{L} = \{1, 2, \ldots, N\}$ and N is an upper bound on the size of the stream, then $\mathcal{D}$ gives vk to both the server $\mathcal{S}$ and all the clients.

To stream a value x_i at time i, $\mathcal{D}$ computes $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, i, x_i)$ and gives (x_i, σ_i) to $\mathcal{S}$.

Upon receiving a query (f, Q), where f is the computation and $Q = (j_1, \ldots, j_n)$ a subset of indices, $\mathcal{S}$ computes $y \leftarrow f(x_{j_1}, \ldots, x_{j_n}, w)$ and $\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, \mathsf{R}_f, y, \sigma_{j_1}, \ldots, \sigma_{j_n}, w)$, and gives (y, σ) to the relevant client. Here R_f is the NP relation containing tuples $(y, x_{j_1}, \ldots, x_{j_n}, w)$ such that $y = f(x_{j_1}, \ldots, x_{j_n}, w)$, and w is a non-deterministic input provided by $\mathcal{S}$. For example, w can be a secret parameter known only by $\mathcal{S}$, or it can include values that are computed by $\mathcal{S}$ and that simplify the verification of some steps of f (e.g., to express the binary decomposition of an integer or to turn divisions into multiplications). The client can in turn execute $\mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}_f, Q), y, \sigma)$ to verify the correctness of y.

Key generation
- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R})$; send vk to $\mathcal{A}$

Signing queries as in Figure 1, repeated $\mathsf{poly}(\lambda)$ times

Choose
- $\mathcal{A}$ outputs $y \in \mathcal{D}_y$, $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ and $w \in \mathcal{D}_w$
- if $\exists j \in [t]$ s.t. $(\tau_j, \cdot, \cdot) \notin T^*$ return $\bot$
- for $i \in [t]$, denote (x_i, σ_i) the pairs s.t. $(\tau_i, x_i, \sigma_i) \in T^*$
- if $(y, (x_1, \ldots, x_t), w) \notin \mathsf{R}$ return $\bot$

Challenge
- $\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \sigma_1, \ldots, \sigma_t, w)$
- send σ to $\mathcal{A}$

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

(a) Experiment $\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}}(\lambda)$

Key generation
- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}_{\mathsf{kg}}(1^\lambda, \mathcal{L}, \mathcal{R})$; send vk to $\mathcal{A}$

Signing queries as in Figure 2a.

Choose as in Figure 2a.

Challenge
- $\sigma \leftarrow \mathcal{S}_{\mathsf{eval}}(\mathsf{vk}, \mathsf{sk}, (\mathsf{R}, \tau_1, \ldots, \tau_t), y)$.
- send σ to $\mathcal{A}$.

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

(b) Experiment $\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}}(\lambda)$

Figure 2: Real/simulated experiments for zero-knowledge.

By the above description, one can immediately see that this solution satisfies property (1) as the communication is unidirectional, and the client only needs to know $\mathcal{D}$'s public vk. The security property (2) follows from the unforgeability of the HSNP scheme. Regarding efficiency (3), the communication from $\mathcal{S}$ to $\mathcal{C}$ is short because of the HSNP succinctness, and so is $\mathcal{C}$'s computational cost to verify each result. Finally, the support for non-deterministic computations (4) is immediate by HSNP correctness, and privacy (5) is guaranteed thanks to the zero-knowledge of HSNP.

4 A GENERIC HSNP SCHEME

Consider data items x_i in some finite field $\mathbb{F}$ and a positive integer $N \in \mathbb{N}$ specifying the maximum number of data items computed upon. In this section, we provide a generic HSNP scheme for the universal relation $\mathcal{R}$ where each $\mathsf{R} \subseteq \mathcal{D}_y \times \mathbb{F}^t \times \mathcal{D}_w$ for some positive integer $t \leqslant N$.

Our construction builds upon:

- A commitment scheme Com = (Setup, Commit, VerCom) for committing to n-variate polynomials of total degree $< N$ with coefficients in $\mathbb{F}$, as well to scalars in $\mathbb{F}$.
- A universal CP-SNARK $\mathsf{CP}_{\mathcal{R}}$ for the commitment Com and the universal relation $\mathcal{R}$ to be supported by our HSNP scheme. Specifically, considering that Com commits to polynomials, given a statement y and a commitment $\mathsf{c}_{\boldsymbol{x}}$, $\mathsf{CP}_{\mathcal{R}}$ proves the existence of witnesses $\tilde{x} \in \mathbb{F}[X_1, \ldots, X_n]$, $o_{\boldsymbol{x}}$, and $w \in \mathcal{D}_w$, such that:

$$(y, (\tilde{x}(h_1), \ldots, \tilde{x}(h_t)), w) \in \mathsf{R} \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_{\boldsymbol{x}}, \tilde{x}, o_{\boldsymbol{x}}) = 1.$$

where $\mathbb{T} = \{h_1, \ldots, h_t\} \subset \mathbb{F}^n$ is a public subset.

For our construction, we assume that from $\mathbb{T}$ one can define t polynomials $\boldsymbol{\chi}(X_1,\dots,X_n) \in (\mathbb{F}[X_1,\dots,X_n])^t$ such that:[3]

(a) for every $i,j \in [t]$ it holds $\chi_i(h_j) = 1$ if $j = i$ and 0 if $i \neq j$;
(b) given $\boldsymbol{r} \in \mathbb{F}^n$, $\boldsymbol{\chi}(\boldsymbol{r}) \in \mathbb{F}^t$ can be computed in time $O(t)$;
(c) every $\chi_i(X)$ has degree at most d such that $d/|\mathbb{F}| = \mathsf{negl}(\lambda)$.

- A universal CP-SNARK $\mathsf{CP_{ev}}$ for *committed* evaluations. Namely, $\mathsf{CP_{ev}}$ works for the commitment Com and the universal relation $\mathcal{R}_{\mathsf{ev}}$ such that each $\mathsf{R_{ev}} \in \mathcal{R}_{\mathsf{ev}}$ is parametrized by an integer $t \leqslant N$, and $\mathsf{R_{ev}} \subset \mathbb{F}^n \times \mathbb{F}[X_1,\dots,X_n] \times \mathbb{F}$ is such that $(\boldsymbol{r}, \tilde{x}(X), z) \in \mathsf{R_{ev}}$ iff $z = \tilde{x}(\boldsymbol{r})$.

  As per the CP-SNARK definition in Section 2.6, given a public statement $\boldsymbol{r} \in \mathbb{F}^n$, and commitments $\mathsf{c}_{\boldsymbol{x}}$ and c_z, $\mathsf{CP_{ev}}$ proves the existence of committed values $\tilde{x} \in \mathbb{F}[X_1,\dots,X_n]$ and $z \in \mathbb{F}$ (and opening values $o_{\boldsymbol{x}}, o_z$) such that:

$$z = \tilde{x}(\boldsymbol{r}) \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_{\boldsymbol{x}}, \tilde{x}(X), o_{\boldsymbol{x}}) = 1 \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_z, z, o_z) = 1.$$

- A hash function[4] $\mathsf{H} : \{0,1\}^* \longrightarrow \mathbb{F}^n$, that is used to generate randomness for the proofs in our evaluation algorithm.
- An HSNP scheme $\Sigma' := (\mathsf{Kg}', \mathsf{Sign}', \mathsf{Eval}', \mathsf{VerSig}')$ for:

$$\mathcal{R}_{\text{com-ip}} := \Big\{ \mathsf{R}^{(t,s)}_{\text{com-ip}} := \{(\mathsf{c}, (x_1,\dots,x_t), o) : \\ \mathsf{VerCom}(\mathsf{ck}, \mathsf{c}, \langle \boldsymbol{x}, \boldsymbol{s} \rangle, o)\} : t \leqslant N, \boldsymbol{s} \in \mathbb{F}^t \Big\}.$$

  Namely, Σ' can prove that c opens to the result of a (public) linear function on signed values. We call an HSNP scheme with this functionality a ComLHS.

4.1 Intuition behind generic HSNP

Before providing the formal description of the protocol, we here give a high level view of how the different components interact, and why they are all necessary for our construction.

To sign a labelled field element (x, τ), one computes $\sigma \leftarrow \mathsf{Sign}'(x, \tau)$ and outputs the pair (x, σ).

Now consider a positive integer $t \leqslant N$; a relation R; a statement y; a tuple of t signatures $\{(x_i, \sigma_i)\}_{i \in [t]}$; and a witness w such that, denoting $\tilde{x}(X_1,\dots,X_n) := \langle \boldsymbol{x}, \boldsymbol{\chi}(X_1,\dots,X_n) \rangle$, it holds that $(y, (\tilde{x}(h_1),\dots,\tilde{x}(h_t)), w) \in \mathsf{R}$. To evaluate a signature for y, one first computes a commitment $\mathsf{c}_{\boldsymbol{x}}$ to $\tilde{x}$. The CP-SNARK $\mathsf{CP}_{\mathcal{R}}$ allows one to compute π_y, a proof of the existence of $\tilde{x}$ (committed to in $\mathsf{c}_{\boldsymbol{x}}$) and of w, such that: $(y, (\tilde{x}(h_1),\dots,\tilde{x}(h_t)), w) \in \mathsf{R}$. Observe that π_y provides no guarantee $\tilde{x}$ was authenticated; hence we devise a technique allowing one to prove that $\mathsf{c}_{\boldsymbol{x}}$ commits to an authenticated vector by using only a ComLHS, i.e., an HSNP for (commitments to) linear functions. The main idea is to have the evaluator:

- evaluate $\tilde{x}$ in a *random* point $\boldsymbol{r}$, which results in $z := \tilde{x}(\boldsymbol{r})$, and to commit to z in c_z;
- use $\mathsf{CP_{ev}}$ to prove that, for the committed values $z, \tilde{x}$, it indeed holds that $z = \tilde{x}(\boldsymbol{r})$;
- use Σ' to prove that c_z commits to $z = \langle \boldsymbol{x}, \boldsymbol{s} \rangle$, where the linear function is $\boldsymbol{s} := \boldsymbol{\chi}(\boldsymbol{r})$ (which can be computed by the verifier).

Note that, so long as $\boldsymbol{r}$ is random and *independent* of the signed data $\{(x_i, \tau_i)\}_{i \in [t]}$ and of the polynomial $\tilde{x}$ committed to in $\mathsf{c}_{\boldsymbol{x}}$, then the proofs above imply that the authenticated $\boldsymbol{x}$ and the vector $(\tilde{x}(h_1),\dots,\tilde{x}(h_t))$ are the same except with probability $d/|\mathbb{F}|$ (chosen to be negligible in λ). In order to make the proof generation non-interactive, we would like to generate $\boldsymbol{r}$ using a random oracle; so as to achieve the independence mentioned above we need to somehow fix the inputs, by including them in the input to H. While the polynomial $\tilde{x}$ can be fixed before the choice of $\boldsymbol{r}$ by including its commitment $\mathsf{c}_{\boldsymbol{x}}$ along with a succinct proof of knowledge (or alternatively, as we do, by sending the proof π_y), the same cannot be done for the authenticated $\{(x_i, \tau_i)\}_{i \in [t]}$ as we do not have a succinct representation (e.g., a commitment) for them. To solve this issue, we show that we can achieve the same result by computing a signature σ'_{sum} for the commitment $\mathsf{c}_{\mathsf{sum}}$ to the *sum of all data items* $\sum_{i \in [t]} x_i$, and by including σ'_{sum} and $\mathsf{c}_{\mathsf{sum}}$ in the input to H, whose output results in $\boldsymbol{r}$.

4.2 Our generic HSNP scheme

Our homomorphic signature scheme HSNP for relations in

$$\mathcal{R} := \left\{ \mathsf{R} \subseteq \mathcal{D}_y \times \mathbb{F}^t \times \mathcal{D}_w : 1 \leqslant t \leqslant N \right\},$$

works as follows.

```
$\mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R})$:
 - Let $N := |\mathcal{L}|$
 - $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$
 - $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathsf{Kg}'(1^\lambda, \mathcal{L}, \mathcal{R}_{\text{com-ip}})$
 - $\mathsf{srs_{ev}} \leftarrow \mathsf{Kg^{ev}}(\mathsf{ck}, \mathcal{R}_{\mathsf{ev}})$
 - $\mathsf{srs}_{\mathcal{R}} \leftarrow \mathsf{Kg}^{\mathcal{R}}(\mathsf{ck}, \mathcal{R})$
 - Output $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs_{ev}}, \mathsf{srs}_{\mathcal{R}}, \mathsf{vk}', \mathsf{H})$ and $\mathsf{sk} := \mathsf{sk}'$.

$\mathsf{Sign}(\mathsf{sk}, \tau, x)$: on input $x \in \mathbb{F}$,
 - $\sigma \leftarrow \mathsf{Sign}'(\mathsf{sk}', \tau, x)$
 - Output $\Sigma := (x, \sigma)$.

$\mathsf{Eval}(\mathsf{vk}, \mathsf{R}, y, \Sigma_1, \dots, \Sigma_t, w)$:
 - For $i \in [t]$, parse $(x_i, \sigma_i) \leftarrow \Sigma_i$. Let $\boldsymbol{x} := (x_1, \dots, x_t)$
 - $(\mathsf{ek_R}, \mathsf{vk_R}) \leftarrow \mathsf{Derive}^{\mathcal{R}}(\mathsf{srs}_{\mathcal{R}}, \mathsf{R})$
 - $(\mathsf{ek_{ev}}, \mathsf{vk_{ev}}) \leftarrow \mathsf{Derive^{ev}}(\mathsf{srs_{ev}}, \mathsf{R_{ev}})$
 - $\tilde{x}(\boldsymbol{X}) \leftarrow \langle \boldsymbol{x}, \boldsymbol{\chi}(X_1, \dots, X_n) \rangle$
 - $(\mathsf{c}_{\boldsymbol{x}}, o_{\boldsymbol{x}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \tilde{x}(\boldsymbol{X}))$
 - $\pi_y \leftarrow \mathsf{Prove}^{\mathcal{R}}(\mathsf{ek_R}, \mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \tilde{x}, o_{\boldsymbol{x}}, w)$
 - $(\mathsf{c_{sum}}, o_{\mathsf{sum}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{1}^\top \cdot \boldsymbol{x})$
 - $\sigma'_{\mathsf{sum}} \leftarrow \mathsf{Eval}'(\mathsf{vk}', \mathsf{R}^{(t,\mathbf{1})}_{\text{com-ip}}, \mathsf{c_{sum}}, \{\sigma_i\}_{i \in [t]}, o_{\mathsf{sum}})$
 - $\boldsymbol{r} \leftarrow \mathsf{H}(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c_{sum}}, \sigma'_{\mathsf{sum}})$
 - $z \leftarrow \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle$
 - $(\mathsf{c}_z, o_z) \leftarrow \mathsf{Commit}(\mathsf{ck}, z)$
 - $\pi_z \leftarrow \mathsf{Prove^{ev}}(\mathsf{ek_{ev}}, \boldsymbol{r}, (\mathsf{c}_{\boldsymbol{x}}, \mathsf{c}_z), (\tilde{x}, z), (o_{\boldsymbol{x}}, o_z))$
 - $\sigma' \leftarrow \mathsf{Eval}'(\mathsf{vk}', \mathsf{R}^{(t, \boldsymbol{\chi}(\boldsymbol{r}))}_{\text{com-ip}}, \mathsf{c}_z, \{\sigma_i\}_{i \in [t]}, o_z)$
 - Output $\Sigma := (\mathsf{c}_{\boldsymbol{x}}, \pi_y, \mathsf{c_{sum}}, \sigma'_{\mathsf{sum}}, \mathsf{c}_z, \pi_z, \sigma')$

$\mathsf{VerPrep}(\mathsf{vk}, \mathsf{R})$:
 - $(\mathsf{ek_R}, \mathsf{vk_R}) \leftarrow \mathsf{Derive}^{\mathcal{R}}(\mathsf{srs}_{\mathcal{R}}, \mathsf{R})$
 - $(\mathsf{ek_{ev}}, \mathsf{vk_{ev}}) \leftarrow \mathsf{Derive^{ev}}(\mathsf{srs_{ev}}, \mathsf{R_{ev}})$
 - Output $\mathsf{vk_R} := (\mathsf{ck}, \mathsf{vk_{ev}}, \mathsf{vk_R}, \mathsf{vk}', \mathsf{H})$

$\mathsf{EffVer}(\mathsf{vk_R}, (\tau_1, \dots, \tau_t), y, \Sigma)$:
 - Parse $(\mathsf{c}_{\boldsymbol{x}}, \pi_y, \mathsf{c_{sum}}, \sigma'_{\mathsf{sum}}, \mathsf{c}_z, \pi_z, \sigma') \leftarrow \Sigma$
```

[3] In our proposed instantiation (Section 6) $\chi_i(X_1,\dots,X_n)$ is defined as $\lambda_{h_i}(X)$, the h_i-th Lagrange basis univariate polynomial for which properties (a)–(b) are satisfied when the Lagrange domain $\mathbb{H} \subset \mathbb{T}$ is FFT-friendly and property (c) is satisfied when $|\mathbb{F}| \geqslant 2^\lambda$. One may also consider other polynomial encodings, e.g. the monomial basis or Laurent polynomials.

[4] H is in fact a family of hash functions parametrised by $\lambda \in \mathbb{N}$; for readability we suppress this dependency in the notation.

- $\boldsymbol{r} \leftarrow \mathsf{H}(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$;
- $\boldsymbol{s} \leftarrow \boldsymbol{\chi}(\boldsymbol{r})$
- If any of the following hold, then reject (return 0):
    - $\mathsf{VerProof}^{\mathcal{R}}(\mathsf{vk}_{\mathsf{R}}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y) = 0$
    - $\mathsf{VerProof}^{\mathsf{ev}}(\mathsf{vk}_{\mathsf{ev}}, \boldsymbol{r}, (\mathsf{c}_{\boldsymbol{x}}, \mathsf{c}_z), \pi_z) = 0$
    - $\mathsf{VerSig}'(\mathsf{vk}', (\mathsf{R}^{(t,1)}_{\mathsf{com\text{-}ip}}, \tau_1, \ldots, \tau_t), \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}}) = 0$
    - $\mathsf{VerSig}'(\mathsf{vk}', (\mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}, \tau_1, \ldots, \tau_t), \mathsf{c}_z, \sigma') = 0$
- Else accept (return 1).

In Section 6 we detail a concrete instantiation of our generic protocol from efficient CP-SNARKs. The only missing link is a ComLHS scheme for relations in $\mathcal{R}_{\mathsf{com\text{-}ip}}$, which we provide in Section 5.

Correctness Authentication and evaluation correctness follow from that of Com, Σ', $\mathsf{CP}_{\mathsf{ev}}$ and $\mathsf{CP}_{\mathcal{R}}$.

Succinctness and amortized efficiency For an evaluated signature $\Sigma := (\mathsf{c}_{\boldsymbol{x}}, \pi_y, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}}, \mathsf{c}_z, \pi_z, \sigma')$, the values $\mathsf{c}_{\boldsymbol{x}}$, $\mathsf{c}_{\mathsf{sum}}$ and c_z are of constant size $O(\lambda)$ (independent of t). The proofs π_y and π_z have size at most $\mathsf{poly}(\lambda + \log w)$ by the succinctness of the CP-SNARKs. Hence, if the ComLHS scheme Σ' is succinct, so is HSNP (as the signatures σ'_{sum} and σ' are short). The amortized efficiency follows from the succinctness of the CP-SNARKs and the property (a) of the $\boldsymbol{\chi}$ polynomials.

Theorem 4.1. *If* Com *is binding;* H *is modelled as a random oracle;* Σ' *is adaptively secure for oracle* $\mathsf{H}(\cdot)$*; and* $\mathsf{CP}_{\mathsf{ev}}$ *and* $\mathsf{CP}_{\mathcal{R}}$ *are knowledge-sound for oracles* $(\mathsf{H}, \Sigma'.\mathsf{Sign})$*, then* HSNP *is adaptively secure. Furthermore, if* Com *is hiding, and* $\mathsf{CP}_{\mathcal{R}}$, $\mathsf{CP}_{\mathsf{ev}}$ *and* Σ' *are zero knowledge, then* HSNP *is zero-knowledge.*

4.2.1 *Adaptive security.*

Consider an adversary $\mathcal{A}$ against the adaptive security of HSNP that takes as input $(\mathcal{R}, \mathsf{vk}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}})$, where $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs}_{\mathsf{ev}}, \mathsf{srs}_{\mathcal{R}}, \mathsf{vk}')$, and that outputs a signature $\Sigma := (\mathsf{c}_{\boldsymbol{x}}, \pi_y, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}}, \mathsf{c}_z, \pi_z, \sigma')$ for a statement y and labelled relation $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ for some $t \leqslant N$. Let T be the transcript of $\mathcal{A}$'s queries to both the random oracle H and to the signing oracle. We denote by T_{H} (resp. T_{Σ}) the subset of T with queries to H (resp. Sign), and by $T_{<j}$ the subset of T with all queries made before the j-th query to H. Let Q_{H} be the number of queries to the random oracle H and let $j^* \in [Q_{\mathsf{H}}]$ be the index of the query to H which contains the tuple $(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$ returned in the forgery phase. Without loss of generality we assume that this oracle query exists, otherwise for any $\mathcal{A}$ that does not do such query we can build another adversary that makes this query at the very end.

We begin by showing that for every $\mathcal{A}$ there is an extractor $\mathcal{E}$. More in detail we show that for every $\mathcal{A}$ there exist extractor algorithms $\{\mathcal{E}_{\mathcal{R},j}\}_{j \in [Q_{\mathsf{H}}]}$, $\mathcal{E}_{\mathsf{ev}}$, $\mathcal{E}'$ that correspond to the schemes $\mathsf{CP}_{\mathcal{R}}$, $\mathsf{CP}_{\mathsf{ev}}$ and $\mathsf{LHS}_{\mathsf{Ped}}$ respectively. Next, we use these extractors to build $\mathcal{E}$.

First, given $\mathcal{A}$ we can define a collection of adversaries $\{\mathcal{A}_{\mathcal{R},j}\}_{j \in [Q_{\mathsf{H}}]}$ for the knowledge soundness of $\mathsf{CP}_{\mathcal{R}}$ for oracles $(\mathsf{H}(\cdot), \mathsf{Sign}'(\mathsf{sk}', \cdot, \cdot))$. $\mathcal{A}_{\mathcal{R},j}$ is the adversary that takes as input the universal commit-and-prove relation $(\mathsf{ck}, \mathcal{R})$, $\mathsf{CP}_{\mathcal{R}}$'s SRS $\mathsf{srs}_{\mathcal{R}}$, the relation's auxiliary input $\mathsf{aux}_{\mathsf{Rg}}$ and auxiliary input $(\mathsf{srs}_{\mathsf{ev}}, \mathsf{vk}', \mathsf{aux}_{\mathcal{Z}})$, it has access to oracles $\mathsf{H}(\cdot)$, $\mathsf{Sign}'(\mathsf{sk}', \cdot, \cdot)$ and runs $\mathcal{A}$ up to its j-th query to H, forwarding $\mathcal{A}$'s queries to its oracles. When receiving the j-th query $(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$, $\mathcal{A}_{\mathcal{R},j}$ outputs $(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y)$. By the oracle knowledge-soundness of $\mathsf{CP}_{\mathcal{R}}$, for every such $\mathcal{A}_{\mathcal{R},j}$ there is an extractor $\mathcal{E}_{\mathcal{R},j}$ that, given the same input of $\mathcal{A}_{\mathcal{R},j}$ and the transcript $T_{<j}$ of its oracle queries, outputs a tuple $\tilde{x}(\boldsymbol{X}), o_{\tilde{x}}, \tilde{w}$ such that, conditioned on that π_y correctly verifies, $(y, (\tilde{x}(h_1), \ldots, \tilde{x}(h_t)), \tilde{w}) \in \mathsf{R}$ and $\mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_{\boldsymbol{x}}, \tilde{x}, o_{\tilde{x}}) = 1$ hold with overwhelming probability.

Second, in a way similar to the previous case, given $\mathcal{A}$ we can define an adversary $\mathcal{A}_{\mathsf{ev}}$ for the knowledge soundness of $\mathsf{CP}_{\mathsf{ev}}$ for oracles $(\mathsf{H}(\cdot), \mathsf{Sign}'(\mathsf{sk}', \cdot, \cdot))$. $\mathcal{A}_{\mathsf{ev}}$ is the adversary that takes as input the universal commit-and-prove relation $(\mathsf{ck}, \mathcal{R}_{\mathsf{ev}})$, $\mathsf{CP}_{\mathsf{ev}}$'s SRS $\mathsf{srs}_{\mathsf{ev}}$, and auxiliary input consisting of $(\mathsf{srs}_{\mathcal{R}}, \mathsf{vk}', \mathcal{R}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}})$, it has access to oracles $(\mathsf{H}(\cdot), \mathsf{Sign}'(\mathsf{sk}', \cdot, \cdot)$ and runs $\mathcal{A}$ until it outputs the forgery $(\mathsf{R}, \tau_1, \ldots, \tau_t, y, \Sigma)$. Then $\mathcal{A}_{\mathsf{ev}}$ defines R_{ev} based on t and outputs $(\mathsf{R}_{\mathsf{ev}}, \boldsymbol{r}, (\mathsf{c}_{\boldsymbol{x}}, \mathsf{c}_z), \pi_z)$. By the oracle knowledge-soundness of $\mathsf{CP}_{\mathsf{ev}}$, for every such $\mathcal{A}_{\mathsf{ev}}$ there is an extractor $\mathcal{E}_{\mathsf{ev}}$ that, given the same input of $\mathcal{A}_{\mathsf{ev}}$ and the transcript T of its oracle queries, outputs values $\hat{x}(\boldsymbol{X}), o_{\hat{x}}, \hat{z}$ and $o_{\hat{z}}$ satisfying, with all but negligible probability, $\hat{z} = \hat{x}(\boldsymbol{r})$; $\mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_{\boldsymbol{x}}, \hat{x}, o_{\hat{x}}) = 1$ and $\mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_z, \hat{z}, o_{\hat{z}}) = 1$.

Third, given $\mathcal{A}$ we can build an adversary $\mathcal{A}'$ against the adaptive security of Σ' relative to a random oracle H. $\mathcal{A}'$ takes as input the verification key vk', the commitment key ck (which is part of the $\mathcal{R}_{\mathsf{com\text{-}ip}}$ description) and auxiliary input $(\mathsf{srs}_{\mathsf{ev}}, \mathsf{srs}_{\mathcal{R}}, \mathcal{R}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}})$ and runs $\mathcal{A}$ until it outputs a forgery, using its oracles to answer the queries of $\mathcal{A}$. By the adaptive security of Σ' (relative to oracle H), for any such $\mathcal{A}'$ there exists an extractor $\mathcal{E}'$ which, given the same input of $\mathcal{A}'$ and the transcript T of its queries, outputs an opening value o'_z satisfying $\mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_z, \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle, o'_z)$, where $\boldsymbol{x}$ is the vector of values queried to the oracle, i.e., such that $\forall i \in [t] : (x_i, \tau_i, \sigma_i) \in T_{\Sigma}$.

We have shown that for every $\mathcal{A}$ there exist extractor algorithms $\{\mathcal{E}_{\mathcal{R},j}\}_{j \in [Q_{\mathsf{H}}]}$, $\mathcal{E}_{\mathsf{ev}}$, $\mathcal{E}'$. Based on this, we define the extractor $\mathcal{E}(\mathcal{R}, \mathsf{vk}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}}, T)$ for $\mathcal{A}$ as follows.

- Run $\mathcal{A}$, using T to simulate its queries, obtaining $(y, \mathsf{R}, \tau_1, \ldots, \tau_t, \Sigma)$. Let $j^* \in [Q_{\mathsf{H}}]$ be the index of the H-oracle query containing the values in the forgery (see above for the precise definition of j^*).
- Run the extractors $\mathcal{E}_{\mathcal{R},j^*}, \mathcal{E}_{\mathsf{ev}}, \mathcal{E}'$: $(\tilde{x}(\boldsymbol{X}), o_{\tilde{x}}, \tilde{w}) \leftarrow \mathcal{E}_{\mathcal{R},j^*}((\mathsf{ck}, \mathcal{R}), \mathsf{srs}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{R}}, (\mathsf{srs}_{\mathsf{ev}}, \mathsf{vk}', \mathsf{aux}_{\mathcal{Z}}), T_{<j^*})$, $(\hat{x}(\boldsymbol{X}), o_{\hat{\boldsymbol{x}}}, \hat{z}, o_{\hat{z}}) \leftarrow \mathcal{E}_{\mathsf{ev}}((\mathsf{ck}, \mathcal{R}_{\mathsf{ev}}), \mathsf{srs}_{\mathsf{ev}}, (\mathsf{srs}_{\mathcal{R}}, \mathsf{vk}', \mathcal{R}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}}), T)$, and $o'_z \leftarrow \mathcal{E}'(\mathsf{ck}, \mathsf{vk}', (\mathsf{srs}_{\mathsf{ev}}, \mathsf{srs}_{\mathcal{R}}, \mathcal{R}, \mathsf{aux}_{\mathcal{R}}, \mathsf{aux}_{\mathcal{Z}}), T)$.
- If any of the extracted witnesses does not satisfy the corresponding relation, or $\tilde{x}(\boldsymbol{X}) \neq \hat{x}(\boldsymbol{X})$, or $\hat{z} \neq \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle$, then output $\perp$. Otherwise output $\tilde{w}$.

Next, we show that $\mathcal{E}$ is such that $\Pr[\mathsf{UF}_{\mathsf{Rg},\mathcal{Z},\mathcal{A},\mathcal{E}}(\lambda) = 1]$ is negligible. We do this via an hybrid argument. We describe a sequence of games, $\mathsf{Game}_0(\lambda), \ldots, \mathsf{Game}_2(\lambda)$, where $\mathsf{Game}_0(\lambda)$ is identical to the experiment $\mathsf{UF}_{\mathsf{Rg},\mathcal{Z},\mathcal{A},\mathcal{E}}(\lambda)$. Let us recall that in this game the output is the bit $b \leftarrow \mathsf{ValidForge} \wedge (\mathsf{ForgeType}_1 \vee \mathsf{ForgeType}_2)$ where $\mathsf{ValidForge} \leftarrow \mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}, \boldsymbol{\tau}), y, \sigma)$, $\mathsf{ForgeType}_1 \leftarrow \exists j \in [t] : (\tau_j, \cdot, \cdot) \notin T_{\Sigma}$, and $\mathsf{ForgeType}_2 \leftarrow (y, \boldsymbol{x}, \tilde{w}) \notin \mathsf{R}$. For $i \geq 1$, Game_i uses flag values $\mathsf{bad}_1, \ldots, \mathsf{bad}_i$ that are initially false. If at the end of the game any of these flags is true, Game_i outputs 0.

$\underline{\mathsf{Game}_1}$ Let us define Bad_1 as the event that there exists a random oracle query $(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$ such that $\mathsf{VerSig}'(\mathsf{vk}', (\mathsf{R}^{(t,1)}_{\mathsf{com\text{-}ip}}, \boldsymbol{\tau}), \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}}) = 1$ and $\exists i \in [t]$ such that τ_i was not yet queried to the signing oracle, namely $(\tau_i, \cdot) \notin T_{<j,\Sigma}$, for the j-th query to H. This experiment sets $\mathsf{bad}_1 \longleftarrow \mathsf{true}$ if Bad_1 occurs.

We claim that $\Pr[\mathsf{Game}_0(\lambda) = 1] - \Pr[\mathsf{Game}_1(\lambda) = 1] \leq \Pr[\mathsf{Bad}_1] \leq \mathsf{negl}(\lambda)$ based on the adaptive security of Σ'. The reduction is rather simple and works as follows. Given an $(\mathcal{A}, \mathcal{E})$ pair that causes Bad_1 to happen we can build an adversary $\mathcal{B}$ that has advantage $\Pr[\mathsf{Bad}_1]$ against the adaptive security of Σ'. $\mathcal{B}$ simulates the experiment up to the first oracle query for which Bad_1 happens. Let this query be $(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$. Then $\mathcal{B}$ returns $((\mathsf{R}^{(t,1)}_{\mathsf{com\text{-}ip}}, \boldsymbol{\tau}), \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$, which clearly makes the adaptive security game return 1.

Notice that if Game_1 outputs 1, Bad_1 did not occur and there cannot be type 1 forgeries, i.e., $\mathsf{ForgeType}_1 = \mathsf{false}$.

$\underline{\mathsf{Game}_2}$ Let Bad_2 be the event that $\mathcal{E}$ outputs $\perp$. This experiment proceeds as Game_1 except that it sets $\mathsf{bad}_2 \longleftarrow \mathsf{true}$ if Bad_2 occurs.

By definition of Game_2 we have

$$\Pr[\mathsf{Game}_1(\lambda) = 1] - \Pr[\mathsf{Game}_2(\lambda) = 1] \leq \Pr[\mathsf{Bad}_2 \wedge \mathsf{ValidForge}].$$

We claim that the latter probability is negligible based on the knowledge-soundness of $\mathsf{CP}_{\mathcal{R}}$ and $\mathsf{CP}_{\mathsf{ev}}$, the adaptive security of Σ' and the binding of the commitment scheme. The reduction is standard and is omitted.

Notice, if Game_2 outputs 1, then it must be that $\mathcal{E}$ did not output $\perp$, and it also holds $\hat{x}(\boldsymbol{X}) = \tilde{x}(\boldsymbol{X})$ and $\hat{z} = \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle = \tilde{x}(\boldsymbol{r})$.

We conclude the proof by showing that $\mathcal{A}$ and $\mathcal{E}$ have negligible probability of causing Game_2 return 1.

Let $\tilde{x}(\boldsymbol{X})$ be the polynomial extracted (internally) by $\mathcal{E}$, and let us define Bad^* as the event that $\tilde{x}(\boldsymbol{X}) \neq \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{X}) \rangle$. Then we have:

$$\Pr[\mathsf{Game}_2(\lambda) = 1] = \Pr[\mathsf{Game}_2(\lambda) = 1 \wedge \mathsf{Bad}^*] \leq d/|\mathbb{F}| = \mathsf{negl}(\lambda)$$

To see this, we first observe that $\Pr[\mathsf{Game}_2(\lambda) = 1 \wedge \neg\mathsf{Bad}^*] = 0$. In order to return 1, neither Bad_1 nor Bad_2 have occurred in Game_2. From $\neg\mathsf{Bad}_1$ we get that the extracted $\tilde{w}$ is such that $(y, \boldsymbol{x}, \tilde{w}) \notin \mathsf{R}$. From $\neg\mathsf{Bad}_2$ we have that $(y, (\tilde{x}(h_1), \ldots, \tilde{x}(h_t)), \tilde{w}) \in \mathsf{R}$. Finally, conditioning on $\neg\mathsf{Bad}^*$ we have $\tilde{x}(\boldsymbol{X}) = \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{X}) \rangle$, and thus for all $i \in [t]$ $\tilde{x}(h_i) = x_i$. Hence, we have $(y, \boldsymbol{x}, \tilde{w}) \in \mathsf{R}$ and thus Game_2 cannot return 1 in this case.

Next, we argue that $\Pr[\mathsf{Game}_2(\lambda) = 1 \wedge \mathsf{Bad}^*] \leq d/|\mathbb{F}| = \mathsf{negl}(\lambda)$ over the random choice of $\boldsymbol{r}$ in the j^*-th query to the random oracle H. Observe that by the validity of $\mathcal{A}$'s forgery and by the fact that Bad_2 did not occur, we have $\langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle = \tilde{x}(\boldsymbol{r})$ and $\tilde{x}(\boldsymbol{X}) \neq \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{X}) \rangle$. If $\boldsymbol{r}$ is random and independent of $\tilde{x}(\boldsymbol{X}), \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{X})$ then the probability that this happens is $d/|\mathbb{F}| = \mathsf{negl}(\lambda)$, by property (c) of $\boldsymbol{\chi}$. In particular, $\boldsymbol{r}$ is *independent* from the random oracle's input $(y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$, which makes it also independent from $\boldsymbol{x}$ and $\tilde{x}$. Independence from $\tilde{x}$ holds because we can extract $\tilde{x}$ using $\mathcal{E}_{\mathcal{R},j^*}$ *before* issuing this random oracle query and receiving the answer $\boldsymbol{r}$. Independence from the queried inputs $\boldsymbol{x}$ follows by observing that $\boldsymbol{r}$ is independent from the random oracle query's input $\boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}}$. Since the signature σ'_{sum} is valid and since Bad_1 did not occur, we obtain that all the data items $\boldsymbol{x}$ must have been already queried to the signing oracle, and thus we can fix them in the view of the game, before $\boldsymbol{r}$ is chosen.

4.2.2 Zero knowledge. Let $\mathcal{A}$ be a PT adversary for the zero-knowledge property of HSNP. In the $\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}}$ experiment, the challenger runs $(\mathsf{vk}, \mathsf{sk}) \longleftarrow \mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R})$, and hands vk to $\mathcal{A}$. Then during the signing queries phase, $\mathcal{A}$ adaptively requests signatures for pairs (τ, x) of its choosing. We denote $T := \{(i, x_i, \Sigma_i)\}_{i \in [q]}$ the list of queries made by $\mathcal{A}$ and corresponding signatures, where $q \in \mathbb{N}$ is the number of answered queries.

In the choose phase $\mathcal{A}$ outputs y, $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ and w. Note that if this output causes either the real or simulated experiment to output the error symbol $\perp$, then both experiments do, and are identical from $\mathcal{A}$'s view. Assuming this does not occur., it holds that, for $i \in [t]$, there exist $\Sigma_i := (x_i, \sigma_i)$ such that $(i, x_i, \Sigma_i) \in T$, and $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$.

To prove the zero-knowledge property of the scheme, we proceed via a sequence of games. Game 0 is the experiment $\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}}$, whereas the final game (Game 3) is $\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}}$, which calls upon the simulator of Figure 4. We denote by S_i the event $\mathcal{A}$ outputs $b' = 1$ in Game i. Hence $\Pr[S_0] = \Pr[\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}} = 1]$, and $\Pr[S_3] = \Pr[\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}} = 1]$. In all games, the signing queries and choose phases are as described above (i.e. as per experiment $\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}}$).

By proving that, from $\mathcal{A}$'s view, each game is indistinguishable from the next, we demonstrate that $\mathcal{A}$ cannot, with significant probability, output a different bit b in experiments $\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}}$ and $\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}}$.

A detailed description of each game is provided in Figure 3, changes from one game to the next are highlighted for improved visibility.

Game 0 to Game 1 Since Σ' is zero-knowledge, there exists a simulator $\mathcal{S}' = (\mathcal{S}'_{\mathsf{kg}}, \mathcal{S}'_{\mathsf{eval}})$ for the ZK property of the scheme. Here we substitute the key generation and evaluation algorithms of Σ' by algorithms $\mathcal{S}'_{\mathsf{kg}}$ and $\mathcal{S}'_{\mathsf{eval}}$ respectively. Note that $\mathcal{S}'_{\mathsf{eval}}$ outputs signatures σ' and σ'_{sum} which are independent of the individual signatures $\Sigma_1, \ldots, \Sigma_n$ in T. By the zero-knowledge property of Σ', both games are indistinguishable to $\mathcal{A}$, so:

$$|\Pr[S_0] - \Pr[S_1]| = \mathsf{negl}(\lambda). \tag{1}$$

Game 1 to Game 2 In Game 2, we call upon the zero-knowledge simulators $(\mathcal{S}^{\mathsf{ev}}_{\mathsf{kg}}, \mathcal{S}^{\mathsf{ev}}_{\mathsf{prv}})$ and $(\mathcal{S}^{\mathcal{R}}_{\mathsf{kg}}, \mathcal{S}^{\mathcal{R}}_{\mathsf{prv}})$ of $\mathsf{CP}_{\mathcal{R}}$ and $\mathsf{CP}_{\mathsf{ev}}$ respectively. Note that these simulators only take input the commitments to $\boldsymbol{x}$ and w, but not the opening values. By the zero-knowledge property of $\mathsf{CP}_{\mathcal{R}}$ and $\mathsf{CP}_{\mathsf{ev}}$, both games are indistinguishable from $\mathcal{A}$'s view, so:

$$|\Pr[S_1] - \Pr[S_2]| = \mathsf{negl}(\lambda). \tag{2}$$

Game 2 to Game 3 Here, instead of committing to the real inputs provided by $\mathcal{A}$, Game 3 samples random $\boldsymbol{x}^* \in \mathbb{F}^n$, and $z^* \in \mathbb{F}$, computes commitments $\mathsf{c}_{\boldsymbol{x}}$, $\mathsf{c}_{\mathsf{sum}}$, and c_z to $\boldsymbol{x}^*$, $\mathbf{1}^\top \cdot \boldsymbol{x}^*$ and z^* respectively, and uses these commitments throughout the experiment. Due to the fact Com is statistically hiding, and since $\mathcal{S}^{\mathsf{ev}}_{\mathsf{prv}}$ and $\mathcal{S}^{\mathcal{R}}_{\mathsf{prv}}$ only see the commitments $\mathsf{c}_{\boldsymbol{x}}$ and c_z (but not the openings), from $\mathcal{A}$'s view, both games are statistically close, and the probability that $\mathcal{A}$ behaves differently in Game 2 than in Game 3 is negligible.

$$|\Pr[S_2] - \Pr[S_3]| = \mathsf{negl}(\lambda). \tag{3}$$

Now in Game 3, the evaluation of the signature sent to $\mathcal{A}$ only relies on the statement y, the labelled relation $(\mathsf{R}, \tau_1, \ldots, \tau_t)$ and on

```
Key generation
 - Let $N := |\mathcal{L}|$; $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$
 - $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathsf{Kg}'(1^\lambda, \mathcal{L}, \mathcal{R}_{\text{com-ip}})$
 - $(\mathsf{ek}_{\mathsf{ev}}, \mathsf{vk}_{\mathsf{ev}}) \leftarrow \mathsf{Kg}^{\mathsf{ev}}(\mathsf{ck}, \mathcal{R}_{\mathsf{ev}})$
 - $(\mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}) \leftarrow \mathsf{Kg}^{\mathcal{R}}(\mathsf{ck}, \mathcal{R})$
 - $\mathsf{vk} := (\mathsf{ck}, \mathsf{ek}_{\mathsf{ev}}, \mathsf{vk}_{\mathsf{ev}}, \mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}, \mathsf{vk}', \mathsf{H})$; $\mathsf{sk} := \mathsf{sk}'$
 - send $\mathsf{vk}$ to $\mathcal{A}$.
Challenge
 - For $i \in [n]$, parse $(x_i, \sigma_i) \leftarrow \Sigma_i$
 - $(\mathsf{c}_{\boldsymbol{x}}, o_{\boldsymbol{x}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \boldsymbol{x})$
 - $\pi_y \leftarrow \mathsf{Prove}^{\mathcal{R}}(\mathsf{ek}_{\mathsf{R}}, \mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \tilde{x}, o_{\boldsymbol{x}}, w)$
 - $(\mathsf{c}_{\mathsf{sum}}, o_{\mathsf{sum}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{1}^\top \cdot \boldsymbol{x})$;
 - $\sigma'_{\mathsf{sum}} \leftarrow \mathsf{Eval}'(\mathsf{vk}', \mathsf{R}'_{t,1}, \mathsf{c}_{\mathsf{sum}}, \{\sigma_i\}_{i\in[t]}, o_{\mathsf{sum}})$
 - $\boldsymbol{r} \leftarrow \mathsf{H}(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$; $z \leftarrow \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle$
 - $(\mathsf{c}_z, o_z) \leftarrow \mathsf{Commit}(\mathsf{ck}, z)$
 - $\pi_z \leftarrow \mathsf{Prove}^{\mathsf{ev}}(\mathsf{ek}_{\mathsf{ev}}, \boldsymbol{r}, (\mathsf{c}_{\boldsymbol{x}}, \mathsf{c}_z), (\tilde{x}, z), (o_{\boldsymbol{x}}, o_z))$
 - $\sigma' \leftarrow \mathsf{Eval}'(\mathsf{vk}', \mathsf{R}'_{t,\boldsymbol{\chi}(\boldsymbol{r})}, \mathsf{c}_z, \{\sigma_i\}_{i\in[n]}, o_z)$
 - send $\Sigma := (\mathsf{c}_{\boldsymbol{x}}, \pi_y, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}}, \mathsf{c}_z, \pi_z, \sigma')$ to $\mathcal{A}$
 - $\mathcal{A}$ outputs $b'$
```

Game 0

```
Key generation
 - Let $N := |\mathcal{L}|$; $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$
 - $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathcal{S}'_{\mathsf{kg}}(1^\lambda, \mathcal{L}, \mathcal{R}_{\text{com-ip}})$
 - $(\mathsf{ek}_{\mathsf{ev}}, \mathsf{vk}_{\mathsf{ev}}) \leftarrow \mathsf{Kg}^{\mathsf{ev}}(\mathsf{ck}, \mathcal{R}_{\mathsf{ev}})$
 - $(\mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}) \leftarrow \mathsf{Kg}^{\mathcal{R}}(\mathsf{ck}, \mathcal{R})$
 - $\mathsf{vk} := (\mathsf{ck}, \mathsf{ek}_{\mathsf{ev}}, \mathsf{vk}_{\mathsf{ev}}, \mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}, \mathsf{vk}', \mathsf{H})$; $\mathsf{sk} := \mathsf{sk}'$
 - send $\mathsf{vk}$ to $\mathcal{A}$.
Challenge
 - $\tilde{x}(\boldsymbol{X}) \leftarrow \langle \boldsymbol{x}, \boldsymbol{\chi}(X_1, \ldots, X_n) \rangle$
 - $(\mathsf{c}_{\boldsymbol{x}}, o_{\boldsymbol{x}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \tilde{x}(\boldsymbol{X}))$
 - $\pi_y \leftarrow \mathsf{Prove}^{\mathcal{R}}(\mathsf{ek}_{\mathsf{R}}, \mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \tilde{x}, o_{\boldsymbol{x}}, w)$
 - $(\mathsf{c}_{\mathsf{sum}}, o_{\mathsf{sum}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{1}^\top \cdot \boldsymbol{x})$;
 - $\sigma'_{\mathsf{sum}} \leftarrow \mathcal{S}'_{\mathsf{eval}}(\mathsf{vk}', \mathsf{sk}', (\mathsf{R}'_{t,1}, \tau_1, \ldots, \tau_n), \mathsf{c}_{\mathsf{sum}})$
 - $\boldsymbol{r} \leftarrow \mathsf{H}(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$; $z \leftarrow \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle$
 - $(\mathsf{c}_z, o_z) \leftarrow \mathsf{Commit}(\mathsf{ck}, z)$
 - $\pi_z \leftarrow \mathsf{Prove}^{\mathsf{ev}}(\mathsf{ek}_{\mathsf{ev}}, \boldsymbol{r}, (\mathsf{c}_{\boldsymbol{x}}, \mathsf{c}_z), (\tilde{x}, z), (o_{\boldsymbol{x}}, o_z))$
 - $\sigma' \leftarrow \mathcal{S}'_{\mathsf{eval}}(\mathsf{vk}', \mathsf{sk}', (\mathsf{R}^{(t,s)}_{\text{com-ip}}, \tau_1, \ldots, \tau_n), \mathsf{c}_z)$
 - send $\Sigma := (\mathsf{c}_{\boldsymbol{x}}, \pi_y, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}}, \mathsf{c}_z, \pi_z, \sigma')$ to $\mathcal{A}$
 - $\mathcal{A}$ outputs $b'$.
```

Game 1

```
Key generation
 - Let $N := |\mathcal{L}|$; $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$
 - $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathcal{S}'_{\mathsf{kg}}(1^\lambda, \mathcal{L}, \mathcal{R}_{\text{com-ip}})$
 - $(\mathsf{ek}_{\mathsf{ev}}, \mathsf{vk}_{\mathsf{ev}}, \mathsf{td}_{\mathsf{ev}}) \leftarrow \mathcal{S}^{\mathsf{ev}}_{\mathsf{kg}}(\mathsf{ck}, \mathcal{R}_{\mathsf{ev}})$
 - $(\mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}, \mathsf{td}_{\mathcal{R}}) \leftarrow \mathcal{S}^{\mathcal{R}}_{\mathsf{kg}}(\mathsf{ck}, \mathcal{R})$
 - $\mathsf{vk} := (\mathsf{ck}, \mathsf{ek}_{\mathsf{ev}}, \mathsf{vk}_{\mathsf{ev}}, \mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}, \mathsf{vk}', \mathsf{H})$; $\mathsf{sk} := (\mathsf{sk}', \mathsf{td}_{\mathsf{ev}}, \mathsf{td}_{\mathcal{R}})$
 - send $\mathsf{vk}$ to $\mathcal{A}$.
Challenge
 - $\tilde{x}(\boldsymbol{X}) \leftarrow \langle \boldsymbol{x}, \boldsymbol{\chi}(X_1, \ldots, X_n) \rangle$
 - $(\mathsf{c}_{\boldsymbol{x}}, o_{\boldsymbol{x}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \tilde{x}(\boldsymbol{X}))$
 - $\pi_y \leftarrow \mathcal{S}^{\mathcal{R}}_{\mathsf{prv}}(\mathsf{ek}_{\mathsf{R}}, \mathsf{td}_{\mathcal{R}}, \mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}})$
 - $(\mathsf{c}_{\mathsf{sum}}, o_{\mathsf{sum}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{1}^\top \cdot \boldsymbol{x})$;
 - $\sigma'_{\mathsf{sum}} \leftarrow \mathcal{S}'_{\mathsf{eval}}(\mathsf{vk}', \mathsf{sk}', (\mathsf{R}'_{t,1}, \tau_1, \ldots, \tau_n), \mathsf{c}_{\mathsf{sum}})$
 - $\boldsymbol{r} \leftarrow \mathsf{H}(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$; $z \leftarrow \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle$
 - $(\mathsf{c}_z, o_z) \leftarrow \mathsf{Commit}(\mathsf{ck}, z)$
 - $\pi_z \leftarrow \mathcal{S}^{\mathsf{ev}}_{\mathsf{prv}}(\mathsf{ek}_{\mathsf{ev}}, \mathsf{td}_{\mathsf{ev}}, \boldsymbol{r}, (\mathsf{c}_{\boldsymbol{x}}, \mathsf{c}_z))$
 - $\sigma' \leftarrow \mathcal{S}'_{\mathsf{eval}}(\mathsf{vk}', \mathsf{sk}', (\mathsf{R}^{(t,s)}_{\text{com-ip}}, \tau_1, \ldots, \tau_n), \mathsf{c}_z)$
 - send $\Sigma := (\mathsf{c}_{\boldsymbol{x}}, \pi_y, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}}, \mathsf{c}_z, \pi_z, \sigma')$ to $\mathcal{A}$
 - $\mathcal{A}$ outputs $b'$.
```

Game 2

```
Key generation
 - Let $N := |\mathcal{L}|$; $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$
 - $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathcal{S}'_{\mathsf{kg}}(1^\lambda, \mathcal{L}, \mathcal{R}_{\text{com-ip}})$
 - $(\mathsf{ek}_{\mathsf{ev}}, \mathsf{vk}_{\mathsf{ev}}, \mathsf{td}_{\mathsf{ev}}) \leftarrow \mathcal{S}^{\mathsf{ev}}_{\mathsf{kg}}(\mathsf{ck}, \mathcal{R}_{\mathsf{ev}})$
 - $(\mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}, \mathsf{td}_{\mathcal{R}}) \leftarrow \mathcal{S}^{\mathcal{R}}_{\mathsf{kg}}(\mathsf{ck}, \mathcal{R})$
 - $\mathsf{vk} := (\mathsf{ck}, \mathsf{ek}_{\mathsf{ev}}, \mathsf{vk}_{\mathsf{ev}}, \mathsf{ek}_{\mathsf{R}}, \mathsf{vk}_{\mathsf{R}}, \mathsf{vk}', \mathsf{H})$; $\mathsf{sk} := (\mathsf{sk}', \mathsf{td}_{\mathsf{ev}}, \mathsf{td}_{\mathcal{R}})$
 - Send $\mathsf{vk}$ to $\mathcal{A}$.
Challenge
 - Sample $\boldsymbol{x}^* \leftarrow\!\!\$\ \mathbb{F}^n$ and $z^* \leftarrow\!\!\$\ \mathbb{F}$
 - $(\mathsf{c}_{\boldsymbol{x}}, o_{\boldsymbol{x}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \langle \boldsymbol{x}^*, \boldsymbol{\chi}(X_1, \ldots, X_n) \rangle)$
 - $\pi_y \leftarrow \mathcal{S}^{\mathcal{R}}_{\mathsf{prv}}(\mathsf{ek}_{\mathsf{R}}, \mathsf{td}_{\mathcal{R}}, \mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}})$
 - $(\mathsf{c}_{\mathsf{sum}}, o_{\mathsf{sum}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{1}^\top \cdot \boldsymbol{x}^*)$;
 - $\sigma'_{\mathsf{sum}} \leftarrow \mathcal{S}'_{\mathsf{eval}}(\mathsf{vk}', \mathsf{sk}', (\mathsf{R}'_{t,1}, \tau_1, \ldots, \tau_n), \mathsf{c}_{\mathsf{sum}})$
 - $\boldsymbol{r} \leftarrow \mathsf{H}(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}})$; $z \leftarrow \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle$
 - $(\mathsf{c}_z, o_z) \leftarrow \mathsf{Commit}(\mathsf{ck}, z^*)$
 - $\pi_z \leftarrow \mathcal{S}^{\mathsf{ev}}_{\mathsf{prv}}(\mathsf{ek}_{\mathsf{ev}}, \mathsf{td}_{\mathsf{ev}}, \boldsymbol{r}, \mathsf{c}_{\boldsymbol{x}}, \mathsf{c}_z)$
 - $\sigma' \leftarrow \mathcal{S}'_{\mathsf{eval}}(\mathsf{vk}', \mathsf{sk}', (\mathsf{R}^{(t,s)}_{\text{com-ip}}, \tau_1, \ldots, \tau_n), \mathsf{c}_z)$
 - Send $\Sigma := (\mathsf{c}_{\boldsymbol{x}}, \pi_y, \mathsf{c}_{\mathsf{sum}}, \sigma'_{\mathsf{sum}}, \mathsf{c}_z, \pi_z, \sigma')$ to $\mathcal{A}$
 - $\mathcal{A}$ outputs $b'$
```

Game 3

Figure 3: Game steps for proving zero-knowledge of HSNP.

the public and secret keys output by the various simulators, but not on the signed data $\{(i, x_i, \Sigma_i)\} \in T$. We can thus see that Game 3 is $\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}}$, which calls upon the simulators of Figure 4. Hence $\Pr[S_3] = \Pr[\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}} = 1]$, and combining eqs. (1) to (3), we get the desired:

$$\left|\Pr[\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}} = 1] - \Pr[\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}} = 1]\right| = \mathsf{negl}(\lambda)$$

5 CONCRETE HSNP SCHEME FOR COMMITMENTS TO LINEAR FUNCTIONS

We here devise a ComLHS scheme for the evaluation of (perfectly hiding) Pedersen commitments. The scheme, dubbed $\mathsf{LHS}_{\mathsf{Ped}}$, can be used to instantiate our HSNP protocol for universal relations of Section 4.

For a given security parameter $\lambda \in \mathbb{N}$, consider a description of bilinear groups $\mathsf{pp}_{\mathcal{G}} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ and group element h sampled uniformly at random from $\mathbb{G}_1$. Let $\mathsf{ck} := (\mathsf{pp}_{\mathcal{G}}, h)$. The message space $\mathcal{M}$ and non-deterministic input space $\mathcal{D}_w$ of $\mathsf{LHS}_{\mathsf{Ped}}$ are $\mathbb{Z}_q$. For a positive integer $N = \mathsf{poly}(\lambda)$, the set of admissible

```
$\mathcal{S}_{\mathsf{kg}}(1^\lambda, \mathcal{L}, \mathcal{R})$
$(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathcal{S}'_{\mathsf{kg}}(1^\lambda, \mathcal{L}, \mathcal{R}_{\mathsf{com\text{-}ip}})$
$N := |\mathcal{L}|,\ \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{ek_{ev}}, \mathsf{vk_{ev}}) \leftarrow \mathsf{Kg^{ev}}(\mathsf{ck}, \mathcal{R}_{\mathsf{ev}})$
$(\mathsf{ek_R}, \mathsf{vk_R}) \leftarrow \mathsf{Kg}^{\mathcal{R}}(\mathsf{ck}, \mathcal{R})$
$\mathsf{vk} := (\mathsf{ck}, \mathsf{ek_{ev}}, \mathsf{vk_{ev}}, \mathsf{ek_R}, \mathsf{vk_R}, \mathsf{vk}', \mathsf{H})$
$\mathsf{sk} := \mathsf{sk}'$
return $(\mathsf{sk}, \mathsf{vk})$
```

```
$\mathcal{S}_{\mathsf{eval}}(\mathsf{vk}, \mathsf{sk}, (\mathsf{R}, \tau_1, \ldots, \tau_n), y)$
Parse: $(\mathsf{ck}, \mathsf{ek_{ev}}, \mathsf{vk_{ev}}, \mathsf{ek_R}, \mathsf{vk_R}, \mathsf{vk}', \mathsf{H}) = \mathsf{vk}$;
Sample random $\tilde{x}_1, \ldots, \tilde{x}_n \leftarrow\!\$\, \mathbb{F}^n$ and $\tilde{w} \in \mathcal{D}_w$.
$(\mathsf{c}_{\boldsymbol{x}}, o_{\boldsymbol{x}}) \leftarrow \mathsf{Commit}(\mathsf{ck}, \tilde{\boldsymbol{x}})$
$\pi_y \leftarrow \mathcal{S}^{\mathcal{R}}_{\mathsf{prv}}(\mathsf{ek_R}, \mathsf{td}_{\mathcal{R}}, \mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}})$
$\boldsymbol{r} \leftarrow \mathsf{H}(\mathsf{R}, y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \boldsymbol{\tau}, \mathsf{c_{sum}}, \sigma'_{\mathsf{sum}});\ z \leftarrow \langle \boldsymbol{x}, \boldsymbol{\chi}(\boldsymbol{r}) \rangle$
$(\mathsf{c}_z, o_z) \leftarrow \mathsf{Commit}(\mathsf{ck}, z)$
$\pi_z \leftarrow \mathcal{S}^{\mathsf{ev}}_{\mathsf{prv}}(\mathsf{ek_{ev}}, \mathsf{td_{ev}}, \boldsymbol{r}, (\mathsf{c}_{\boldsymbol{x}}, \mathsf{c}_z))$
$\sigma' \leftarrow \mathcal{S}'_{\mathsf{eval}}(\mathsf{vk}', \mathsf{sk}', (\mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}, \tau_1, \ldots, \tau_t), \mathsf{c}_z)$
return $\Sigma := (y, \mathsf{c}_{\boldsymbol{x}}, \pi_y, \mathsf{c}_z, \pi_z, \sigma')$
```

Figure 4: Zero knowledge simulators for HSNP.

relations is

$$\mathcal{R}_{\mathsf{com\text{-}ip}} := \Big\{ \mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}} := \{ (y, (x_1, \ldots, x_t), w) \in \mathbb{G}_1 \times \mathbb{Z}_q^{t+1} : y = g_1^{\langle \boldsymbol{x}, \boldsymbol{s} \rangle} h^w \} : t \in [N], \boldsymbol{s} \in (\mathbb{Z}_q^*)^t \Big\}.$$

$\mathsf{LHS_{Ped}}$ builds upon a hash function $\mathsf{H} : \{0,1\}^* \longrightarrow \mathbb{G}_1$; a PRF $\mathsf{F} : \mathcal{K} \times \{0,1\}^* \longrightarrow \mathbb{Z}_q$ whose key space is denoted $\mathcal{K}$; and a proof system $\mathsf{NIZK} := (\mathsf{K}, P, V)$ for relation $\mathsf{R_c} := \{(y; z, w) : y = g_1^z h^w\}$. We let K take as additional input ck, so that the resulting CRS may depend on pp_G and h.

5.1 Intuition behind $\mathsf{LHS_{Ped}}$

Before formally describing the protocol, we here give a high level view of how the different components interact, and why they are all necessary for our construction.

The signing key contains two elements $a, b \in \mathbb{Z}_q$ and a PRF key. The public key contains, among other things, group elements $h, h^a \in \mathbb{G}_1$.

To sign a labelled field element (x, τ), one first computes a hash $R_\tau \in \mathbb{G}_1$ of the label. Including this hash in the signature, as $\Lambda := (R_\tau g_1^x)^a$, will ensure that the signature for x is tied to label τ. However, such a Λ is not simulatable (when proving unforgeability). Indeed, the natural way to give simulated signatures to the adversary here would be, knowing h^γ and $(h^a)^\gamma$ for some randomly chosen γ, to program $R_\tau := (h^\gamma) g_1^{-x}$, so that one can simulate the signature component $\Lambda := (h^a)^\gamma$. Unfortunately, the adversary may query the random oracle H on input τ *before* requesting the signature of the pair (x, τ). To overcome this, we introduce an extra degree of liberty: the signer will also evaluate the PRF F on input τ to obtain $r \in \mathbb{Z}_q$, and computes $\Lambda := (R_\tau g_1^{x+br})^a$. Now since the PRF key is known only to the signer, the reduction will be able to simulate the PRF output r in such a way that it cancels out x, as desired. The full signature contains x, Λ and r.

To evaluate a relation $\mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}$, given signatures $\sigma_1, \ldots, \sigma_t$ for pairs $(x_1, \tau_1), \ldots, (x_t, \tau_t)$, along with y, w such that $(y; \boldsymbol{x}, w) \in \mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}$, the process is quite intuitive. Having extracted x_i, Λ_i, r_i from each σ_i, the evaluator first computes the linear function of $\boldsymbol{x}$ which is committed to in y, i.e. $\langle \boldsymbol{x}, \boldsymbol{s} \rangle$; and applies the same linear function to the pseudo-randomness $(r_1, \ldots, r_t)$, i.e., $\langle \boldsymbol{r}, \boldsymbol{s} \rangle$. It also evaluates an analogue computation over the Λ_i's, while introducing the non-deterministic input w. Precisely, using the value h^a included in the public key, it computes $(h^a)^w \prod_{i \in [t]} \Lambda_i^{s_i}$. This is equal to $(h^w R g_1^{\langle \boldsymbol{x}, \boldsymbol{s} \rangle + b \langle \boldsymbol{r}, \boldsymbol{s} \rangle})^a$, where R is simply a multi-exponentiation of the R_{τ_i} to the exponent s_i. Enabling this randomization by w in an authenticated way is the main innovation of our scheme, i.e., where it departs from existing linearly-homomorphic signatures techniques. The latter only support deterministic computations in which any bias of the result should be considered a forgery. So the challenge here is to actually allow one to "bias" the result but only according to a specific distribution, namely a multiple of the group element h. We achieve this by making h^a public and showing that this does not harm the security. Such a change however required us to have the evaluator including an NIZKPoK π of z, w such that $(y, z, w) \in \mathsf{R_c}$. This will be used to extract w in our proof of adaptive security. Finally, we note that the reason why we generate r using a PRF, rather than fully at random, is due to proving the zero-knowledge of $\mathsf{LHS_{Ped}}$. Deriving the r_i for the i-th input deterministically from the label τ_i makes the r of the evaluated signature a deterministic function of the labels of the inputs, the relation $\mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}$ and the secret key. Since all this information is known to the simulator, it can simulate r.

5.2 Our HSNP scheme for $\mathcal{R}_{\mathsf{com\text{-}ip}}$

Our homomorphic signature scheme $\mathsf{LHS_{Ped}}$ for relations in $\mathcal{R}_{\mathsf{com\text{-}ip}}$ works as follows.

$\mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R}_{\mathsf{com\text{-}ip}})$**:** Let $N := |\mathcal{L}|$;

- $\mathsf{pp}_{\mathcal{G}} \leftarrow \mathcal{G}(1^\lambda)$; $h \leftarrow\!\$\, \mathbb{G}_1$
- $\mathsf{ck} := (\mathsf{pp}_{\mathcal{G}}, h)$
- $\mathsf{srs} \leftarrow \mathsf{K}(1^\lambda, \mathsf{ck})$
- $\kappa \leftarrow\!\$\, \mathcal{K}$ for the PRF F
- $a, b \leftarrow\!\$\, \mathbb{Z}_q$
- $\Gamma_1 \leftarrow h^a$, $\Gamma_2 \leftarrow g_2^{1/a}$ and $B \leftarrow g_1^b$
- $\mathsf{sk} := (a, b, \kappa)$ and $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs}, \Gamma_1, \Gamma_2, B, \mathsf{H}, \mathsf{F})$
- Output $(\mathsf{sk}, \mathsf{vk})$

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

- $R_\tau \leftarrow \mathsf{H}(\tau)$;
- $r \leftarrow \mathsf{F}_\kappa(\tau)$;
- $\Lambda \leftarrow (R_\tau g_1^{x+br})^a$
- Output $\sigma := (x, \Lambda, r, \emptyset)$

$\mathsf{Eval}(\mathsf{vk}, \mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}, y, \sigma_1, \ldots, \sigma_t, w)$**:**

- Parse $\boldsymbol{s} \in \mathbb{Z}_q^t$ from $\mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}$
- For $i \in [t]$, parse $(x_i, \Lambda_i, r_i, \emptyset) = \sigma_i$
- $z \leftarrow \langle \boldsymbol{x}, \boldsymbol{s} \rangle$ and $r \leftarrow \langle \boldsymbol{r}, \boldsymbol{s} \rangle$

```
- $\pi \leftarrow \mathsf{NIZK}.P(\mathsf{R_c}, \mathsf{srs}, y, (z, w))$
- $\Lambda \leftarrow \Gamma_1^w \cdot \prod_{i \in [t]} (\Lambda_i)^{s_i}$
- Output $\sigma := (y, \Lambda, r, \pi)$
```

```
VerSig(vk, $(\mathsf{R}^{(t,s)}_{\text{com-ip}}, \tau_1, \ldots, \tau_t), y, \sigma$):
- Parse $(y, \Lambda, r, \pi) = \sigma$
- If $\boldsymbol{s} \notin (\mathbb{Z}_q^*)^t$, output 0
- $R_\tau \leftarrow \prod_{i \in [t]} \mathsf{H}(\tau_i)^{s_i}$
- If $(\pi_1 = \emptyset)$ then:
        - if $\left(e(g_1^y B^r R_\tau, g_2) = e(\Lambda, \Gamma_2)\right)$ output 1, else output 0
- If $(\mathsf{NIZK}.V(\mathsf{R_c}, \mathsf{srs}, c, \pi)) \wedge (e(yB^r R_\tau, g_2) = e(\Lambda, \Gamma_2))$ output 1,
      else output 0
```

Correctness and succinctness As detailed next, $\mathsf{LHS_{Ped}}$ satisfies perfect authentication correctness, and (if NIZK is perfectly complete) perfect evaluation correctness. As evaluated signatures are of constant size, $\mathsf{LHS_{Ped}}$ is succinct.

Consider $(\mathsf{sk} := (a, b, \kappa), \mathsf{vk}) \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{L})$, where $N := |\mathcal{L}|$, and $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs}, \Gamma_1, \Gamma_2, B, \mathsf{H}, \mathcal{R}^N_{\text{com-ip}})$ specifies $h \in \mathbb{G}_1$, $\Gamma_1 := h^a$, $\Gamma_2 := g_2^{1/a}$, and $B := g_1^b$.

Perfect authentication correctness For any $\tau \in \mathcal{L}$, and $x \in \mathbb{Z}_q$, let $\sigma := (x, \Lambda, r, \emptyset)$ denote the output of $\mathsf{Sign}(\mathsf{sk}, \tau, x)$. Then it holds that $r = \mathsf{F}_\kappa(\tau)$, and $\Lambda = (\mathsf{H}(\tau) g_1^{x+br})^a = (\mathsf{H}(\tau) g_1^x B^r)^a$. Hence $e(g_1^x B^r \mathsf{H}(\tau), g_2) = e((g_2^x g_1^{br} \mathsf{H}(\tau))^a, g_2^{1/a}) = e(\Lambda, \Gamma_2)$, and $\mathsf{Ver}(\mathsf{vk}, \mathcal{I}_\tau, x, \sigma) = 1$.

Perfect evaluation correctness Consider any $t \in [N]$ and $\boldsymbol{s} \in \mathbb{Z}_q^t$, and any set of label/message/signature triples $\{\tau_i, x_i, \sigma_i\}_{i=1}^t$, where $\sigma_i := (x_i, \Lambda_i, r_i, \emptyset)$. For these to be valid signatures, it must hold that for $i \in [t]$:

$$e(g_1^{x_i} B^{r_i} \mathsf{H}(\tau_i), g_2) = e(\Lambda_i, g_2^{1/a}).$$

Let $z := \sum_{i=1}^t x_i s_i$, $r := \sum_{i=1}^t r_i s_i$, and, for any $w \in \mathbb{Z}_q$ let $\sigma := (y, \Lambda, r, \pi) \leftarrow \mathsf{Eval}(\mathsf{vk}, \mathsf{R}^{(t,s)}_{\text{com-ip}}, \sigma_1, \ldots, \sigma_t, w)$. In particular, $y := g_1^z h^w$, $\Lambda := (h^a)^w \prod_{i=1}^t (\Lambda_i)^{s_i}$, and $\pi := P(\mathsf{R_c}, \mathsf{srs}, y, (z, w))$.

By correctness of the proof system it holds that $V(\mathsf{R_c}, \mathsf{srs}, y, \pi) = 1$, and, denoting $R_\tau := \prod_{i=1}^t \mathsf{H}(\tau_i)^{s_i}$,

$$\begin{aligned} e\left(yB^r \prod_{i=1}^{t} \mathsf{H}(\tau_i)^{s_i}, g_2\right) &= e\left(h^w \prod_{i=1}^{t} \left(g_1^{x_i} B^{r_i} \mathsf{H}(\tau_i)\right)^{s_i}, g_2\right) \\ &= e\left((h^a)^w \prod_{i \in [t]} \Lambda_i^{s_i}, g_2^{1/a}\right) = e\left(\Lambda, \Gamma_2\right). \end{aligned}$$

And so $\mathsf{Ver}(\mathsf{vk}, (\mathsf{R}^{(t,s)}_{\text{com-ip}}, \tau_1, \ldots, \tau_t), y, \sigma) = 1$.

5.3 Security of $\mathsf{LHS_{Ped}}$

The scheme's adaptive security relies on the following assumption.

Assumption 1 (SCDH). *The square computational Diffie-Hellman assumption holds for the asymmetric group generator $\mathcal{G}$ if for any PT adversary $\mathcal{A}$, and for all large enough λ:*

$$\Pr\Big[\mathsf{pp}_{\mathcal{G}} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda), g_1 \leftarrow\!\$\, \mathbb{G}_1, g_2 \leftarrow\!\$\, \mathbb{G}_2, \\ a \leftarrow\!\$\, \mathbb{Z}_q,\ g_1^{a^2} \leftarrow \mathcal{A}(\mathsf{pp}_{\mathcal{G}}, g_1, g_1^a, g_2, g_2^a)\Big] = \mathsf{negl}(\lambda).$$

Theorem 5.1. *If* NIZK *is a proof of knowledge for* $\mathsf{R_c}$, H *is modelled as a random oracle, and* F *is a PRF, then the ComLHS scheme* $\mathsf{LHS_{Ped}}$ *described above is adaptively secure under the SCDH assumption for group generator $\mathcal{G}$. Furthermore, if* NIZK *satisfies zero-knowledge, then* $\mathsf{LHS_{Ped}}$ *is zero-knowledge.*

Remark 1. *The relation $\mathcal{R}_{\text{com-ip}}$ is defined for $\boldsymbol{s} \in (\mathbb{Z}_q^*)^t$ where each coordinate is $\not\equiv 0 \bmod q$. We add this restriction to achieve the strongest notion of unforgeability. It states that a signature, verifying for labels which were not queried during the signature queries phase, is considered a forgery. We could avoid this restriction, by slightly sacrificing performance, via a change in our scheme. However, we preferred to achieve the best efficiency, since working with the restriction $\boldsymbol{s} \in (\mathbb{Z}_q^*)^t$ is sufficient for our application: when the* HSNP *scheme of Section 4 calls upon* $\mathsf{LHS_{Ped}}$, *the coordinates of $\boldsymbol{s}$ are random in $\mathbb{Z}_q$, and thus non-zero with overwhelming probability.*

5.3.1 Adaptive security.

Proof. Consider an adversary $\mathcal{A}$ for the adaptive security of $\mathsf{LHS_{Ped}}$, which can forge signatures with probability ϵ. We devise an algorithm $\mathcal{B}$ which uses $\mathcal{A}$ to solve an SCDH challenge with the same success probability as $\mathcal{A}$.

Key generation: Algorithm $\mathcal{B}$ gets as input $(f_1, f_1^a, f_2, f_2^a) \in \mathbb{G}_1^2 \times \mathbb{G}_2^2$ for some unknown integer a. It sets $g_1 := f_1^a$, $g_2 := f_2^a$, $\Gamma_2 := f_2$, samples $\alpha, b \leftarrow\!\$\, \mathbb{Z}_q$ uniformly at random and sets $h := f_1^\alpha$, $\Gamma_1 := (f_1^a)^\alpha$ and $B := (f_1^a)^b$. As in the experiment $\mathsf{UF}_{\mathsf{Rg}, \mathcal{Z}, \mathcal{A}, \mathcal{E}}$, $\mathcal{B}$ initialises an empty list T to keep track of $\mathcal{A}$'s signing queries. Next, denoting $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ the knowledge extractor for the proof of knowledge NIZK, algorithm $\mathcal{B}$ runs $(\mathsf{srs}, \xi) \leftarrow \mathcal{E}_1(1^\lambda)$. The resulting verification key sent to $\mathcal{A}$ is $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs}, \Gamma_1, \Gamma_2, B)$. Due to the *perfect knowledge extraction* property of NIZK, vk follows a distribution which is identical to that of keys produced by the real key generation algorithm.

Simulating the random oracle: Algorithm $\mathcal{B}$ initialises an empty table T_H. Whenever $\mathcal{A}$ queries H on input τ, $\mathcal{B}$ checks if there exists an entry in T_H for τ. If not, it samples $\gamma_\tau, \bar{x}_\tau \leftarrow\!\$\, \mathbb{Z}_q$ uniformly at random, sets $T_\mathsf{H}[\tau] := (\gamma_\tau, \bar{x}_\tau, R_\tau := g_1^{-\bar{x}_\tau} f_1^{\gamma_\tau})$, and sends R_τ to $\mathcal{A}$. If an entry already exists, $\mathcal{B}$ sends $\mathcal{A}$ the previously stored R_τ.

Signing queries. When $\mathcal{A}$ requests a signature for (τ, x_τ), $\mathcal{B}$ first checks in T if this pair was previously queried, in which case it returns the same signature as before; or if τ was previously queried for a different message, in which case it ignores the query (as in the real unforgeability experiment).

Now if it is the first time a signing query is made with label τ, $\mathcal{B}$ checks if $T_\mathsf{H}[\tau]$ exists. If not, it creates an entry as detailed above; denote it $(\gamma_\tau, \bar{x}_\tau, R_\tau)$. Then $\mathcal{B}$ computes $r := (\bar{x}_\tau - x_\tau) b^{-1} \bmod q$ (so that $\bar{x}_\tau = x_\tau + br$); and $\Lambda := g_1^{\gamma_\tau}$. It sends $\sigma := (x_\tau, \Lambda, r, \emptyset)$ to $\mathcal{A}$ and adds (τ, x_τ, σ) to T. Observe that $\Lambda = (R_\tau g_1^{x_i + b \cdot r})^a$, as expected by $\mathcal{A}$. Furthermore, since $\gamma_\tau, \bar{x}_\tau$ are sampled uniformly at random in $\mathbb{Z}_q$, the distribution of r from $\mathcal{A}$'s view is also uniformly random in $\mathbb{Z}_q$. Hence, due to the pseudo-randomness of F, σ follows a distribution which is indistinguishable from that of signatures produced by the real signing algorithm.

Forgery. After a polynomial number of signing queries, $\mathcal{A}$ outputs $((\mathsf{R}, \tau_1 \ldots, \tau_t), \sigma, y)$, where $\sigma := (y, \Lambda, r, \pi)$, and R specifies an integer $t \in [N]$ and a vector $\boldsymbol{s} \in (\mathbb{Z}_q^*)^t$. Since, from $\mathcal{A}$'s view, its interactions with $\mathcal{B}$ are indistinguishable from a real execution of experiment $\mathsf{UF}_{\mathsf{Rg},\mathcal{Z},\mathcal{A},\mathcal{E}}$, with probability negligibly close to ϵ, this output is a forgery for $\mathsf{LHS}_{\mathsf{Ped}}$. This implies that $\mathsf{VerSig}(\mathsf{vk}, (\mathsf{R}, \tau_1, \ldots, \tau_t), y, \sigma) = 1$, and that either (1) $\mathcal{A}$ has not previously queried signatures for all labels $(\tau_1, \ldots, \tau_t)$, or (2) there exist answered queries (τ_i, x_i, σ_i) for each $i \in [t]$, but one cannot efficiently extract, from $\mathcal{A}$'s view, a witness w satisfying $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$.

Let us first consider case (1). Note that if any τ_i for $i \in [t]$ has not been queried to H, then $\mathsf{H}(\tau_i)$ can take any value in $\mathbb{Z}_q$ with equal probability $1/q$. Since all coordinates of $\boldsymbol{s}$ are non zero modulo q, due to the pairing checks in the verification algorithm, $1/q$ would also upper bound the (negligible) probability that VerSig returns 1. We hence assume that for all $i \in [t]$, there exist $\gamma_i, \bar{x}_i, R_i$ such that $T_{\mathsf{H}}[\tau_i] = (\gamma_i, \bar{x}_i, R_i)$. Let $\boldsymbol{\gamma} := (\gamma_1, \ldots, \gamma_t)$ and $\bar{\boldsymbol{x}} := (\bar{x}_1, \ldots, \bar{x}_t)$.

If, in $\mathcal{A}$'s forgery, $\pi = \emptyset$, then $t = 1$ and it holds that

$$e(g_1^y B^r f_1^{-a\bar{x}_1+\gamma_1}, g_2) = e(\Lambda, g_2^{1/a})$$

$$\text{and so} \quad f_1^{ay} f_1^{abr} f_1^{-a\bar{x}_1+\gamma_1} = \Lambda^{1/a}$$

$$\Leftrightarrow f_1^{a^2(y+br-\bar{x}_1)} f_1^{a\gamma_1} = \Lambda \Leftrightarrow (f_1^{a^2})^{y+br-\bar{x}_1} = \Lambda \cdot g_1^{-\gamma_1}.$$

Since $\bar{x}_1$ is sampled uniformly at random modulo q, it is non-zero with overwhelming probability, and hence under the hardness of computing discrete logarithms, $y + br - \bar{x}_1 \neq 0 \bmod q$ (otherwise $\mathcal{A}$ could compute b). Hence $\mathcal{B}$ solves its SCDH challenge by outputting:

$$f_1^{a^2} = \left(\Lambda \cdot g_1^{-\gamma_1}\right)^{(y+br-\bar{x}_1)^{-1}}.$$

If $\pi \neq \emptyset$, then it holds that $\mathsf{NIZK}.V(\mathsf{R_c}, \mathsf{srs}, y, \pi) = 1$, so $\mathcal{B}$ runs $(z_{\mathcal{E}}, w_{\mathcal{E}}) \leftarrow \mathcal{E}_2(\mathsf{srs}, \xi, y, \pi)$, thus obtaining $(z_{\mathcal{E}}, w_{\mathcal{E}}) \in \mathsf{R_c}$, i.e., satisfying $y = g_1^{z_{\mathcal{E}}} h^{w_{\mathcal{E}}} = f_1^{az_{\mathcal{E}}+\alpha w_{\mathcal{E}}}$.

It also holds that:

$$e\left(yB^r \prod_{i\in[t]} (f_1^{-a\bar{x}_i+\gamma_i})^{s_i}, g_2\right) = e\left(\Lambda^{1/a}, g_2\right)$$

$$\Leftrightarrow \quad f_1^{az_{\mathcal{E}}+\alpha w_{\mathcal{E}}} f_1^{abr} f_1^{\sum_{i\in[t]} s_i(-a\bar{x}_i+\gamma_i)} = \Lambda^{1/a}$$

$$\Leftrightarrow \quad f_1^{a^2(z_{\mathcal{E}}+br-\langle\bar{\boldsymbol{x}},\boldsymbol{s}\rangle)+a(\alpha w_{\mathcal{E}}+\langle\boldsymbol{\gamma},\boldsymbol{s}\rangle)} = \Lambda$$

$$\Leftrightarrow \quad (f_1^{a^2})^{(z_{\mathcal{E}}+br-\langle\bar{\boldsymbol{x}},\boldsymbol{s}\rangle)} = \Lambda \cdot g_1^{-(\alpha w_{\mathcal{E}}+\langle\boldsymbol{\gamma},\boldsymbol{s}\rangle)}.$$

Finally, recall that for $i \in [t]$, the $\bar{x}_i$ and γ_i are sampled uniformly at random from $\mathbb{Z}_q$. As we assume $\mathcal{A}$ knows $R_i = f_1^{\gamma_i-a\bar{x}_i}$, the value of $(\gamma_i - a\bar{x}_i \bmod q)$ is information theoretically fixed from $\mathcal{A}$'s view, but $\bar{x}_i$ can still take any value in $\mathbb{Z}_q$ with equal probability. Now for every label τ_i for which $\mathcal{A}$ has queried a signature, the values of $\bar{x}_i$ and γ_i are fixed modulo q from $\mathcal{A}$'s view, as it is also granted $\Lambda_i = g_1^{\gamma_i}$. However we know that there exists at least one label for which $\mathcal{A}$ has not queried a signature, and since, for $i \in [t]$, $s_i \neq 0 \bmod q$, it holds that $\langle\bar{\boldsymbol{x}}, \boldsymbol{s}\rangle \neq z_{\mathcal{E}} \bmod q$ with all but negligible probability. So once again, under the assumption that computing discrete logarithms is hard, $z_{\mathcal{E}} + br - \langle\bar{\boldsymbol{x}}, \boldsymbol{s}\rangle$ is invertible modulo q, and $\mathcal{B}$ can solve its SCDH challenge by outputting:

$$f_1^{a^2} = (\Lambda \cdot g_1^{-(\alpha w_{\mathcal{E}}+\langle\boldsymbol{\gamma},\boldsymbol{s}\rangle)})^{(z_{\mathcal{E}}+br-\langle\bar{\boldsymbol{x}},\boldsymbol{s}\rangle)^{-1}}.$$

Let us now consider case (2): there exist $(\tau_i, x_i, \sigma_i) \in T$ for each $i \in [t]$, but one cannot efficiently extract, from $\mathcal{A}$'s view, a witness w satisfying $(y, (x_1, \ldots, x_t), w) \in \mathsf{R}$, i.e., such that $y = g_1^{\langle\boldsymbol{x},\boldsymbol{s}\rangle} h^w$. As before, we denote, for $i \in [t]$, $T_{\mathsf{H}}[\tau_i] = (\gamma_i, \bar{x}_i, R_i)$, and $\sigma_i := (x_i, \Lambda_i, r_i, \emptyset)$. The case $\pi = \emptyset$ is identical to that of case (1). Now if $\pi \neq \emptyset$, once again $\mathcal{B}$ runs $(z_{\mathcal{E}}, w_{\mathcal{E}}) \leftarrow \mathcal{E}_2(\mathsf{srs}, \xi, y, \pi)$, thus obtaining $(z_{\mathcal{E}}, w_{\mathcal{E}}) \in \mathsf{R_c}$, which satisfy $y = g_1^{z_{\mathcal{E}}} h^{w_{\mathcal{E}}} = f_1^{az_{\mathcal{E}}+\alpha w_{\mathcal{E}}}$. However, since $(y, (x_1, \ldots, x_t), w_{\mathcal{E}}) \notin \mathsf{R}$ (otherwise $\mathcal{B}$ would have efficiently extracted a witness), it must be that $\langle\boldsymbol{x}, \boldsymbol{s}\rangle \neq z_{\mathcal{E}} \bmod q$.

From the pairing check, it holds that:

$$yB^r \prod_{i\in[t]} R_i^{s_i} = \Lambda^{1/a}$$

$$\Leftrightarrow \quad f_1^{az_{\mathcal{E}}+\alpha w_{\mathcal{E}}+abr} \prod_{i\in[t]} (f_1^{-a(x_i+br_i)+\gamma_i})^{s_i} = \Lambda^{1/a}$$

$$\Leftrightarrow \quad f_1^{a^2(z_{\mathcal{E}}+br-\langle\boldsymbol{x},\boldsymbol{s}\rangle-b\langle\boldsymbol{r},\boldsymbol{s}\rangle)+a(\alpha w_{\mathcal{E}}+\langle\boldsymbol{\gamma},\boldsymbol{s}\rangle)} = \Lambda$$

$$\Leftrightarrow \quad (f_1^{a^2})^{z_{\mathcal{E}}-\langle\boldsymbol{x},\boldsymbol{s}\rangle+b(r-\langle\boldsymbol{r},\boldsymbol{s}\rangle)} = \Lambda \cdot g_1^{-(\alpha w_{\mathcal{E}}+\langle\boldsymbol{\gamma},\boldsymbol{s}\rangle)}.$$

Now if $r = \langle\boldsymbol{r}, \boldsymbol{s}\rangle \bmod q$, since $\langle\boldsymbol{x}, \boldsymbol{s}\rangle \neq z_{\mathcal{E}} \bmod q$, it holds that $z_{\mathcal{E}} - \langle\boldsymbol{x}, \boldsymbol{s}\rangle + b(r - \langle\boldsymbol{r}, \boldsymbol{s}\rangle)$ is invertible modulo q. And as before, it will also be invertible if $r \neq \langle\boldsymbol{r}, \boldsymbol{s}\rangle \bmod q$, since otherwise $\mathcal{A}$ could compute $b \bmod q$. So $\mathcal{B}$ solves its challenge by outputting:

$$f_1^{a^2} = \left(\Lambda \cdot g_1^{-(\alpha w_{\mathcal{E}}+\langle\boldsymbol{\gamma},\boldsymbol{s}\rangle)}\right)^{(z_{\mathcal{E}}-\langle\boldsymbol{x},\boldsymbol{s}\rangle+b(r-\langle\boldsymbol{r},\boldsymbol{s}\rangle))^{-1}}.$$

Hence, if NIZK is a proof of knowledge for $\mathsf{R_c}$, $\mathcal{B}$ solves its SCDH challenge with probability negligibly close to ϵ, which concludes the proof that, in the SCDH assumption, and in the random oracle model, the signature scheme is adaptively secure. □

5.3.2 *Zero knowledge.*

We here show that evaluated signatures guarantee the privacy of signed data.

Proof. Let $\mathcal{A}$ be a PT adversary for the zero-knowledge property of $\mathsf{LHS}_{\mathsf{Ped}}$. In the $\mathsf{ZK}_{\mathcal{A}}^{\mathsf{real}}$ experiment, the challenger runs $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^\lambda, \mathcal{L}, \mathcal{R}_{\mathsf{com\text{-}ip}})$, and hands vk to $\mathcal{A}$.

During the signing queries phase, $\mathcal{A}$ adaptively sends queries (τ, x) of its choosing to the challenger, who honestly computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, x)$, and sends σ to $\mathcal{A}$.

In the choose phase $\mathcal{A}$ outputs $(y, (\mathsf{R}_{\mathsf{com\text{-}ip}}^{(t,s)}, \tau_1, \ldots, \tau_t), w)$. Note that if this output causes either the real or simulated experiment to output the error symbol $\perp$, then both experiments do, and are identical from $\mathcal{A}$'s view. We hereafter assume this does not occur, so for $i \in [t]$, there exist $\sigma_i := (x_i, \Lambda_i, r_i, \emptyset)$ such that $(\tau_i, x_i, \sigma_i) \in T$, and $(y, (\mathsf{R}_{\mathsf{com\text{-}ip}}^{(t,s)}, \tau_1, \ldots, \tau_t), w) \in \mathsf{R}_{\mathsf{com\text{-}ip}}^{(t,s)}$.

We proceed via a sequence of games, depicted in Figure 5. Game 0 is experiment $\mathsf{ZK}_{\mathcal{A}}^{\mathsf{real}}$, and the final game (Game 2) is experiment $\mathsf{ZK}_{\mathcal{A},\mathcal{S}}^{\mathsf{sim}}$. In all games, the signing queries and choose phases proceed as described above (i.e. as in experiment $\mathsf{ZK}_{\mathcal{A}}^{\mathsf{real}}$, *cf.* Figure 2a).

We denote by S_i the event $\mathcal{A}$ outputs $b' = 1$ in Game i. Hence $\Pr[S_0] = \Pr[\mathsf{ZK}_{\mathcal{A}}^{\mathsf{real}} = 1]$, and $\Pr[S_2] = \Pr[\mathsf{ZK}_{\mathcal{A},\mathcal{S}}^{\mathsf{sim}} = 1]$. By proving that, from $\mathcal{A}$'s view, each game is indistinguishable from the next, we demonstrate that $\mathcal{A}$ cannot, with significant probability, output a different bit b in experiments $\mathsf{ZK}_{\mathcal{A}}^{\mathsf{real}}$ and $\mathsf{ZK}_{\mathcal{A},\mathcal{S}}^{\mathsf{sim}}$. Changes from one game to the next are highlighted for improved visibility.

Game 0 to Game 1 Since NIZK is zero-knowledge, there exists a simulator $\mathcal{S}^{\mathsf{NIZK}} = (\mathcal{S}^{\mathsf{NIZK}}_{\mathsf{kg}}, \mathcal{S}^{\mathsf{NIZK}}_{\mathsf{prv}})$ for the ZK property of the scheme. Here we substitute the key generation and proving algorithms of NIZK by algorithms $\mathcal{S}^{\mathsf{NIZK}}_{\mathsf{kg}}$ and $\mathcal{S}^{\mathsf{NIZK}}_{\mathsf{prv}}$ respectively. Note that the proof π no longer depends on the witness w provided by $\mathcal{A}$. By the zero-knowledge property of NIZK, both games are indistinguishable to $\mathcal{A}$, so:

$$|\Pr[S_0] - \Pr[S_1]| = \mathsf{negl}(\lambda). \tag{4}$$

Game 1 to Game 2 Instead of using the signatures $\sigma_1, \dots, \sigma_t$ to evaluate Λ, Game 2 uses the secret signing key (a, b, κ) and the adversarially chosen labelled relation $(\mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}, \tau_1, \dots, \tau_t)$. Precisely,

$$\Lambda := y^a \cdot \prod_{i\in[t]} (\mathsf{H}(\tau_i) \cdot B^{\mathsf{F}_\kappa(\tau_i)})^{a\cdot s_i}.$$

Note that though they are computed differently, the resulting values of Λ in Game 1 and Game 2 are identical, hence:

$$\Pr[S_2] = \Pr[S_1]. \tag{5}$$

One can now easily see that Game 2 is experiment $\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}}$, which calls upon the simulators $(\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{eval}})$ described in Figure 5a. This concludes the proof, since :

$$\left|\Pr[\mathsf{ZK}^{\mathsf{sim}}_{\mathcal{A},\mathcal{S}}(\lambda) = 1] - \Pr[\mathsf{ZK}^{\mathsf{real}}_{\mathcal{A}}(\lambda) = 1]\right| = |\Pr[S_2] - \Pr[S_0]|\,,$$

which, combining Equations (4) and (5), is proven to be negligible. □

5.4 Efficiency of $\mathsf{LHS}_{\mathsf{Ped}}$

The main cost for signing is a multi-exponentiation in $\mathbb{G}_1$, roughly $2\log(q)$ group operations in $\mathbb{G}_1$. A signature consists of one group element in $\mathbb{G}_1$, and two integers modulo q.

The time to evaluate a signature for a statement $y \in \mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}$ for $t \in \mathbb{N}$, $\boldsymbol{s} \in \mathbb{Z}_q^t$ is dominated by that of the multi-exponentiation in $\mathbb{G}_1$ computing Λ, which involves $t+1$ terms. Using Pippenger's algorithm, this requires $\approx t\log(q)/\log(t\log(q))$ group operations. An evaluated signature consists of three group elements in $\mathbb{G}_1$ (y, Λ, and one in π), and three integers modulo q (due to r and π).

The time to verify a signature is dominated by that of the multi-exponentiation computing R_τ, involving t terms, which requires $\approx t\log(q)/\log(t\log(q))$ group operations.

6 EFFICIENT INSTANTIATION AND EVALUATION

We here show how to efficiently instantiate the building blocks of our HSNP scheme from Section 4. We then analyse the efficiency of the resulting construction (that we call SPHINX), and compare it with that of the most promising contender to our solution, based on digital signatures and zkSNARKs.

6.1 Building blocks

Our building blocks are instantiated over bilinear groups $\mathsf{pp}_{\mathcal{G}} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ of order q; hence $\mathbb{F} = \mathbb{Z}_q$.

Commitment scheme Com. For Com we use the KZG polynomial commitment [40] in which the commitment c_p to a univariate polynomial $p(X)$ is $g_1^{p(s)} h_1^{o_p}$ for a random $o_p \leftarrow\$\, \mathbb{F}$. Here s is a random secret point chosen in Setup such that $\mathsf{ck} := (\{g_1^{s^i}, h_1^{s^i}\}_{i=0}^{D}, g_2^s)$ for some degree bound D. In our instantiation $D \geqslant N$.[5] To commit to a scalar $z \in \mathbb{F}$ we use a Pedersen commitment with the same key, i.e., $\mathsf{c}_z = g_1^z h_1^{o_z}$. The Com scheme is binding under the power discrete logarithm assumption [43].

Universal CP-SNARK $\mathsf{CP}_{\mathcal{R}}$**.** We instantiate this CP-SNARK with a commit-and-prove version of the Marlin universal zkSNARK [21] that we propose. This CP-SNARK, denoted $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$, is detailed in Appendix A and is knowledge-sound under the SDH assumption [8] in the algebraic group model [30]. We here give a brief overview of it and state its efficiency.

Marlin is a zkSNARK for the NP-complete language of rank-1 constraint systems (R1CS) [33]. In R1CS a vector $\boldsymbol{y}$ is in the language, defined by matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, if there exists a vector $\boldsymbol{w}'$ such that, for $\boldsymbol{z} = (1, \boldsymbol{y}, \boldsymbol{w}')$, it holds that $\mathbf{A}\boldsymbol{z} \circ \mathbf{B}\boldsymbol{z} = \mathbf{C}\boldsymbol{z}$. Our $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$ turns Marlin into a commit-and-prove SNARK in the sense that, given a commitment c_x and a public $\boldsymbol{y}$, $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$ proves that the R1CS relation holds for $\boldsymbol{z} = (1, \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w})$ where c_x opens to a polynomial $\tilde{x}(X)$ that, for a public subset $\mathbb{T} = \{h_1, \dots, h_t\}$, satisfies $\tilde{x}(h_i) = x_i$. More precisely, in Marlin, every R1CS relation is associated to a multiplicative subgroup $\mathbb{H} \subset \mathbb{F}$ of cardinality $|z|$. Considering a canonical ordering of $\mathbb{H} = \{\eta, \eta^2, \dots, \eta^{|\mathbb{H}|}\}$, we define the set $\mathbb{T} = \{\eta^{|\boldsymbol{y}|+1}, \dots, \eta^{|\boldsymbol{y}|+t}\}$ and the polynomial $\chi_i(X) = \frac{h_i(X^{|\mathbb{H}|}-1)}{|\mathbb{H}|(X-h_i)}$. Namely, $\chi_i(X)$ is the Lagrange-basis polynomial over the domain $\mathbb{H}$ corresponding to the point h_i. This allows one to verify that $\boldsymbol{\chi}$ satisfies the properties (a)–(b)–(c).

Let us provide intuition on how we turn Marlin into a CP-SNARK. Notably, although any zkSNARK can be turned into a CP one by encoding the commitment verification in the R1CS relation (which can be concretely expensive, see [14]), we use an alternative technique to do this transformation efficiently. We start from the observation that a proof in Marlin includes a commitment to a polynomial $\hat{w}'(X)$ that interpolates the vector $\boldsymbol{w}'$ over a subset of points, such that the first t of them are exactly those in $\mathbb{T}$. So our $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$ augments Marlin by adding a proof that $\boldsymbol{w}' = (\boldsymbol{x}, \boldsymbol{w})$ for the $\boldsymbol{x}$ committed (as a polynomial) in c_x. The latter proof is built by using a CP-SNARK for this specific "prefix" relation.

Of course, compared to Marlin, $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$ has some additional costs. The proof of $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$ includes three more elements in $\mathbb{G}_1$ and one element in $\mathbb{F}$. To generate a proof about a statement $(\boldsymbol{y}, (\boldsymbol{x}, \boldsymbol{w})) \in \mathsf{R}$, the prover of $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$ performs one extra multiexponentiation of length $|w|$ and two of length t (as well as a few constant exponentiations). $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$'s verifier needs to compute 4 more pairings. Hence, the only non-negligible overhead are the $|\boldsymbol{w}|+2t$ exponentiations for the prover. However, we observe that in comparison to the remaining cost for generating a proof in Marlin, proof generation in $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$ is only $\approx 10\%$ slower.

[5] For the plain commitment, $D = N$ would be enough, and we wouldn't need the $h_1^{s^j}$ for $j \geqslant 1$. The additional elements are for using this same commitment to instantiate Marlin (see next building block).

```
Key generation
  - $N := |\mathcal{L}|$; $\mathsf{pp}_{\mathcal{G}} \leftarrow \mathcal{G}(1^{\lambda})$; $h \leftarrow\$ \mathbb{G}_1$;
  - $\mathsf{ck} := (\mathsf{pp}_{\mathcal{G}}, h)$; $\mathsf{srs} \leftarrow \mathsf{K}(1^{\lambda}, \mathsf{ck})$
  - $\kappa \leftarrow\$ \mathcal{K}$
  - $a, b \leftarrow\$ \mathbb{Z}_q$
  - $\Gamma_1 \leftarrow h^{a}$; $\Gamma_2 \leftarrow g_2^{1/a}$; $B \leftarrow g_1^{b}$
  - $\mathsf{sk} := (a, b, \kappa)$; $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs}, \Gamma_1, \Gamma_2, B, \mathsf{H})$
  - send vk to $\mathcal{A}$.

Challenge
  - For $i \in [t]$, $\sigma_i := (x_i, \Lambda_i, r_i, \emptyset) \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau_i, x_i)$
  - $z \leftarrow \langle \boldsymbol{x}, \boldsymbol{s} \rangle$; $r \leftarrow \langle \boldsymbol{r}, \boldsymbol{s} \rangle$
  - $\pi \leftarrow \mathsf{NIZK}.P(\mathsf{R_c}, \mathsf{srs}, y, (z, w))$
  - $\Lambda \leftarrow \Gamma_1^{w} \cdot \prod_{i \in [t]} (\Lambda_i)^{s_i}$
  - send $\sigma := (y, \Lambda, r, \pi)$ to $\mathcal{A}$
  - $\mathcal{A}$ outputs $b'$
```

Game 0

```
Key generation
  - $N := |\mathcal{L}|$; $\mathsf{pp}_{\mathcal{G}} \leftarrow \mathcal{G}(1^{\lambda})$; $h \leftarrow\$ \mathbb{G}_1$;
  - $\mathsf{ck} := (\mathsf{pp}_{\mathcal{G}}, h)$; $(\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{S}^{\mathsf{NIZK}}_{\mathsf{kg}}(1^{\lambda}, \mathsf{ck})$
  - $\kappa \leftarrow\$ \mathcal{K}$
  - $a, b \leftarrow\$ \mathbb{Z}_q$
  - $\Gamma_1 \leftarrow h^{a}$; $\Gamma_2 \leftarrow g_2^{1/a}$; $B \leftarrow g_1^{b}$
  - $\mathsf{sk} := (a, b, \kappa, \mathsf{td})$; $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs}, \Gamma_1, \Gamma_2, B, \mathsf{H})$
  - send vk to $\mathcal{A}$.

Challenge
  - For $i \in [t]$, $\sigma_i := (x_i, \Lambda_i, r_i, \emptyset) \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau_i, x_i)$
  - $z \leftarrow \langle \boldsymbol{x}, \boldsymbol{s} \rangle$; $r \leftarrow \langle \boldsymbol{r}, \boldsymbol{s} \rangle$
  - $\pi \leftarrow \mathcal{S}^{\mathsf{NIZK}}_{\mathsf{prv}}(\mathsf{R_c}, \mathsf{srs}, \mathsf{td}, y)$
  - $\Lambda \leftarrow \Gamma_1^{w} \cdot \prod_{i \in [t]} (\Lambda_i)^{s_i}$
  - send $\sigma := (y, \Lambda, r, \pi)$ to $\mathcal{A}$
  - $\mathcal{A}$ outputs $b'$
```

Game 1

```
Key generation
  - $N := |\mathcal{L}|$; $\mathsf{pp}_{\mathcal{G}} \leftarrow \mathcal{G}(1^{\lambda})$; $h \leftarrow\$ \mathbb{G}_1$;
  - $\mathsf{ck} := (\mathsf{pp}_{\mathcal{G}}, h)$; $(\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{S}^{\mathsf{NIZK}}_{\mathsf{kg}}(1^{\lambda}, \mathsf{ck})$
  - $\kappa \leftarrow\$ \mathcal{K}$
  - $a, b \leftarrow\$ \mathbb{Z}_q$
  - $\Gamma_1 \leftarrow h^{a}$; $\Gamma_2 \leftarrow g_2^{1/a}$; $B \leftarrow g_1^{b}$
  - $\mathsf{sk} := (a, b, \kappa)$; $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs}, \Gamma_1, \Gamma_2, B, \mathsf{H})$
  - send vk to $\mathcal{A}$.

Challenge
  - For $i \in [t]$: $r_i \leftarrow \mathsf{F}_{\kappa}(\tau_i)$
  - $r \leftarrow \langle \boldsymbol{r}, \boldsymbol{s} \rangle$
  - $\pi \leftarrow \mathcal{S}^{\mathsf{NIZK}}_{\mathsf{prv}}(\mathsf{R_c}, \mathsf{srs}, \mathsf{td}, y)$
  - $\Lambda \leftarrow y^{a} \cdot B^{a \cdot r} \prod_{i \in [t]} (\mathsf{H}(\tau_i))^{a \cdot s_i}$
  - send $\sigma := (y, \Lambda, r, \pi)$ to $\mathcal{A}$
  - $\mathcal{A}$ outputs $b'$
```

Game 2

```
$\mathcal{S}_{\mathsf{kg}}(1^{\lambda}, \mathcal{L}, \mathcal{R}_{\mathsf{com\text{-}ip}})$
  - $N := |\mathcal{L}|$; $\mathsf{pp}_{\mathcal{G}} \leftarrow \mathcal{G}(1^{\lambda})$; $h \leftarrow\$ \mathbb{G}_1$;
  - $\mathsf{ck} := (\mathsf{pp}_{\mathcal{G}}, h)$; $(\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{S}^{\mathsf{NIZK}}_{\mathsf{kg}}(1^{\lambda}, \mathsf{ck})$
  - $\kappa \leftarrow\$ \mathcal{K}$
  - $a, b \leftarrow\$ \mathbb{Z}_q$
  - $\Gamma_1 \leftarrow h^{a}$; $\Gamma_2 \leftarrow g_2^{1/a}$; $B \leftarrow g_1^{b}$
  - $\mathsf{sk} := (a, b, \kappa)$; $\mathsf{vk} := (\mathsf{ck}, \mathsf{srs}, \Gamma_1, \Gamma_2, B, \mathsf{H})$
  - output $(\mathsf{sk}, \mathsf{vk})$.

$\mathcal{S}_{\mathsf{eval}}(\mathsf{vk}, \mathsf{sk}, (\mathsf{R}^{(t,s)}_{\mathsf{com\text{-}ip}}, \tau_1, \ldots, \tau_t), y)$
  - For $i \in [t]$: $r_i \leftarrow \mathsf{F}_{\kappa}(\tau_i)$
  - $r \leftarrow \langle \boldsymbol{r}, \boldsymbol{s} \rangle$
  - $\pi \leftarrow \mathcal{S}^{\mathsf{NIZK}}_{\mathsf{prv}}(\mathsf{R_c}, \mathsf{srs}, \mathsf{td}, y)$
  - $\Lambda \leftarrow y^{a} \cdot B^{a \cdot r} \prod_{i \in [t]} (\mathsf{H}(\tau_i))^{a \cdot s_i}$
  - output $\sigma := (y, \Lambda, r, \pi)$
```

(a) Description of simulators $\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{eval}}$ for $\mathsf{LHS}_{\mathsf{Ped}}$.

Figure 5: Game steps and simulators proving zero-knowledge of $\mathsf{LHS}_{\mathsf{Ped}}$.

CP-SNARK $\mathsf{CP}_{\mathsf{ev}}$ We instantiate this CP-SNARK using a commit-and-prove version of the KZG polynomial evaluation argument from [27], which we describe in Appendix B and prove knowledge-sound under the SDH assumption [8] in the AGM [30]. So as to ensure that scalars $z \in \mathbb{F}$ are indeed committed as $\mathsf{c}_z := g_1^{z} h_1^{o_z}$, where $o_z \in \mathbb{F}$ (as opposed to being a polynomial in $\mathbb{F}$), our instantiation of $\mathsf{CP}_{\mathsf{ev}}$ also includes a Schnorr PoK of $z, o_z \in \mathbb{F}$. The cost of generating the proof π_z is essentially one multi-exponentiation of length $deg(\chi_i(X)) = |\mathbb{H}|$, while verification only requires 3 pairings, and a few constant exponentiations.

ComLHS for $\mathcal{R}_{\mathsf{com\text{-}ip}}$ We use the scheme $\mathsf{LHS}_{\mathsf{Ped}}$ of Section 5. See Section 5.4 for a summary of its costs.

6.2 Theoretical comparison and evaluation

Consider a computation f, and corresponding relation R_f containing tuples $(y, \boldsymbol{x}, w)$ such that $y = f(x_1, \ldots, x_t, w)$, where $t = |\boldsymbol{x}|$. We analyse the cost of generating a proof asserting the existence of a vector of data items $\boldsymbol{x} \in \mathbb{F}^t$, of signatures $\{\sigma_i\}_{i \in [t]}$, and of $w \in \mathcal{D}_w$ satisfying $(y, \boldsymbol{x}, w) \in \mathsf{R}_f$ and, for $i \in [t]$, $\mathsf{VerSig}(\mathsf{vk}, x_i, \sigma_i) = 1$.

We compare our SPHinx HSNP instantiation to the most promising alternative of Table 1, combining a standard signature scheme with a zkSNARK. We recall that the Sig+SNARK solution uses a zkSNARK to prove both the correctness of the computation output y, and the knowledge of messages and signatures $(x_1, \sigma_1), \ldots, (x_t, \sigma_t)$ such that $y = f(x_1, \ldots, x_t)$, and each σ_i is a valid signature for x_i. For Sig+SNARK we consider an instantiation using Marlin; for brevity we call this solution SigMarlin.

For both solutions, the protocol's cost can be split into (1) that of proving correct evaluation of $y = f(x_1, \ldots, x_t)$, denoted $\mathsf{P}(|\boldsymbol{x}|+|w|)$, which is essentially the cost of running the universal (CP)zkSNARK (e.g. Marlin or $\mathsf{CP}^{\mathsf{Marlin}}_{\mathcal{R}}$); and (2) that of proving the input data was authenticated. Note that (1) is common to both SPHinx and SigMarlin. So we are interested in comparing them for (2).

In SigMarlin, (2) is the cost of running Marlin to prove that, for $i \in [t]$, $\mathsf{VerSig}(\mathsf{vk}, x_i, \sigma_i) = 1$. Let us denote $\mathsf{c}_{\mathsf{Ver}}$ the number of constraints in the associated R1CS instance. Since there are t such signatures to verify for a single computation, the extra cost for the prover will be $\approx t\, \mathsf{c}_{\mathsf{Ver}} \log(t\, \mathsf{c}_{\mathsf{Ver}})$, while verification requires an extra $\approx t \log(t\, \mathsf{c}_{\mathsf{Ver}})$ operations. As arguments in Marlin are constant size, the size of the proof is not affected.

On the other hand, in SPHinx, (2) consists of the costs of committing $\mathsf{c}_{\boldsymbol{x}}, \mathsf{c}_z, \mathsf{c}_{\mathsf{sum}}$, running $\mathsf{CP}_{\mathsf{svec}}$ and $\mathsf{CP}_{\mathsf{ev}}$, and evaluating and verifying two $\mathsf{LHS}_{\mathsf{Ped}}$ signatures. For q of 255 bits, and values of $t \geqslant 100$ (which we expect for most practical applications), these operations entail less than $2|\mathbb{H}| + 4t$ extra operations for the prover; less than $2t + O(1)$ operations for the verifier; and an increase in the proof size of 11 elements in $\mathbb{G}_1$ and 11 field elements.

Table 2 summarizes this theoretical comparison.

Solution	Prover	Verifier	Proof size
SPHinx	$\mathsf{P}(\vert\boldsymbol{x}\vert + \vert w\vert) + 4t + 2\vert\mathbb{H}\vert$	$3t + o(1)$	$24\vert\mathbb{G}_1\vert + 19\vert\mathbb{F}\vert$
SigMarlin	$\mathsf{P}(\vert\boldsymbol{x}\vert + \vert w\vert + \mathsf{c}_{\mathsf{Ver}} t)$	$t \log(t\, \mathsf{c}_{\mathsf{Ver}})$	$13\vert\mathbb{G}_1\vert + 8\vert\mathbb{F}\vert$

Table 2: Theoretical cost comparison of HSNP solutions.

Estimating $\mathsf{c}_{\mathsf{Ver}}$. To make the comparison in Table 2 more concrete, we consider various signature schemes, and estimate the number of constraints $\mathsf{c}_{\mathsf{Ver}}$ to express signature verification with R1CS.

For EdDSA (without the hash evaluation, which, if using SHA2-256, adds $\approx 79\,000$ constraints[6]) the cost is $\mathsf{c}_{\mathsf{Ver}} \approx 7\,000$, if implemented over a SNARK-friendly elliptic curve. [7] Note that a single fixed base-point (resp. variable-base) scalar multiplication requires ≈ 500 (resp $\approx 1\,500$) constraints (2 and 6 constraints respectively per scalar bit). Schemes such as ECDSA or EC-Schnorr have a comparable number of constraints, as they require similar operations.

To avoid the cost of encoding (elliptic-curve) group operations, one may be drawn to lattice-based schemes such as that of Lyubashevsky [44]. Here verification requires two matrix-vector multiplications. Due to the large sizes of involved matrices, and range proofs required for modular operations, the number of constraints will be far over 5 000.

Finally, modular arithmetic in SNARKs is expensive, hence both RSA and Schnorr signature verification will require well over 10 000 constraints. Indeed, the state of the art implementation (cf. xJsnark [41]) of an RSA-2048 exponentiation requires 90 800 constraints.

Given these observations, we generously assume that $\mathsf{c}_{\mathsf{Ver}} \approx 5\,000$. Plugging this value into the comparison of Table 2, one can appreciate the important efficiency gains we expect.

7 EXPERIMENTAL EVALUATION

7.1 Implementation

We implemented a library for homomorphic signature schemes for NP in Rust based on the arkworks[8] libraries. The code is available at https://github.com/dariofiore/hsnp. Pairing-friendly curves are instantiated with BLS12-381[12], a Barreto–Lynn–Scott curve of embedding degree 12, defined over a 381-bit prime field. Our library, hsnp, implements and combines together each of the building blocks of SPHinx except for Marlin (namely, referring to Appendix A.2, we implement $\mathsf{CP}_{\mathcal{R}}$ with $\mathsf{CP}_{\mathsf{svec}}$ but not Marlin, and benchmark Marlin separately using its publicly available implementation[9]).

We ran our experiments on a virtual machine running Debian GNU/Linux with 8 cores Xeon-Gold-6154 clocked at 3GHz and with 98 GB of RAM. All the reported timings correspond to the median of measurements over 10 executions.

7.2 Experimental set up

We evaluate the performance of our implementation (called SPHinx) on different benchmarks, comparing it with that of the solution outlined in Section 6.2 (SigMarlin). That is, using Marlin to both prove the given R1CS statement, and knowledge of signatures which verify for all data items used in the statement. To benchmark SPHinx, we run all the components of our HSNP protocol on the given R1CS instance, and add the resulting cost to that of running Marlin on this same instance. To benchmark SigMarlin, we increase the size of the given R1CS instance by adding 5 000 constraints per signed input; this is done to account for the cost of signature verification, as explained in Section 6.2. We stress that 5 000 is a very optimistic estimate as the state-of-the-art suggests it is at least twice this cost.

We consider various benchmarks aiming to measure the performance impact of signed inputs in computations of varying complexity. We start with a general benchmark, which takes a circuit of fixed size and measures the performance for varying numbers of signed inputs. The goal of this benchmark is to see the relative performance degradation of both solutions while the number of signed inputs increases. Next, we evaluate and compare the performance of SigMarlin and SPHinx when applied to realistic applications; we consider computations typical in data streams such as *sliding window statistics* (variance, histograms), and a computation that models the application of *predicting stock prices.*

Remark 2 (On the role of labels). *We recall that in the usage of HSNP for VCS from Section 3.2 the labelling mechanism allows expressing queries that first filter a portion of the stream, and then apply a computation on the filtered portion. In all our benchmarks by "signed inputs" we mean the signed values that are fed as input to the R1CS relation* after *filtering, and not all the signed values in the data stream. This property of labelling is important for the scaling of our VCS solution on large streams as the complexity of the proof system (notably the R1CS size) depends only on the size of the filtered portion of the stream. This is significant in concrete terms. For instance, we may have a stream comprising $N = 1$ billion of values, yet we may wish to compute the variance of the last $t = 1$ million of them, in which case the size of the R1CS encoding the variance depends only on t and not on N.*

7.3 Benchmarks

7.3.1 *General circuits.*

We fix the size of the R1CS to $2^{22} \approx 4M$ and consider a variable number of signed inputs ranging from 2^8 to 2^{20}. The goal here is to measure the performance impact of handling signed inputs in the different protocols.

7.3.2 *Sliding window statistics.*

In the sliding window model for data streams, a data owner streams data items to a server, and clients are interested in the result of computations on the "window" of the

[6] Using the Zokrates library

[7] EthSnarks [https://ethresear.ch/t/low-overhead-secret-single-leader-election/5994/9]

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

[9] https://github.com/arkworks-rs/marlin

last t items in the stream. This allows extracting useful information about the stream, such as classical data science statistical tools. We consider two such tools described hereafter. We assume data items and thresholds are represented as 32-bit integers, and denote by t the number of measurements in the window.

Variance. This is a common tool to quantify the spread of a data[10]. Computing the variance can be expressed by an R1CS with $t + 2$ constraints and $2t + 2$ variables.

Histograms. Histograms allow one to roughly assess the probability distribution of a given variable, by depicting the frequencies of observations occurring in certain ranges of values. Denoting k the number of prescribed intervals, constructing a histogram can be expressed by an R1CS with $36tk$ constraints, and $96tk$ variables.

We also consider a more involved application of sliding window statistics to real-time market data: that of model-driven prediction, i.e. *predicting stock prices*. This task should be done frequently in order to learn from recent price fluctuations and better predict future ones. We consider a client querying the server for a prediction of some company's stock price in 5 days time, requiring that the prediction model be based on data streamed over the past n days.

Multi-linear regression. A potent model for this task is multilinear regression (MLR). Let k be the number of indicators, used as additional features for the model, computed by the server. Computing both the coefficients of the prediction model, and the predicted value itself, can be expressed 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 (see appendix D for details). Choosing $k := 20$ results in $964n + 10\,186$ constraints, and $754n + 9\,744$ variables.

For these three computations, we run our experiments by synthesizing R1CSs of the given size. Typically, one can extract more precise information from the stream by increasing the number of items on which the statistic is computed, i.e., t in variance and histograms and n in MLR. Hence we evaluate the systems' scalability by running these benchmarks on growing values of t (resp. n). We used: $t \in [2^8, 2^{20}]$ for variance; $t \in [2^8, 2^{13}]$ and $k = 4$ intervals for histograms; $n \in [2^5, 2^{10}]$ and $k = 30$ indicators for MLR. Though $t = 2^{13}$ (resp. $n = 2^{10}$) may look small, note that they imply R1CS of size $\approx 3M$ for histograms with $k = 4$ (resp. $\approx 2M$ for MLR with $k = 30$).

7.4 Evaluation

7.4.1 Signing and proof size. SPHinx's signature generation takes $500\mu s$ and only 80 Bytes of space. Moreover, both solutions have constant-size proofs: SigMarlin's proof is 880 Bytes while SPHinx's proof is 1760 Bytes.

7.4.2 Proving and verification time. Figure 6 (left) shows the proving and verification times for the benchmark with fixed-size R1CS; Figure 6 (right) shows the results for the variance benchmark; Table 3 highlights a selection of measurements for the histograms and MLR benchmarks.

Our experiments show that SPHinx is much more scalable than SigMarlin w.r.t. prover's performance. In the variance benchmark, the difference is significant with SPHinx being between 544× and 1340× faster than SigMarlin. For histograms and MLR, SPHinx's prover is still 7×–20× faster than SigMarlin's.

Besides proving time, the other scalability limitation of SigMarlin is its memory consumption. This is due to the large R1CS sizes induced by encoding signature verifications. In none of our benchmarks we were able to execute SigMarlin on an instance with more than 2048 signed inputs due to excessive RAM consumption. In contrast, we find that the prover's performance of SPHinx is minimally affected by the number of signed inputs and it can scale to large instances. In the fixed-size R1CS benchmark (Figure 6–left) we can observe the impact of signed inputs: in SPHinx generating a proof for 2^{20} signed inputs is only 2% slower than a proof for 2^8 inputs, whereas SigMarlin's performance degrades quickly. Another metric to evaluate the impact of our technique is to compare SPHinx and SigMarlin against a baseline represented by Marlin used to prove the same computation *with no authenticated inputs*. This metric highlights the overhead of proving validity of signed inputs in the two solutions. In Figure 6 we observe that SPHinx is 1.07×–1.3× slower than Marlin, whereas SigMarlin is at least 7× slower than Marlin (MLR with $n = 32$ days) and up to 1300× slower (variance with $t = 1024$).

On the downside, SPHinx has slower verification than SigMarlin:[11] the concrete time remains feasible though: e.g., verification takes below 4s on the largest instances with 2^{20} signed inputs and less than 100ms in benchmarks with moderate input sizes. We see the slower, yet feasible, verification time as a tradeoff to pay in order to make proving feasible. Indeed, we note that running SigMarlin on a computation with 2^{20} signed inputs would require an R1CS with ≈ 5.2 billions constraints, which is virtually prohibitive.

		Prover time (s)		Verifier time (ms)	
Benchmark	Input size	SPHinx	SigMarlin	SPHinx	SigMarlin
Histogram $k = 4$	$t = 256$	10	146	38	9
	$t = 1024$	39	606	46	10
	$t = 2048$	75	1255	53	11
	$t = 4096$	134	—	65	—
MLR $k = 30$	$n = 32$ days	17	105	39	9
	$n = 128$ days	47.7	417.3	45	10
	$n = 512$ days	117	1733	54	11
	$n = 1024$ days	220	—	69	—

Table 3: Costs in prover and verifier time, of SPHinx and SigMarlin **for histograms and MLR.**

ACKNOWLEDGMENTS

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under projects SCUM (ref. RTI2018-102043-B-I00), CRYPTOEPIC (ref. EUR2019-103816), and RED2018-102321-T, and by the Madrid Regional Government under project BLOQUES (ref. S2018/TCS-4339). This work is also supported by a grant from Nomadic Labs and the Tezos foundation.

[10] The spread of data is the extent to which it is squeezed towards a single value or spread out across a wider range.

[11] The slowdown is due to the need of performing a multi-exponentiation of length t in $\mathsf{LHS}_{\mathsf{Ped}}$ verification, while SigMarlin only needs $O(t)$ field operations.

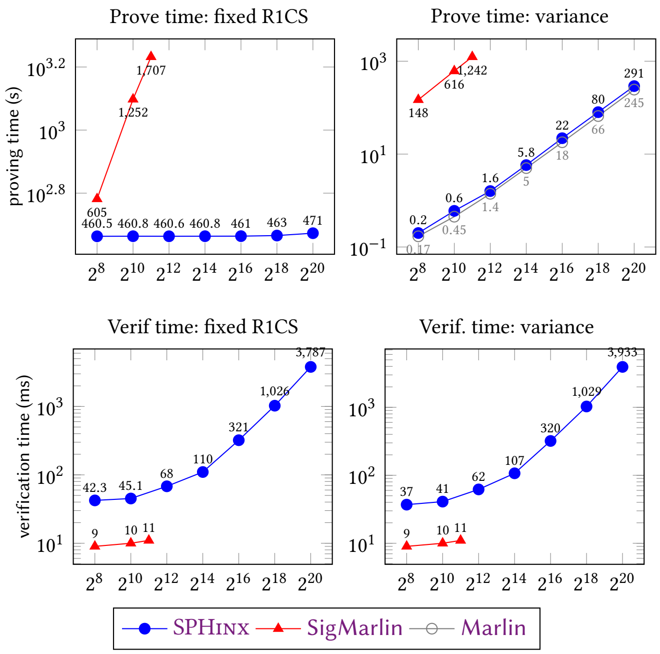

Figure 6: Comparison of SPHINX and SigMarlin on: R1CS of fixed size 2^{22} and varying inputs (left), and variance (right). In x-axis the number of signed inputs. Plots in log-scale.

REFERENCES

[1] Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. 2022. ECLIPSE: Enhanced Compiling Method for Pedersen-Committed zkSNARK Engines. In *Public-Key Cryptography – PKC 2022*, Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe (Eds.). Springer International Publishing, Cham, 584–614.

[2] Nuttapong Attrapadung and Benoît Libert. 2011. Homomorphic Network Coding Signatures in the Standard Model. In *PKC 2011 (LNCS, Vol. 6571)*, Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi (Eds.). Springer, Heidelberg, Germany, Taormina, Italy, 17–34. https://doi.org/10.1007/978-3-642-19379-8_2

[3] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. 2012. Computing on Authenticated Data: New Privacy Definitions and Constructions. In *ASIACRYPT 2012 (LNCS, Vol. 7658)*, Xiaoyun Wang and Kazue Sako (Eds.). Springer, Heidelberg, Germany, Beijing, China, 367–385. https://doi.org/10.1007/978-3-642-34961-4_23

[4] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. 2013. Efficient Completely Context-Hiding Quotable and Linearly Homomorphic Signatures. In *PKC 2013 (LNCS, Vol. 7778)*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.). Springer, Heidelberg, Germany, Nara, Japan, 386–404. https://doi.org/10.1007/978-3-642-36362-7_24

[5] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. 2015. ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 271–286. https://doi.org/10.1109/SP.2015.24

[6] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. 2017. The Hunting of the SNARK. *J. Cryptology* 30 (2017), 989–1066. https://doi.org/10.1007/s00145-016-9241-9

[7] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, Cambridge, MA, USA, 326–349. https://doi.org/10.1145/2090236.2090263

[8] Dan Boneh and Xavier Boyen. 2008. Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *Journal of Cryptology* 21, 2 (April 2008), 149–177. https://doi.org/10.1007/s00145-007-9005-7

[9] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. 2009. Signing a Linear Subspace: Signature Schemes for Network Coding. In *PKC 2009 (LNCS, Vol. 5443)*, Stanislaw Jarecki and Gene Tsudik (Eds.). Springer, Heidelberg, Germany, Irvine, CA, USA, 68–87. https://doi.org/10.1007/978-3-642-00468-1_5

[10] Dan Boneh and David Mandell Freeman. 2011. Homomorphic Signatures for Polynomial Functions. In *EUROCRYPT 2011 (LNCS, Vol. 6632)*, Kenneth G. Paterson (Ed.). Springer, Heidelberg, Germany, Tallinn, Estonia, 149–168. https://doi.org/10.1007/978-3-642-20465-4_10

[11] Dan Boneh and David Mandell Freeman. 2011. Linearly Homomorphic Signatures over Binary Fields and New Tools for Lattice-Based Signatures. In *PKC 2011 (LNCS, Vol. 6571)*, Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi (Eds.). Springer, Heidelberg, Germany, Taormina, Italy, 1–16. https://doi.org/10.1007/978-3-642-19379-8_1

[12] Sean Bowe. 2017. ebfull/pairing source code, BLS12-381 – README.md as of commit e726600. https://github.com/ebfull/pairing/tree/e72660056e00c93d6b054dfb08ff34a1c67cb799/src/bls12_381

[13] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. 2013. Verifying computations with state. In *Proc. of the ACM SOSP*.

[14] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 2075–2092. https://doi.org/10.1145/3319535.3339820

[15] Dario Catalano, Dario Fiore, Rosario Gennaro, and Konstantinos Vamvourellis. 2013. Algebraic (Trapdoor) One-Way Functions and Their Applications. In *TCC 2013 (LNCS, Vol. 7785)*, Amit Sahai (Ed.). Springer, Heidelberg, Germany, Tokyo, Japan, 680–699. https://doi.org/10.1007/978-3-642-36594-2_38

[16] Dario Catalano, Dario Fiore, and Luca Nizzardo. 2015. Programmable Hash Functions Go Private: Constructions and Applications to (Homomorphic) Signatures with Shorter Public Keys. In *CRYPTO 2015, Part II (LNCS, Vol. 9216)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 254–274. https://doi.org/10.1007/978-3-662-48000-7_13

[17] Dario Catalano, Dario Fiore, and Luca Nizzardo. 2018. On the Security Notions for Homomorphic Signatures. In *ACNS 18 (LNCS, Vol. 10892)*, Bart Preneel and Frederik Vercauteren (Eds.). Springer, Heidelberg, Germany, Leuven, Belgium, 183–201. https://doi.org/10.1007/978-3-319-93387-0_10

[18] Dario Catalano, Dario Fiore, and Bogdan Warinschi. 2012. Efficient Network Coding Signatures in the Standard Model. In *PKC 2012 (LNCS, Vol. 7293)*, Marc Fischlin, Johannes Buchmann, and Mark Manulis (Eds.). Springer, Heidelberg, Germany, Darmstadt, Germany, 680–696. https://doi.org/10.1007/978-3-642-30057-8_40

[19] Dario Catalano, Dario Fiore, and Bogdan Warinschi. 2014. Homomorphic Signatures with Efficient Verification for Polynomial Functions. In *CRYPTO 2014, Part I (LNCS, Vol. 8616)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 371–389. https://doi.org/10.1007/978-3-662-44371-2_21

[20] Dario Catalano, Antonio Marcedone, and Orazio Puglisi. 2014. Authenticating Computation on Groups: New Homomorphic Primitives and Applications. In *ASIACRYPT 2014, Part II (LNCS, Vol. 8874)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C., 193–212. https://doi.org/10.1007/978-3-662-45608-8_11

[21] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. 2020. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *EUROCRYPT 2020, Part I (LNCS, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, Germany, Zagreb, Croatia, 738–768. https://doi.org/10.1007/978-3-030-45721-1_26

[22] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. 2011. Memory Delegation. In *CRYPTO 2011 (LNCS, Vol. 6841)*, Phillip Rogaway (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 151–168. https://doi.org/10.1007/978-3-642-22792-9_9

[23] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, Cambridge, MA, USA, 90–112. https://doi.org/10.1145/2090236.2090245

[24] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. 2015. Geppetto: Versatile Verifiable Computation. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 253–270. https://doi.org/10.1109/SP.2015.23

[25] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. 2016. Hash First, Argue Later: Adaptive Verifiable Computations on Outsourced Data. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, Vienna, Austria, 1304–1316. https://doi.org/10.1145/2976749.2978368

[26] Dario Fiore and Anca Nitulescu. 2016. On the (In)Security of SNARKs in the Presence of Oracles. In *TCC 2016-B, Part I (LNCS, Vol. 9985)*, Martin Hirt and Adam D. Smith (Eds.). Springer, Heidelberg, Germany, Beijing, China, 108–138. https://doi.org/10.1007/978-3-662-53641-4_5

[27] Dario Fiore, Anca Nitulescu, and David Pointcheval. 2020. Boosting Verifiable Computation on Encrypted Data. In *PKC 2020, Part II (LNCS, Vol. 12111)*, Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas (Eds.). Springer, Heidelberg, Germany, Edinburgh, UK, 124–154. https://doi.org/10.1007/978-3-030-45388-6_5

[28] Matthew Fredrikson and Benjamin Livshits. 2014. ZØ: An Optimizing Distributing Zero-Knowledge Compiler. In *USENIX Security 2014*, Kevin Fu and Jaeyeon Jung (Eds.). USENIX Association, San Diego, CA, USA, 909–924.

[29] David Mandell Freeman. 2012. Improved Security for Linearly Homomorphic Signatures: A Generic Framework. In *PKC 2012 (LNCS, Vol. 7293)*, Marc Fischlin, Johannes Buchmann, and Mark Manulis (Eds.). Springer, Heidelberg, Germany, Darmstadt, Germany, 697–714. https://doi.org/10.1007/978-3-642-30057-8_41

[30] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. 2018. The Algebraic Group Model and its Applications. In *CRYPTO 2018, Part II (LNCS, Vol. 10992)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 33–62. https://doi.org/10.1007/978-3-319-96881-0_2

[31] S.D. Galbraith, K.G. Paterson, and N.P. Smart. 2006. Pairings for Cryptographers. Cryptology ePrint Archive, Report 2006/165. http://eprint.iacr.org/2006/165.

[32] Rosario Gennaro, Craig Gentry, and Bryan Parno. 2010. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In *CRYPTO 2010 (LNCS, Vol. 6223)*, Tal Rabin (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 465–482. https://doi.org/10.1007/978-3-642-14623-7_25

[33] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In *EUROCRYPT 2013 (LNCS, Vol. 7881)*, Thomas Johansson and Phong Q. Nguyen (Eds.). Springer, Heidelberg, Germany, Athens, Greece, 626–645. https://doi.org/10.1007/978-3-642-38348-9_37

[34] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. 2010. Secure Network Coding over the Integers. In *PKC 2010 (LNCS, Vol. 6056)*, Phong Q. Nguyen and David Pointcheval (Eds.). Springer, Heidelberg, Germany, Paris, France, 142–160. https://doi.org/10.1007/978-3-642-13013-7_9

[35] Rosario Gennaro and Daniel Wichs. 2013. Fully Homomorphic Message Authenticators. In *ASIACRYPT 2013, Part II (LNCS, Vol. 8270)*, Kazue Sako and Palash Sarkar (Eds.). Springer, Heidelberg, Germany, Bengalore, India, 301–320. https://doi.org/10.1007/978-3-642-42045-0_16

[36] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2008. Delegating computation: interactive proofs for muggles. In *40th ACM STOC*, Richard E. Ladner and Cynthia Dwork (Eds.). ACM Press, Victoria, BC, Canada, 113–122. https://doi.org/10.1145/1374376.1374396

[37] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. 2015. Leveled Fully Homomorphic Signatures from Standard Lattices. In *47th ACM STOC*, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM Press, Portland, OR, USA, 469–477. https://doi.org/10.1145/2746539.2746576

[38] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. 2018. Updatable and Universal Common Reference Strings with Applications to zk-SNARKs. In *CRYPTO 2018, Part III (LNCS, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 698–728. https://doi.org/10.1007/978-3-319-96878-0_24

[39] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. 2002. Homomorphic Signature Schemes. In *CT-RSA 2002 (LNCS, Vol. 2271)*, Bart Preneel (Ed.). Springer, Heidelberg, Germany, San Jose, CA, USA, 244–262. https://doi.org/10.1007/3-540-45760-7_17

[40] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT 2010 (LNCS, Vol. 6477)*, Masayuki Abe (Ed.). Springer, Heidelberg, Germany, Singapore, 177–194. https://doi.org/10.1007/978-3-642-17373-8_11

[41] Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. 2018. xJsnark: A Framework for Efficient Verifiable Computation. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 944–961. https://doi.org/10.1109/SP.2018.00018

[42] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. 2013. Linearly Homomorphic Structure-Preserving Signatures and Their Applications. In *CRYPTO 2013, Part II (LNCS, Vol. 8043)*, Ran Canetti and Juan A. Garay (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 289–307. https://doi.org/10.1007/978-3-642-40084-1_17

[43] Helger Lipmaa. 2012. Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In *TCC 2012 (LNCS, Vol. 7194)*, Ronald Cramer (Ed.). Springer, Heidelberg, Germany, Taormina, Sicily, Italy, 169–189. https://doi.org/10.1007/978-3-642-28914-9_10

[44] Vadim Lyubashevsky. 2012. Lattice Signatures without Trapdoors. In *EUROCRYPT 2012 (LNCS, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Heidelberg, Germany, Cambridge, UK, 738–755. https://doi.org/10.1007/978-3-642-29011-4_43

[45] David Pointcheval and Jacques Stern. 2000. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology* 13, 3 (June 2000), 361–396. https://doi.org/10.1007/s001450010003

[46] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 863–880. https://doi.org/10.1109/SP.2017.43

A TURNING Marlin INTO A CP-SNARK, EFFICIENTLY

We here describe how to turn the Marlin universal zkSNARK [21] into a commit-and-prove one.

We observe that a recent work [1], called ECLIPSE, proposes a generic method that can be also used to turn Marlin into a CP-SNARK. Their method is more general than ours as it can for example deal with inputs committed across several commitments. However, for the specific use case of our work, the advantage of our conversion over ECLIPSE's is that we can keep the size of the proof and the verification time constant in the length of the committed vector, whereas in ECLIPSE these are logarithmic and linear respectively (cf. [1, Table 1]).

Background Marlin is a zkSNARK for the R1CS in which a relation is defined by three matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and an instance vector $\boldsymbol{y}$ is considered in the language if there is a vector $\boldsymbol{w'}$ such that, for $\boldsymbol{z} = (1, \boldsymbol{y}, \boldsymbol{w'})$, it holds $\mathbf{A}\boldsymbol{z} \circ \mathbf{B}\boldsymbol{z} = \mathbf{C}\boldsymbol{z}$. Underlying Marlin is a polynomial commitment scheme like the scheme Com we mentioned in Section 6, where the commitment to a polynomial $p(X)$ is the group element $g_1^{p(s)} h_1^{o_p}$ for a random $o_p \leftarrow\$ \mathbb{F}$. For the sake of our conversion, we note that a proof in Marlin includes a commitment $\mathsf{c}_{w'}$ to the polynomial $\hat{w}'(X)$ that interpolates the vector $\boldsymbol{w'}$ (precisely the evaluations $(\hat{w}'(h_1), \ldots, \hat{w}'(h_{|\boldsymbol{w'}|}))$ on a specific subset of points is the vector $\boldsymbol{w'}$).

Turning Marlin **into a CP-SNARK** Our goal is to build a CP-SNARK for R1CS and the commitment scheme Com where, given a commitment c_x and a public $\boldsymbol{y}$, one can prove that the R1CS relation holds for $\boldsymbol{z} = (1, \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w})$ and c_x opens to $\boldsymbol{x}$. To this end, our $\mathsf{CP}_{\mathcal{R}}^{\mathsf{Marlin}}$ scheme augments Marlin with a proof that the vector $\boldsymbol{w'}$ encoded in the commitment $\mathsf{c}_{w'}$ is of the form $\boldsymbol{w'} = (\boldsymbol{x}, \boldsymbol{w})$, where $\boldsymbol{x}$ is the opening of c_x. More precisely, if c_x is a commitment to a polynomial $\hat{x}(X)$, we prove that in a specific public subset $\mathbb{T} \subset \mathbb{F}$ it holds $\forall h \in \mathbb{T} : \hat{w}'(h) = \hat{x}(h)$. We generate the latter proof via a CP-SNARK for this specific relation that we call $\mathsf{CP}_{\mathsf{svec}}$ and that we describe in the next section.

A.1 A CP-SNARK for subvectors

Let $\mathbb{T}$ be a subset of $\mathbb{F}$ of cardinality t and consider the following relation

$$\mathsf{R}_{\mathbb{T}} := \{(z(X), x(X)) : \forall h \in \mathbb{T} : z(h) = x(h)\}$$

We here present the main building block for making Marlin a CP-SNARK, which is a CP-SNARK $\mathsf{CP}_{\mathsf{svec}}$ for $\mathsf{R}_{\mathbb{T}}$.

The main idea of the proof system is that the statement $\forall h \in \mathbb{T} : z(h) = x(h)$ holds if and only if the polynomial $z(X) - x(X)$ is divisible by $Z_{\mathbb{T}}(X) = \prod_{h \in \mathbb{T}}(X - h)$, i.e., the polynomial that vanishes in the subset $\mathbb{T} \subset \mathbb{F}$, and thus there exists a polynomial $w(X)$ such that $z(X) - x(X) = w(X) Z_{\mathbb{T}}(X)$.

Let $\lambda_i(X)$, for $i = 1$ to t, be the Lagrange polynomials over domain $\mathbb{T}$. By polynomial long division, $z(X), x(X)$ can be uniquely decomposed as

$$z(X) = \sum_{i=1}^{t} z_i \lambda_i(X) + Z_{\mathbb{T}}(X) q_z(X),$$

$$x(X) = \sum_{i=1}^{t} x_i \lambda_i(X) + Z_{\mathbb{T}}(X) q_x(X).$$

Therefore, if $\forall h \in \mathbb{T} : z(h) = x(h)$ we can see that $z(X) - x(X)$ is divisible by $Z_{\mathbb{T}}(X)$. In the other direction, if there is $w(X)$ such that $z(X) - x(X) = w(X)Z_{\mathbb{T}}(X)$, then we can write $z(X) = x(X) + w(X)Z_{\mathbb{T}}(X)$ from which we can conclude $\forall h \in \mathbb{T} : z(h) = x(h)$, since $\mathbb{Z}_{\mathbb{T}}(h) = 0$ on every $h \in \mathbb{T}$.

To prove that the committed polynomials $(z(X), x(X)) \in \mathsf{R}_{\mathbb{T}}$, the prover first commits to the polynomial $w(X) = \frac{z(X)-x(X)}{Z_{\mathbb{T}}(X)}$ using a degree-2 polynomial $r_w(X) \leftarrow r_{w,0} + r_{w,1}X + r_{w,2}X^2$ to make it hiding, i.e., $\mathsf{c}_w \leftarrow g_1^{w(s)} h_1^{r_w(s)}$. Notice that after revealing c_w, there are still $|\mathbb{F}|^2$ possible choices for $r_w(X)$. Furthermore, by the homomorphic property of Com, $\mathsf{c}_z/\mathsf{c}_x$ is a commitment to the polynomial $w(X)Z_{\mathbb{T}}(X)$ with opening $r_z(X) - r_x$; the rest of the proof is devoted to showing that this is the case. To this end, the prover creates a commitment c_0 to the 0 polynomial with opening $r_0(X) := r_z(X) - r_x - r_w(X)Z_{\mathbb{T}}(X)$, i.e., $\mathsf{c}_0 \leftarrow h_1^{r_0(s)}$, and proves that this the case. The latter is done by showing that the evaluation of the polynomial in c_0 in a random point ρ (chosen after c_0) is 0. This proof requires to reveal $r_0(\rho)$. Note that we still have that $h_1^{r_0(s)}$ and $r_0(\rho)$ are uniformly distributed thanks to the entropy of $r_w(X)$.

The full description of the CP-SNARK $\mathsf{CP}_{\mathsf{svec}}$ follows.

```
Kg(ck): output ck = $(\{g_1^{s^i}, h_1^{s^i}\}_{i=0}^{D}, g_2^s)$ as universal SRS.
Derive(ck, $\mathbb{T}$): output ek = ck and vk = $(g_1, h_1, g_2, g_2^s, g_2^{Z_{\mathbb{T}}(s)})$
Prove(ek, $(\mathsf{c}_z, \mathsf{c}_x)$, $(z(X), \hat{x}(X))$, $(r_z(X), r_x)$):
 - $w(X) \leftarrow \frac{z(X)-\hat{x}(X)}{Z_{\mathbb{T}}(X)}$
 - $r_{w,0}, r_{w,1}, r_{w,2} \leftarrow\$ \mathbb{F}$
 - $r_w(X) \leftarrow r_{w,0} + r_{w,1}X + r_{w,2}X^2$
 - $\mathsf{c}_w \leftarrow g_1^{w(s)} h_1^{r_w(s)}$
 - $\mathsf{c}_0 \leftarrow h_1^{r_0(s)}$ where $r_0(X) := r_z(X) - r_x - r_w(X)Z_{\mathbb{T}}(X)$
 - $\rho \leftarrow \mathsf{H}(\mathsf{c}_z, \mathsf{c}_x, \mathsf{c}_w, \mathsf{c}_0)$
 - $q(X) \leftarrow \frac{r_0(X)-r_0(\rho)}{X-\rho}$, $\mathsf{c}_q \leftarrow h_1^{q(s)}$, $y \leftarrow r_0(\rho)$
 - Output $\pi := (\mathsf{c}_w, \mathsf{c}_0, \mathsf{c}_q, y)$
VerProof(vk, $(\mathsf{c}_z, \mathsf{c}_x)$, $\pi$):
 - $\rho \leftarrow \mathsf{H}(\mathsf{c}_z, \mathsf{c}_x, \mathsf{c}_w, \mathsf{c}_0)$
 - Output 1 iff the following checks are satisfied
```

$$e(\mathsf{c}_z/(\mathsf{c}_x \cdot \mathsf{c}_0), g_2) = e(\mathsf{c}_w, g_2^{Z_{\mathbb{T}}(s)})$$
$$e(\mathsf{c}_0 h_1^{-y}, g_2) = e(\mathsf{c}_q, g_2^{s-\rho})$$

Completeness Follows from the discussion above, and the completeness of the polynomial commitment scheme used in Marlin.

Succinctness The proof size is constant (3 group elements and one field element). For verification, the cost is poly(λ).

Knowledge soundness Let us denote $\mathbb{T}$, $(\mathsf{c}_z, \mathsf{c}_x)$, $\pi := (\mathsf{c}_w, \mathsf{c}_0, \mathsf{c}_q, y)$ the output of $\mathcal{A}$ in the knowledge soundness experiment $\mathsf{Game}^{\mathsf{KSND}}$; and ek := ck, vk := $(g_1, h_1, g_2, g_2^s, g_2^{Z_{\mathbb{T}}(s)})$ the corresponding output of Derive.

The polynomial commitment scheme used in Marlin is extractable, hence the extractor $\mathcal{E}$ for $\mathsf{CP}_{\mathsf{svec}}$ can extract, from $\mathcal{A}$'s view, polynomials $(\tilde{r}, r_{0,\mathsf{E}})$ from c_0, while the evaluation proof (c_q, y) (verified in the last check of our verification algorithm) ensures that $\tilde{r}(\rho) = 0$ and $r_{0,\mathsf{E}}(\rho) = y$. Now since ρ is chosen randomly *after* computing c_0, with overwhelming probability it holds that $\tilde{r}$ is the zero polynomial.

Similarly, from commitments c_z, c_x and c_w, one can extract polynomials $z_{\mathsf{E}}(X), \hat{x}_{\mathsf{E}}(X), w_{\mathsf{E}}(X)$, and openings $r_{z_{\mathsf{E}}}(X), r_{x_{\mathsf{E}}}(X), r_{w_{\mathsf{E}}}(X)$, which, from the first equality check; from the homomorphic property of the commitment scheme; and since we have verified that c_0 commits to the zero polynomial, satisfy $z_{\mathsf{E}}(X) - \hat{x}_{\mathsf{E}}(X) = w_{\mathsf{E}}(X)Z_{\mathbb{T}}(X)$. Finally, since $Z_{\mathbb{T}}(X)$ vanishes in $\mathbb{T}$, $\mathcal{E}$ has successfully extracted $(z_{\mathsf{E}}(X), \hat{x}_{\mathsf{E}}(X)) \in \mathsf{R}_{\mathbb{T}}$.

Composable zero-knowledge Intuitively, thanks to the fact r_w is chosen as a random polynomial of degree 2, even given commitments c_w, c_0 and c_q, the value of y is uniformly distributed from the adversary's view. Hence a simulator can chose the polynomial r_0 at random, compute $\mathsf{c}_0 \leftarrow h_1^{r_0(s)}$, and then compute c_w from c_0, its inputs $(\mathsf{c}_z, \mathsf{c}_x)$, and $Z_{\mathbb{T}}(s)$ (which it can compute as s is included in the trapdoor), i.e.,

$$\mathsf{c}_w \leftarrow \left(\frac{\mathsf{c}_z}{\mathsf{c}_x \mathsf{c}_0}\right)^{Z_{\mathbb{T}}(s)^{-1}}.$$

From there the simulator computes ρ, $q(X)$, c_q and y as per protocol, and the distribution of resulting simulated proofs is identical to that output by real executions of the protocol.

A.2 The CP-SNARK $\mathsf{CP}_{\mathcal{R}}^{\mathsf{Marlin}}$

Here we detail our CP version of Marlin.

```
Derive(ck):
 - $(\mathsf{ek}', \mathsf{vk}') \leftarrow \mathsf{Marlin.Derive}(\mathsf{ck})$
 - $(\mathsf{ek}_{\mathsf{svec}}, \mathsf{vk}_{\mathsf{svec}}) \leftarrow \mathsf{CP}_{\mathsf{svec}}.\mathsf{Derive}(\mathsf{ck})$
 - Output ek = $(\mathsf{ek}', \mathsf{ek}_{\mathsf{svec}})$ and vk = $(\mathsf{vk}', \mathsf{vk}_{\mathsf{svec}})$
Prove(ek, $\boldsymbol{y}$, $\mathsf{c}_x$, $\boldsymbol{x}$, $r_x$, $\boldsymbol{w}$):
 - $\pi' \leftarrow \mathsf{Marlin.Prove}(\mathsf{ek}', \mathsf{R}, y, (\boldsymbol{x}, \boldsymbol{w}))$
 - Parse $\pi' = (\mathsf{c}_{w'}, \pi'')$ with $\mathsf{c}_{w'} = g_1^{w'(s)} h_1^{r_{w'}(s)}$
 - $\pi_{\mathsf{svec}} \leftarrow \mathsf{CP}_{\mathsf{svec}}.\mathsf{Prove}(\mathsf{ek}_{\mathsf{svec}}, (\mathsf{c}_{w'}, \mathsf{c}_x),$
        $(w'(X), \langle \boldsymbol{x}, \boldsymbol{\lambda}(X) \rangle), (r_{w'}(X), r_x))$
 - Output $\pi := (\pi', \pi_{\mathsf{svec}})$
VerProof(vk, $\boldsymbol{y}$, $\mathsf{c}_x$, $\pi$):
 - Parse $\pi := (\pi', \pi_{\mathsf{svec}})$ and $\pi' = (\mathsf{c}_{w'}, \pi'')$
 - Output 1 iff $\mathsf{Marlin.VerProof}(\mathsf{vk}', \boldsymbol{y}, \pi') = 1$ and
        $\mathsf{CP}_{\mathsf{svec}}.\mathsf{VerProof}(\mathsf{vk}_{\mathsf{svec}}, \mathsf{c}_x, \mathsf{c}_{w'}, \pi_{\mathsf{svec}}) = 1$.
```

Correctness, knowledge-soundness and zero-knowledge of $\mathsf{CP}_{\mathcal{R}}^{\mathsf{Marlin}}$ follow from the properties of the underlying schemes.

B AN EFFICIENT INSTANTIATION OF THE CP-SNARK $\mathsf{CP}_{\mathsf{ev}}$

This is a CP-SNARK for committed polynomial evaluation.

```
Prove(ek, $y$, $(\mathsf{c}_p, \mathsf{c}_z)$, $(p(X), z)$, $(o_x, o_z)$):
 - $q(X) \leftarrow \frac{p(X)-z}{(X-y)}$
 - $o_q \leftarrow\$ \mathbb{F}$, $\mathsf{c}_q \leftarrow g_1^{q(s)} h_1^{o_q}$
 - $\tilde{g} \leftarrow h_1^{s-y}$
 - $\gamma, \delta, \epsilon, \eta \leftarrow\$ \mathbb{F}$
 - $U \leftarrow e(h_1^{\gamma} \tilde{g}^{\delta}, g_2)$
 - $u \leftarrow g_1^{\eta} h_1^{\epsilon}$
 - $\rho \leftarrow \mathsf{H}(y, \mathsf{c}_p, \mathsf{c}_z, \mathsf{c}_q, u, U)$
```

- $\sigma \leftarrow \gamma - (o_z - o_x)\rho$
- $\tau \leftarrow \delta - o_q\rho$
- $\mu \leftarrow \epsilon + o_z\rho$
- $\nu \leftarrow \eta + z\rho$
- Output $\pi := (\mathsf{c}_q, u, \mu, \nu, \rho, \sigma, \tau)$

$\mathsf{VerProof}(\mathsf{vk}, y, (\mathsf{c}_p, \mathsf{c}_z), \pi)$:

- $A \leftarrow e(\mathsf{c}_q, g_2^{s-y})e(\mathsf{c}_z/\mathsf{c}_p, g_2)$
- $U \leftarrow e(h_1^{\sigma}\tilde{g}^{\tau}, g_2)A^{\rho}$, for $\tilde{g} \leftarrow h_1^{s-y}$
- If $\rho \neq \mathsf{H}(y, \mathsf{c}_p, \mathsf{c}_z, \mathsf{c}_q, u, U)$ output 0
- If $g_1^{\nu}h_1^{\mu} \neq u \cdot \mathsf{c}_z^{\rho}$ output 0
- Else output 1

Remark 3.

• *In the* HSNP *scheme of Section 5, the vector of signed input data is encoded into a polynomial. This allows us to use a CP-SNARK for polynomial evaluation, rather than for the evaluation of inner product.*

• *Elements (u, ρ, μ, ν) of proof π constitute a Schnorr NIZKPoK of $z \in \mathbb{F}$ committed to in c_z. This enforces that the opening o_z to z be a scalar. Conversely, the opening o_x to $p(X)$ can itself be a polynomial, since in the proof for knowledge soundness of our HSNP, the extractor outputs $o_x(X) \in \mathbb{F}[X]$, of degree $\leqslant D$.*

Completeness If $\pi := (\mathsf{c}_q, u, \mu, \nu, \rho, \sigma, \tau)$ is computed as per the protocol description, then the verifier computes:

$$\begin{aligned} A &= e(g_1^{\frac{p(s)-z}{s-y}} h_1^{o_q}, g_2^{s-y})e(g_1^{z-p(s)} h_1^{o_z-o_x}, g_2) \\ &= e(g_1^{p(s)-z} h_1^{(s-y)o_q}, g_2)e(g_1^{z-p(s)} h_1^{o_z-o_x}, g_2) \\ &= e(h_1^{(s-y)o_q-o_x+o_z}, g_2) = e(\tilde{g}^{o_q} h_1^{o_z-o_x}, g_2) \end{aligned}$$

and

$$\begin{aligned} U &= e(h_1^{\gamma-(o_z-o_x)\rho}\tilde{g}^{\delta-o_q\rho}, g_2)\left(e(\tilde{g}^{o_q}h_1^{o_z-o_x}, g_2)\right)^{\rho} \\ &= e(h_1^{\gamma}\tilde{g}^{\delta}, g_2). \end{aligned}$$

This is exactly how U is computed by the prover, and hence the value ρ computed by the verifier and that provided by the prover in π are identical, if both parties follow the protocol. The final equality holds by the perfect completeness of Schnorr proofs of knowledge.

Zero-knowledge At a high level, the zero knowledge simulator works as follows. For key-generation, $\mathcal{S}_{\mathsf{kg}}$ runs the key generation simulator for the commitment scheme, from which it obtains a trapdoor $\mathsf{td} := (s, \alpha := \log_{g_1}(h_1))$. This is also the trapdoor output by $\mathcal{S}_{\mathsf{kg}}$ for $\mathsf{CP_{ev}}$.

To simulate a proof for an instance $y, (\mathsf{c}_p, \mathsf{c}_z)$, algorithm $\mathcal{S}_{\mathsf{prv}}$ commits to any polynomial q^*, say the zero polynomial, i.e. $\mathsf{c}_q^* \leftarrow h_1^{o_q^*}$ for $o_q^* \leftarrow\!\$\, \mathbb{F}$. Then it simulates the proof for this commitment, by sampling random $\rho, \sigma, \tau, \nu, \mu \leftarrow\!\$\, \mathbb{F}$, and computing $A^* \leftarrow e(\mathsf{c}_q^*, g_2^{s-y})\cdot e(\mathsf{c}_z/\mathsf{c}_p, g_2)$; $U^* \leftarrow e(h_1^{\sigma}\tilde{g}^{\tau}, g_2)A^{\rho}$; and $u^* \leftarrow g_1^{\nu}h_1^{\mu}\mathsf{c}_z^{-\rho}$. Then it sets the output of the random oracle H on input $(y, \mathsf{c}_p, \mathsf{c}_z, \mathsf{c}_q^*, u^*, U^*)$ to be ρ. If this output value had already been set, then $\mathcal{S}_{\mathsf{prv}}$ aborts, however this occurs with negligible probability since U^* is random in $\mathbb{G}_T$ (based on the randomness of σ and τ). Finally $\mathcal{S}_{\mathsf{prv}}$ outputs $\pi := (\mathsf{c}_q^*, u^*, \mu, \nu, \rho, \sigma, \tau)$. From the perfect hiding property of Com, and the zero-knowledge property of the Schnorr proof of knowledge for c_z, the simulation is indistinguishable from a real execution.

B.1 Knowledge soundness

Knowledge soundness holds under the d – SDH assumption (which implies the hardness of computing discrete logarithms), and in the algebraic group model (AGM) [30], which we recall below.

B.1.1 *d-Strong Diffie-Hellman assumption.*

The security of our $\mathsf{CP_{ev}}$ CP-SNARK relies on the following assumption, which is a slight variation of the original Strong Diffie-Hellman assumption [8], as we provide less group elements to the adversary (it does not see g_2^s).

Assumption 2 (d-SDH). *The d-Strong Diffie-Hellman assumption holds for the bilinear group generator $\mathcal{G}$ if for any PT adversary $\mathcal{A}$, and for all large enough λ, on the probability space $\mathsf{pp}_{\mathcal{G}} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^{\lambda})$, $\mathsf{chall} := ((g_1, g_1^s, \ldots, g_1^{s^d}); g_2)$, $g_1 \xleftarrow{\$} \mathbb{G}_1$, $g_2 \xleftarrow{\$} \mathbb{G}_2$ and $s \xleftarrow{\$} \mathbb{Z}_q$, the following probability is negligible in λ:*

$$\mathsf{Adv}_{\mathcal{A}}^{d\text{-}\mathsf{sdh}}(\lambda) := \Pr[(r, y) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathcal{G}}, \mathsf{chall}) \wedge y = g_1^{\frac{1}{s-r}}].$$

B.1.2 *The algebraic group model.*

For additional efficiency, our instantiation for $\mathsf{CP_{ev}}$ is proven extractable in the Algebraic Group Model (AGM) [30], which replaces a Power Knowledge of Exponent assumption. In the AGM, algorithms are modelled as algebraic, this means that whenever an algorithm outputs a group element g, the algorithm must also output an *explanation* of g in terms of the group elements that it has seen.

Definition B.1 (algebraic algorithm). Consider a cyclic group $\mathbb{G}$ of prime order q, and a probabilistic algorithm $\mathcal{A}_{\mathsf{alg}}$ whose initial input includes a description pp of $\mathbb{G}$. During its execution $\mathcal{A}_{\mathsf{alg}}$ may interact with oracles or other parties and receive further inputs including obliviously sampled group elements (these cannot be sampled directly). Let $\boldsymbol{L} \in \mathbb{G}^n$ be the list of all group elements $\mathcal{A}_{\mathsf{alg}}$ has been given so far such that all other inputs it has received do not depend in any way on group elements. Then $\mathcal{A}_{\mathsf{alg}}$ is said to be *algebraic* if whenever it outputs a group element $g \in \mathbb{G}$, it also outputs a vector $\boldsymbol{a} \in \mathbb{Z}_q^n$ such that $g = \prod_{i=1}^n L_i^{a_i}$. The coefficients $\boldsymbol{a}$ are called the *representation* of g with respect to $\boldsymbol{L}$.

Knowledge-soundness proof of $\mathsf{CP_{ev}}$ The proof strategy is similar to that of [27, Thm. 10], only we rely on the AGM instead of a knowledge assumption. First observe that an algebraic adversary for knowledge soundness $\mathcal{A}_{\mathsf{alg}}$ must output representations of group elements c_p and c_q with respect to $(g_1, g_1^s, \ldots, g_1^{s^D})$, $(h_1, \ldots, h_1^{s^D})$. These representations allow the simulator to effortlessly extract polynomials $p(X), q(X)$ (of degree $\leqslant D$), $o_x(X)$ and $o_q(X)$ (of degree $\leqslant D$) which satisfy $\mathsf{c}_p = g_1^{p(s)}h_1^{o_x(s)}$ and $\mathsf{c}_q = g_1^{q(s)}h_1^{o_q(s)}$.

Next we use the rewinding technique [45] for proving the soundness of the Schnorr proofs to extract scalars z, o_z, ψ, ϕ such that $\mathsf{c}_z = g_1^z h_1^{o_z}$ and $A = e(\mathsf{c}_q, g_2^{s-y})e(\mathsf{c}_z/\mathsf{c}_p, g_2) = e(h_1^{\psi}\tilde{g}^{\phi}, g_2)$.

In more detail, consider the game between the challenger and a malicious prover P^* against the soundness of the Schnorr proof. The challenger runs P^* by fixing the values $\mathsf{c}_p, \mathsf{c}_z, \mathsf{c}_q$ and changing the oracle definition to get a fork with $\rho' = \mathsf{H}(y, \mathsf{c}_p, \mathsf{c}_z, \mathsf{c}_q, u, U) \neq \rho$.

Algorithm P^* will output two distinct forgeries corresponding to the same random oracle query, but for distinct challenges ρ and ρ'.

By the Forking Lemma, it holds that rewinding $O(q_{\mathsf{H}}/\epsilon_{P^*})$ times, where q_{H} is the maximal number of random oracle queries made by P^* and ϵ_{P^*} its success probability, then one obtains μ, ν, σ, τ, and $\mu', \nu', \sigma', \tau'$ with constant probability. These openings allow one to compute $z := (\nu - \nu')/(\rho - \rho')$, $o_z := (\mu - \mu')/(\rho - \rho')$, as well as $\psi := (\sigma - \sigma')/(\rho' - \rho)$ and $\phi := (\tau - \tau')/(\rho' - \rho)$ which satisfy the aforementioned requirements.

Now assume the algebraic adversary $\mathcal{A}_{\mathsf{alg}}$ for knowledge soundness of $\mathsf{CP}_{\mathsf{ev}}$ outputs a cheating statement $(y, (\mathsf{c}_p, \mathsf{c}_z))$ and proof $\pi = (\mathsf{c}_q, u, \mu, \nu, \rho, \sigma, \tau)$ such that it passes verification checks, but the extracted values $(p(X), q^*(X), z^*, o_x(X), o_q(X), o_z, \psi, \phi)$ do not satisfy the expected relation $p(y) = z^*$. We denote $\tilde{g} := h^{s-y}$.

Assuming that the commitment scheme is binding, one of the following must hold:

Case 1 The extracted polynomials do not satisfy the correct relation, even when evaluated in s, i.e., $q^*(s) \neq \frac{p(s)-z^*}{s-y}$. This type of forgery can be reduced to the discrete logarithm problem for $(g_1, h_1) \in \mathbb{G}_1$. Indeed, consider an adversary $\mathcal{B}_{\mathsf{DL}}$ for the discrete logarithm problem, which gets as input a pair $(g_1, h_1) \in \mathbb{G}_1$. Adversary $\mathcal{B}_{\mathsf{DL}}$ samples a random $s \leftarrow\$ \mathbb{F}$ which it uses to simulate the srs for $\mathcal{A}_{\mathsf{alg}}$ (using g_1 and h_1 as in the real protocol), so as to extract a tuple as described above. Now assuming the binding of the commitment scheme, the verification check gives us $A = e(h_1^{(s-y)o_q(s)} g_1^{q^*(s)(s-y)}, g_2) e(h_1^{o_z - o_x(s)} g_1^{z^* - p(s)}, g_2) = e(\tilde{g}^{\phi} h_1^{\psi}, g_2)$. By the non-degeneracy of the pairing, we obtain:

$$h_1^{(s-y)o_q(s)+o_z-o_x(s)-\psi-(s-y)\phi} = g_1^{(s-y)q^*(s)+z^*-p(s)},$$

where $(s-y)q^*(s) + z^* - p(s)$ is invertible modulo q, so $\mathcal{B}_{\mathsf{DL}}$ can output the discrete logarithm of g_1 in base h_1.

Case 2 The polynomial $q^*(X)$ committed to in c_q does not satisfy the correct relation w.r.t. $p(X)$ and z^*, but evaluated in s it holds that $q^*(s) = \frac{p(s)-z^*}{s-y}$. This type of forgery can be reduced to the $D-\mathsf{SDH}$ assumption. Consider an adversary $\mathcal{B}_{\mathsf{SDH}}$ for the $D-\mathsf{SDH}$ problem, which gets as input a tuple $(g_1, g_1^s, \ldots, g_1^{s^D})$, it samples a random $\alpha \in \mathbb{F}$ and sets $h_1 := g_1^{\alpha}, h_1^s := (g_1^s)^{\alpha}, \ldots, h_1^{s^D} := (g_1^{s^D})^{\alpha}$. It uses these values to simulate the srs for $\mathcal{A}_{\mathsf{alg}}$, so as to extract a tuple as described above. The checks on verification imply:

$$\begin{aligned} A &= e(g_1^{q^*(s)} h_1^{o_q(s)}, g_2^{s-y}) \cdot e(g_1^{z^*-p(s)} h_1^{o_z-o_x(s)}, g_2) \\ &= e(h_1^{o_z-o_x(s)+(s-y)o_q(s)}, g_2) \cdot e(g_1^{(s-y)q^*(s)+z^*-p(s)}, g_2) \\ &= e(h_1^{o_z-o_x(s)+(s-y)o_q(s)}, g_2) = e(h_1^{(s-y)\phi+\psi}, g_2) \end{aligned}$$

Hence $(s-y)(o_q(s) - \phi) + (o_z - o_x(s)) - \psi = 0$ in $\mathbb{F}$. So either $\mathcal{B}_{\mathsf{SDH}}$ can efficiently compute s as a root of the polynomial $(X-y)(o_q(X) - \phi) + (o_z - o_x(X)) - \psi \in \mathbb{F}[X]$, which allows it to solve any SDH challenge; or o_x and o_q are degree zero polynomials, and $\phi = o_q$ and $\psi = o_z - o_x$. Assuming this is the case, denote $z := p(y)$, and define the polynomial $q(X) := \frac{p(X)-z}{X-y} \in \mathbb{F}[X]$, so that $g_1^{(s-y)q(s)} = g_1^{p(s)-z}$. Then $\mathcal{B}_{\mathsf{SDH}}$ computes $g_1^{z-z^*} = g^{(s-y)(q^*(s)-q(s))}$. Define the polynomial $q'(X) := (X-y)(q(X) - q^*(X)) - z + z^*$ and denote $q'(X) = \sum_i q'_i X^i$. We have that $g_1^{q'(s)} = 1$. By the hypothesis that $q^*(X)$ does not satisfy the correct relation w.r.t. $p(X)$ and z^*, it holds that $q'(X) \neq 0$, and hence s is a root of q'. This allows $\mathcal{B}_{\mathsf{SDH}}$ to compute $g_1^{1/s}$ as follows. Consider q'_{i_0} the first non-zero coefficient of the polynomial $q'(X)$. Then there is a polynomial $q''(X)$ of lower degree (which can be computed from the initial instance), satisfying $q'_{i_0} s^{i_0} = s^{i_0+1} q''(s)$, and so $g_1^{q'_{i_0}} = g_1^{s \cdot q''(s)}$, or equivalently $g_1^{1/s} = (g_1^{q''(s)})^{1/q'_{i_0}}$. Hence $\mathcal{B}_{\mathsf{SDH}}$ is able to solve its $D-\mathsf{SDH}$ challenge.

C APPLICATIONS

We here provide additional concrete applications for verifiable computation on delegated data streams.

Statistics on health data Governments must periodically publish health statistics to inform the public of the status of healthcare systems. Clearly, the original data cannot be made public as it contains sensitive information pertaining to the people receiving health care. However, this raw information can be authenticated by medical practitioners, who operate as trusted data providers. The amount of medical data to be stored being large, the hospital delegates its' entire memory to the cloud. The government (the VCS client) may then ask the could to compute authorised functions of this data, and expects the answers to be accompanied by a proof. In this case, the government (and the public being informed) is ensured that the result of the computations are correct and originated in legitimate medical data by using VCS.

Driving statistics Pay as you drive auto insurance involves paying a rate proportional to the number of miles driven. For privacy, the driver may not want to reveal her personal driving habits to the insurance company. For integrity, the company wants to be sure that every driver pays the correct premium. Though solutions for this specific problem have been provided [5, 28], where the driver also performs the computations on the data, by using VCS, we allow for much more flexibility. Precisely, the driver (i.e. the data-provider) can stream the data collected by the vehicle onto some cloud server. The insurance company (the VCS client) can request from the server the required function of this data to compute the driver's premium, ensuring the pay-as-you-drive functionality. Furthermore, if say, the driver has an accident, both the insurance company, and law enforcement officials, may request functions of the stored data ascertaining whether the driver was speeding at the time of the accident.

Hence this same data stored by the server can serve a wide variety of analysis purposes, as it will be stored in full by the server, and the choice of applied function only comes into play upon request by the insurance company. This contrasts to AD-SNARKS or previous pay-as-you drive solutions, where the driver herself applies some aggregate function to the data (so as to not reveal it in full), and sends the results, along with a proof of integrity and authenticity to the insurance company.

Sliding window statistics Several applications naturally generate data streams as opposed to data sets. In telecommunications call records are generated continuously. Typically, most processing is done by examining a call record once or operating on a 'window' of recent call records (e.g., to update customer billing information), after which records are archived and not examined again. Other applications include network traffic engineering, web tracking and personalisation (where the data streams are web log entries or click-streams), medical monitoring (vital signs, treatments, and other measurements), sensor databases, financial monitoring, and data mining applications. In most of these applications, the goal is to make decisions based on the statistics or models gathered over the recently observed data elements. For example, one might be interested in gathering statistics about packets processed by a set of routers over the last day. In the sliding window model, data elements arrive at every instant; each data element expires after exactly N time steps; and the portion of data that is relevant to gathering statistics or answering queries is the set of the last N elements to arrive. The sliding window refers to the window of active data elements at a given time instant.

Predicting market data An application of sliding window statistics to real-time market data is that of model-driven prediction, i.e. *predicting stock prices.* This difficult task should be done frequently in order to learn from recent price fluctuations and try to better predict future ones. A simple yet potent model for this task is multi-linear regression (MLR). This model aims to predict the outcome of an event which depends on multiple factors.

Using VCS, the server can compute the coefficients of the MLR model, as well as the prediction of some stock price requested by a client, while proving computations where performed honestly on the expected data.

Machine learning classifiers on portions of streamed data The goal of statistical classification in machine learning is to use an object's characteristics to identify which class it belongs to. These characteristics are typically presented to the machine in a vector. For our example these vectors are streamed to the server, which computes the corresponding classifications. A client may then query the server to ask if, among the previous 1000 vectors, any of them are categorised in a certain class (e.g. a 'high risk' class).

A linear classifier makes a classification decision based on the value of a linear combination of the characteristics, i.e. based on the result of a dot product between the input vector and some weight vector. Such classifiers work well for problems with many variables, reaching accuracy levels comparable to non-linear classifiers while taking less time to train and use.

Quadratic classifiers generalise the linear model, and are generally used to extend the classifier's ability to represent more complex separating surfaces. Denoting the input vector $\boldsymbol{x}$, and for a learned matrix $\boldsymbol{A}$, vector $\boldsymbol{B}$ and scalar c, the class of $\boldsymbol{x}$ will be decided based on $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c$.

For both types of classifiers, a final non-linear step is applied, which checks if the output is in a given range.

D MULTI-LINEAR REGRESSION FOR PREDICTING STOCK PRICES

An application of sliding window statistics to real-time market data is that of model-driven prediction, i.e. *predicting stock prices.* This difficult task should be done frequently in order to learn from recent price fluctuations and try to better predict future ones.

A simple yet potent model for this task is multi-linear regression (MLR). This model aims to predict the outcome of an event which depends on multiple factors.

Assume a data provider streams Microsoft's daily stock price to a server, and a client queries the server for a prediction of Microsoft's closing price in 5 days time, requiring that the prediction model be based on data streamed the past n days. Using VCS, the server can compute the coefficients of the MLR model, and the required prediction, while proving to the client that the computations where performed honestly on the expected data.

Until recently, the high infrastructure and maintenance costs of handling real-time market data made it difficult for many firms to adopt. However, cloud-based delivery has made real-time market data accessible to a wide array of applications. One application that benefits from this expanded access to real-time data sources is that of model-driven prediction, i.e. *predicting stock prices.* This difficult task should be done frequently in order to learn from recent price fluctuations and try to better predict future ones. For efficiency reasons, on may wish to delegate both the offline training and the online prediction work (on authenticated data) to a third party server. One simple, yet very efficient model for this task is multiple linear regression (MLR). This models aims to predict the outcome of an event which depends on multiple factors.

For our concrete example, assume a powerful server receives a signed stream of Microsoft's daily stock price. And a client queries the server for a target it wants to predict: Microsoft's Closing stock price in 5 days, where the prediction model is computed based on data received over the past N days.

We hereafter explain how the server computes the coefficients of an MLR, based on data received in the past N days, to predict the closing price of Microsoft in 5 days time.

The data streamed each day consists of four data items, which provide the high, low, open and close prices. Hence the size of the input used for computations is $4N$.

The R1CS instance vector will thus contain at least $4N+2$ entries, to encode the signed data, the constant 1, and how far in the future the prediction is for (5 days).

Evaluated computations. As the price itself is not sufficient to produce useful predictions, the server first computes additional indicators which will be used as features (inputs) for computing the model. The server computes:

- the average price of the last T days for varying periods T. This adds $n - T + 1$ entries (e.g. variables) to the R1CS instance vector, and $n - T + 1$ constraints (e.g. rows).

  We also add these features shifted by varying time lags from 0 to 10 days, each inducing an extra $n - T$ variables and constraints.
- the aforementioned averages plus/minus a certain amount of standard deviations, again shifted by varying numbers of days.

  To compute the rolling values of the standard deviation requires an extra $(n-T)(T+2)$ constraints, and $(n-T)(2T+2)$ variables. To then compute the value of a given feature adds an extra $(n - T)$ variables and constraints (per feature).
- shifts of the original high, low, open and close price. This adds n variables and constraints per additional feature.

Days / Input size	Feat.	Constraints	Variables
n / 4n	k	$n(2k^2 + 8k + 4) + k^3 + 5k^2 + 9k + 6$	$n\left(\frac{3}{2}k^2 + \frac{15}{2}k + 4\right) + k^3 + 4k^2 + 7k + 4$
n / 4n	20	$964n + 10\,186$	$754n + 9\,744$
n / 4n	30	$2\,044n + 31\,776$	$1\,579n + 30\,814$
n / 4n	40	$3\,524n + 72\,366$	$2\,704n + 70\,684$

Table 4: Size of R1CS instance for MLR application.

Denoting k the total number of computed features, it will always hold that the number of variables and constraints to prove correct computation of these features is between kn and $2kn$.

For each of the above features, an extra column is added to the data table. We also add a new column for the value we want to predict: the close price of 5 days in the future. This column copies the close column and shifts it by 5 rows.

Let us denote Z the resulting $n \times (k+1)$ dataset (each column corresponds either to the target or to one of the aforementioned features), and X the matrix obtained by removing the target column from Z, and inserting a column of 1s as the first column of X. Finally let Y be the $n \times 1$ target column.

The server now computes a vector of coefficients $\boldsymbol{b} = (b_0, b_1, \ldots, b_{50})$ such that $Y = X \cdot \boldsymbol{b}$. This can be done as:

$$\boldsymbol{b} := (X^\top \cdot X)^{-1} X^\top Y.$$

The computation of $(X^\top \cdot X)$ requires first computing all the products of entries in X and $X^\top$, so we add $n(k+1)^2$ constraints and $\frac{n(k+1)(k+2)}{2}$ new variables.

Then to compute the $(k+1) \times (k+1)$ entries of $(X^\top \cdot X)$ by summing the aforementioned products, we need an extra $(k+1)^2$ variables and constraints.

The inversion adds an extra $(k+1)^3$ variables and constraints for intermediate computations And to prove that $(X^\top \cdot X)(X^\top \cdot X)^{-1}$ is the identity matrix requires $(k+1)^2$ constraints (but no extra variables).

So $(X^\top \cdot X)^{-1}$ adds $n \times (k+1)^2 + 2(k+1)^2 + (k+1)^3$ constraints and $\frac{n(k+1)(k+2)}{2} + (k+1)^2 + (k+1)^3$ variables.

Multiplying by $X^\top$ adds an extra $(k+1)^2 n$ variables and constraints for intermediate computations. Then for the actual matrix $(X^\top \cdot X)^{-1} X^\top$, we add an extra $(k+1)n$ constraints and variables. So a total of $((k+1)^2 + (k+1))n$ constraints and variables for the multiplication by $X^\top$.

Multiplying by Y, adds an extra $(k+1)n$ intermediate variables and constraints, and an extra $(k+1)$ of each for computing the values $(b_0, b_1, \ldots, b_k)$.

Finally, from the row of data $\boldsymbol{x} = (x_1, \ldots, x_k)$ consistent with Z corresponding to the most recent streamed record, the server predicts the close price 5 days in the future by computing:

$$\widehat{\boldsymbol{Y}} = b_0 + b_1 x_1 + \cdots + b_{50} x_{50}.$$

This adds $k+1$ variables to the R1CS instance (the claimed output $\widehat{\boldsymbol{Y}}$), and an extra $k+2$ constraints.

The total size of the resulting R1CS is given in Table 4, where we use the upper bound $2kn$ for number of variables and constraints needed for the generation of additional features.