A Linearly Homomorphic Signature Scheme From Weaker Assumptions

Lucas Schabhüser, Johannes Buchmann, and Patrick Struck

Technische Universität Darmstadt, Germany,
buchmann,lschabhueser@cdc.informatik.tu-darmstadt.de,
patrick.struck@stud.tu-darmstadt.de

Abstract. In delegated computing, prominent in the context of cloud computing, guaranteeing both the correctness and authenticity of computations is of critical importance. Homomorphic signatures can be used as cryptographic solutions to this problem. In this paper we solve the open problem of constructing a linearly homomorphic signature scheme that is secure against an active adversary under standard assumptions. We provide a construction based on the DL and CDH assumption. Furthermore we show how our scheme can be combined with homomorphic encryption under the framework of Linearly Homomorphic Authenticated Encryption with Public Verifiability. This way we can provide the first such scheme that is context hiding. Furthermore our solution even allows verification in constant time (in an amortized sense).

Keywords: homomorphic signatures, homomorphic encryption, delegated computation, cloud computing

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No 644962.

1 Introduction

1.1 Motivation

Homomorphic signature schemes allow to check the correctness of a computations result without having to perform the computation oneself. This allows a client to delegate computations to a computationally more powerful server, such that the server can verify the result. In this scenario the server is asked to not only perform a computation but also to evaluate this function over the signatures as well. The resulting signature to the output can be used to verify the correctness of the result. There have been multiple schemes proposed for this. Their security however is mostly based on weaker non-standard assumptions. Besides outsourcing of computations, homomorphic signatures offer security in *network coding* [9]. These constructions however do not consider confidentiality, i.e. the client might not be comfortable with the server knowing what data he is computing on. Homomorphic encryption schemes allow the evaluation of functions over encrypted messages. That servers can perform computations, learning neither the input nor the output. However, clients still have to trust the server to a degree, as there is no way to check whether the function has been evaluated as claimed. Combining both approaches allows a client to verify the correctness by checking a signature and decrypting the cipher returned by the server. However, naively combining both primitives requires the cipher space of the encryption scheme to be (a subset of) the message space of the homomorphic signature scheme. In the case of Paillier encryption for instance the underlying message space is $\mathbb{Z}_n$ while the cipher space is $\mathbb{Z}_{n^2}$, i.e. a doubling the signature size. Catalano et al. [14] proposed a method which allows to combine the Paillier encryption scheme with a homomorphic signature scheme instantiated to support only the message space of the Paillier encryption scheme. Currently there exists only one instantiation of this, whose security is also based on strong assumptions.

1.2 Related work

Linearly homomorphic signature schemes: The idea of linearly homomorphic signature schemes was introduced in [16] and later refined in [19]. Freeman proposed stronger security definitions in [17]. An instantiation based on 2-3-Diffie Hellmann was proposed in [9]. Later realizations are based on subgroup decision problems [2,3], the k-Simultaneous Flexible Pairing Problem [4], the RSA problem [18] (offering only security against *weak adversaries*), the strong RSA problem [11], the Flexible DH Inversion problem [10], and the lattice based k-SIS problem [8]. As already mentioned in [17] the construction of such a scheme, that is secure against a strong adversary, and based on weak assumptions, has been solved by using so called *chameleon hash functions*, which are very computationally expensive. Constructing such a scheme without them, has remained an open problem ever since. The idea of homomorphic signatures with efficient verification was introduced in [12]. Intuitively, this means that the outcome of a computation can be checked faster by using the schemes verification algorithm than computing it oneself. However, this only holds in an *amortized* sense, as an expensive preprocessing phase has to be amortized over multiple datasets (see [5,14]).

Authenticated Homomorphic Encryption An and Bellare [1] introduced a new paradigm called *encryption with redundancy* which allows to achieve both privacy and authentication. In [6] the idea of *authenticated encryption* is formalized. Analogous notions for the homomorphic setting were given in [20] and [14]. In the latter the notion of *linearly homomorphic authenticated encryption with public verifiability (LAEPuV)* was introduced which will be used in this paper. As pointed out in [23], the candidate instantiation of [14] suffers from false negatives however, and an improved version was proposed.

1.3 Contribution and Roadmap

In this paper we propose a linearly homomorphic signature scheme that is unforgeable against strong (adaptive) adversaries under the computational Diffie-Hellman assumption, which is one of the most well studied cryptographic problems and thereby solve a problem left open in [17]. This scheme has several desirable properties. The size of a signature does not depend on the size of the dataset over which computations are executed, so it is in particular *succinct*, it allows for *efficient verification*, in our case even *constant time verification*, and is *context hiding*, i.e. no information about the input values can be learned from the signature to the output of a computation (not even if the secret key is compromised). We then show how our scheme can be used in conjunction with Paillier encryption [21] in order to instantiate a LAEPuV scheme. This is both the first context hiding construction and the first to support vectors of messages.

We introduce notation and preliminaries in Section 2. We present a new homomorphic signature scheme in Section 3 and prove its properties, while Section 4 shows how our scheme can be combined with homomorphic encryption.

2 Preliminaries

To accurately describe what both correct and legitimate operations for homomorphic signatures are, we will make use of *multi-labeled programs* similar to [5]. On a high level a function is appended by several identifiers, in our case input identifiers and dataset identifiers. Input identifiers label in which order the input values are to be used and dataset identifiers determine which signatures can be homomorphically combined. The idea is that only signatures created under the same dataset identifier can be combined. We will now give formal definitions.

A *labeled program* $\mathcal{P}$ consists of a tuple $(f, \tau_1, \ldots, \tau_k)$, where $f : \mathcal{M}^k \to \mathcal{M}$ is a function with k inputs and $\tau_i \in \mathcal{T}$ is a label for the i-th input of f from some set $\mathcal{T}$. Given a set of labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and a function $g : \mathcal{M}^t \to \mathcal{M}$, they can be composed by evaluating g over the labeled programs, i.e. $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. The identity program with label τ is given by $\mathcal{I}_\tau = (f_{id}, \tau)$, where $f_{id} : \mathcal{M} \to \mathcal{M}$ is the identity function. Note that program $\mathcal{P} = (f, \tau_1, \ldots, \tau_k)$ can be expressed as the composition of k identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_k})$.

A *multi-labeled program* $\mathcal{P}_\Delta$ is a pair $(\mathcal{P}, \Delta)$ of the labeled program $\mathcal{P}$ and a dataset identifier Δ. Given a set of t multi-labeled programs with the same data set identifier Δ, i.e. $(\mathcal{P}_1, \Delta), \ldots, (\mathcal{P}_t, \Delta)$, and a function $g : \mathcal{M}^t \to \mathcal{M}$, a composed multi-label program $\mathcal{P}^*_\Delta$ can be computed, consisting of the pair $(\mathcal{P}^*, \Delta)$, where $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. Analogous to the identity program for labeled programs we refer to a multi-labeled identity program by $\mathcal{I}_{(\Delta,\tau)} = ((f_{id}, \tau), \Delta)$.

Definition 1 (Homomorphic Signature Scheme). *A homomorphic signature scheme is a tuple of the following probabilistic polynomial-time algorithms:*

$\mathsf{HKeyGen}(1^\lambda, k)$: *On input a security parameter λ and an integer k, the algorithm returns a key pair $(\mathsf{sk}, \mathsf{pk})$, where sk is the secret key kept private and pk is the public key which determines the message space $\mathcal{M}$, the signature space $\mathcal{Y}$, and the set $\mathcal{F}$ of admissible labeled programs $\mathcal{P} : \mathcal{M}^k \to \mathcal{M}$.*

$\mathsf{HSign}(\mathsf{sk}, \Delta, \tau, m)$: *On input a secret key sk, a dataset identifier Δ, an input identifier τ, and a message $m \in \mathcal{M}$, the algorithm returns a signature $\sigma \in \mathcal{Y}$ which is the signature for the message labeled by τ in the dataset identified by Δ.*

$\mathsf{HEval}(\mathsf{pk}, \mathcal{P}_\Delta, \boldsymbol{\sigma})$: *On input a public key pk, a multi-labeled program $\mathcal{P}_\Delta$, and a set of signatures $\boldsymbol{\sigma} \in \mathcal{Y}^k$, the algorithm returns a signature*

$\sigma' \in \mathcal{Y}$ for the multi-labeled program $\mathcal{P}$ over the (tuple of) signatures $\boldsymbol{\sigma}$ identified by Δ.

$\mathsf{HVerify}(\mathsf{pk}, \mathcal{P}_\Delta, m, \sigma)$: *On input a public key* pk*, a multi-labeled program $\mathcal{P}_\Delta$, a message $m \in \mathcal{M}$, and a signature $\sigma \in \mathcal{Y}$, the algorithm either accepts the signature σ, for the multi-labeled program $\mathcal{P}$ over the dataset identified by Δ, i.e. it returns* $\mathsf{1}$*, or rejects the signature, i.e. it returns* $\mathsf{0}$.

We will now define the relevant properties for homomorphic signatures.

Definition 2 (Correctness). *A homomorphic signature scheme* $(\mathsf{HKeyGen}, \mathsf{HSign}, \mathsf{HEval}, \mathsf{HVerify})$ *is called correct, if for any security parameter λ, any integer k, and any key pair* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{HKeyGen}(1^\lambda, k)$ *the following two conditions are satisfied.*

Condition 1 *For any dataset identifier Δ, any input identifier τ, and any message $m \in \mathcal{M}$, it holds that*

$$\mathsf{HVerify}(\mathsf{pk}, \mathcal{I}_{\Delta,\tau}, m, \mathsf{HSign}(\mathsf{sk}, \Delta, \tau, m)) = 1.$$

Condition 2 *For any dataset identifier Δ, any multi-labeled program $\mathcal{P}_\Delta = ((f_1, \ldots, f_k), \tau_1, \ldots, \tau_k, \Delta)$ containing a linear function, and any set of messages $\boldsymbol{m} \in \mathcal{M}^k$ with $\boldsymbol{m} = (m_1, \ldots, m_k)$, it holds that*

$$\mathsf{HVerify}(\mathsf{pk}, \mathcal{P}_\Delta, f(m_1, \ldots, m_k), \mathsf{HEval}(\mathsf{pk}, \mathcal{P}_\Delta, \boldsymbol{\sigma})) = 1$$

where $\boldsymbol{\sigma} = (\sigma_{\tau_1}, ..., \sigma_{\tau_k}) \in \mathcal{Y}^k$ with $\sigma_\tau \leftarrow$ $\mathsf{HSign}(\mathsf{sk}, \Delta, \tau, m_\tau)$.

Definition 3 (Succinctness). *A homomorphic signature scheme* $(\mathsf{HKeyGen}, \mathsf{HSign}, \mathsf{HEval}, \mathsf{HVerify})$ *is called* succinct *if for a fixed security parameter λ the size of the signatures depends at most logarithmically on the dataset size k.*

For the security notion of our homomorphic signature scheme we first provide a definition for *well defined programs* and *forgeries* on these programs. Then, we introduce an experiment the attacker can run in order to make a successful forgery and present a definition for unforgeability based on this experiment.

Definition 4 (Well Defined Program). *A labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_k)$ is well defined with respect to a list $\mathcal{L}$ if one of the two following cases holds: First, there exists exactly one m_i such that $(\tau_i, m_i) \in \mathcal{L}\ \forall i = 1, \ldots, k$. Second, there is an $i \in \{1, \ldots, k\}$ such that $(\tau_i, \cdot) \notin \mathcal{L}$ and $f(\{m_j\}_{(\tau_j, m_j) \in \mathcal{L}} \cup \{\tilde{m}_l\}_{(\tau_l, \cdot) \notin \mathcal{L}})$ does not depend on the choice of $\tilde{m}_l \in \mathcal{M}$.*

Definition 5 (Forgery). *A forgery is a tuple* $(\mathcal{P}_\Delta, m^*, \sigma^*)$ *such that* $\mathsf{HVerify}(\mathsf{pk}, \mathcal{P}_\Delta, m^*, \sigma^*) = 1$ *holds and one of the following conditions is met:*

Type 1: *The list* $\mathcal{L}$ *was not initialized during the game, i.e. no message was ever committed under the dataset identifier* Δ*.*
Type 2: $\mathcal{P}_\Delta$ *is well defined with respect to the list* $\mathcal{L}$ *and* m^* *is not the correct output of the computation, i.e* $m^* \neq f(\{m_j\}_{(\tau_j, m_j) \in \mathcal{L}})$*.*
Type 3: $\mathcal{P}_\Delta$ *is not well defined with respect to* $\mathcal{L}$*.*

For the notion of unforgeability we define the following experiments $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda)$ and $\mathsf{Weak-HomUF-CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

$\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda)$:

Key Generation $\mathcal{C}$ calls $(\mathsf{sk}, \mathsf{pk}) \leftarrow_\$ \mathsf{HKeyGen}(1^\lambda, k)$ and gives pk to $\mathcal{A}$.

Queries $\mathcal{A}$ adaptively submits queries for (Δ, τ, m) where Δ is a dataset, τ is an input identifier, and m is a message. $\mathcal{C}$ proceeds as follows: if (Δ, τ, m) is the first query with dataset identifier Δ, it initializes an empty list $\mathcal{L} = \emptyset$ for Δ. If $\mathcal{L}$ does not contain a tuple $(\tau, \cdot)$, i.e. $\mathcal{A}$ never queried $(\Delta, \tau, \cdot)$, $\mathcal{C}$ calls $\sigma \leftarrow \mathsf{HSign}(\mathsf{sk}, \Delta, \tau, m)$, updates the list $\mathcal{L} = \mathcal{L} \cup (\tau, m)$, and gives σ to $\mathcal{A}$. If $(\tau, m) \in \mathcal{L}$ then $\mathcal{C}$ returns the same signature σ as before. If $\mathcal{L}$ already contains a tuple (τ, m') for $m \neq m'$, $\mathcal{C}$ returns $\perp$.

Forgery $\mathcal{A}$ outputs a tuple $(\mathcal{P}_\Delta, m, \sigma)$. The experiment outputs 1, if $(\mathcal{P}_\Delta, m, \sigma)$ is a forgery according to Definition 5.

In the following experiment $\mathsf{Weak-HomUF-CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda)$, the adversary has to declare the message components of the later signing queries before the key generation and can later on specify in which dataset Δ_j it wants to query it.

$\mathsf{Weak-HomUF-CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda)$:

Declaration of Messages $\mathcal{A}$ outputs a list of possible messages $\{m_{\tau,j}\}_{\tau \in \mathcal{L}, j=1}^{Q} \subset \mathcal{M}$ where Q is the number of datasets to be queried.

Key Generation $\mathcal{C}$ calls $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{HKeyGen}(1^\lambda, k)$ and gives pk to $\mathcal{A}$.

Queries $\mathcal{A}$ adaptively submits queries for $(\Delta_j, \tau, m_{\tau,j})$ where Δ is a dataset, τ is an identifier, and $m_{\tau,j}$ is a message. $\mathcal{C}$ proceeds as follows: if $(\Delta_j, \tau, m_{\tau,j})$ is the first query with dataset identifier Δ_j, it initializes an empty list $\mathcal{L} = \emptyset$ for Δ_j. If $\mathcal{L}$ does not contain a tuple $(\tau, \cdot)$, i.e. $\mathcal{A}$ never queried $(\Delta_j, \tau, \cdot)$, $\mathcal{C}$ calls $\sigma \leftarrow \mathsf{HSign}(\mathsf{sk}, \Delta_j, \tau, m)$, updates the list $\mathcal{L} = \mathcal{L} \cup (\tau, m_{\tau,j})$, and gives σ to $\mathcal{A}$. If $(\tau, m_{\tau,j}) \in \mathcal{L}$ then $\mathcal{C}$ returns the same signature σ as before. If $\mathcal{L}$ already contains a tuple $(\tau, m'_{\tau,j})$ for $m \neq m'$ $\mathcal{C}$ returns $\perp$.

Forgery $\mathcal{A}$ outputs a tuple $(\mathcal{P}_\Delta, m, \sigma)$. The experiment outputs 1, if $(\mathcal{P}_\Delta, m, \sigma)$ is a forgery according to Definition 5

Definition 6 (Unforgeability). *A linearly homomorphic signature scheme is* unforgeable *if for any PPT adversary $\mathcal{A}$ we have*

$$Pr[\mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

It is weakly unforgeable *if for any PPT adversary $\mathcal{A}$ we have*

$$Pr[\mathsf{Weak} - \mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

However any homomorphic signature scheme weakly-unforgeable under a computational assumption can be transformed into one that is unforgeable under the same assumption by [12, Theorem 1].

Additionally we will make use of the following statement.

Lemma 1 (Proposition 2.3 of [17]).

Let $\mathcal{H} = (\mathsf{HKeyGen}, \mathsf{HSign}, \mathsf{HEval}, \mathsf{HVerify})$ be a linearly homomorphic signature scheme over a message space $\mathcal{M} \subset R^T$ for some ring R. If $\mathcal{H}$ is secure against Type 2 forgeries, then $\mathcal{H}$ is also secure against Type 3 forgeries.

Definition 7 (Context-Hiding). *A homomorphic signature scheme for multi-labeled programs is called* context hiding *if there exist additional PPT procedures $\tilde{\sigma} \leftarrow \mathsf{HHide}(\mathsf{pk}, m, \sigma)$ and $\mathsf{HHideVer}(\mathsf{pk}, \mathcal{P}_\Delta, m, \tilde{\sigma})$ such that:*

Correctness: *For any $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{HKeyGen}(1^\lambda, k)$ and tuple $(\mathcal{P}_\Delta, m, \sigma)$, such that $\mathsf{HVerify}(\mathsf{pk}, \mathcal{P}_\Delta, m, \sigma) = 1$, and $\tilde{\sigma} \leftarrow \mathsf{HHide}(\mathsf{pk}, m, \sigma)$, it holds that $\mathsf{HHideVer}(\mathsf{pk}, \mathcal{P}_\Delta, m, \tilde{\sigma}) = 1$.*

Unforgeability: *The homomorphic signature scheme is unforgeable (see Definition 6) when replacing the algorithm* HVerify *with* HHideVer *in the security experiment.*

Context-Hiding Security: *There is a simulator* Sim *such that, for any fixed (worst-case) choice of $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{HKeyGen}(1^\lambda, k)$, any multi-labeled program $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_k, \Delta)$, messages $m_1, \ldots, m_l$, and distinguisher $\mathcal{D}$ there exists a function $\epsilon(\lambda)$ such that the following equation holds:*

$$|Pr[\mathcal{D}(I, \mathsf{HHide}(\mathsf{pk}, m, \sigma) = 1] - Pr[\mathcal{D}(I, \mathsf{Sim}(\mathsf{sk}, \mathcal{P}_\Delta, m)) = 1]| = \epsilon(\lambda)$$

where $I = (\mathsf{sk}, \mathsf{pk}, \mathcal{P}_\Delta, \{m_i, \sigma_i\}_{i=1}^l, m, \sigma)$ *for* $\sigma_i \leftarrow \mathsf{HSign}(\mathsf{sk}, \Delta, \tau_i, m_i)$, $m \leftarrow f(m_1, \ldots, m_k)$, $\sigma \leftarrow \mathsf{HEval}(\mathsf{pk}, \mathcal{P}_\Delta, \sigma_1, \ldots, \sigma_k)$,*and the probabilities are taken over the randomness of* $\mathsf{HSign}, \mathsf{HHide}$ *and* Sim*. If* $\epsilon(\lambda) \leq \mathsf{negl}(\lambda)$ *we call the homomorphic signature scheme* statistically context-hiding*, if* $\epsilon(\lambda) = 0$ *we call it* perfectly context hiding*.*

Definition 8 (Efficient Verification). *A homomorphic signature scheme for multi-labeled programs allows for* efficient verification*, if there exist two additional algorithms* $(\mathsf{VerPrep}, \mathsf{EffVer})$ *such that:*

$\mathsf{VerPrep}(\mathsf{pk}, \mathcal{P})$: *Given a public key* pk *and a labeled program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_k)$*, this algorithm generates a concise public key* $\mathsf{pk}_\mathcal{P}$*. This does* not *depend on a dataset identifier* Δ*.*

$\mathsf{EffVer}(\mathsf{pk}_\mathcal{P}, m, \sigma, \Delta)$**:** *Given a concise public key* $\mathsf{pk}_\mathcal{P}$*, a message* m*, a signature* σ *and a dataset* Δ*, it outputs* `1` *or* `0`*.*

The above algorithms are required to satisfy the following two properties:

Correctness: *Let* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{HKeyGen}(1^\lambda, k)$ *be honestly generated keys and* $(\mathcal{P}, m, \sigma)$ *be a tuple such that for* $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$ *we have* $\mathsf{HVerify}(\mathsf{pk}, \mathcal{P}_\Delta, m, \sigma) = 1$*.*

Then for every $\mathsf{pk}_\mathcal{P} \overset{\$}{\leftarrow} \mathsf{VerPrep}(\mathsf{pk}, \mathcal{P})$, $\mathsf{EffVer}(\mathsf{pk}_\mathcal{P}, m, \sigma, \Delta) = 1$ *holds except with negligible probability.*

Amortized Efficiency: *Let* $\mathcal{P}$ *be a program,* $m_1, \ldots, m_k$*, be valid input values and let* $t(k)$ *be the time required to compute* $\mathcal{P}(m_1, \ldots, m_k)$*. Then for* $\mathsf{pk}_\mathcal{P} \overset{\$}{\leftarrow} \mathsf{VerPrep}(\mathsf{pk}, \mathcal{P})$ *the time required to compute* $\mathsf{EffVer}(\mathsf{pk}_\mathcal{P}, m, \sigma, \Delta)$ *is* $t' = o(t(k))$*.*

Note that efficiency here is used in an amortized sense. There is a function dependent preprocessing so that the cost of verification amortizes over multiple datasets.

2.1 Notation

Definition 9 (Asymmetric bilinear groups). *An* asymmetric bilinear group *is a tuple* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ *such that (1)* $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$ *are cyclic groups of order* q*, (2) the* Discrete Logarithm Problem *is hard to be computed in* $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$*, (3)* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is bilinear, i.e.* $e({g_1}^a, {g_2}^b) = e(g_1, g_2)^{ab}$ *holds for all* $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$*, and* $a, b \in \mathbb{Z}_q$*, (4)* e *is non-degenerate, i.e.* $e(g_1, g_2) \neq 1$*, and (5)* e *is efficiently computable. The function* e *is called* bilinear map *or* pairing*.*

During our constructions we will have multiple input messages m_i where the messages are vectors. For reasons of clarity we will make the following convention: m_i will be used to identify a certain message, while $m[j]$ will be used to denote the j-th entry of the message vector m. Thus $m_i[j]$ is the j-th entry of the i-th message.

2.2 Assumptions

Definition 10. *(DL) Let $\mathbb{G}$ be a group of order q (not necessarily prime): We say the Discrete Logarithm assumption holds in $\mathbb{G}$. if there exists no ppt adversary $\mathcal{A}$ that given (g, g^a) for a random generator $g \in \mathbb{G}$ and random $a \in \mathbb{Z}_q$ can output a with more than negligible probability.*

Note that there exist different variations of Diffie-Hellman assumptions in bilinear groups (see for example [15]). We will use the following definition.

Definition 11. *(CDH in Bilinear Groups [15])*
Let $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ be a description of a bilinear group. We say the Computational Diffie-Hellman assumption holds in bgp, if there exists no ppt adversary $\mathcal{A}$ that given $(\mathsf{bgp}, g_1^a, g_2^b)$ where $a, b \overset{\$}{\leftarrow} \mathbb{Z}_q$ can output g_1^{ab} with more than negligible probability.

Definition 12. *(DCRA) Let n be the product of two (safe) primes, i.e. $n = pq$. We say the Decisional composite residuosity assumption (DCRA) holds if there exists no ppt adversary $\mathcal{A}$ that can distinguish between an element drawn uniformly random from the set $\mathbb{Z}^*_{n^2}$ and an element from the set $\{z^n | z \in \mathbb{Z}^*_{n^2}\}$, that is the set of the n-th residues modulo n^2.*

3 Construction

In the following we will describe a linearly homomorphic signature scheme $\mathsf{HSig} = (\mathsf{HKeyGen}, \mathsf{HSign}, \mathsf{HEval}, \mathsf{HVerify})$ based on CDH in bilinear groups. In this instantiation the input identifiers are simply the integers from 1 to k. Multi-labeled programs contain linear functions f given by their coefficients, i.e. $f = (f_1, \dots, f_k)$.

$\mathsf{HKeyGen}(1^\lambda, k, T)$**:** On input a security parameter λ, an integer k, and an integer T, the algorithm runs $\mathcal{G}(1^\lambda)$ to obtain a bilinear group $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$, and samples $k+T$ elements $R_1, \dots, R_k, h_1, \dots, h_T \leftarrow \mathbb{G}_1$. Additionally it generates a key pair $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{KeyGen}'(1^\lambda)$

of a regular signature scheme and a key $K \overset{\$}{\leftarrow} \mathcal{K}$ for a pseudorandom function $\mathsf{PRF} : \mathcal{K} \times \{0,1\}^* \rightarrow \mathbb{Z}_q$. It returns the key pair $(\mathsf{sk}, \mathsf{pk})$ with $\mathsf{sk} = (\mathsf{sk}', K)$ and $\mathsf{pk} = (\mathsf{pk}', \mathsf{bgp}, \{h_j\}_{j=1}^T, \{R_i\}_{i=1}^k)$.

$\mathsf{HSign}(\mathsf{sk}, \Delta, i, m)$**:** On input a secret key sk, a dataset identifier Δ, an input identifier $i \in [k]$, and a message $m \in \mathbb{Z}_q^T$, the algorithm generates the parameters for the dataset identified by Δ, by running $z \leftarrow \mathsf{PRF}_K(\Delta)$ and computing $Z = g_2^z$. It binds Z to the dataset identifier Δ by using the regular signature scheme, i.e. it sets $\sigma_\Delta \leftarrow \mathsf{Sign}'(\mathsf{sk}', Z|\Delta)$. Then, it computes $\Lambda \leftarrow (R_i \cdot \prod_{j=1}^T h_j^{-m[j]})^z$ and returns the signature $\sigma = (\sigma_\Delta, Z, \Lambda)$.

$\mathsf{HEval}(\mathsf{pk}, \mathcal{P}_\Delta, \boldsymbol{\sigma})$**:** On input a public key pk, a multi-labeled program $\mathcal{P}_\Delta$ containing a linear function f, and signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$, where $\sigma_i = (\sigma_{\Delta,i}, Z_i, \Lambda_i)$, the algorithm checks if the signatures share the same public values, i.e. if $\sigma_{\Delta,1} = \sigma_{\Delta,i}$ and $Z_1 = Z_i$ for all $i = 2, \ldots, k$, and the signature for each set of public values is correct and matches the dataset identifier Δ, i.e. $\mathsf{Verify}'(\mathsf{pk}', Z_i|\Delta, \sigma_{\Delta,i}) = 1$ for any $i = 1, \ldots, k$. If that is not the case the algorithm rejects the signature, otherwise, it proceeds as follows. It computes $m = \sum_{i=1}^k f_i m_i$ and $\Lambda = \prod_{i=1}^k \Lambda_i^{f_i}$, and returns the signature $\sigma = (Z_1, \sigma_{\Delta,1}, \Lambda)$.

$\mathsf{HVerify}(\mathsf{pk}, \mathcal{P}_\Delta, m, \sigma)$**:** On input a public key pk, a message m, a signature $\sigma = (\sigma_\Delta, Z, \Lambda)$, and a multi-labeled program containing a linear function f, the algorithm returns $\mathsf{1}$, if $\mathsf{Verify}'(\mathsf{pk}', Z|\Delta, \sigma_\Delta) = 1$ and $e\left(R \cdot \prod_{j=1}^T h_j^{-m[j]}, Z\right) = e\left(\Lambda, g_2\right)$, where $R \leftarrow \prod_{i=1}^k R_i^{f_i}$. Otherwise, it returns $\mathsf{0}$.

Theorem 1. HSig *is a correct linearly homomorphic signature scheme according to Definiton 2.*

Proof. Throughout this proof, let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{HKeyGen}(1^\lambda, k, T)$ be an honestly generated key pair with $\mathsf{sk} = (\mathsf{sk}', K)$ and $\mathsf{pk} = (\mathsf{pk}', \mathsf{bgp}, \{h_j\}_{j=1}^T,$ $\{R_i\}_{i=}^k)$.

Condition 1: Let Δ be a dataset identifier, $i \in [k]$ be an input identifier, $m \in \mathbb{Z}_q^T$ be a message, and $\sigma = (\sigma_\Delta, Z, \Lambda) \leftarrow \mathsf{HSign}(\mathsf{sk}, \Delta, i, m)$ be the signature of m. Furthermore, let $\mathcal{I}_{(\Delta,i)}$ be the identity function for the i-th input under the tag Δ. By construction it holds that $\mathsf{Verify}'(\mathsf{pk}', Z|\Delta, \sigma_\Delta) = 1$ and $R = \prod_{i=1}^k R_i^{f_i} = R_i^1 = R_i$, which yields $e\left(R_i \cdot \prod_{j=1}^T h_j^{-m[j]}, Z\right) = e\left(R_i \cdot \prod_{j=1}^T h_j^{-m[j]}, g_2^z\right) = e\left(R_i h_1^{-m}, g_2\right)^z = e\left(\left(R_i \cdot \prod_{j=1}^T h_j^{-m[j]}\right)^z, g_1\right) = e\left(\Lambda, g_2\right)$. Thus, we have
$\mathsf{HVerify}(\mathsf{pk}, \mathcal{I}_{(\Delta,i)}, m, \sigma) = \mathsf{HVerify}(\mathsf{pk}, \mathcal{I}_{(\Delta,i)}, m, \mathsf{HSign}(\mathsf{sk}, \Delta, i, m)) = 1$.

Condition 2: Let Δ be a dataset identifier, $m_i \in \mathbb{Z}_q^T$ for $i \in [k]$ be messages, $\mathcal{P}_\Delta = ((f_1, \dots, f_k), 1, \dots, k, \Delta)$, and $\sigma_i \leftarrow \mathsf{HSign}(\mathsf{sk}, \Delta, i, m_i)$, with $\sigma_i = (\sigma_{\Delta,i}, Z_i, \Lambda_i)$, be a signature of m_i. Furthermore, let $\sigma = (\sigma_\Delta, Z, \Lambda) \leftarrow \mathsf{HEval}(\mathsf{pk}, \mathcal{P}_\Delta, \boldsymbol{\sigma})$ be the signature obtained by evaluating f over the signatures in the dataset identified by Δ.
By construction we have $Z = Z_1$ and $\sigma_{\Delta,i} = \sigma_{\Delta,1}$, hence we have $\mathsf{Verify}'(\mathsf{pk}', Z|\Delta, \sigma_\Delta) = 1$. To prove the correctness it remains to show that $e\left(R \cdot \prod_{j=1}^T h_j^{-m[j]}, Z\right) = e(\Lambda, g_2)$, where $R = \prod_{i=1}^k R_i^{f_i}$. It holds that

$$e\left(R \cdot \prod_{j=1}^{T} h_j^{-m[j]}, Z\right) = e\left(\prod_{i=1}^{k} R_i^{f_i} \cdot \prod_{j=1}^{T} h_j^{-\sum_{i=1}^k f_i m_i[j]}, g_2^z\right)$$

$$= e\left(\prod_{i=1}^{k} R_i^{f_i} \cdot \prod_{i=1}^{k} (\prod_{j=1}^{T} h_j^{-m_i[j]})^{f_i}, g_2\right)^z = e\left(\prod_{i=1}^{k} (R_i \cdot \prod_{j=1}^{T} h_j^{-m_i[j]})^{f_i}, g_2\right)^z$$

$$= e\left(\prod_{i=1}^{k} ((R_i \cdot \prod_{j=1}^{T} h_j^{-m_i[j]})^z)^{f_i}, g_2\right) = e\left(\prod_{i=1}^{k} \Lambda_i^{f_i}, g_2\right) = e(\Lambda, g_2)$$

hence $\mathsf{HVerify}(\mathsf{pk}, \mathcal{P}_\Delta, f(m_1, \dots, m_k), \mathsf{HEval}(\mathsf{pk}, \mathcal{P}_\Delta, \boldsymbol{\sigma})) = 1$.

Theorem 2. *If* Sig' *is an unforgeable signature scheme,* PRF *is a pseudorandom function, and the CDH assumption (see Definition 11) holds in* bgp*, then the signature scheme describe above is a weakly-unforgeable homomorphic signature scheme for linear functions.*

Proof. To prove this Theorem we define a series of games with the adversary $\mathcal{A}$ and we will show that the adversary $\mathcal{A}$ wins, i.e. the game outputs $\mathtt{1}$, only with negligible probability. Following the notation of [10] we will write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns $\mathtt{1}$. We will make use of flag values bad_i initially set to false. If at the end of the game any of these flags is set to true, the game simply outputs $\mathtt{0}$. Let $\mathsf{Bad_i}$ denote the event that $\mathsf{bad_i}$ is set to true during a game.

Game 1: This is the experiment $\mathsf{Weak-HomUF-CMA}_{\mathcal{A},\mathsf{HomSign}}$ (see Definition 6) where $\mathcal{A}$ only outputs Type-1 or Type-2 forgeries.

Game 2: This game is defined as Game 1 apart from the fact that whenever $\mathcal{A}$ outputs a forgery $(\mathcal{P}_\Delta, m^*, \sigma^*)$, where $\sigma^* = (\sigma^*_\Delta, Z^*, \Lambda^*)$ such that Z^* was not generated by the challenger, then Game 2 sets $\mathsf{bad}_2 \leftarrow \mathsf{true}$.

Game 3: This game is the same as Game 2, except that the pseudorandom function of the scheme is replaced with a true random function $\Phi : \{0,1\}^* \to \mathbb{Z}_q$.

Game 4: This game is the same as Game 3, except for an additional check. When given a forgery $({\mathcal{P}^*}_{\Delta^*}, m^*, \sigma^*)$ where ${\mathcal{P}^*}_{\Delta^*} = ((f^*, 1, \ldots, k), \Delta^*)$ the simulator computes $m \leftarrow f^*(m_{1,\Delta}, \ldots, m_{k,\Delta})$. It checks whether $\prod_{j=1}^{T} h_j^{m[j]} = \prod_{j=1}^{T} h_j^{m^*[j]}$ holds. If it does it sets $\mathsf{bad}_4 = \mathsf{true}$.

We will first show that these games are computationally indistinguishable under our assumptions and then proceed by showing how to construct a simulator $\mathcal{S}$ which uses an efficient adversary $\mathcal{A}$ against the signature scheme to solve the CDH problem.

Games 1 and 2 are only different if Bad_2 occurs. By constructions this means that $\mathcal{A}$ produced a forgery containing a valid signature σ_Δ^* on $(\Delta^*|Z^*)$ even though no signature has ever been queried for datatset Δ^*. This means that the adversary $\mathcal{A}$ can be used to obtain an existential forgery for the signature scheme Sig'.

If PRF is a pseudorandom function then Game 2 is computationally indistinguishable from Game 3.

We obviously have $|Pr[G_3(\mathcal{A})] - \Pr[G_4(\mathcal{A})]| \leq Pr[\mathsf{Bad}_4]$.

In Lemma 3 in the Appendix we show how an adversary $\mathcal{A}$, such that $Pr[\mathsf{Bad}_4]$ is non negligible, can be used to break the DL assumption. Afterwards in Lemma 2 we show how a simulator can use an adversary winning Game 4 to break the CDH assumption.

Theorem 3. *The homomorphic signature scheme* HSig *is succinct.*

Proof. The signature size is independent of the size k of the datasets.

Theorem 4. *The homomorphic signature scheme* HSig *allows for efficient verification.*

Proof. We describe the two algorithms $(\mathsf{VerPrep}, \mathsf{EffVer})$.

$\mathsf{VerPrep}(\mathsf{pk}, \mathcal{P})$: It parses $\mathcal{P} = ((f_1, \ldots, f_k), 1, \ldots, k)$ and takes the R_i for $i \in [k]$ contained in the public key. It computes $R_\mathcal{P} \leftarrow \prod_{i=1}^{k} R_i^{f_i}$ and outputs $\mathsf{pk}_\mathcal{P} = (\mathsf{pk}', \mathsf{bgp}, \{h_j\}_{j=1}^{T}, R_\mathcal{P})$ where $\mathsf{pk}', \mathsf{bgp}, \{h_j\}_{j=1}^{T}$ are taken from pk.

$\mathsf{EffVer}(\mathsf{pk}_\mathcal{P}, m, \sigma, \Delta)$**:** This algorithm does the same as $\mathsf{HVerify}$ only the value R has been precomputed as $R_\mathcal{P}$.

Obviously this satisfies correctness and the running time of EffVer is now independent of k and therefore the runtime complexity of $\mathcal{P}$. Thus our construction is constant time (in an amortized sense).

Theorem 5. *The linearly homomorphic signature scheme* HSig *is perfectly context hiding according to Definition 7 if* Sig' *is a deterministic signature scheme.*

For the proof we refer to the appendix (see Theorem 7).

4 Linearly Homomorphic Authenticated Encryption

We will give the formal definitions for Linearly Homomorphic Authenticated Encryption with Public Verifiability.

Definition 13. *(LAEPuV [14]). A LAEPuV scheme is a tuple of five ppt algorithms* $(\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AEval}, \mathsf{AVerify}, \mathsf{ADecrypt})$ *such that:*

$\mathsf{AKeyGen}(1^\lambda, k)$**:** *It takes a security parameter* λ *and the maximum number* k *of encrypted messages in each dataset as input. It returns a key pair* $(\mathsf{sk}, \mathsf{pk})$*, where* sk *is the secret key for encrypting and signing and* pk *is the public key used for verification and evaluation. The message space* $\mathcal{M}$*, the cipher space* $\mathcal{C}$ *and dataset identifier space* $\mathcal{D}$ *are implicitly defined by the public key* pk.

$\mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m)$**:** *The input is a secret key* sk*, a dataset identifier* Δ*, an input identifier* τ*, and a message* m*. The output is a cipher* c*.*

$\mathsf{AEval}(\mathsf{pk}, \mathcal{P}_\Delta, \{c_i\}_{i=1}^k)$**:** *The input is a public key* pk*, a multi-labeled program* $\mathcal{P}_\Delta$*, and a set of* k *ciphers* $\{c_i\}_{i=1\dots k}$*. The output is a cipher* c*.*

$\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c)$**:** *The input is a public key* pk*, a multi-labeled program* $\mathcal{P}_\Delta$ *containing a linear function* f*, and a cipher* c*. The output is either* 1*, i.e. the cipher is valid, or* 0*, i.e. the cipher is invalid.*

$\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c)$**:** *It gets a secret key* sk*, a multi-labeled program* $\mathcal{P}_\Delta$*, and a cipher* c *as input and outputs a message* m *if* c *is valid and* $\perp$ *if* c *is invalid, respectively.*

Definition 14 (Correctness).
Let $\mathsf{LAE} = (\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AEval}, \mathsf{AVerify}, \mathsf{ADecrypt})$ *be a LAEPuV scheme. We say* LAE *is* correct *if the following two conditions all hold.*

1. *For any key pair* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, k)$ *and any cipher* $c \in \mathcal{C}$ *we have*

$$\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 1 \Leftrightarrow \exists m \in \mathcal{M} : \mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c) = m.$$

2. *Let* $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, k)$ *be a key pair,* $\Delta \in \{0,1\}^*$ *be any dataset identifier,* $m_1,\dots,m_k \in \mathcal{M}$ *be a tuple of messages, and let* $c_i \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau_i, m_i)$*. For any admissible multi-labeled program* $\mathcal{P}_\Delta = ((f_1,\dots,f_k),\tau_1,\dots,\tau_k,\Delta)$ *it holds that*

$$\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, \mathsf{AEval}(\mathsf{pk}, \mathcal{P}_\Delta, \{c_i\}_{i=1}^k)) = f(m_1,\dots,m_k).$$

Note that in particular, if we have $\mathcal{P}_\Delta = \mathcal{I}_{(\Delta,\tau_i)}$ *the identity program, then* $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{I}_{(\Delta,\tau)}, c_i) = m_i$ *holds.*

We will give a security definition for a LAEPuV scheme in the Appendix (Definition15).

We will now show how our linearly homomorphic signature scheme can be used to instantiate such a LAEPuV scheme $\mathsf{LAE} = (\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AEval}, \mathsf{AVerify}, \mathsf{ADecrypt})$ when using bilinear groups of *composite order.* In [7] it is shown how to construct even asymmetric bilinear groups of composite order $n = pq$. Note that previous instantiations of LAEPuV schemes can only sign messages in $\mathbb{Z}_n$, i.e. vectors of length 1, while we show the first use of LAEPuV for vectors of *polynomial length.* Note again, that in this case the input identifiers are integers $i \in [k]$.

$\mathsf{AKeyGen}(1^\lambda, k, T)$**:** On input a security parameter λ, an integer k, and an integer T, it chooses two (safe) primes p, q and computes the modulus $n \leftarrow p \cdot q$. It runs $\mathcal{G}(1^\lambda)$ to obtain a bilinear group $\mathsf{bgp} = (n, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, e)$ of composite order and samples $k+T$ elements $R_1,\dots,R_k, h_1,\dots h_T \leftarrow \mathbb{G}_1$ uniformly at random. Additionally, the algorithm generates a key pair $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{KeyGen}'(1^\lambda)$ of a regular signature scheme and a key $K \xleftarrow{\$} \mathcal{K}$ for the pseudorandom function PRF. Furthermore it chooses an element $g \in \mathbb{Z}_{n^2}^*$ of order n as well as a hash function $H : \{0,1\}^* \to \mathbb{Z}_{n^2}^*$. It returns the key pair $(\mathsf{sk}, \mathsf{pk})$ with $\mathsf{sk} = (\mathsf{sk}', K, p, q)$ and $\mathsf{pk} = (\mathsf{bgp}, H, \mathsf{pk}', g, \{h_j\}_{j=1}^T, \{R_i\}_{i=1}^k)$.

$\mathsf{AEncrypt}(\mathsf{sk}, \Delta, i, m)$**:** On input a secret key sk, a dataset identifier Δ, an input identifier $i \in [k]$, and a message $m \in \mathbb{Z}_n^T$, it chooses β_j uniformly at random from $\mathbb{Z}_{n^2}^*$ for $j \in [T]$. It computes the cipher $C[j] \leftarrow g^{m[j]} \cdot \beta[j]^n \mod n^2$, computes $S[j] \leftarrow H(\Delta|i|j)$ and computes $(a[j], b[j]) \in \mathbb{Z}_n \times \mathbb{Z}_n^*$ such that $g^{a[j]} \cdot b[j]^n = C[j]S[j] \mod n^2$ using the factorization of n (see [21] for a detailed description). It generates the parameters for the dataset identified by Δ, by running $z \leftarrow \mathsf{PRF}_K(\Delta)$ and computing $Z = g_2^z$. It binds Z to the dataset identifier Δ by using the regular signature scheme, i.e. it sets $\sigma_\Delta \xleftarrow{\$} \mathsf{Sign}'(\mathsf{sk}', Z|\Delta)$. Then, it computes $\Lambda \leftarrow (R_i \cdot \prod_{j=1}^T h_j^{-a[j]})^z$ and returns the the cipher $c = (C, a, b, \sigma_\Delta, Z, \Lambda)$.

$\mathsf{AEval}(\mathsf{pk}, \mathcal{P}_\Delta, \{c_i\}_{i=1}^k)$**:** On input a public key pk, a multi-labeled program $\mathcal{P}_\Delta$, and a set of cipers c_i, it parses $\mathcal{P}_\Delta = ((f_1, \dots, f_k), 1, \dots, k, \Delta)$ and $c_i = (C_i, a_i, b_i, \sigma_{\Delta,i}, Z_i, \Lambda_i)$. If $Z_i \neq Z_1$ for any $i \in [k]$, it aborts. Otherwise, it sets

$$C \leftarrow \prod_{i=1}^{k} C_i^{f_i} \mod n^2 \qquad a \leftarrow \sum_{i=1}^{k} f_i a_i \mod n$$

$$b[j] \leftarrow \prod_{i=1}^{k} b_i[j]^{f_i} \mod n^2,\ \text{for } j \in [T] \qquad \Lambda \leftarrow \prod_{i=1}^{k} \Lambda_i^{f_i} \mod n$$

It returns the cipher $c = (C, a, b, \sigma_{\Delta,1}, Z_1, \Lambda)$.

$\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c)$**:** On input a public key pk, a multi-labeled program $\mathcal{P}_\Delta$, and a cipher c, it parses $\mathcal{P}_\Delta = ((f_1, ..., f_k), \tau_1, \dots, \tau_k, \Delta)$ and $c = (C, a, b, \sigma_\Delta, Z, \Lambda)$. The algorithm checks whether the following equations hold:
$\mathsf{Verify}'(\mathsf{pk}', Z|\Delta, \sigma_\Delta) = 1$, $e\left(R \cdot \prod_{j=1}^{T} h_j^{-a[j]}, Z\right) = e(\Lambda, g_2)$, and $g^{a[j]} \cdot b[j]^n = C[j] \prod_{i=1}^{k} H(\Delta|i|j)^{f_i} \mod n^2$. If all checks are satisfied, it returns $\mathtt{1}$. Otherwise, it returns $\mathtt{0}$.

$\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c)$**:** Returns $\perp$ if $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 0$. Otherwise, compute (m, β) such that $g^{m[j]}\beta[j]^n = C[j] \mod n^2$ and return m.

We will formally show the correctness of LAE in Theorem 8 in the Appendix.

Theorem 6. *([14]). In the random oracle model, if the DCR Assumption (see Definition 12)and the CDH Assumption (see Definition 11) hold and H is a random oracle the LAEPuV scheme* LAE *is a LH-IND-CCA secure (see Definition 15) LAEPuV scheme.*

Proof. This is a direct corollary of [14, Theorem 1] and Theorem 2.

5 Conclusion

We provide a new linearly homomorphic signature scheme directly based on the CDH assumption, without using a chameleon hash function thereby solving the problem introduced in [17]. Additionally we provide the first LAEPuV scheme that supports vectors as inputs and hereby give an alternative to the instantiation provided in [23]. Our construction achieves two additional properties, that are constant time verification and context hiding. It would be interesting to see if the security of homomorphic schemes supporting a larger class of computations can also be based on such well studied assumptions.

A Appendix: Postponed Proofs

Lemma 2. *An efficient adversary $\mathcal{A}$ winning Game 4 in Theorem 2, can be used to break the CDH assumption.*

Proof. We will now show how to construct a simulator $\mathcal{S}$ which uses an efficient adversary $\mathcal{A}$ against Game 4 to solve the *CDH* problem. Let $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathcal{G}(1^\lambda)$ be a bilinear group of order q. The simulator $\mathcal{S}$ is given g_1, g_1^a, g_2^b, where $a, b \xleftarrow{\$} \mathbb{Z}_q$, and intends to compute g_1^{ab}.

Initialization: Let Q be the number of datasets in which the adversary makes signature queries. The adversary gives the simulator all messages $\{m_{(i,l)}\}_{i=1}^k$, for $l \in [Q]$ on which he makes signature queries.

Setup: The simulator runs the key generation algorithm of the regular signature scheme to obtain a key pair $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{KeyGen}'(1^\lambda)$ and samples a key $K \xleftarrow{\$} \mathcal{K}$ for the pseudorandom function PRF. The simulator guesses the dataset in which the adversary produces a forgery, in the following identified by the dataset identifier Δ. Then, it chooses $r_i \xleftarrow{\$} \mathbb{Z}_q$ for $i \in [k]$ as well as $s_j \xleftarrow{\$} \mathbb{Z}_q$ for $j \in [T]$. It sets $R_i \leftarrow g_1^{r_i} \cdot g_1^{a \cdot \sum_{j=1}^T m_{(i,\Delta)}[j]}$, sets $h_j = (g_1^a)^{s_j}$, and sends the public key $\mathsf{pk} = (\mathsf{pk}', g_1, \{R_i\}_{i=1}^k, \{h_j\}_{j=1}^T)$ to the adversary. Note that since the s_j and r_i are chosen uniformly at random this is perfectly indistinguishable from an honest setup.

Query: While the adversary queries signatures for messages, we distinguish between the following two cases.

- Case I: The adversary queries signatures for the dataset $\Delta_l \neq \Delta$.
- Case II: The adversary queries signatures for the dataset $\Delta_l = \Delta$.

Case I: In this case, the simulator answers the signing queries by the adversary with honestly generated signatures. More precisely, let $m_1, \ldots, m_k$ be the messages and Δ_l be the dataset identifier. The simulator computes $z \leftarrow \mathsf{PRF}_K(\Delta_l)$, sets $Z = g_2^z$, and $\sigma_\Delta \leftarrow \mathsf{Sign}'(\mathsf{sk}', \Delta_l | Z)$. Then, for any $i \in [k]$, it computes $\Lambda_i \leftarrow (R_i \cdot \prod_{j=1}^T h_j^{-m_i[j]})^z$ and returns the set of signatures $\boldsymbol{\sigma} = \{(\sigma_{\Delta_l}, Z, \Lambda_i)\}_{i=1}^k$. The validity of the signatures can be easily verified.

Case II: In this case the adversary $\mathcal{A}$ queries signatures in the dataset the simulator expects $\mathcal{A}$ to produce a forgery in. It chooses $u \leftarrow \mathbb{Z}_q$ uniformly at random and sets $Z = (g_2^b)^u$. Let $m_1, \ldots, m_k$ be the messages and Δ be the dataset identifier. The simulator sets $\sigma_\Delta \leftarrow \mathsf{Sign}'(\mathsf{sk}', Z | \Delta)$. Then, for $i =\in [k]$, the simulator computes $\Lambda_i \leftarrow \left(g_2^b\right)^{ur_i}$ and returns the signatures $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_k)$, where $\sigma_i = (\sigma_\Delta, Z, \Lambda_i)$. Note that for any $i = 1, \dots, k$, it holds that

$$\begin{aligned} &e(R_i \cdot \prod_{j=1}^{T} h_j^{-m_i[j]}, Z) = e(g_1^{r_i} \cdot g_1^{a \cdot s_j \cdot m_i[j]} \cdot (g_1^{a \cdot s_j})^{-m_i[j]}, g_2^{ub}) \\ &= e(g_1^{r_i}, g_2)^{ub} = e((g_1^{ur_i})^b, g_2) = e(\Lambda, g_2) \end{aligned}$$

Thus, σ_i is a valid signature for any $i \in [k]$ and the simulated signatures are perfectly indistinguishable from honestly generated signatures.

Challenge Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by the adversary $\mathcal{A}$. Parse $\sigma^* = (\sigma^*_{\Delta^*}, Z^*, \Lambda^*))$ and $\mathcal{P}^*_{\Delta^*} = (f^*, 1, \dots, k, \Delta^*)$. If $\Delta^* \neq \Delta$, restart the simulation. Otherwise, the simulator evaluates the function f^* over the dataset identified by Δ, i.e. it computes $m \leftarrow f^*(m_1, ..., m_k)$ and $\sigma = (\sigma_\Delta, Z, \Lambda) \leftarrow \mathsf{HEval}(\mathsf{pk}, \Delta, \boldsymbol{\sigma}, f^*)$. Note that we have $\prod_{j=1}^{T} h_j^{m[j]} \neq \prod_{j=1}^{T} h_j^{m^*[j]}$, since $\mathsf{bad}_4 = \mathsf{false}$ and therefore also $\sum_{j=1}^{T} s_j \cdot m[j] \neq \sum_{j=1}^{T} s_j \cdot m^*[j]$. It returns
$(\Lambda \cdot (\Lambda^*)^{-1})^{(\sum_{j=1}^{T} s_j (m^*[j] - m[j]))^{-1}}$ as a solution. Let $R \leftarrow \prod_{i=1}^{k} R_i^{f_i}$. Since Λ and Λ^* are valid signatures for the function f^*, it holds that

$$\begin{aligned} \Lambda &= \left(R \cdot \prod_{j=1}^{T} h_j^{-m[j]} \right)^{ub} = \left(R^{ub} \cdot (g_1^a)^{-ub(\sum_{j=1}^{T} s_j m[j])} \right) \\ &= R^{ub} \cdot g_1^{-(\sum_{j=1}^{T} s_j m[j]) uab} \\ \Lambda^* &= \left(R \cdot \prod_{j=1}^{T} h_j^{-m^*[j]} \right)^{ub} \\ &= R^{ub} (g_1^a)^{-ub(\sum_{j=1}^{T} s_j m^*[j])} = R^{ub} g_1^{-(\sum_{j=1}^{T} s_j m^*[j]) uab} \end{aligned}$$

Therefore, we have

$$\begin{aligned} &\Lambda \cdot (\Lambda^*)^{-1} = (R^{ub} \cdot g_1^{-u(\sum_{j=1}^{T} s_j \cdot m[j])ab}) \cdot (R^{-ub} \cdot g_1^{u(\sum_{j=1}^{T} s_j \cdot m^*[j])ab}) \\ &= g_1^{(\sum_{j=1}^{T} s_j \cdot m^*[j]) uab} \cdot g_1^{-(\sum_{j=1}^{T} s_j \cdot m[j]) uab} = g_1^{ab(u \sum_{j=1}^{T} s_j (m^*[j] - m[j]))} \end{aligned}$$

which yields

$$(\Lambda \cdot (\Lambda^*)^{-1})^{\frac{1}{u \sum_{j=1}^{T} s_j (m^*[j] - m[j])}} = (g_1^{ba})^{\frac{u \sum_{j=1}^{T} s_j (m^*[j] - m[j])}{u \sum_{j=1}^{T} s_j (m^*[j] - m[j])}} = g_1^{ab}$$

Since the simulator guesses the right dataset with probability at least $1/Q$, it holds that

$$\mathsf{Pr}[\mathbf{Adv}(\mathcal{S})] \geq \frac{1}{Q} \cdot \mathsf{Pr}[G_4(\mathcal{A})]$$

which proves the statement.

Lemma 3. *Assuming the DL assumption holds in $\mathbb{G}_1$ then $Pr[\mathsf{Bad}_4] \leq \mathsf{negl}(\lambda)$*

Proof. Given $g_1, g_1' \in \mathbb{G}_1$ from bgp we show how to simulate the game in order to break the discrete logarithm in $\mathbb{G}_1$, i.e. computing x for $g_1' = g_1^x$. The simulator chooses an index $\nu \in [T]$. It follows the protocol faithfully except for the generation of the h_j. It chooses $s_j \xleftarrow{\$} \mathbb{Z}_q$. and sets $h_j = g_1^{s_j}$ for all $j \neq \nu$ and sets $h_\nu = {g_1'}^{s_\nu}$. This is perfectly indistinguishable from a real execution of the game. It answers all queries faithfully. When the adversary returns a forgery $(\mathcal{P}^*{}_{\Delta^*}, m^*, \sigma^*)$ it checks whether $m[\nu] \neq m^*[\nu]$. If not it restarts the simulation. Otherwise we know that $\prod_{j=1}^T h_j^{m[j]} = \prod_{j=1}^T h_j^{m^*[j]}$ and therefore we have

$$s_\nu m[\nu]x + \sum_{j=1, j\neq\nu}^{T} s_j m[j] = s_\nu m^*[\nu]x + \sum_{j=1, j\neq\nu}^{T} s_j m^*[j]$$
$$\Leftrightarrow x = \frac{1}{s_\nu(m[\nu] - m^*[\nu])} \sum_{j=1, j\neq\nu}^{T} s_j(m^*[j] - m[j])$$

and found the discrete logarithm $g_1^x = g_1'$.

Theorem 7. *The linearly homomorphic signature scheme* HSig *is perfectly context hiding according to Definition 7 if* Sig' *is a deterministic signature scheme.*

Proof. First we note that in our case the algorithm HHide is just the identity function, i.e. $\sigma \leftarrow \mathsf{HHide}(\mathsf{pk}, m, \sigma)$ for all pk, m, σ and we have $\mathsf{HHideVer} = \mathsf{HVerify}$. We will show how to construct a simulator Sim that outputs signatures perfectly indistinguishable from the ones obtained by running HEval. Parse the simulator's input as $\mathsf{sk} = (\mathsf{sk}', K)$, $\mathcal{P}_\Delta = ((f_1, \ldots, f_k), 1, \ldots, k, \Delta)$, and $\tilde{m} = (\tilde{m}[1], \ldots, \tilde{m}[T])$. With this information the simulator computes the following:

$Z' = g_2^z$ where $z \leftarrow \mathsf{PRF}_K(\Delta)$
$\sigma'_\Delta \xleftarrow{\$} \mathsf{Sign}'(\mathsf{sk}', Z|\Delta)$
$\Lambda' = (\prod_{i=1}^k R_i^{f_i} \cdot \prod_{j=1}^T h_j^{-m[j]})^z$

The simulator outputs the signature $\sigma' = (\sigma'_\Delta, Z', \Lambda')$.

We will now show that this simulator allows for perfectly context hiding security. We will fix an arbitrary key pair $(\mathsf{sk}, \mathsf{pk})$, a multi-labeled program $((f_1, \dots, f_k), 1, \dots, k, \Delta)$, and messages $m_1, \dots, m_k \in \mathbb{Z}_q^T$.
Let $\sigma \leftarrow \mathsf{HEval}(\mathsf{pk}, \mathcal{P}_\Delta, \boldsymbol{\sigma})$ and parse it as $\sigma = (\sigma_\Delta, Z, \Lambda)$.
We look at each component of the signature.
We have $Z = \mathsf{PRF}_K(\Delta)$ by definition and therefore also $Z = Z'$. In particularly we also have $z = z'$ where $Z = g_2^z$ and $Z' = g_2^{z'}$.
We have $\sigma_\Delta = \mathsf{Sign}'(\mathsf{sk}', Z|\Delta)$ by definition and since $Z = Z'$ therefore also $\sigma_\Delta = \sigma'_\Delta$ since Sign' is deterministic.
We have
$\Lambda = \prod_{i=1}^k (R_i \cdot \prod_{j=1}^T h_j^{-m_i[j]})^{z \cdot f_i} = (\prod_{i=1}^k R_i^{f_i})^z \cdot (\prod_{j=1}^T \prod_{i=1}^k h_j^{-f_i \cdot m_i[j]})^z = (\prod_{i=1}^k R_i^{f_i} \cdot \prod_{j=1}^T h_j^{-m[j]})^z$, where the last equation holds since $m = \sum_{i=1}^k f_i \cdot m_i$. Thus we also have $\Lambda = \Lambda'$.
We can see that we have *identical* elements and therefore even a computationally unbounded distinguisher has no advantage distinguishing the two cases.

Definition 15 (LH-IND-CCA [13]).
Let $\mathsf{LAE} = (\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AEval}, \mathsf{AVerify}, \mathsf{ADecrypt})$ *be a LAEPuV scheme. We define the following experiment* $LH-IND-CCA_{\mathcal{H},\mathcal{A}}(1^\lambda, k)$ *between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{A}$*:*

Setup: *The challenger runs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, k)$*. Then it initializes an empty list* $\mathcal{L}$ *and gives* pk *to the adversary* $\mathcal{A}$*.*

Queries I: $\mathcal{A}$ *can ask a polynomial number of both encryption and decryption queries. The former are of the form* (m, Δ, τ) *where* $m \in \mathcal{M}$ *is a message,* $\Delta \in \{0,1\}^*$ *is a dataset identifier, and* $\tau \in \mathcal{T}$ *is an input identifier. The challenger computes* $c \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m)$*, gives* c *to* $\mathcal{A}$ *and updates the list* $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, \Delta, \tau)\}$*. If* $\mathcal{L}$ *already contains a query* $(\cdot, \Delta, \tau)$ *the challenger* $\mathcal{C}$ *will answer* $\perp$*.*
The latter queries are of the form $(\mathcal{P}_\Delta, c)$ *and* $\mathcal{A}$ *receives the output of* $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c)$*. Note that this can be* $\perp$ *if* c *is not a valid cipher.*

Challenge: $\mathcal{A}$ *produces a challenge tuple* $(m_0, m_1, \Delta^*, \tau^*)$*. If a query of the form* $(\cdot, \Delta^*, \tau^*)$ *is contained in* $\mathcal{L}$*, the challenger returns* $\perp$ *as before. The challenger chooses a random bit* $b \xleftarrow{\$} \{0,1\}$ *and gives* $c^* \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta^*, \tau^*, m_b)$ *to* $\mathcal{A}$*. Then it updates the list* $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m_b, \Delta^*, \tau^*)\}$*.*

Queries II: *This phase is carried out similar to the Queries I phase. Any decryption query* $(\mathcal{P}_{\Delta^*}, c)$ *with* $\mathcal{P}_{\Delta^*} = ((f_1, \dots, f_k), \tau_1, \dots, \tau_k, \Delta^*)$

where $f_{\tau^*} \neq 0$ is answered with $\perp$. All other queries are answered as in phase Queries I.

Output: *Finally $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. The challenger outputs* $\mathsf{1}$*if $b = b'$ and* $\mathsf{0}$*otherwise.*

We say that a LAEPuV scheme is LH-IND-CCA secure if for any ppt adversary $\mathcal{A}$ we have

$$|Pr[LH - IND - CCA_{\mathsf{LAE},\mathcal{A}}(1^\lambda, k) = 1] - 1/2| \leq \mathsf{negl}(\lambda).$$

Theorem 8. *The LAEPuV scheme* LAE *is correct in the sense of Definition 14.*

Proof. We fix a random key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, k, T)$,
with $\mathsf{sk} = (\mathsf{sk}', K, p, q)$ and $\mathsf{pk} = (\mathsf{bgp}, H, \mathsf{pk}', g, \{h_j\}_{j=1}^T, \{R_i\}_{i=1}^k)$.

1. If $g \in \mathbb{Z}_{n^2}^*$ has order n then the map: $\mathbb{Z}_n \times \mathbb{Z}_n^* \to \mathbb{Z}_{n^2}^*$, $(a,b) \mapsto g^a \cdot b^n$ is an isomorphism (see [21]). If $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 1$ holds then we have in particular $g^{a[j]} \cdot b[j]^n = C[j] \prod_{i=1}^k H(\Delta|i|j)^{f_i} \mod n^2$, where each $g^{a[j]} \cdot b[j]^n$ and $H(\Delta|i|j)^{f_i}$ is an element of $\mathbb{Z}_{n^2}^*$. Since this is a group so is every $C[j]$ which means every Paillier decryption yields a valid message m.
2. We choose messages $m_i \xleftarrow{\$} \mathbb{Z}_n^T$ as well as a dataset identifier $\Delta \in \{0,1\}^*$ and a multi-labeled program $\mathcal{P}_\Delta = ((f_1, \dots, f_k), \tau_1, \dots, \tau_k, \Delta)$. Let $c_i \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta, i, m_i)$ and $c \leftarrow \mathsf{AEval}(\mathsf{pk}, \mathcal{P}_\Delta, \{c_i\}_{i=1}^k)$.
By definition we have $c = (C, a, b, \sigma_\Delta, Z, \Lambda)$. Where for each $j \in [T]$ we have

$$C[j] = \prod_{i=1}^k \left(g^{m_i[j]} \beta_i[j]^n\right)^{f_i} = g^{\sum_{i=1}^k f_i m_i[j]} \left(\prod_{i=1}^k \beta_i[j]^{f_i}\right)^n \mod n^2$$

$$g^{a[j]} \cdot b[j]^n = g^{\sum_{i=1}^k f_i a_i[j]} \cdot \left(\prod_{i=1}^k b_i[j]^{f_i}\right)^n = \prod_{i=1}^k \left(C[j]^{f_i} \cdot H(\Delta|i|j)^{f_i}\right)$$

$$= C \cdot \prod_{i=1}^k (H(\Delta|i|j)^{f_i}) \mod n^2$$

$$z = \mathsf{PRF}_K(\Delta),\ Z = g_2^z,\ \sigma_\Delta = \mathsf{Sign}'(\mathsf{sk}', Z|\Delta)$$

$$\Lambda = \prod_{i=1}^k \Lambda_i^{f_i} = \left(\prod_{i=1}^k R_i^{f_i} \cdot \prod_{j=1}^T h_j^{-\sum_{i=1}^k f_i a_i[j]}\right)^z = \left(R \cdot \prod_{j=1}^T h_j^{-a[j]}\right)^z$$

Therefore we have $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, C) = 1$ and due to the first equation Paillier decryption of $C[j]$ yields $\sum_{i=1}^k f_i m_i[j]$ for each $j \in [T]$.

References

1. An, J.H., Bellare, M.: Does encryption with redundancy provide authenticity? In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 512–528. Springer (2001)
2. Attrapadung, N., Libert, B.: Homomorphic Network Coding Signatures in the Standard Model. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 17–34. Springer (2011)
3. Attrapadung, N., Libert, B., Peters, T.: Computing on Authenticated Data: New Privacy Definitions and Constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 367–385. Springer (2012)
4. Attrapadung, N., Libert, B., Peters, T.: Efficient Completely Context-Hiding Quotable and Linearly Homomorphic Signatures. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 386–404. Springer (2013)
5. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) ACM CCS. pp. 863–874. ACM (2013)
6. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. J. Cryptology 21(4), 469–491 (2008), http://dx.doi.org/10.1007/s00145-008-9026-x
7. Boneh, D., Rubin, K., Silverberg, A.: Finding composite order ordinary elliptic curves using the cocks–pinch method. Journal of Number Theory 131(5), 832 – 841 (2011), http://www.sciencedirect.com/science/article/pii/S0022314X10001344
8. Boneh, D., Freeman, D.M.: Linearly Homomorphic Signatures over Binary Fields and New Tools for Lattice-Based Signatures. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 1–16. Springer (2011)
9. Boneh, D., Freeman, D.M., Katz, J., Waters, B.: Signing a Linear Subspace: Signature Schemes for Network Coding. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 68–87. Springer (2009)
10. Catalano, D., Fiore, D., Nizzardo, L.: Programmable Hash Functions Go Private: Constructions and Applications to (Homomorphic) Signatures with Shorter Public Keys. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 254–274. Springer (2015)
11. Catalano, D., Fiore, D., Warinschi, B.: Efficient Network Coding Signatures in the Standard Model. In: Fischlin, M., Buchmann, J.A., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 680–696. Springer (2012)
12. Catalano, D., Fiore, D., Warinschi, B.: Homomorphic Signatures with Efficient Verification for Polynomial Functions. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 371–389. Springer (2014)
13. Catalano, D., Marcedone, A., Puglisi, O.: Authenticating computation on groups: New homomorphic primitives and applications. Cryptology ePrint Archive, Report 2013/801 (2013), http://eprint.iacr.org/2013/801
14. Catalano, D., Marcedone, A., Puglisi, O.: Authenticating computation on groups: New homomorphic primitives and applications. In: Sarkar and Iwata [22], pp. 193–212
15. Chatterjee, S., Hankerson, D., Knapp, E., Menezes, A.: Comparing two pairing-based aggregate signature schemes. Des. Codes Cryptography 55(2-3), 141–167 (2010), https://doi.org/10.1007/s10623-009-9334-7
16. Desmedt, Y.: Computer security by redefining what a computer is. In: Michael, J.B., Ashby, V., Meadows, C.A. (eds.) NSPW. pp. 160–166. ACM (1993)

17. Freeman, D.M.: Improved Security for Linearly Homomorphic Signatures: A Generic Framework. In: Fischlin, M., Buchmann, J.A., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 697–714. Springer (2012)
18. Gennaro, R., Katz, J., Krawczyk, H., Rabin, T.: Secure Network Coding over the Integers. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 142–160. Springer (2010)
19. Johnson, R., Molnar, D., Song, D.X., Wagner, D.A.: Homomorphic Signature Schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer (2002)
20. Joo, C., Yun, A.: Homomorphic authenticated encryption secure against chosen-ciphertext attack. In: Sarkar and Iwata [22], pp. 173–192
21. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT '99. LNCS, vol. 1592, pp. 223–238. Springer (1999)
22. Sarkar, P., Iwata, T. (eds.): ASIACRYPT 2014 - Part II, LNCS, vol. 8874. Springer (2014)
23. Struck, P., Schabhüser, L., Demirel, D., Buchmann, J.A.: Linearly homomorphic authenticated encryption with provable correctness and public verifiability. In: Hajji, S.E., Nitaj, A., Souidi, E.M. (eds.) C2SI 2017. LNCS, vol. 10194, pp. 142–160. Springer (2017)