Linearly Homomorphic Structure Preserving Signatures: New Methodologies and Applications

Dario Catalano[1], Antonio Marcedone[2,⋆], and Orazio Puglisi[1]

[1] University of Catania, Catania, Italy,
[2] Scuola Superiore di Catania, University of Catania, Catania, Italy

Abstract At Crypto 2013 Libert, Peters, Joye and Yung introduced the notion of Linearly Homomorphic Structure Preserving Signatures (LHSPS) as a tool to perform verifiable computation on encrypted data and to create constant-size non malleable commitments to group elements. In this paper we improve our understanding of LHSPS by putting forward new methodologies and applications. First, we present a generic transform that converts LHSPS which are secure against weak random message attack (RMA) into ones that achieve full security guarantees. Next we give evidence that RMA secure linearly homomorphic structure preserving signatures are interesting in their own right by showing applications in the context of on-line/off-line homomorphic and network coding signatures. This notably provides what seems to be the first instantiations of homomorphic signatures achieving on-line/off-line efficiency trade-offs.

⋆ Part of the work done while visiting Aarhus University.

1 Introduction

Structure Preserving cryptography provides a simple and elegant methodology to compose algebraic tools within the framework of Groth Sahai proof systems [36]. In the last years, this methodology has been widely used to design simple and modular realizations of cryptographic protocols and primitives. These include structure preserving signatures (SPS) [6,3,4,1,2,16,23,24,30,35,37], commitments [34,7] and encryption schemes [17]. Very recently Libert *et al.* [41] further enlarged this set of primitive by introducing and realizing the notion of *Linearly Homomorphic Structure Preserving signatures* (LHSPS for short). Informally LHSPS are like ordinary SPS but they come equipped with a linearly homomorphic property that makes them interesting even beyond their usage within the Groth Sahai framework. In particular Libert *et al.* showed that LHSPS can be used to enable simple verifiable computation mechanisms on encrypted data. More surprisingly, they observed that linearly homomorphic SPS (generically) yield efficient simulation sound trapdoor commitment schemes [31], which in turn imply non malleable trapdoor commitments [27] to group elements.

Our Contribution. In this paper we improve our understanding of linearly homomorphic structure preserving signatures by proposing new constructions and applications for this primitive. More precisely, our contributions are as follows. First, we put forward an efficient and *generic* methodology to convert LHSPS which are secure against a very weak *random* message attack into ones that achieve full security. Specifically, by random message security here we mean that the unforgeability guarantee holds only with respect to adversaries that are allowed to see signatures corresponding to messages randomly chosen by the signer. We stress that while similar transforms were known for structure preserving signatures (e.g. [25]), to our knowledge this is the first such transform for the case of *linearly homomorphic* signatures in general. On a technical level, we note that the main difficulty in proving this result comes from the fact that we require it to hold in the strong security model introduced by Freeman in [29].

As a second contribution of this paper we show how to construct a very simple LHSPS which is secure against random message attack (RMA). Our construction relies on a variant of the Computational Diffie-Hellman assumption introduced by Kunz-Jacques and Pointcheval in [40]. Our construction is less general (but also conceptually simpler) than that given in [41]. Indeed, while the latter allows to sign vectors of arbitrary dimension, ours allow to sign only scalars (i.e. vectors composed by one single component).

Perhaps surprisingly, however, we show that this simple scalar construction has useful applications in the context of *on-line/off-line* (homomorphic) signatures. Very informally, on-line/off-line signatures allow to split the cost of signing in two phases. An (expensive) offline phase that can be carried out *without* needing to know the message m to be signed and a much more efficient on-line phase that is done once m becomes available. In this sense, we show that RMA-secure linearly homomorphic SPS naturally fit the on-line/off-line scenario. Specifically, we prove that if one combines a RMA-secure LHSPS (to sign scalars) with a Σ protocols with some specific homomorphic properties, one gets a fully fledged linearly homomorphic signature achieving a very efficient online phase. Moreover, since the resulting signature scheme supports vectors of arbitrary dimensions as underlying message space, our results readily generalize to the case of network coding signatures [12]. More concretely, by combining our RMA-secure scheme together with (a variant of) Schnorr's identification protocol

we get what seems to be the first constructions of strongly secure homomorphic and network coding signatures offering online/offline efficiency tradeoffs.

1.1 Other Related Work

Structure Preserving Signature (SPS). Structure preserving signatures were introduced (with a different terminology) in 2006 by Groth [35] as a tool to realize group signatures in the standard model. The scheme from [35], is mainly of theoretical interest as each signature consists of thousands of group elements. More efficient realizations were provided by Cathalo, Libert and Yung [23] and by Fuchsbauer [30]. In 2011 Abe, Groth, Haralambiev, Ohkubo [4] proved that any secure SPS scheme using asymmetric bilinear groups must consist of at least 3 group elements. Moreover they proposed a realization matching this lower bound. Later Abe, Groth an Ohkubo in [5] prove that any SPS scheme matching this lower bound must be based on some interactive assumptions. Constructions based on the much more standard decision linear assumption [11] were later given in [37,16,24]. More recently Abe *et al.* [1,2] managed to realize constant size constructions under simple assumptions.

Over the last years SPS have been used in different application [17,34]. Recently Sakai *et al.* in [42] use SP Identity Based Encryption to obtain a new property called $message - dependent\ opening$, that restricts the power of the opening authority.

Linearly homomorphic signature The concept of homomorphic signature scheme was originally introduced in 1992 by Desmedt [26], and then refined by Johnson, Molnar, Song, Wagner in 2002 [38]. A very useful class of homomorphic signatures are linearly homomorphic ones, introduced in 2009 by Boneh *et al.* [12] as a way to prevent pollution attacks in network coding. Linearly homomorphic signatures allow one to sign linear functions of signed data without needing to know the secret signing key. Following the original construction from [12] many other works further explored the notion of homomorphic signatures by proposing new frameworks and realizations [32,8,14,13,21,9,22,29,10,20]. In the symmetric setting constructions of homomorphic message authentication codes have been proposed by [12,33,19]

On-line/Off-line Signatures. On-Line/Off-Line digital signature were introduced by Even, Goldreich and Micali in [28]. In such schemes the signature process consists of two parts: a computationally intensive one that can be done Off-Line (i.e. when the message to be signed is not known) and a much more efficient online phase that is done once the message becomes available. There are two general ways to construct on-line/off-line signatures: using one time signatures [28] or using chameleon hash [44]. In [18] Catalano *et al.*, unified the two approaches by showing that they can be seen as different instantiations of the same paradigm.

2 Preliminaries and notation

We denote with $\mathbb{Z}$ the set of integers, with $\mathbb{Z}_p$ the set of integers modulo p. An algorithm $\mathcal{A}$ is said to be PPT if it's modelled as a probabilistic Turing machine that runs in polynomial time in its inputs. If S is a set, then $x \xleftarrow{\$} S$ denotes the process of selecting one element x from S uniformly at random. A function f is said to be negligible if for all polynomial p there exists $n_0 \in \mathbb{N}$ such that for each $n > n_0$

$$|f(n)| < \frac{1}{p(n)}$$

2.1 Computational assumptions

We recall below a few well known computational assumptions.
Let $\mathbb{G}$ be a finite (multiplicative) group of prime order p.

Definition 1 (CDH). We say that the Computational Diffie-Hellman assumption holds in $\mathbb{G}$ if, given a random generator $g \in \mathbb{G}$, there exists no PPT $\mathcal{A}$ that on input g, g^a, g^b outputs g^{ab} with more than negligible probability. Here the probability is taken over the uniform choices of $a, b \xleftarrow{\$} \mathbb{Z}_p$ and the internal coin tosses of $\mathcal{A}$.

The 2-out-of-3 Computational Diffie-Hellman assumption was introduced by Kunz-Jacques and Pointcheval in [40] as a relaxation of the standard CDH assumption. It is defined as follows.

Definition 2 (2-3CDH). We say that the 2-out-of-3 Computational Diffie-Hellmann assumption holds in $\mathbb{G}$ if, given a random generator $g \in \mathbb{G}$, there exists no PPT $\mathcal{A}$ that on input (g, g^a, g^b) (for random $a, b \xleftarrow{\$} \mathbb{Z}_p$) outputs h, h^{ab} $(h \neq 1)$ with more than negligible probability.

2.2 Linearly Homomorphic Structure Preserving Signatures Schemes

Following [41], we define linearly homomorphic structure preserving signatures by adapting to the structure preserving context the definition of linearly homomorphic signature scheme. In particular we assume that the message space is some multiplicative group $\mathcal{M}$ and

- We use as set of functions $\mathcal{F}$ the set of linear combinations of elements of the group, so each function $f \in \mathcal{F}$ can be uniquely expressed as $f(m_1, \ldots, m_k) = \prod_{i=1}^{k} m_i^{\alpha_i}$, and therefore can be identified by a proper vector $(\alpha_1, \ldots, \alpha_k) \in \mathbb{Z}^k$.
- We identify each dataset by a string fid, and use an additional argument $i \in \{1, \ldots, n\}$ for the signing algorithm to specify that the signed message can be used only as the i-th argument for each function $f \in \mathcal{F}$.

As a technical note we stress that the following definitions are slightly different with respect to those given in [41]. This is to accommodate the fact that our constructions focuses on signing vectors rather than subspaces as in [41]. Formally, we have the following:

Definition 3 (LHSPSS). A Linearly homomorphic structure-preserving signature scheme is a tuple of PPT algorithms (**KeyGen**, **Sign**, **Verify**, **Eval**) such that:

- **KeyGen**$(1^\lambda, n, k)$ takes as input the security parameter λ, an integer n denoting the length of vectors to be signed and an upper bound k for the number of messages signed in each dataset. It outputs a secret signing key sk and a public verification key vk; the public key implicitly defines a message space of the form $\mathcal{M} = \mathbb{G}^n$, a file identifier space $\mathcal{D} = \{0,1\}^{n_d}$ and a signature space $\Sigma \subseteq \mathbb{G}^{n_s}$, for some $n_s, n_d \in poly(\lambda)$.
- **Sign**$(\mathrm{sk}, m, \mathrm{fid}, i)$ takes as input the secret key, an element $m \in \mathcal{M}$, a dataset identifier fid, an index $i \in \{1, \ldots, k\}$ and outputs a signature σ.
- **Verify** (vk, σ, m, fid, f) takes as input the pubic key vk, a signature $\sigma \in \Sigma$, a message $m \in \mathcal{M}$ a dataset identifier fid $\in \mathcal{D}$ and a function $f \in \mathcal{F}$ and outputs 1 (accept) or 0 (reject).

– **Eval** (vk, fid, f, $\{\sigma_i\}_{i=1\dots k}$) takes as input the public key vk, a dataset identifier fid, an admissible function f in its vector form $(\alpha_1, \dots, \alpha_k)$, a set of k signatures $\{\sigma_i\}_{i=1\dots k}$ and outputs a signature $\sigma \in \Sigma$. Note that this algorithm should also work if less than k signatures are provided, as long as their respective coefficients in the function f are 0, but we avoid to explicitly account this to avoid heavy notation.

The correctness conditions of our scheme are the following:

– Let $(\text{sk}, \text{vk}) \leftarrow \mathbf{KeyGen}(1^\lambda, n, k)$ be an honestly generated keypair, $m \in \mathcal{M}$, fid any dataset identifier and $i \in 1, \dots, k$. If $\sigma \leftarrow \mathbf{Sign}(\text{sk}, m, \text{fid}, i)$, then with overwhelming probability

$$\mathbf{Verify}(\text{vk}, \sigma, m, \text{fid}, e_i) = 1,$$

where e_i is the i-th vector of the standard basis of $\mathbb{Z}^k$.

– Let $(\text{sk}, \text{vk}) \leftarrow \mathbf{KeyGen}(1^\lambda, n, k)$ be an honestly generated keypair, $m_1, \dots, m_k \in \mathcal{M}$ any tuple of messages signed w.r.t the same fid, and let $\sigma_1, \dots, \sigma_k \in \Sigma$, $f_1, \dots, f_k \in \mathcal{F}$ such that for all $i \in \{1, \dots, k\}$, $\mathbf{Verify}(\text{vk}, \sigma_i, m_i, \text{fid}, f_i) = 1$. Then, for any admissible function $f = (\alpha_1, \dots, \alpha_k) \in \mathbb{Z}^k$, with overwhelming probability

$$\mathbf{Verify}(\text{vk}, \mathbf{Eval}(\text{vk}, \text{fid}, f, \{\sigma_i\}_{i=1\dots k}), f(m_1, \dots, m_k), \text{fid}, \sum_{i=0}^{k} \alpha_i f_i) = 1$$

Security Roughly speaking, a LHSPSS is said to be secure if no PPT adversary $\mathcal{A}$ can produce with more than negligible probability one of the following:

– A signature for a message w.r.t. a new fid (i.e. one that it has never seen before)
– A signature w.r.t. a previously seen identifier, for a message m different from the one obtained applying the claimed function f to the previously signed messages of the same dataset
– A signature it has not seen but that has been used in the **Eval** algorithm to compute signatures it has seen (under certain independence constraints, see the formal definition for details).

We distinguish between notions where the adversary has no control over the signed messages he can see, and the standard one where he can adaptively choose them by itself.

Definition 4 (Random message attack security). An LHSPSS is unforgeable against a random message attack if for all n the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible in the security parameter λ:
Setup The challenger runs $\mathbf{KeyGen}(1^\lambda, n, k)$ and gives vk to $\mathcal{A}$. The message space $\mathcal{M}$, the signature space Σ and the dataset space $\mathcal{D}$ are all implicitly defined by the verification key.
Queries $\mathcal{A}$ can ask a polynomial number of queries of the following types:

– **Signing Queries** $\mathcal{A}$ asks for a new message/signature couple w.r.t. to a specific fid $\in \mathcal{D}$ and a specific index $i \in \{1, \dots, k\}$. The challenger checks that this query has not been previously answered (otherwise it returns $\perp$), then it picks a random message $m \xleftarrow{\$} \mathcal{M}$ and uses the secret key sk to compute a signature σ for m w.r.t.

fid and the index i. Finally it picks a handle h (from a proper set of identifiers), stores $(h, (\text{fid}, m, \sigma, e_i))$ in a table T and returns h to $\mathcal{A}$. Note that $\mathcal{A}$ does not see neither the message nor the signature, and that here $e_i \in \mathbb{Z}^k$ is the i-th vector of the canonical basis, used to indicate the (trivial) function with respect to which the signature has been issued.

- **Derivation Queries** $\mathcal{A}$ chooses a set of handles $h = (h_1, \ldots, h_k)$ and a vector of coefficients $f = (\alpha_1, \ldots, \alpha_k)$. The challenger retrieves $\{(h_i, (\text{fid}_i, m_i, \sigma_i, f_i))\}_{i=1,\ldots k}$ from T and returns $\perp$ if any of these does not exist or if $\text{fid}_i \neq \text{fid}_j$ for some $i, j \in 1, \ldots, k$. Else, it computes $m = \prod_{i=1}^{k} m_i^{\alpha_i}$, $\sigma = \mathbf{Eval}(\text{vk}, \text{fid}, f, \{\sigma_i\}_{i=1\ldots k})$, chooses a handle h, stores $(h, (\text{fid}, m, \sigma, \sum_{i=0}^{k} \alpha_i f_i))$ in T and returns h to $\mathcal{A}$.

- **Reveal Queries** $\mathcal{A}$ chooses a handle h. If this handle is not in T, the challenger returns $\perp$. Otherwise it retrieves the corresponding record $(h, (\text{fid}, m, \sigma, f))$ from table T and gives $(\text{fid}, m, \sigma, f)$ to $\mathcal{A}$. Next it adds $(h, (\text{fid}, m, \sigma, f))$ to a different table Q.

Forgery $\mathcal{A}$ outputs a dataset identifier fid^*, a message m^*, a signature σ^* and a function f^*.
Let $Q_{\text{fid}^*} = \{(h_i, (\text{fid}^*, m_i, \sigma_i, f_i))\}_{i=1,\ldots,s} \subseteq Q$ be the set of entries in Q for which $\text{fid} = \text{fid}^*$.
The Adversary wins the game if $\mathbf{Verify}(\text{vk}, \text{fid}^*, m^*, \sigma^*, f^*) = 1$ and one of the following conditions hold:

1 Q_{fid^*} is empty
2 f^* (interpreted as a vector) is in the span of $\{f_1, \ldots, f_s\}$ but, for any $\alpha_1, \ldots, \alpha_s$ such that $f = \sum_{i=1}^{s} \alpha_i f_i$, it holds $m^* \neq \prod_{i=1}^{s} m_i^{\alpha_i}$
3 f^* (interpreted as a vector) is not in the span of $\{f_1, \ldots, f_s\}$

Finally we define the advantage $\mathbf{Adv}^{LHSPSS-RMA}(\mathcal{A})$ of $\mathcal{A}$ as the probability that $\mathcal{A}$ wins the game.

Known Random Message Security. We also extend the above definition to consider a slightly stronger adversarial model (that we call *Known* Random Message Attack security). Informally the KRMA security game is almost identical to the RMA game above. The only difference concerns Signing queries which are dealt as follows.

- **Signing Queries** $\mathcal{A}$ asks for a new message/signature couple w.r.t. to a specific $\text{fid} \in \mathcal{D}$ and a specific index $i \in \{1, \ldots, k\}$. The challenger checks that this query has not been previously answered (otherwise it returns $\perp$), then it picks a random message $m \xleftarrow{\$} \mathcal{M}$ and uses the secret key sk to compute a signature σ for m w.r.t. fid and the index i. Finally it picks a handle h (from a proper set of identifiers), stores $(h, (\text{fid}, m, \sigma, e_i))$ in a table T and returns h and m to $\mathcal{A}$. Thus, in this case $\mathcal{A}$ actually knows the (random) message (but not the corresponding signature) being signed by the challenger.

Definition 5 (Chosen message attack security). An LHSPSS is unforgeable against chosen message attack if for all n the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible in the security parameter λ:
Setup The challenger runs $\mathbf{KeyGen}(1^\lambda, n, k)$ and gives vk to $\mathcal{A}$. The message space $\mathcal{M}$ and the signature space Σ are implicitly defined by the verification key.
Queries $\mathcal{A}$ can ask a polynomial number of queries of the following types:

- **Signing Queries** When $\mathcal{A}$ asks for a signature on the triple (fid, m, i) (where fid is a file identifier, $m \in \mathcal{M}$ and $i \in 1, \ldots, k$), the challenger first checks that no other signature of the form $(\text{fid}, \cdot, i)$ has been requested (if this is not the case, it returns $\perp$). Then it uses the secret key sk to compute a signature σ for m w.r.t. fid and the index i. Finally it picks a handle h (from a proper set identifiers), stores $(h, (\text{fid}, m, \sigma, e_i))$ in a table T and returns h.

- **Derivation Queries** $\mathcal{A}$ chooses a set of handles $h = (h_1, \ldots, h_k)$ and a set of coefficients $f = (\alpha_1, \ldots, \alpha_k)$. The challenger retrieves $\{(h_i, (\text{fid}_i, m_i, \sigma_i))\}_{i=1,\ldots k}$ from T and returns $\perp$ if any of these does not exists or if $\text{fid}_i \neq \text{fid}_j$ for some $i, j \in 1, \ldots, k$. Else, it computes $m = \prod_{i=1}^{k} m_i^{\alpha_i}$, $\sigma = \textbf{Eval}(\text{vk}, \text{fid}, f, \{\sigma_i\}_{i=1\ldots k})$, chooses a handle h, stores $(h, (\text{fid}, m, \sigma, \sum_{i=0}^{k} \alpha_i f_i)))$ in T and returns h to $\mathcal{A}$.

- **Reveal Queries** $\mathcal{A}$ chooses a handle h. If this handle is not in T, the challenger returns $\perp$. Otherwise it retrieves the corresponding record $(h, (\text{fid}, m, \sigma, f))$ from table T and gives $(\text{fid}, m, \sigma, f)$ to $\mathcal{A}$. Next it adds $(h, (\text{fid}, m, \sigma, f))$ to a different table Q.

Forgery $\mathcal{A}$ outputs a dataset identifier fid^*, a message m^*, a signature σ^* and a function f^*.
Let $Q_{\text{fid}^*} = \{(h_i, (\text{fid}^*, m_i, \sigma_i, f_i)\}_{i=1,\ldots,s} \subseteq Q$ be the set of entries in Q for which $\text{fid} = \text{fid}^*$.
The Adversary wins the game if $\textbf{Verify}(\text{vk}, \text{fid}^*, m^*, \sigma^*, f^*) = 1$ and one of the following conditions hold:

1 Q_{fid^*} is empty
2 f^* (interpreted as a vector) is in the span of $\{f_1, \ldots, f_s\}$ but, for any $\alpha_1, \ldots, \alpha_s$ such that $f = \sum_{i=1}^{s} \alpha_i f_i$, it holds $m^* \neq \prod_{i=1}^{s} m_i^{\alpha_i}$
3 f^* (interpreted as a vector) is not in the span of $\{f_1, \ldots, f_s\}$

Finally we define the advantage $\textbf{Adv}^{LHSPSS-CMA}(\mathcal{A})$ of $\mathcal{A}$ the probability that $\mathcal{A}$ wins the game.

2.3 Homomorphic Online/Offline signatures

Definition 6 (LHOOS). A Linearly homomorphic Online/Offline signature scheme is a tuple of PPT algorithms (**KeyGen**, **OffSign**, **OnSign**, **Verify**, **Eval**) such that:

- $\textbf{KeyGen}(1^\lambda, n, k)$ takes as input the security parameter λ, an integer n denoting the length of vectors to be signed and an upper bound k for the number of messages signed in each dataset. It outputs a secret signing key sk and a public verification key vk; the public key implicitly defines a message space that can be seen as a vector space of the form $\mathcal{M} = \mathbb{F}^n$ (where $\mathbb{F}$ is a field), a file identifier space $\mathcal{D} = \{0,1\}^{n_d}$ and a signature space Σ.
- $\textbf{OffSign}(\text{sk}, \text{fid})$ takes as input the secret key, and a file identifier $\text{fid} \in \mathcal{D}$ and outputs some information I_{fid}.
- $\textbf{OnSign}(\text{sk}, \text{fid}, I_{\text{fid}}, \mathbf{m}, i)$ takes as input the secret key, an element $\mathbf{m} \in \mathcal{M}$, an index $i \in \{1, \ldots, k\}$, a dataset identifier fid and the related information I_{fid} output by **OffSign**. It outputs a signature σ.

- **Verify** $(\text{vk}, \text{fid}, \mathbf{m}, \sigma, \alpha)$ takes as input the pubic key vk, a signature $\sigma \in \Sigma$, a message $\mathbf{m} \in \mathcal{M}$, a dataset identifier fid $\in \mathcal{D}$ and a vector $\alpha \in \mathbb{Z}^k$; it outputs 1 (accept) or 0 (reject).
- **Eval** $(\text{vk}, fid, \{m_i, \sigma_i, \alpha_i\}_{i=1,\dots,d}, \beta)$ takes as input the public key vk, a dataset identifier fid, a vector $\beta = (\beta_1, \dots, \beta_d) \in \mathbb{Z}^d$ (for some integer $d \in \mathbb{N}$), a set of d tuples $\{m_i, \sigma_i, \alpha_i\}$ of a message m_i, a signature σ_i and a vector $\alpha_i \in \mathbb{Z}^k$. It outputs a signature $\sigma \in \Sigma$.

The correctness conditions of our scheme are the following:

- Let $(\text{sk}, \text{vk}) \leftarrow \mathbf{KeyGen}(1^\lambda, n, k)$ be an honestly generated keypair, $m \in \mathcal{M}$, fid any dataset identifier and $i \in 1, \dots, k$. If $\sigma \leftarrow \mathbf{Sign}(\text{sk}, \text{fid}, \mathbf{OffSign}(\text{sk}, \text{fid}), m, i)$, then with overwhelming probability

$$\mathbf{Verify}(\text{vk}, \sigma, m, \text{fid}, e_i) = 1,$$

  where e_i is the i^{th} vector of the standard basis of $\mathbb{Z}^k$.
- Let $(\text{sk}, \text{vk}) \leftarrow \mathbf{KeyGen}(1^\lambda, n, k)$ be an honestly generated keypair, $\mathbf{m}_1, \dots, \mathbf{m}_d \in \mathcal{M}$ any tuple of messages signed w.r.t the same fid, and let $\sigma_1, \dots, \sigma_d \in \Sigma$, $\alpha_1, \dots, \alpha_d \in \mathbb{Z}^k$ such that for all $i \in \{1, \dots, d\}$, $\mathbf{Verify}(\text{vk}, \sigma_i, \mathbf{m}_i, \text{fid}, \alpha_i) = 1$. Then, for any vector $\beta = (\beta_1, \dots, \beta_d) \in \mathbb{Z}^d$, with overwhelming probability

$$\mathbf{Verify}(\text{vk}, \mathbf{Eval}(\text{vk}, \text{fid}, \beta, \{\mathbf{m}_i, \sigma_i, \alpha_i\}_{i=1,\dots,d}), \sum_{i=1}^{d} \beta_i \mathbf{m}_i, \text{fid}, \sum_{i=0}^{d} \beta_i \alpha_i) = 1$$

Definition 7 (LHOOS CMA). An LHOOS is unforgeable against a chosen message attack if for all n the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible in the security parameter λ:
Setup The challenger runs $\mathbf{KeyGen}(1^\lambda, n, k)$ and gives vk to $\mathcal{A}$. The message space $\mathcal{M}$, the signature space Σ and the dataset space $\mathcal{D}$ are all implicitly defined by the verification key.
Signing Queries $\mathcal{A}$ can ask a polynomial number of queries of the form (m_i, fid, i) (where $m_i \in \mathcal{M}$ is a message, fid is a dataset identifier and $i \in \{1, \dots, k\}$ is an index), and get the corresponding signatures by the challenger. No two queries where only the message $\mathbf{m}_i$ changes can be asked by the adversary (if this happens, the answer to the second query is $\perp$).
Forgery $\mathcal{A}$ outputs a dataset identifier fid^*, a message m^*, a signature σ^* and a vector $\alpha^* \in \mathbb{Z}^k$.
The Adversary wins the game if $\mathbf{Verify}(\text{vk}, \text{fid}^*, \mathbf{m}^*, \sigma^*, \alpha^*) = 1$ and, called $\mathbf{m}_1, \dots, \mathbf{m}_k$ the messages (possibly) queried by the adversary for the identifier fid^*, either

- there exists i such that $\alpha_i^* \neq 0$, but no message w.r.t. index i and fid^* has been queried by the adversary.
- the previous condition does not occur, and $\mathbf{m}^* \neq \sum_{i=1}^{k} \alpha_i \mathbf{m}_i$

Finally we define the advantage $\mathbf{Adv}^{LHOOS-RMA}(\mathcal{A})$ of $\mathcal{A}$ as the probability that $\mathcal{A}$ wins the game.

One can give an analogous notion of strong security against a chosen message attack, where even a new signature for a message the adversary received from the simulator (or computed itself from the **Eval** algorithm and signatures obtained from the simulator) is considered a forgery.

Remark 8. In the definition above, the dataset identifiers need to be prepared in advance during the off-line phase. While this assumption is reasonable for several applications, the use of a chameleon hash [39] function could easily allow the signer to change the fid during the on-line phase at a negligible computational cost. Indeed, in the practical instantiation presented in appendix D we achieve this better outcome for free.

3 From random message security to chosen message security

In this section we present our main result: a general transform to construct an LHSPS secure against chosen message attack from one secure under random message attack. This transform comes in two flavours, depending on whether the underlying scheme is RMA secure or known RMA secure. In this latter case the conversion is totally generic. In the first case, on the other hand, the RMA secure scheme needs to satisfy some additional, but reasonable, requirements. In particular we require it to be *almost deterministic*. Informally, this means that given a file identifier $\text{fid} \in \mathcal{D}$ and a signature on a message m with respect to fid, the signature of any other $m' \in \mathcal{M}$ w.r.t. to any admissible function $f \in \mathcal{F}$ and the same fid is uniquely determined.

We stress that while we present our theorems in the context of (linearly homomorphic) structure preserving signatures, our results apply essentially to *any* linearly homomorphic signature scheme.

Let $\mathcal{HSPS} = (\mathbf{HKeyGen}, \mathbf{HSign}, \mathbf{HVerify}, \mathbf{HEval})$ be a LHSPS which is either known RMA-secure or RMA-secure and almost deterministic. The transformation below shows how to produce a new LHSPS $\mathcal{T} = (\mathbf{TKeyGen}, \mathbf{TSign}, \mathbf{TVerify}, \mathbf{TEval})$ which is secure under CMA.

- $\mathbf{TKeyGen}(1^\lambda, n, k)$ takes as input the security parameter λ, the vector size n and an upper bound k for the number of messages signed in each dataset. It runs two times the **HKeyGen** algorithm to obtain $(\text{sk}_1, \text{vk}_1) \leftarrow \mathbf{HKeyGen}(1^\lambda, n, k)$ and $(\text{sk}_2, \text{vk}_2) \leftarrow \mathbf{HKeyGen}(1^\lambda, n, k)$.
  It outputs $\text{sk} = (\text{sk}_1, \text{sk}_2)$ as the secret signing key and $\text{vk} = (\text{vk}_1, \text{vk}_2)$ as the public verification key. The message space $\mathcal{M}$ is the same of $\mathcal{HSPS}$.
- $\mathbf{TSign}(\text{sk}, \mathbf{m}, \text{fid}, i)$ It chooses random $\mathbf{m}_1 = (m_{1,1}, \ldots, m_{1,n}) \xleftarrow{\$} \mathcal{M}$ and computes $\mathbf{m}_2 \leftarrow \left(\frac{m_1}{m_{1,1}}, \ldots, \frac{m_n}{m_{1,n}}\right)$ (where $\mathbf{m} = (m_1, \ldots, m_n)$).
  Then it computes $\sigma_1 \leftarrow \mathbf{HSign}(\text{sk}_1, \mathbf{m}_1, i, \text{fid})$, $\sigma_2 \leftarrow \mathbf{HSign}(\text{sk}_2, \mathbf{m}_2, i, \text{fid})$ and outputs $\sigma = (\text{fid}, \mathbf{m}_1, \sigma_1, \sigma_2)$.
- $\mathbf{TVerify}(\text{vk}, \sigma, \mathbf{m}, \text{fid}, f)$ parses σ as $(\text{fid}, \mathbf{m}_1, \sigma_1, \sigma_2)$, computes $\mathbf{m}_2 \leftarrow \left(\frac{m_1}{m_{1,1}}, \ldots, \frac{m_n}{m_{1,n}}\right)$ and checks that the following equations hold:

$$\mathbf{HVerify}(\text{sk}_i, m_i, \sigma_i, \text{fid}, f) = 1 \quad \text{for} \quad i = 1, 2.$$

- $\mathbf{Eval}(\text{vk}, \text{fid}, f, \{\sigma^{(i)}\}_{i=1\ldots k})$ parses $\sigma^{(i)}$ as $(\text{fid}^{(i)}, \mathbf{m}_1^{(i)}, \sigma_1^{(i)}, \sigma_2^{(i)})$ and f as $(\alpha_1, \ldots, \alpha_k)$, then checks that $\text{fid} = \text{fid}^{(i)}$ for all i and, if not, aborts. Finally it sets

$$\sigma_1 \leftarrow \mathbf{HEval}(\text{vk}_1, \text{fid}, \{\sigma_1^{(i)}\}_{i=1\ldots k}, f),$$

$$\sigma_2 \leftarrow \mathbf{HEval}(\text{vk}_2, \text{fid}, \{\sigma_2^{(i)}\}_{i=1\ldots k}, f),$$

$$\mathbf{m}_1 = \left(\prod_{i=1}^{k} (m_{1,1}^{(i)})^{\alpha_i}, \ldots, \prod_{i=1}^{k} (m_{1,n}^{(i)})^{\alpha_i} \right)$$

and returns

$$\sigma \leftarrow (\mathrm{fid}, \mathbf{m}_1, \sigma_1, \sigma_2)$$

Theorem 9. *Suppose $\mathcal{HSPS}$ is a LHSPS secure against a random message attack with almost deterministic signatures. Moreover assume that the underlying message space is a group where one can efficiently solve systems of group equations. Then the scheme $\mathcal{T}$ described above is a LHSPS secure against a chosen message attack.*

Proof. We prove the theorem by reducing the security of $\mathcal{T}$ to the one of $\mathcal{HSPS}$, and showing how to build a simulator $\mathcal{B}$ that uses an adversary $\mathcal{A}$ against $\mathcal{T}$ to break the RMA security of $\mathcal{HSPS}$.
First of all one can notice that, by construction, if $(\mathbf{m}^*, \mathbf{Sign}^* = (\mathrm{fid}^*, \mathbf{m}_1^*, \mathbf{Sign}_1^*, \mathbf{Sign}_2^*), f^*)$ is a forgery for $\mathcal{T}$ then at least one between $(\mathbf{m}_1^*, \mathbf{Sign}_1, f^*)$ (case 1) and $(\mathbf{m}^*/\mathbf{m}_1^*, \mathbf{Sign}_2, f^*)$ (case 2) is a forgery for the corresponding instance of $\mathcal{HSPS}$.
The simulator $\mathcal{B}$ works as follows:
It receives a public key vk' for an instance of $\mathcal{HSPS}$ from its challenger. First of all it flips a coin to guess in which case he will be (as usual, his guess will be right with probability at least 1/2). Without loss of generality, we will describe the simulation in the case where its guess is case 1.

Setup $\mathcal{B}$ runs once the **HKeyGen** algorithm to obtain (sk_2, vk_2), sets $\mathrm{vk}_1 \leftarrow \mathrm{vk}'$ and gives $\mathrm{vk} = (\mathrm{vk}_1, \mathrm{vk}_2)$ to $\mathcal{A}$.

Signing Queries Each time $\mathcal{A}$ asks a query of the form $(\mathrm{fid}, \mathbf{m}, i)$, $\mathcal{B}$ forwards a query of the form (fid, i) to its challenger and gets back an handle h (if the challenger returns an error $\perp$, $\mathcal{B}$ simply forwards it to $\mathcal{A}$). Then it chooses a random message[1] $\mathbf{m}_2$, computes $\mathbf{Sign}_2 \leftarrow \mathbf{Sign}(\mathrm{sk}_2, \mathbf{m}_2, \mathrm{fid}_2, i)$ and returns h to $\mathcal{A}$. The handle h, the messages $\mathbf{m}$ and $\mathbf{m}_2$, the signature $\mathbf{Sign}_2$ and the index i are stored in a table T, like in the real experiment.

Derivation Queries In response to a derivation query $(h_1, \ldots, h_k, \mathbf{f})$, the simulator forwards the query to its challenger, and gets back a new handle h (or an error $\perp$, which gets forwarded to $\mathcal{A}$). Then it executes itself the query on the second part of the signature by computing $\mathbf{Sign}^{(h)} \leftarrow \mathbf{Eval}(vk, \mathrm{fid}, f, \{\mathbf{Sign}^{(h_i)}\}_{i=1,\ldots,k})$, computes the corresponding messages $\mathbf{m}^{(h)} = \prod_{i=1}^{k} (\mathbf{m}^{(h_i)})^{f_i}$, $\mathbf{m}_2^{(h)} = \prod_{i=1}^{k} (\mathbf{m}_2^{(h_i)})^{f_i}$ (the components $\mathbf{m}^{(h_i)}, \mathbf{m}_2^{(h_i)}$ $\mathbf{Sign}^{(h_i)}$ corresponding to each handle h_i are retrieved from the table T). Finally, $\mathcal{B}$ gives h to $\mathcal{A}$ and stores the messages, signature, handles and function f in T.

Reveal Queries When $\mathcal{A}$ provides a handle h in a reveal query, $\mathcal{B}$ forwards the reveal query to the challenger. If the answer is $\perp$, $\mathcal{B}$ simply forwards it to $\mathcal{A}$. Otherwise, it gets a tuple $(\mathrm{fid}, \mathbf{m}_1, \mathbf{Sign}_1, f)$. Since the adversary expects to receive a valid signature for a certain message $\mathbf{m}$ (that the simulator knows since it is stored

[1] We stress that, since the simulator does not know what random message $\mathbf{m}_1$ the challenger has chosen to sign, at this point there is no guarantee that $\mathbf{m} = \mathbf{m}_1\mathbf{m}_2$. However, the adversary only gets a random handle, and we will deal with this problem later.

in its own table T together with the handle h, the message $\mathbf{m}_2$ and signature $\mathbf{Sign}_2$), it must now modify the table T in such a way that it is compliant with the information that the adversary has requested and the ones it has already obtained in the previous reveal queries. In particular, for each reveal query (associated with a function f), the adversary knows a message $\mathbf{m}_2$ such that, called $\mathbf{m}_2^{(1)}, \ldots, \mathbf{m}_2^{(k)}$ the messages corresponding to the second part of the signatures issued by the simulator in response to the signing queries for the same fid, it holds that $\mathbf{m}_2 = \prod_{i=1}^{k} (\mathbf{m}_2^{(h_i)})^{f_i}$. It can modify the table by choosing a random simultaneous solution for all these equations[2] (in the unknowns $\mathbf{m}_2^{(1)}, \ldots, \mathbf{m}_2^{(k)}$) and computing new signatures[3] for each entry in T (except for those who have been already given to the adversary) by either using the **Sign** or the **Eval** algorithm. Finally, it can compute $\mathbf{m}_1 = \mathbf{m}/\mathbf{m}_2$ and give the updated signature to $\mathcal{A}$ in response to the query.

Forgery Suppose $\mathcal{A}$ returns $\mathbf{m}^*, \sigma^* = (\text{fid}^*, \mathbf{m}_1^*, \sigma_1^*, \sigma_2^*), f^*$ as a valid forgery and that $\mathcal{B}$'s guess was correct. Then $\mathcal{B}$ can return $(\text{fid}^*, \mathbf{m}_1^*, \sigma_1^*, f^*)$ as a valid forgery against $\mathcal{HSPS}$ to its challenger.

We remark that the proof above becomes much simpler if the simulator were allowed to know the messages signed by the challenger when answering signing queries. Formalizing this observation leads to the following theorem (whose proof is omitted):

Theorem 10. *If $\mathcal{HSPS}$ is a LHSPS secure against known random message attack the scheme $\mathcal{T}$ described above is a LHSPS secure against a chosen message attack.*

Remark 11. We described this generic construction w.r.t. multiplicative schemes, but it can be trivially extended to other kinds of homomorphic schemes modifying the class of functions accordingly. The only requirement for this extension to work is that each function in this class of functions must be homomorphic w.r.t. the same operation of the signature scheme.

3.1 A random message secure construction for the scalar case

Here we present a randomly secure instantiation for the case where $n = 1$, that is when the vectors in the message space have only one component. In appendix C we show how to to derive a fully secure scheme using the conversion methodology described in the previous section[4]. Our construction uses as underlying building block the strongly secure variant of Waters signature given in [15]. For completeness, such a scheme is recalled in appendix A

Let $\mathbb{G}$, $\mathbb{G}_T$ be groups of prime order p such that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map and $\mathcal{H} = \{H_K\}_{K \in \mathcal{K}}$ be a family of collision-resistant hash functions $H_K : \{0,1\}^* \to \mathbb{Z}_p$. The scheme works as follows:

[2] A solution always exists, since if the simulator was given the actual messages chosen by the challenger, he could set $m_2^{(i)} = \mathbf{m}/\mathbf{m}_1^i$ for all i. Moreover, we assumed that such a solution can be efficiently computed

[3] by the the property that the scheme is almost deterministic, the adversary cannot distinguish whether or not the signatures it has not seen have been modified during the game because for each message there is only one signature and therefore this signature does not contain any information about how it was generated.

[4] More precisely the scheme given in appendix C is a slightly optimized version of what one would get by naively converting our random message secure scheme. See appendix C for details.

KeyGen$(1^\lambda, 1, k)$**:** Choose two random generators $g, g_2 \in \mathbb{G}$ and a random hashing key $K \xleftarrow{\$} \mathcal{K}$.
Pick random $\alpha, w \xleftarrow{\$} \mathbb{Z}_p$ and set $g_1 = g^\alpha, W \leftarrow g^w$, $l \leftarrow \lceil \log p \rceil$.
Select random group elements $A_0, \ldots, A_l, h_1, \ldots, h_k, h \xleftarrow{\$} \mathbb{G}$.
Set $\mathrm{vk} \leftarrow (g, g_1, g_2, W, A_0, \ldots, A_l, h, h_1, \ldots, h_k, K)$ as the public verification key and $\mathrm{sk} = (w, g_2^\alpha)$ as the secret signing key.

Sign$(\mathbf{sk}, m, \mathbf{fid}, i)$**:** This algorithm stores a list $\mathcal{L}$ of all previously returned dataset identifiers fid (together with the related secret information r and public information σ, τ, s defined below) and works according to the type of fid it is given in input):

If fid $\notin \mathcal{L}$, then choose[5] $r, s \xleftarrow{\$} \mathbb{Z}_p$, set $\sigma \leftarrow g^r$, compute $t \leftarrow H_K(\mathrm{fid} \| \sigma)$, $\overline{\mathrm{fid}} \leftarrow H_K(g^t h^s)$ and $\tau \leftarrow g_2^\alpha (A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\mathrm{fid}}]_\zeta})^r$ (where $[\overline{\mathrm{fid}}]_\zeta$ is the ζ-th bit of $\overline{\mathrm{fid}}$)

Else if fid $\in \mathcal{L}$, then retrieve the associated r, σ, τ, s from memory.
Then set $M \leftarrow m^w, T \leftarrow (h_i M)^r$ (if a signature for the same fid and the same index i was already issued, then abort). Finally output $\mathbf{Sign} \leftarrow (\mathrm{fid}, \sigma, \tau, T, M, s)$ as a signature for m w.r.t. the function e_i (where e_i is the i-th vector of the canonical basis of $\mathbb{Z}^n$).

Verify$(\mathrm{vk}, m, \mathbf{Sign}, f)$: Parse the signature **Sign** as $(\mathrm{fid}, \sigma, \tau, T, M, s)$, f as $(f_1, \ldots, f_k)$ and compute $t \leftarrow H_K(\mathrm{fid} \| \sigma)$, $\overline{\mathrm{fid}} \leftarrow H_K(g^t h^s)$. Then check that:

$$e(\tau, g) = e(g_2, g_1) \cdot e(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\mathrm{fid}}]_\zeta}, \sigma)$$

$$e(M, g) = e(m, W)$$

$$e(T, g) = e(\prod_{i=1}^{k} h_i^{f_i} M, \sigma)$$

If all the above equations hold output 1, else output 0.

Eval $(\mathrm{vk}, \alpha, \mathbf{Sign}_1, \ldots, \mathbf{Sign}_k)$: Parse α as $(\alpha_1, \ldots, \alpha_k)$ and $\mathbf{Sign}_i$ as $(\mathrm{fid}_i, \sigma_i, \tau_i, T_i, M_i, s_i)$, $\forall i = 1, \ldots, k$. Then verify that all $\mathbf{Sign}_i$ share the same $\mathrm{fid}, \sigma, \tau, s$ components and, if not, reject. Finally, compute $T \leftarrow \prod_{i=1}^{k} T_i^{\alpha_i}$, $M \leftarrow \prod_{i=1}^{k} M_i^{\alpha_i}$ and output $\mathbf{Sign} = (\mathrm{fid}, \sigma, \tau, T, M, s)$.

The security of the scheme follows from the following theorem (whose proof is deferred to appendix B.1)

Theorem 12. *If the 2-3CDH assumption holds and $\mathcal{H}$ is a family of collision resistant hash functions then the scheme described above is a secure Linearly homomorphic signature scheme according to definition 4.*

[5] We stress that, even if $s \in \mathbb{Z}$ is not an element of the group $\mathbb{G}$, it is only used to authenticate the identifier fid and therefore it does not compromise the usefulness of SPS in any application that we are aware of.

4 Applications to On-Line/Off-Line Homomorphic Signatures

In this section we show applications to the case of on-line/off-line homomorphic signatures. More precisely we show that combining a LHSPS secure against a random message attack, with a certain class of sigma protocols yields to efficient on-line/off-line linearly homomorphic (and network coding) signature schemes. Informally the properties we require from the underlying sigma protocol are: (1) it is linearly homomorphic, (2) its challenge space can be seen as a vector space and (3) the third message of the protocol can be computed in a very efficient way (as it is used in the online phase of the resulting scheme). More precise details follow.
We start by recalling the notion of Σ-Protocol.

Σ-Protocol. Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be an arbitrary binary relation, with the only restriction that if $(x,w) \in \mathcal{R}$, then the length of w is polynomial in the length of x (typically, $(x,w) \in \mathcal{R}$ if x is part of an NP language L and w is one of its associated witnesses). A Σ protocol for $\mathcal{R}$ is an interactive (three rounds) protocol involving two parties: a prover P and a verifier V. We assume that both parties are PPT machines and that they agree on some value x in advance, and the goal of the protocol is to let the prover convince the verifier that he knows w such that $(x,w) \in \mathcal{R}$. The three rounds are carried out as follows: in the first round P sends a message to V, who replies with a string (chosen at random from a well defined set and called a challenge string), and finally gets back a third message from P and outputs 1 or 0 depending on whether he is convinced by this interaction.
More formally, a Σ protocol consists of four PPT algorithms $\Sigma = (\Sigma\textbf{-Setup}, \Sigma\textbf{-Com}, \Sigma\textbf{-Resp}, \Sigma\textbf{-Verify})$ defined as follows:

- $\Sigma\textbf{-Setup}(1^\lambda, \mathcal{R}) \rightarrow (x,w)$ It takes as input a security parameter λ and a relation $\mathcal{R}$. It returns a statement x and a witness w such that $(x,w) \in \mathcal{R}$.
- $\Sigma\textbf{-Com}(x) \rightarrow (R,r)$ Is a probabilistic algorithm run by the prover to get the first message R to be sent to the verifier and some private state r to be stored and used later in the protocol.
- $\Sigma\textbf{-Resp}(x,w,r,c) \rightarrow s$ Is an algorithm run by the prover to compute the last (third) message of the protocol (to be sent to the verifier). It takes as input the statement x, its witness w, the challenge string (chosen at random by V in a well defined set **ChSp** and sent as the second message of the protocol), and some state information r. It outputs the third message of the protocol.
- $\Sigma\textbf{-Verify}(x,R,c,s) \rightarrow \{0,1\}$ Is the verification algorithm that on input the message R, a challenge $c \in \textbf{ChSp}$ and a response s, outputs 1 (accept) or 0 (reject).

We assume that the protocol satisfies the following three proprieties:

- **Completeness** $\forall (x,w) \in \mathcal{R}$, any $(R,r) \leftarrow \Sigma\textbf{-Com}(x,r)$, any $c \in \textbf{ChSp}$ and $s \leftarrow \Sigma\textbf{-Resp}(x,w,r,c)$, it holds that $\Sigma\textbf{-Verify}(x,R,c,s) = 1$ with overwhelming probability.
- **Special Soundness** There exists a PPT extractor algorithm $\Sigma\textbf{-Ext}$ such that $\forall x \in L$, $\forall R,c,s,c',s'$ such that $(c,s) \neq (c',s')$, $\Sigma\textbf{-Verify}(x,R,c,s) = 1$ and $\Sigma\textbf{-Verify}(x,R,c',s') = 1$, outputs $w' \leftarrow \Sigma\textbf{-Ext}(x,R,c,s,c',s')$ such that $(x,w') \in \mathcal{R}$
- **Special Honest Verifier Zero Knowledge (HVZK)** There exists a PPT algorithm S such that $\forall x \in L, \forall c \in \textbf{ChSp}$, $S(x,c)$ generates a pair (R,s) such that $\Sigma\textbf{-Verify}(x,R,c,s) = 1$ and the probability distribution of (R,c,s) is identical to the one obtained by running the real algorithms.

4.1 Vector and Homomorphic Σ-protocols

Given a language L and an integer $n \in \mathbb{N}$, we can consider the language $L^n = \{(x_1, \ldots, x_n) \mid x_i \in L \ \forall i = 1, \ldots, n\}$. A trivial witness for a tuple (vector) in this language is the tuple of the witnesses of each of its components for the language L. As before we can consider the relation $\mathcal{R}^n$ associated to L^n, where $(\boldsymbol{x}, \boldsymbol{w}) = (x_1, \ldots, x_n, w_1, \ldots, w_n) \in \mathcal{R}^n$ if $(x_1, \ldots, x_n)$ is part of L^n and w_i is a witness for x_i. A *vector* Σ protocol for $\mathcal{R}^n$ is a three round protocol defined similarly as above with the relaxation that the special soundness property is required to hold in a weaker form. Namely, we require the existence of an efficient extractor algorithm Σ_n-**Ext** such that $\forall\, \boldsymbol{x} \in L^n$, $\forall\, R, \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{c'}, \boldsymbol{s'}$ such that $(c, s) \neq (c', s')$, Σ_n-**Verify**$(\boldsymbol{x}, R, \boldsymbol{c}, \boldsymbol{s}) = 1$ and Σ_n-**Verify**$(\boldsymbol{x}, R, \boldsymbol{c'}, \boldsymbol{s'}) = 1$, outputs $(x, w) \leftarrow \Sigma_n$-**Ext** $(\boldsymbol{x}, R, \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{c'}, \boldsymbol{s'})$ where x is one of the components of $\boldsymbol{x}$ and $(x, w) \in \mathcal{R}$.

Definition 13. A Sigma protocol $\Sigma = (\Sigma$-**Setup**,Σ-**Com**,Σ-**Resp**,Σ-**Verify**$)$ for a relation $\mathcal{R}$ is called *group homomorphic* if

- The outputs of the Σ-**Com** algorithm and the challenge space of the protocol can be seen as elements of two groups $(\mathbb{G}_1, \circ_1)$ and $(\mathbb{G}_2, \circ_2)$ respectively
- There exists a PPT algorithm **Combine** such that, for all $(x, w) \in \mathcal{R}$ and all $\alpha \in \mathbb{Z}^n$, if transcripts $\{(R_i, c_i, s_i)\}_{i=1,\ldots,n}$ are such that Σ-**Verify**$(x, R_i, c_i, s_i) = 1$ for all i, then

$$\Sigma\text{-}\mathbf{Verify}\left(x, R_1^{\alpha_1} \circ_1 \cdots \circ_1 R_n^{\alpha_n}, c_1^{\alpha_1} \circ_2 \cdots \circ_2 c_n^{\alpha_n}, \mathbf{Combine}(\alpha, \{(R_i, c_i, s_i)\}_{i=1,\ldots,n})\right) = 1$$

We stress that the above definition can be trivially extended to the case of vector sigma protocols.

In appendix D we show that a simple variant of the well known identification protocol by Schnorr fits the requirements of (efficient) homomorphic vector Σ-protocol

4.2 On-Line/Off-Line Signature

In this section we describe a generic framework to obtain an On-Line/Off-Line Signature Scheme.
Suppose $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{Eval})$ is a randomly secure LHSPS (even one that only allows to sign scalars) and $\Sigma_n = (\Sigma_n$-**Setup**,Σ_n-**Com**,Σ_n-**Resp**,Σ_n-**Verify**$)$ a group homomorphic vector Sigma protocol associated to a relation $\mathcal{R}^n$ on a language L^n. Our generic construction works as follows:

ON/OFFKeyGen $(1^\lambda, k, n)$: It runs $(\mathrm{vk}_1, sk_1) \leftarrow \mathbf{KeyGen}(1^\lambda, k, n)$ and $(\mathbf{x}, \mathbf{w}) \leftarrow \Sigma_n$-**Setup**$(1^\lambda, \mathcal{R}^n)$. It outputs $\mathrm{vk} \leftarrow (vk_1, \mathbf{x})$, $\mathrm{sk} \leftarrow (sk_1, \mathbf{w})$.

OFFSign $(\mathrm{sk}, \mathrm{fid})$: This algorithm runs the Σ_n-**Com** algorithm k times to obtain $(R_i, r_i) \leftarrow \Sigma_n$-**Com**$(\mathbf{x})$. Then it signs each R_i using the LHSPS signing algorithm $\overline{\sigma_i} \leftarrow \mathbf{Sign}(sk_1, \mathrm{fid}, R_i, i)$ and outputs $I_{\mathrm{fid}} = \{(i, r_i, R_i, \overline{\sigma_i})\}_{i=1,\ldots,k}$.

ONSign $(\mathrm{vk}, \mathrm{sk}, \mathbf{m}, \mathrm{fid}, I_{\mathrm{fid}}, i)$: It parses I_{fid} as[6] $\{(i, r_i, R_i, \overline{\sigma_i})\}_{i=1,\ldots,k}$, computes $s \leftarrow \Sigma_n$-**Resp**$(\mathbf{x}, \mathbf{w}, r_i, \mathbf{m})$ and outputs $\sigma \leftarrow (R_i, \overline{\sigma}_i, s)$

ON/OFFVerify $(\mathrm{vk}, \mathbf{m}, \sigma)$: It parses σ as $(R, \overline{\sigma}, s)$ and vk as $(\mathrm{vk}_1, \mathbf{x})$. Then it checks that

$$\mathbf{Verify}(\mathrm{vk}_1, R, \sigma_1) = 1 \quad \text{and} \quad \Sigma_n\text{-}\mathbf{Verify}(\mathbf{x}, R, \mathbf{m}, s) = 1.$$

If both the above equations hold it returns 1, else it returns 0.

[6] As said in remark 8, the use of a chameleon hash function (if the scheme does not support this feature itself) allows the signer to compute a valid signature even on inputs $\mathrm{fid}_1, I_{\mathrm{fid}_2}$ where $\mathrm{fid}_1 \neq \mathrm{fid}_2$

ON/OFFEval $(\mathrm{vk}, \alpha, \sigma_1, \ldots, \sigma_k)$: it parses σ_i as $(R_i, \overline{\sigma}_i, s_i)$ for each $i = 1, \ldots, k$ and vk as $(\mathrm{vk}_1, \mathbf{x})$. Then it computes:

$$R \leftarrow R_1^{\alpha_1} \circ_1 \cdots \circ_1 R_k^{\alpha_k}, \quad \overline{\sigma} \leftarrow \mathbf{Eval}(\mathrm{vk}_1, \alpha, \overline{\sigma}_1, \ldots, \overline{\sigma}_k),$$

$$s \leftarrow \mathbf{Combine}\left(\alpha, \{(R_i, c_i, s_i)\}_{i=1,\ldots,k}\right).$$

Finally it returns $(R, \overline{\sigma}, s)$ (as a signature for the message $\mathbf{m}_1^{\alpha_1} \circ_2 \cdots \circ_2 \mathbf{m}_k^{\alpha_k}$).

Theorem 14. *If* $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{Eval})$ *is a randomly secure LHSPS and* $\Sigma_n = (\Sigma_n\textbf{-Setup}, \Sigma_n\textbf{-Com}, \Sigma_n\textbf{-Resp}, \Sigma_n\textbf{-Verify})$ *is a group homomorphic vector sigma protocol for a non trivial relation* $\mathcal{R}^n$*, then the on line/off line scheme described above is secure against a chosen message attack according to definition 7.*

Suppose $\mathbf{m}^*, (R^*, \overline{\sigma}^*, s^*)$ is the forgery returned by an adversary $\mathcal{A}$ w.r.t the identifier fid^* and the vector $\boldsymbol{\alpha}^* = (\alpha_1^*, \ldots, \alpha_k^*)$. Let $\{\sigma_1, \ldots, \sigma_k\}$ the set of signatures seen by $\mathcal{A}$ w.r.t. the same identifier fid^* and the messages $(\mathbf{m}_1, \ldots, \mathbf{m}_k)$. Note that, because $\mathcal{S}$ is secure against random messages attacks, it must be that $\prod_{i=1}^k R_i^{\alpha_i} = R^*$ and $\mathrm{fid}^* = \mathrm{fid}$ for some fid that $\mathcal{A}$ has received during the security game. Therefore, by the security definition, the only kind of forgery the adversary can make is one where $\mathbf{m}^* \neq \mathbf{m}_1^{\alpha_1^*} \circ_2 \cdots \circ_2 \mathbf{m}_k^{\alpha_k^*}$.
In this case we describe a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to extract a witness for the language L such that L^n is the language associated to the relation $\mathcal{R}^n$. $\mathcal{B}$ takes as input a vector of statements $\boldsymbol{x} \in L^n$. It must then return a couple (x, w) such that x is a component of $\boldsymbol{x}$ and $(x, w) \in \mathcal{R}$. It works as follows.
Key Generation. $\mathcal{B}$ runs $(\mathrm{vk}_1, \mathrm{sk}_1) \leftarrow \mathbf{KeyGen}(1^\lambda, k, n)$ and gives to $\mathcal{A}$ $\mathrm{vk} = (\mathrm{vk}_1, \mathbf{x})$. It is easy to check that this key is correctly distributed as in the real case.
Signing queries. Each time $\mathcal{A}$ asks for a signature on a message $\mathbf{m}_i$ w.r.t. an identifier fid and to an index $i \in 1, \ldots, k$, $\mathcal{B}$ uses the HVZK simulator of the sigma protocol to compute $(R_i, s_i) \leftarrow S(\mathbf{x}, m)$. Then it computes a signature $\overline{\sigma_i} \leftarrow \mathbf{Sign}(\mathrm{sk}_1, \mathrm{fid}, R_i, i)$ on R_i and returns the signature $\sigma \leftarrow (R_i, \overline{\sigma}_i, s_i)$ to $\mathcal{A}$.
Forgery Suppose $\mathcal{A}$ returns a forgery of type 1 $(R^*, \overline{\sigma}^*, s^*)$ for the message $\mathbf{m}^*$. Let $\overline{\mathbf{m}} = \mathbf{m}_1^{\alpha_1^*} \circ_2 \cdots \circ_2 \mathbf{m}_k^{\alpha_k^*}$ and $\overline{\mathbf{s}} = \mathbf{Combine}\left(\alpha^*, \{(R_i, \mathbf{m}_i, s_i)\}_{i=1,\ldots,k}\right)$.
$\mathcal{B}$ uses the extractor $(x, w) \leftarrow \Sigma\textbf{-Ext}(\mathbf{x}, R, \mathbf{m}^*, s^*, \overline{\mathbf{m}}, \overline{\mathbf{s}})$ from the vector special soundness to obtain a witness for one of the components of $\boldsymbol{x}$.

The security obtained by this construction can be strengthened by assuming additional properties on the underlying LHSPS scheme. If we assume that either $\mathcal{S}$ has almost deterministic encryptions or that is strongly secure against a random message attack, we can prove that the resulting construction is strongly secure as well. The proofs are straightforward and similar to the previous one and therefore we omit them. Below is a formal statement of one of these two theorems.

Theorem 15. *If* $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{Eval})$ *is an* almost deterministic *randomly secure LHSPS and* $\Sigma_n = (\Sigma_n\textbf{-Setup}, \Sigma_n\textbf{-Com}, \Sigma_n\textbf{-Resp}, \Sigma_n\textbf{-Verify})$ *is a group homomorphic vector sigma protocol for a non trivial relation* $\mathcal{R}^n$*, then the on line/off line scheme described above is* ***strongly*** *secure against chosen message attacks.*

In appendix D we show two practical examples of vector sigma protocols, built upon Schnorr's sigma protocol [43]. We also present a further modification of the vector special soundness property, which preserves the security of the construction but allows to build more efficient schemes.

References

1. Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 4–24. Springer, December 2012.
2. Masayuki Abe, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Tagged one-time signatures: Tight security and optimal tag size. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 312–331. Springer, February / March 2013.
3. Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 209–236. Springer, August 2010.
4. Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 649–666. Springer, August 2011.
5. Masayuki Abe, Jens Groth, and Miyako Ohkubo. Separating short structure-preserving signatures from non-interactive assumptions. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 628–646. Springer, December 2011.
6. Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Signing on elements in bilinear groups for modular protocol design. Cryptology ePrint Archive, Report 2010/133, 2010. http://eprint.iacr.org/.
7. Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Group to group commitments do not shrink. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 301–317. Springer, April 2012.
8. Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 17–34. Springer, March 2011.
9. Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Computing on authenticated data: New privacy definitions and constructions. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 367–385. Springer, December 2012.
10. Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 386–404. Springer, February / March 2013.
11. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, August 2004.
12. Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 68–87. Springer, March 2009.
13. Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 149–168. Springer, May 2011.
14. Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 1–16. Springer, March 2011.
15. Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational Diffie-Hellman. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 229–240. Springer, April 2006.

16. Jan Camenisch, Maria Dubovitskaya, and Kristiyan Haralambiev. Efficient structure-preserving signature scheme from standard assumptions. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12: 8th International Conference on Security in Communication Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 76–94. Springer, September 2012.
17. Jan Camenisch, Kristiyan Haralambiev, Markulf Kohlweiss, Jorn Lapon, and Vincent Naessens. Structure preserving CCA secure encryption and applications. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 89–106. Springer, December 2011.
18. Dario Catalano, Mario Di Raimondo, Dario Fiore, and Rosario Gennaro. Off-line/on-line signatures: Theoretical aspects and experimental results. In Ronald Cramer, editor, *PKC 2008: 11th International Conference on Theory and Practice of Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 101–120. Springer, March 2008.
19. Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 336–352. Springer, May 2013.
20. Dario Catalano, Dario Fiore, Rosario Gennaro, and Konstantinos Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 680–699. Springer, mar 2012.
21. Dario Catalano, Dario Fiore, and Bogdan Warinschi. Adaptive pseudo-free groups and applications. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 207–223. Springer, May 2011.
22. Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 680–696. Springer, May 2012.
23. Julien Cathalo, Benoît Libert, and Moti Yung. Group encryption: Non-interactive realization in the standard model. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 179–196. Springer, December 2009.
24. Melissa Chase and Markulf Kohlweiss. A new hash-and-sign approach and structure-preserving signatures from DLIN. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12: 8th International Conference on Security in Communication Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 131–148. Springer, September 2012.
25. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 281–300. Springer, April 2012.
26. Yvo Desmedt. Computer security by redefining what a computer is. NSPW, 1993.
27. Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552. ACM Press, May 1991.
28. Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.
29. David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 697–714. Springer, May 2012.
30. Georg Fuchsbauer. Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. Cryptology ePrint Archive, Report 2009/320, 2009. http://eprint.iacr.org/.
31. Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 177–194. Springer, May 2003.
32. Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 142–160. Springer, May 2010.
33. Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, page 290. Springer, December 2012.
34. J. Groth. Homomorphic trapdoor commitments to group elements. Cryptology ePrint Archive, Report 2009/007, 2009. http://eprint.iacr.org/.

35. Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459. Springer, December 2006.
36. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, April 2008.
37. Dennis Hofheinz and Tibor Jager. Tightly secure signatures and public-key encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 590–607. Springer, August 2012.
38. Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262. Springer, February 2002.
39. Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *ISOC Network and Distributed System Security Symposium – NDSS 2000*. The Internet Society, February 2000.
40. Sébastien Kunz-Jacques and David Pointcheval. About the security of MTI/C0 and MQV. In Roberto De Prisco and Moti Yung, editors, *SCN 06: 5th International Conference on Security in Communication Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 156–172. Springer, September 2006.
41. Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly homomorphic structure-preserving signatures and their applications. In Ran Canetti; Juan A. Garay, editor, *Advances in Cryptology – CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 289–307. Springer, August 2013.
42. Y. Sakai, K. Emura, G. Hanaoka, Y. Kawai, T. Matsuda, and K. Omote. G. Group signatures with message-dependent opening. In Michel Abdalla and Tanja Lange, editors, *PAIRING 2012: 6th International Conference on Pairing-based Cryptography*, volume 7708 of *Lecture Notes in Computer Science*, pages 270–294. Springer, may 2013.
43. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, August 1990.
44. Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer, August 2001.
45. Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, May 2005.

A Waters Signature

For sake of completeness we now describe the Waters signature schemes [45], which we use in some of the practical constructions presented in this paper.
Let $\mathbb{G}$, $\mathbb{G}_T$ be groups of prime order p such that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map.

KeyGen(p) : It sets $l \leftarrow \lceil \log p \rceil$ and chooses $g, g_2 \xleftarrow{\$} \mathbb{G}$, $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and $A_0, A_1 \ldots, A_l \xleftarrow{\$} \mathbb{G}$. Then it sets $g_1 \leftarrow g^\alpha$ and returns $\mathrm{vk} = (g, g_1, g_2, A_0, A_1, \ldots, A_l)$, $\mathrm{sk} = g_2^\alpha$

Sign$(m, \mathbf{sk})$: This algorithm chooses a random $r \xleftarrow{\$} \mathbb{Z}_p$ and sets $\sigma \leftarrow g^r$ and $\tau \leftarrow g_2^\alpha \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[m]_\zeta} \right)^r$. Then it returns $\Sigma = (\sigma, \tau)$

Verify$(\mathbf{vk}, m, \Sigma)$: It checks that

$$e(g, \tau) = e(\sigma, A_0 \prod_{\zeta=1}^{l} A_\zeta^{[m]_\zeta}) \cdot e(g_1, g_2).$$

If the above equation holds it returns 1, else it returns 0.

This scheme is secure under CDH assumption.
In [15] a strongly secure variant of the previously described scheme (under the same computational assumption) is provided. We will refer to this latter scheme as the *Strong* Waters Signature Scheme.
The signature scheme works as follows.
Let $\mathbb{G}$, $\mathbb{G}_T$ be groups of prime order p such that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map and $\mathcal{H} = \{H_K\}_{K \in \mathcal{K}}$ (where $\mathcal{K}$ is the keys' space) a family of collision resistant hash functions

KeyGen(p) : It sets $l \leftarrow \lceil \log p \rceil$ and chooses $g, g_2, h \xleftarrow{\$} \mathbb{G}$, $\alpha \xleftarrow{\$} \mathbb{Z}_p$, $A_0, A_1 \ldots, A_l \xleftarrow{\$} \mathbb{G}$, $K \xleftarrow{\$} \mathcal{K}$. Then it sets $g_1 \leftarrow g^\alpha$ and returns $\mathrm{vk} = (g, g_1, g_2, h, A_0, A_1, \ldots, A_l, k)$, $\mathrm{sk} = g_2^\alpha$

Sign$(m, \mathbf{sk})$: This algorithm chooses random $r, s \xleftarrow{\$} \mathbb{Z}_p$ and sets $\sigma \leftarrow g^r$. Then it computes $t \leftarrow H(m \| \sigma)$, $M \leftarrow H(g^t h^s)$ and $\tau \leftarrow g_2^\alpha \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[M]_\zeta} \right)^r$. Next it returns $\Sigma = (\sigma, \tau, s)$

Verify$(\mathbf{vk}, m, \Sigma)$: To verify a signature $\Sigma = (\sigma, \tau, s)$ the algorithm computes $t \leftarrow H(m \| \sigma)$, $M \leftarrow H(g^t h^s)$. Then it checks that

$$e(g, \tau) = e(\sigma, A_0 \prod_{\zeta=1}^{l} A_\zeta^{[M]_\zeta}) \cdot e(g_1, g_2).$$

If the above equation holds it returns 1, else it returns 0.

B Postponed Proofs

B.1 Proof of theorem 12

Proof. Proving correctness is straightforward, given the bilinear property of the pairing function. For what concerns security, we split the proof in 4 different cases. In

each of them, we will show how an adversary that breaks the security of the scheme can be used to build a simulator that breaks the 2-3CDH assumption. In particular, let $m^*, \mathbf{Sign}^* = (\text{fid}^*, \sigma^*, \tau^*, T^*, M^*, s^*), f^*$ be the forgery returned by the adversary $\mathcal{A}$, Q the set of answers returned to $\mathcal{A}$ in response to its reveal queries, and let $Q_{fid^*} = \{(h_\eta, (fid^*, m_\eta, \mathbf{Sign}_\eta, f_\eta))\}_{\eta=1,\ldots,\nu} \subseteq Q$ be the set of signatures seen by $\mathcal{A}$ for which the file identifier is fid^*, where $\mathbf{Sign}_\eta = (\text{fid}_\eta, \sigma_\eta, \tau_\eta, T_\eta, M_\eta, s_\eta)$.
Then (at least) one of the following conditions hold:

Case 1: Q_{fid^*} is empty.
Case 2: Q_{fid^*} is not empty, but $(\sigma^*, \tau^*, s^*) \neq (\sigma_\eta, \tau_\eta, s_\eta)$ for all $\eta = 1, \ldots, \nu$ (note that, by construction, all the signatures in Q_{fid^*} share the same σ, τ and s components).
Case 3: Q_{fid^*} is not empty, $(\sigma^*, \tau^*, s^*) = (\sigma_\eta, \tau_\eta, s_\eta)$ for all $\eta = 1, \ldots, \nu$, f^* (interpreted as a vector) is in the span of $\{f_1, \ldots, f_\nu\}$ but, for any $\alpha_1, \ldots, \alpha_\nu$ such that $f = \sum_{\eta=1}^{\nu} \alpha_\eta f_\eta$, it holds $m^* \neq \prod_{\eta=1}^{\nu} m_\eta^{\alpha_\eta}$.
Case 4: Q_{fid^*} is not empty, $(\sigma^*, \tau^*, s^*) = (\sigma_\eta, \tau_\eta, s_\eta)$ for all $\eta = 1, \ldots, \nu$ and f^* (interpreted as a vector) is not in the span of $\{f_1, \ldots, f_\nu\}$.

As one can notice, the simulator can guess in which case he will be in advance with probability at least 1/4.

Case 1. We deal with this type of adversary by constructing a simulator that is similar to that used in [41,15] to prove the security of the strongly secure variant of Waters signature scheme. The simulator $\mathcal{B}$ receives on input (g, g^a, g^b), and behaves as follows:

Key Generation It sets $g_1 \leftarrow g^a, g_2 \leftarrow g^b$ (thus implicitly defining part of the secret key as $g_2^a = g^{ab}$) and picks an hash function $H_K \xleftarrow{\$} \mathcal{H}$. Then it chooses a random value $w \in \mathbb{Z}_p$ and random elements $h_1 = g^{l_1}, \ldots, h_k = g^{l_k}, h = g^\ell \in \mathbb{G}$ for random $\ell, l_1, \ldots, l_k \in \mathbb{Z}_p$. Next it computes $W = g^w$ and chooses $A_0, A_1, \ldots, A_l$ in the same way as in security proof of Waters' signature [45]. Because of this choice, there exist two functions $J, K : \{0,1\}^l \to \mathbb{Z}$ (these functions are all kept internal to the simulator) hidden to the adversary such that, for any string $\text{fid} \in \{0,1\}^l$, the expression $Y(\text{fid}) := A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\text{fid}]_\zeta}$ can be written as $Y(\text{fid}) = g_2^{J(\text{fid})} g^{K(\text{fid})}$. In addition, it was proven that for any distinct $\tau, \tau_1, \ldots, \tau_q \in \{0,1\}^l$ we will have $J(\tau) = 0 \mod p$ and $J(\tau_i) \neq 0 \, \forall i \in \{1, \ldots, q\}$ with non negligible probability $\eta = \frac{1}{8q(l+1)}$.
Finally, $\mathcal{B}$ creates two empty tables T and Q (used to store the output of signing and reveal queries, as explained in the security definition) and gives vk to $\mathcal{A}$.

Signing queries When $\mathcal{A}$ ask a new signing query on a dataset identifier fid with respect to index i, $\mathcal{B}$ does the following:

if fid $\in T$ it retrieves the corresponding (σ, τ, s) from the memory.

if fid $\notin T$, it chooses a random $\gamma \in \mathbb{Z}_p$ and sets $\overline{\text{fid}} \leftarrow H_K(g^\gamma)$. if $J(\overline{\text{fid}}) = 0 \mod p$ aborts. Else it chooses random $r \leftarrow \mathbb{Z}_p$ and sets

$$\tau = (Y(\overline{\text{fid}}))^r g_1^{-\frac{K(\overline{\text{fid}})}{J(\overline{\text{fid}})}}$$

$$\sigma = g^r g_1^{-\frac{1}{J(\overline{\text{fid}})}}.$$

In fact they also can be written as $\tau = g^{ab}(Y_G(\overline{\text{fid}}_i))^{r'}$ and $\sigma = g^{r'}$ where $r = r' + \frac{a}{J(\overline{\text{fid}})}$. (See Waters [45] for details).

Then it sets $t \leftarrow H_k(\text{fid}\|\sigma)$ and $s = \frac{\gamma - t}{\ell}$

Then it chooses $\lambda \xleftarrow{\$} \mathbb{Z}_p$, computes $m \leftarrow g^{\lambda}$ and signs it by setting $M \leftarrow m^w, T \leftarrow \sigma^{l_i + \lambda w} = (h_i M)^{r'}$.

The signature $(\text{fid}, \sigma, \tau, T, M, s, e_i)$ and the message m are not directly returned to $\mathcal{A}$, but associate with a new handle h and stored in the table T.

Derivation and Reveal Queries are handled as in the real experiment.

Forgery Once $\mathcal{A}$ provides a forgery $\mathbf{Sign}^* = (\text{fid}^*, \sigma^*, \tau^*, T^*, M^*, s^*)$ $\mathcal{B}$ computes $J(\overline{\text{fid}}^*)$ and aborts if $J(\overline{\text{fid}}^*) \neq 0$

From this forgery $\mathcal{A}$ can extract a CDH solution as follows. First notice that by correctness the components τ and σ of the forgery will be of the form

$$\tau = g^{ab}\left(Y(\overline{\text{fid}}^*)\right)^{r'} \quad \sigma = g^{r'}$$

Thus the required value can be extracted as

$$g^{ab} = \tau / \sigma^{K(\overline{\text{fid}}^*)}$$

Case 2. In this case, where $(\sigma^*, \tau^*, s^*) \neq (\sigma_\eta, \tau_\eta, s_\eta)$, it is possible to reduce the security to the one of the strongly secure Waters signature scheme. The simulator is quite simple: it uses Waters' scheme as a signing oracle to compute the part of the signatures regarding the fid, and can easily fake the other part of each signature by using the discrete logarithms of the messages and of the h_i and the value w.

Case 3. First of all one can notice that, because the forgery must satisfy (in particular) the third and fourth verification equations, it must be that

$$M^* = (m^*)^w \quad \text{and} \quad T^* = \left(\prod_{i=1}^{k} h_i^{\alpha_i^*} M^*\right)^r$$

Moreover, the same two equations must also hold for the honestly computed signature for the function f^* on the messages signed by the challenger (we call $\overline{m}, \overline{\mathbf{Sign}}$ such couple). So it must be that:

$$T^* \overline{T}^{-1} = \left(\prod_{i=1}^{k} M^* M_i^{-\alpha_i^*}\right)^r$$

If the left hand side of the equation is equal to 1 we don't have any forgery (in fact $M^* = \prod_{i=1}^{k} M_i^{\alpha_i}$ and $T^* = \overline{T}$).

Else, in the case when

$$m^* \prod_{i=1}^{k} m_i^{-\alpha_i^*} \neq 1$$

we describe a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to break the 2-3CDH assumption. $\mathcal{B}$ works as follows. It takes in input a 2-3CDH tuple (g, g^w, g^r) and guesses the dataset identifier fid' for which it will receive a forgery[7].

[7] Note that the simulator does not need to predict the exact value of the identifier it will receive a forgery about, but only to pick one among the ones it will be asked to sign (for example, it might pick a random integer i from a large enough domain and choose fid' to be the i-th identifier it will be queried about). So the probability to guess correctly is not negligible and the reduction still works.

Key Generation $\mathcal{B}$ chooses a random hash key $K \leftarrow \mathcal{K}$ and a random generator $g_2 \in \mathbb{G}$. It sets $W \leftarrow g^w$ (so w is implicitly part of the secret key), $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and $g_1 \leftarrow g^\alpha$. It selects $b_i \xleftarrow{\$} \mathbb{Z}_p$ for $i = 1, \ldots, k$ and for each b_i it computes $m_i = g^{b_i}$, $h_i = g^{\delta_i} m_i^{-w}$, for random $\delta_1, \ldots, \delta_k \xleftarrow{\$} \mathbb{Z}_p$ and $h \xleftarrow{\$} \mathbb{G}$. Finally it picks random $a_0, a_1, \ldots, a_l$, sets $A_\zeta \leftarrow g^{a_\zeta}$, $\zeta = 0, \ldots, l$ and gives $(g, g_1, g_2, W, A_0, \ldots, A_l, h, h_1, \ldots, h_k, K)$ to $\mathcal{A}$.

Signing Queries To answer to the queries about the dataset fid' in the position i from $\mathcal{A}$, $\mathcal{B}$ uses the previously created messages m_i and answers with the following.

if $\mathrm{fid}' \notin T$, it chooses a rand $s \in \mathbb{Z}_p$ and sets

$$\sigma = g^r,$$

$$t \leftarrow H_K(\mathrm{fid}' \| \sigma),$$

$$\overline{\mathrm{fid}'} \leftarrow H_K(g^t h^s),$$

$$\tau \leftarrow g_2^\alpha \left(\sigma^{a_0} \prod_{\zeta=1}^{l} \sigma^{a_\zeta [\overline{\mathrm{fid}'}]_\zeta} \right) = g_2^\alpha \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\mathrm{fid}'}]_\zeta} \right)^r,$$

if $\mathrm{fid}' \in T$, it retrieves the corresponding (σ, r, τ, s) from memory.

Then it sets $T \leftarrow \sigma^{\delta_i}$ and $M \leftarrow W^{b_i} = m_i^w$.
By inspection, one can check that τ, M and T are correctly distributed as in the real case.
To answer the other queries with dataset identifier $\mathrm{fid} \neq \mathrm{fid}'$ w.r.t. index i it does the following.

if $\mathrm{fid} \notin T$, it chooses fresh random $s, r \xleftarrow{\$} \mathbb{Z}_p$ and sets

$$\sigma \leftarrow g^r,$$

$$t \leftarrow H_K(\mathrm{fid} \| \sigma),$$

$$\overline{\mathrm{fid}} \leftarrow H_K(g^t h^s),$$

$$\tau \leftarrow g_2^\alpha \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\mathrm{fid}}]_\zeta} \right)^r,$$

if $\mathrm{fid} \in T$, it retrieves the corresponding (σ, r, τ, s) from memory.

Then it sets $m \leftarrow g^{b_i}$ for a random $b_i \xleftarrow{\$} \mathbb{Z}_p$, $T \leftarrow (h_i M_i)^r$, $M \leftarrow W^{b_i}$. Like in the previous case the signature is not directly returned to $\mathcal{A}$ but associated with a new handle h and stored in a table T.

Derivation and Reveal Queries are handled as in the real experiment.

Forgery Assume that the adversary $\mathcal{A}$ produced a forgery $\mathbf{Sign}^*$ for the function $f^* = (\alpha_1^*, \ldots, \alpha_k^*)$ w.r.t fid^*. If $\mathrm{fid}^* \neq \mathrm{fid}'$, then it aborts. Otherwise it proceeds as follows.
Considering the signature $\overline{\mathbf{Sign}} = (\overline{\mathrm{fid}}, \overline{\sigma}, \overline{\tau}, \overline{T}, \overline{M}, s)$ for the message $\overline{m}$ and the function f^* (that the simulator can compute by the **Eval** algorithm from the function f^* provided by $\mathcal{A}$ and the messages m_i chosen by the simulator itself),

we can and extract a 2-3CDH solution by the couple $\left(\frac{m^*}{\prod_{i=1}^k m_i^{\alpha_i^*}}, \frac{T^*}{\overline{T}}\right)$; in fact the elements of the couple are not trivial by the definition of this subcase.

Case 4. In this case we can use exactly the same simulation of case 3, and assume, just to simplify the notation, that the adversary asks for exactly k signing queries (otherwise the simulator can just compute them on his own). In fact, since f^* is not in the span of the vectors $\{f_1, \ldots, f_\nu\}$, the probability that $f^*(m_1, \ldots, m_k) = m^*$ (where $m_1, \ldots, m_k$ are the vectors signed by the simulator in response to signing queries) is negligible and so we can extract a 2-3CDH solution as in the previous case. This is true because, in response to a signing query, the adversary is not even given the message that the simulator chooses at random, but only a handle. So the only information the adversary learns about those messages are the outputs of the reveal queries (where it can basically choose a vector $f = (\alpha_1, \ldots, \alpha_k)$ and learn m such that $m = \prod_{i=1}^k m_i^{\alpha_i}$. Therefore, by the definition of this case, $f^*(m_1, \ldots, m_k)$ is information theoretically hidden from the adversary, and it can only guess it with negligible probability.

C A scalar scheme secure under CMA

Let $\mathbb{G}$, $\mathbb{G}_T$ be groups of prime order p such that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map, let $l \leftarrow \lceil \log p \rceil$ and $\mathcal{H} = \{H_K : \{0,1\}^* \to \{0,1\}^l\}$ be a family of collision resistant hash functions.

KeyGen($1^\lambda, k$): Chooses two random generators $g, g_2 \in \mathbb{G}$ and picks an hash function H_K from $\mathcal{H}$
Picks random $\alpha, w_1, w_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets $g_1 = g^\alpha, W_1 \leftarrow g^{w_1}, W_2 \leftarrow g^{w_2}$.
Selects random group elements $A_1, \ldots A_l, h_1^{(1)}, \ldots, h_1^{(k)}, h_2^{(1)}, \ldots, h_2^{(k)} \xleftarrow{\$} \mathbb{G}$.
Sets $\mathrm{vk} \leftarrow (g, g_1, g_2, W_1, W_2, A_0, \ldots, A_l, h_1^{(1)}, \ldots, h_1^{(k)}, h_2^{(1)}, \ldots, h_2^{(k)}, K)$ as the public verification key and $\mathrm{sk} = (w_1, w_2, g_2^\alpha)$ as the secret signing key.

Sign(sk, m, fid, i): Stores a list $\mathcal{L}$ of all previously returned dataset identifiers fid (together with the related secret information r and public information σ, τ, s defined below) and works according to the type of fid it is given in input:

If fid $\notin \mathcal{L}$, then it chooses $r_1, r_2, s \xleftarrow{\$} \mathbb{Z}_p$ and sets $\sigma_1 \leftarrow g^{r_1}$, $\sigma_2 \leftarrow g^{r_2}$, $t \leftarrow H_K(\mathrm{fid} \| \sigma_1 \| \sigma_2)$, $\overline{fid} \leftarrow H_K(g^t h^s), \tau \leftarrow g_2^\alpha (A_0 \prod_{\zeta=1}^l A_\zeta^{[\overline{fid}]_\zeta})^{r_1 + r_2}$.

Else if fid $\in \mathcal{L}$, then it retrieves the associated $r_1, r_2, s, \sigma_1, \sigma_2, \tau$ from memory.

The message m to be signed is written as $m_1 m_2$ by choosing random $m_1 \xleftarrow{\$} \mathcal{M}$ and computing $m_2 \leftarrow m(m_1)^{-1}$. Then it sets $M_1 \leftarrow {m_1}^{w_1}, M_2 \leftarrow {m_2}^{w_2}, T_1 \leftarrow (h_i^{(1)} M_1)^{r_1}, T_2 \leftarrow (h_i^{(2)} M_2)^{r_2}$ (if a signature for the same fid and the same index i was already issued, then it aborts). Finally it outputs the signature $\mathbf{Sign} \leftarrow (\mathrm{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, s, e_j)$, where e_i is the i-th vector of the canonical base of $\mathbb{Z}^k$

Verify $(\mathrm{vk}, m, \mathbf{Sign}, \boldsymbol{f})$: Parses the signature **Sign** as $(\mathrm{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, s)$ and $\boldsymbol{f}$ as $(f_1, \ldots, f_k)$, computes $\overline{\mathrm{fid}} \leftarrow H_K(g^{H_K(\mathrm{fid} \| \sigma_1 \| \sigma_2)} h^s)$ and $m_2 \leftarrow m m_1^{-1}$. Then it checks that:

$$e(\tau, g) = e(g_2, g_1) \cdot e(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\mathrm{fid}}]_\zeta}, \sigma_1 \sigma_2)$$

$$e(M_1, g) = e(m_1, W_1)$$

$$e(M_2, g) = e(m_2, W_2)$$

$$e(T_1, g) = e(\prod_{i=1}^{k} h_1^{(i)^{f_i}} M_1, \sigma_1)$$

$$e(T_2, g) = e(\prod_{i=1}^{k} h_2^{(i)^{f_i}} M_2, \sigma_2)$$

If all the above equations hold outputs 1, else outputs 0.

Eval $(\mathrm{vk}, \boldsymbol{\alpha}, \mathbf{Sign}_1, \ldots, \mathbf{Sign}_k)$: Parse $\boldsymbol{\alpha}$ as $(\alpha_1, \ldots, \alpha_k)$ and $\mathbf{Sign}_i$ as

$$(\mathrm{fid}_i, \sigma_1^{(i)}, \sigma_2^{(i)}, \tau^{(i)}, T_1^{(i)}, T_2^{(i)}, m_1^{(i)}, M_1^{(i)}, M_2^{(i)}, s_i)\, \forall i = 1, \ldots, k.$$

Then check that all $\mathbf{Sign}_i$ share the same $\mathrm{fid}, \sigma_1, \sigma_2, \tau, s$ components and, if not, reject. Otherwise, compute $T_1 \leftarrow \prod_{i=1}^{k} T_1^{(i)^{\alpha_i}}, T_2 \leftarrow \prod_{i=1}^{k} T_2^{(i)^{\alpha_i}}$, $M_1 \leftarrow \prod_{i=1}^{k} M_1^{(i)^{\alpha_i}}, M_2 \leftarrow \prod_{i=1}^{k} M_2^{(i)^{\alpha_i}}$ and $m_1 \leftarrow \prod_{i=1}^{k} m_1^{(i)^{\alpha_i}}$. Finally output $\mathbf{Sign} = (\mathrm{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, s)$.

Theorem 16. *If 2-3CDH and the discrete logarithm (DL) assumptions hold and $\mathcal{H}$ is a family of collision resistant hash functions than the scheme described above is a linearly homomorphic structure preserving signature scheme unforgeable against a chosen message attack.*

Proof. Let $m^*, \mathbf{Sign}^* = (\mathrm{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, T_1^*, T_2^*, m_1^*, M_1^*, M_2^*, s^*), f^*$ be the forgery returned by the adversary $\mathcal{A}$, Q the set of answers returned to $\mathcal{A}$ in response to its reveal queries, and let $Q_{fid^*} = \{(h_\eta, (fid^*, m_\eta, \sigma_\eta, f_\eta))\}_{\eta=1,\ldots,\nu} \subseteq Q$ be the set of signatures seen by $\mathcal{A}$ for which $\mathrm{fid} = \mathrm{fid}^*$.
Then (at least) one of the following conditions hold:

Case 1: Q_{fid^*} is empty.
Case 2: Q_{fid^*} is not empty, but $(\sigma_1^*, \sigma_2^*, \tau^*, s^*) \neq (\sigma_1^{(\eta)}, \sigma_2^{(\eta)}, \tau_\eta, s_\eta)$ for all $\eta = 1, \ldots, \nu$
Case 3: Q_{fid^*} is not empty, $(\sigma_1^*, \sigma_2^*, \tau^*, s^*) = (\sigma_1^{(\eta)}, \sigma_2^{(\eta)}, \tau_\eta, s_\eta)$ for all $\eta = 1, \ldots, \nu$, the function f^* (interpreted as a vector) is in the span of $\{f_1, \ldots, f_\nu\}$ but, for any $\alpha_1, \ldots, \alpha_\nu$ such that $f = \sum_{\eta=1}^{\nu} \alpha_\eta f_\eta$, it holds $m^* \neq \prod_{\eta=1}^{\nu} m_\eta^{\alpha_\eta}$.
Case 4: Q_{fid^*} is not empty, $(\sigma_1^*, \sigma_2^*, \tau^*, s^*) = (\sigma_1^{(\eta)}, \sigma_2^{(\eta)}, \tau_\eta, s_\eta)$ for all $\eta = 1, \ldots, \nu$ and f^* (interpreted as a vector) is not in the span of $\{f_1, \ldots, f_\nu\}$.
As one can notice, the simulator can guess in which case he will be in advance with probability at least $1/4$.
Case 1. In this case we construct a simulator $\mathcal{B}$ that solves CDH using an adversary $\mathcal{A}$ against the signature scheme described above. The simulator receives as input (g, g^a, g^b) (for a and b he does not know), and behaves as follows:

Key Generation It sets $g_1 \leftarrow g^a, g_2 \leftarrow g_b$ (thus implicitly defining part of the secret key as $g_2^a = g^{ab}$) and picks an hash function $H_K \xleftarrow{\$} \mathcal{H}$. Then it chooses $w_1, w_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets $W_1 \leftarrow g^{w_1}, W_2 \leftarrow g^{w_2}$, selects random group elements $h_1^{(1)} = g^{l_1^{(1)}}, \ldots, h_1^{(k)} = g^{l_1^{(k)}}, h_2^{(1)} = g^{l_2^{(1)}}, \ldots, h_2^{(k)} = g^{l_2^{(k)}}, h = g^\ell \in \mathbb{G}$ for random $\ell, l_1^{(1)}, \ldots, l_1^{(k)}, l_2^{(1)}, \ldots, l_2^{(k)} \in \mathbb{Z}_p$.
Next it chooses $A_0, A_1, \ldots, A_l$ in the same way as in security proof of Waters' signature [45]. Because of this choice, there exist two functions $J, K : \{0,1\}^l \to \mathbb{Z}$

(these functions are all kept internal to the simulator) hidden to the adversary such that, for any string $\text{fid} \in \{0,1\}^l$, the expression $Y(\text{fid}) := A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\text{fid}]_\zeta}$ can be written as $Y(\text{fid}) = g_2^{J(\text{fid})} g^{K(\text{fid})}$. In addition, it was proven that for any distinct $\tau, \tau_1, \ldots, \tau_q \in \{0,1\}^l$ we will have $J(\tau) = 0 \mod p$ and $J(\tau_i) \neq 0 \, \forall i \in \{1, \ldots, q\}$ with non negligible probability $\eta = \frac{1}{8q(l+1)}$.
Finally, $\mathcal{B}$ creates two empty tables T and Q (used to store the output of signing and reveal queries, as explained in the security definition) and gives vk to $\mathcal{A}$.

Signing queries When $\mathcal{A}$ asks a new signing query (m, fid, i) for a message m w.r.t. dataset identifier fid and index i, $\mathcal{B}$ does the following:

if fid $\in T$, it retrieves the corresponding $(\sigma_1, \sigma_2, r, \tau, s)$ from memory.

if fid $\notin T$, it chooses a random $\gamma \in \mathbb{Z}_p$ and sets $\overline{\text{fid}} \leftarrow H_K(g^\gamma)$. if $J(\overline{\text{fid}}) = 0 \mod p$ it aborts. Else it chooses random $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets[8]

$$\tau \leftarrow (Y(\overline{\text{fid}}))^{r_1+r_2} g_1^{-\frac{K(\overline{\text{fid}})}{J(\overline{\text{fid}})}}$$

$$\sigma_1 \leftarrow g^{r_1} g_1^{-\frac{1}{J(\overline{\text{fid}})}}; \quad \sigma_2 \leftarrow g^{r_2}$$

Then $\mathcal{B}$ sets $t \leftarrow H_K(\text{fid} \| \sigma_1 \| \sigma_2)$ and $s \leftarrow \frac{\gamma - t}{\ell}$

The rest of the signature is computed as follows. First $\mathcal{B}$ chooses a random $\lambda \xleftarrow{\$} \mathbb{Z}_p$ and sets $m_1 = g^\lambda$, $m_2 = m/m_1$. Then it sets $M_1 \leftarrow m_1^{w_1}$, $M_2 \leftarrow m_2^{w_2}$, $T_1 \leftarrow \sigma_1^{l_1^{(i)} + \lambda_1 w} = \left(h_1^{(i)} M_1\right)^{r'_1}$, $T_2 \leftarrow \left(h_2^{(i)} M_2\right)^{r'_2}$. The signature $(\text{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, s)$ and the message m are not directly returned to $\mathcal{A}$, but associated with a new handle h (together with the trivial function e_i) and stored in the table T.

Derivation and Reveal Queries are handled as in the real experiment

Forgery Once $\mathcal{A}$ provides a forgery $\mathbf{Sign}^* = (\text{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, T_1^*, T_2^*, m_1^*, M_1^*, M_2^*, s^*)$ $\mathcal{B}$ computes $J(\overline{\text{fid}}^*)$ and aborts if $J(\overline{\text{fid}}^*) \neq 0$

From this forgery, $\mathcal{A}$ can extract a CDH solution as follows. First, notice that by correctness the components τ^*,σ_1^* and σ_2^* of the forgery will be of the form

$$\tau^* = g^{ab} \left(Y(\overline{\text{fid}}^*)\right)^{r'_1+r'_2} \quad \sigma_1^* = g^{r'_1} \quad \sigma_2^* = g^{r'_2}$$

Thus the solution of the CDH instance can be computed as

$$g^{ab} = \tau / (\sigma_1^* \sigma_2^*)^{K(\overline{\text{fid}^*})}$$

Case 2. In this case, the adversary produces a forgery for a fid it has seen a signature about, but the part of the forgery used to verify the fid, namely $(\sigma_1^*, \sigma_2^*, \tau^*, s^*)$, is different from what the simulator stored in the table Q (to fix the notation, we will assume $(m, \mathbf{Sign} = (\text{fid}^*, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, s), f)$ is recorded in Q during the simulation).

Let $t^* \leftarrow H_K(\text{fid}^* \| \sigma_1^* \| \sigma_2^*), \overline{\text{fid}}^* \leftarrow H_K(g^{H_K(\text{fid}^* \| \sigma_1^* \| \sigma_2^*)} h^{s^*})$,and let $t, \overline{\text{fid}}$ the corresponding values computed from $(m, \mathbf{Sign})$ (as in the real experiment, signatures in Q_{fid^*} will all lead to the same values). Depending on these quantities, we have three different sub-cases:

[8] These values are correctly distributed, as one can easily check that they can be written as $\tau = g^{ab}(Y(\overline{\text{fid}}))^{r'_1+r'_2}$, $\sigma_1 = g^{r'_1}, \sigma_1 = g^{r'_2}$ where $r'_1 = r_1 - \frac{a}{J(\overline{\text{fid}})}$, $r'_2 = r_2$.

2.a $\overline{\text{fid}^*} = \overline{\text{fid}}$ and $t^* = t$
2.b $\overline{\text{fid}^*} = \overline{\text{fid}}$ and $t^* \neq t$
2.c $\overline{\text{fid}^*} \neq \overline{\text{fid}}$

Case 2.a. It is easy to build a simulator against the collision resistance of $\mathcal{H}$. Namely, the simulator $\mathcal{B}$ receives in input an hash key k' and has to come up with a couple of elements (y_1, y_2) such that $H_{k'}(y_1) = H_{k'}(y_2)$. Supposing he will get a forgery in this sub-case, $\mathcal{B}$ can just run the ideal experiment but set $K \leftarrow k'$ as the hashing key inside vk. When $\mathcal{A}$ outputs a forgery $(\text{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, T_1^*, T_2^*, m_1^*, M_1^*, M_2^*, s^*)$, by the fact that $\overline{\text{fid}^*} = \overline{\text{fid}}$, we have $H_K(g^{t^*} h^{s^*}) = H_K(g^t h^s)$. So if $s \neq s^*$, because $t = t^*$, we already have a collision (here we require that each element of the group $\mathbb{G}$ and each value in $\mathbb{Z}_p$ have a unique encoding). Otherwise, if $s = s^*$, it must be that $(\sigma_1^*, \sigma_2^*) \neq (\sigma_1, \sigma_2)$ (if this was not the case, then it must be that $\tau^* = \tau$ and therefore this cannot be a case 2 forgery).
So we have that $H_K(\text{fid}^* \| \sigma_1^* \| \sigma_2^*) = t^* = t = H_K(\text{fid}^* \| \sigma_1 \| \sigma_2)$ but $\text{fid}^* \| \sigma_1^* \| \sigma_2^* \neq \text{fid}^* \| \sigma_1 \| \sigma_2$, and $\mathcal{B}$ can return those values as a collision against the hash function.

Case 2.b. In this case, we can build a simulator $\mathcal{B}$ that breaks the DL problem. $\mathcal{B}$ receives in input a couple (g', h'), and its goal is to output β such that $g'^{\beta} = h'$. $\mathcal{B}$ can run the simulation as follows:

Key Generation $\mathcal{B}$ sets $g \leftarrow g'$, $h \leftarrow h'$, and computes the other elements of the public key vk as in the real case. Then it gives vk to $\mathcal{A}$ and stores the secret key sk.
Queries All types of queries are handled as in the real experiment.
Forgery Suppose $\mathcal{A}$ returns a type 2.*b* forgery. Then it must be that $H_K(g^{t^*} h^{s^*}) = H_K(g^t h^s)$ and $t^* \neq t$. If $g^{t^*} h^{s^*} \neq g^t h^s$ then we have a collision for H_K (and we can run a simulation similar to the previous case). If $g^{t^*} h^{s^*} = g^t h^s$, then $\mathcal{B}$ can return $\beta = \frac{t - t^*}{s^* - s}$ as a solution for the DL instance (note that it can't be $s^* = s$ because otherwise $t^* = t$).

Case 2.c. Suppose $\mathcal{A}$ returns a 2.*c* type forgery. In this case, we want to reduce the security of this scheme to the one of Waters' weak signature scheme, by showing how to construct a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to break that scheme. $\mathcal{B}$ receives in input a public key $\text{vk} = (g, g_1, g_2, A_0, A_1, \ldots, A_l)$ for Waters' weak signature scheme. It needs to output a valid forgery for this scheme.

Key Generation $\mathcal{B}$ chooses $H_K \leftarrow \mathcal{H}$, $a \xleftarrow{\$} \mathbb{Z}_p$ and sets $h \leftarrow g^a$. Then it chooses $w_1, w_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets $W_1 \leftarrow g^{w_1}, W_2 \leftarrow g^{w_2}$, selects random $l_1^{(1)}, \ldots, l_1^{(k)}, l_2^{(1)}, \ldots, l_2^{(k)} \in \mathbb{Z}_p$ and sets $h_1^{(1)} \leftarrow g^{l_1^{(1)}}, \ldots, h_1^{(k)} \leftarrow g^{l_1^{(k)}}, h_2^{(1)} \leftarrow g^{l_2^{(1)}}, \ldots, h_2^{(k)} \leftarrow g^{l_2^{(k)}}$. Finally a $\mathcal{B}$ gives to $\mathcal{A}$ the public key $\text{vk}_1 = (g, g_1, g_2, W_1, W_2, A_0, \ldots, A_l, h_1^{(1)}, \ldots, h_1^{(k)}, h_2^{(1)}, \ldots, h_2^{(k)}, K)$.
Signing Queries Each time $\mathcal{A}$ asks for a new query (m, fid, i) on a message m w.r.t. dataset fid and index i, $\mathcal{B}$ responds in this way.
if fid $\in T$ it retrieves the corresponding $(\sigma_1, \sigma_2, \tau, s)$ from the memory.

if fid $\notin T$ it sets $\overline{\text{fid}} = H_K(g^{\beta})$ for $\beta \xleftarrow{\$} \mathbb{Z}_p$; then it asks its challenger for a signature on $\overline{\text{fid}}$ and receives (σ_1, τ_1). Next it chooses a random $r_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets $\sigma_2 \leftarrow g^{r_2}$ $t \leftarrow H_k(\text{fid} \| \sigma_1 \| \sigma_2)$, $s \leftarrow \frac{\beta - t}{a}$. Then it chooses $\lambda \xleftarrow{\$} \mathbb{Z}_p$ and sets:

$$m_1 \leftarrow g^{\lambda}; \quad m_2 \leftarrow m / m_1$$

$$M_1 \leftarrow (m_1)^{w_1}; \quad M_2 \leftarrow (m_2)^{w_2}$$

$$T_1 \leftarrow \sigma_1^{l_1^{(i)}+\lambda w_1}; \quad T_2 \leftarrow (h_2^{(i)} M_2)^{r_2}$$

$$\tau \leftarrow \tau_1 (A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\text{fid}}]_\zeta})^{r_2}.$$

As in the real experiment, the signature $(\text{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, s)$ is stored in the table T together with an handle h which is returned to $\mathcal{A}$.

Derivation and Reveal Queries are handled as in the real experiment.

Forgery When $\mathcal{A}$ returns a type 2.*c* forgery $(\text{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, T_1^*, T_2^*, m_1^*, M_1^*, M_2^*, s^*)$, $\mathcal{B}$ computes $t^* \leftarrow H_k(\text{fid}^*, \|\sigma_1^*\|\sigma_2^*)$, $\overline{\text{fid}^*} \leftarrow H(g^t h^{s^*})$ and outputs $(\overline{\text{fid}^*}, \sigma_1^* \sigma_2^*, \tau^*)$ as a forgery against Waters' scheme.

We stress that this is a valid forgery for the signature scheme, as no other signature has been requested by the simulator for $\overline{\text{fid}^*}$. In fact, by definition of this subcase we have that $\overline{\text{fid}^*} \neq \overline{\text{fid}}$ (where $\overline{\text{fid}}$ is the one computed from the signatures in Q_{fid^*}). In addition, if there were another $\text{fid}' \in Q$ such that $\overline{\text{fid}^*} = \overline{\text{fid}'}$ then it would be trivial to find a collision for H_K.

Case 3. First of all one can notice that, by definition of a valid forgery, it must be that

$$M_1^* = (m_1^*)^{w_1} \quad \text{and} \quad T_1^* = \left(M_1^* \prod_{i=1}^{k} (h_1^{(i)})^{\alpha_i^*} \right)^{r_1}$$

$$M_2^* = (m_2^*)^{w_2} \quad \text{and} \quad T_2^* = \left(M_2^* \prod_{i=1}^{k} (h_2^{(i)})^{\alpha_i^*} \right)^{r_2}$$

Moreover, the same two equations must also hold for the honestly computed signature for the function f^* computed on the messages originally signed by the simulator (we call $m = \prod_{i=1}^{k} (m^{(i)})^{\alpha_i^*}, \mathbf{Sign}$ such couple, and $(m^{(i)}, \mathbf{Sign}^{(i)})$ each of the message/signature made by the simulator in response to a query for index i and dataset fid^*) . So it must be that:

$$T_1^* T_1^{-1} = \left(M_1^* \prod_{i=1}^{k} M_1^{(i)^{-\alpha_i^*}} \right)^{r_1}, \quad T_2^* T_2^{-1} = \left(M_2^* \prod_{i=1}^{k} M_2^{(i)^{-\alpha_i^*}} \right)^{r_2} \tag{1}$$

By the assumption of this subcase, it cannot be that both the left hand sides of these equations are 1. In fact

$$m_1^* m_2^* = m^* \neq m = m_1 m_2 = \prod_{i=1}^{k} \left(m_1^{(i)}\right)^{\alpha_i^*} \left(m_2^{(i)}\right)^{\alpha_i^*}$$

and therefore

$$m_b^* \prod_{i=1}^{k} m_b^{(i)^{-\alpha_i^*}} \neq 1$$

for at least one value of $b \in \{1, 2\}$. In this case, we describe a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to break the 2-3CDH assumption. $\mathcal{B}$ works as follows. It takes in input a 2-3CDH tuple (g, g^w, g^r) and guesses[9] the dataset identifier fid' for which it will receive a forgery

[9] The probability of guessing correctly is polynomial. See footnote 7 for details.

and the value $b \in \{1,2\}$ for which the previous inequality will hold. For the sake of simplicity (and wlog), in the following we will assume it chooses bit $b = 1$

Key Generation $\mathcal{B}$ chooses a random hash key $H_K \xleftarrow{\$} \mathcal{H}$ and random elements $g_2, h \xleftarrow{\$} \mathbb{G}$, sets $W_1 \leftarrow g^w$ (so $w_1 = w$ is not known by the simulator but is implicitly part of the secret key), $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and $g_1 \leftarrow g^\alpha$. Next it chooses $b_i \xleftarrow{\$} \mathbb{Z}_p$ for $i = 1, \ldots, k$ and for each b_i it computes $m_1^{(i)} \leftarrow g^{b_i}$, $M_1^{(i)} \leftarrow m_1^{(i)^{w_1}} = {W_1}^{b_i}$, $h_1^{(i)} \leftarrow g^{\delta_i} {m_1^{(i)}}^{-w_1}$, for random $\delta_1, \ldots, \delta_k \xleftarrow{\$} \mathbb{Z}_p$. Finally it picks random $a_0, a_1, \ldots, a_l \xleftarrow{\$} \mathbb{Z}_p$ and defines $A_i \leftarrow g^{a_i}$, $i = 0, \ldots, l$. The other parts of the public key are generated as in the real experiment. Finally $\mathcal{B}$ gives $\mathrm{vk} = (g, g_1, g_2, W_1, W_2, A_0, \ldots, A_l, h, h_1^{(1)}, \ldots, h_1^{(k)}, h_2^{(1)}, \ldots, h_2^{(k)}, K)$ to $\mathcal{A}$ and stores all the other computed values in memory.

Signing Queries To answer to the queries $(m^{(i)}, \mathrm{fid}', i)$ about identifier fid' and index i asked by $\mathcal{A}$, $\mathcal{B}$ works as follows.

First, if this is the first query asked for identifier fid' by $\mathcal{A}$, it chooses random $s, r_2 \xleftarrow{\$} \mathbb{Z}_p$, sets $\sigma_1 \leftarrow g^r$ (note that it does not know r) and computes

$$\sigma_2 \leftarrow g^{r_2}, \quad t \leftarrow H_K(\mathrm{fid}' \| \sigma_1 \| \sigma_2), \quad \overline{\mathrm{fid}'} \leftarrow H_K(g^t h^s),$$

$$\tau \leftarrow g_2^\alpha \left(\sigma_1^{a_0} \prod_{\zeta=1}^{l} \sigma^{a_\zeta [\overline{\mathrm{fid}'}]_\zeta} \right) \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\mathrm{fid}'}]_\zeta} \right)^{r_2} = g_2^\alpha \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\mathrm{fid}'}]_\zeta} \right)^{r+r_2}.$$

Otherwise, it retrieves all this information from memory.

Then, it fetches the values $m_1^{(i)}, M_1^{(i)}$ generated in the previous phase from memory and computes $m_2^{(i)} \leftarrow m^{(i)}/m_1^{(i)}$ (so that $m^{(i)} = m_1^{(i)} m_2^{(i)}$), $M_2^{(i)} \leftarrow m_2^{(i)}$, $T_1^{(i)} \leftarrow \sigma_1^{\delta_i}$, $T_2^{(i)} \leftarrow (h_2^{(i)} M_2^{(i)})^{r_2}$ (it is easy to check that the signature is valid and each of its components is correctly distributed).

To answer queries $(m^{(i)}, \mathrm{fid}, i)$ about an identifier $\mathrm{fid} \neq \mathrm{fid}'$, $\mathcal{B}$ works in a different way.

First, if this is the first query asked for identifier fid by $\mathcal{A}$, it chooses random $s, r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets

$$\sigma_1 \leftarrow g^{r_1}, \quad \sigma_2 \leftarrow g^{r_2},$$

$$t \leftarrow H_K(\mathrm{fid} \| \sigma_1 \| \sigma_2), \quad \overline{\mathrm{fid}} \leftarrow H_K(g^t h^s), \quad \tau \leftarrow g_2^\alpha (A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\mathrm{fid}}]_\zeta})^{r_1+r_2}$$

Otherwise, it retrieves all this information from memory.

Then, it chooses $c \xleftarrow{\$} \mathbb{Z}_p$, and computes $m_1^{(i)} \leftarrow g^c, m_2^{(i)} \leftarrow m^{(i)}/m_1^{(i)}$ (so that $m^{(i)} = m_1^{(i)} m_2^{(i)}$),

$$M_1^{(i)} \leftarrow W_1^c, \quad M_2^{(i)} \leftarrow (m_2^{(i)})^{w_2}, \quad T_1^{(i)} \leftarrow (h_1^{(i)} M_1^{(i)})^{r_1}, \quad T_2^{(i)} \leftarrow (h_2^{(i)} M_2^{(i)})^{r_2}.$$

In both cases, the signatures are not directly returned to $\mathcal{A}$ but associated with a new handle h and stored in a table T.

Derivation and Reveal Queries are handled as in the real experiment.

Forgery Assume that the adversary $\mathcal{A}$ produced a forgery $\mathbf{Sign}^*$ for the function $f^* = (\alpha_1^*, \ldots, \alpha_k^*)$ and the identifier fid^*. If fid^* was not guessed correctly, $\mathcal{B}$ aborts.
Otherwise it proceeds as follows.
Consider the signature $\overline{\mathbf{Sign}} = (\text{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, \overline{T_1}, \overline{T_2}, \overline{m_1}, \overline{M_1}, \overline{M_2}, s^*)$ for the function f^* and the message $\overline{m} = \prod_{i=1}^{k} (m^{(i)})^{\alpha_i^*}$ computed using the **Eval** algorithm on the couples $(m^{(i)}, \mathbf{Sign}^{(i)})$ stored in T in response to the signing queries made by $\mathcal{A}$. Then, by the assumption of this subcase and supposing $\mathcal{B}$ guessed the correct index b (otherwise it aborts), it must be that $m_b^* \prod_{i=1}^{k} (m_b^{(i)})^{-\alpha_i^*} \neq 1$). Therefore $\mathcal{B}$ can extract a 2-out-of-3 CDH solution by computing $\left(\frac{m_b^*}{\prod_{i=1}^{k} (m_b^{(i)})^{\alpha_i^*}}, \frac{T_b^*}{\overline{T_b}} \right)$ (this can be easily verified by recalling equation 1).

Case 4. The idea to handle this case is the same as the one used in the analogous case 4 of theorem 12. Basically, we use the same simulator of case 3, because the probability that $f^*(m_b^{(1)}, \ldots, m_b^{(k)}) = m_b^*$ is negligible (as in this case $f^*(m_b^{(1)}, \ldots, m_b^{(k)})$ is information theoretically hidden from the adversary).

D Examples of Vector Σ Protocols

As special case of vector sigma protocol we introduce the 1-n vector sigma protocol notion.

Definition 17 (1-n vector sigma protocol). Let $(\mathbb{G}_1, \circ_1)$, $(\mathbb{G}_2, \circ_2)$ two computational groups and let $\mathbf{ChSp} \subseteq \mathbb{G}_2^n$. A 1-$n$ vector sigma protocol consists of four PPT algorithm $\Sigma_n = (\Sigma_n\textbf{-Setup}, \Sigma_n\textbf{-Com}, \Sigma_n\textbf{-Resp}, \Sigma_n\textbf{-Verify})$ defined as follows:

Σ_n**-Setup** $(1^\lambda, n, \mathcal{R}) \rightarrow (\mathbf{x}, \mathbf{w})$. It takes as input a security parameter λ, a vector size n and a relation $\mathcal{R}^n$ over an L^n language. It returns a vector of statements and witnesses $(x_1, \ldots, x_n, w_1, \ldots, w_n)$.

Σ_n**-Com** $(\mathbf{x}) \rightarrow (R, r)$. It's a PPT algorithm run by the prover to get the first message R to send to the verifier and some private state to be stored. We require that $R \in \mathbb{G}_1$.

Σ_n**-Resp** $(\mathbf{x}, \mathbf{w}, r, \mathbf{c}) \rightarrow s$. It's a PPT algorithm run by the prover to compute the last message of the protocol. It takes as input the statements and witnesses $(\mathbf{x}, \mathbf{w})$ the challenge string $\mathbf{c} \in \mathbf{ChSp}$ (sent as second message of the protocol) and some state information r. It outputs the third message of the protocol, s.

Σ_n**-Verify** $(\mathbf{x}, R, \mathbf{c}, s) \rightarrow \{0, 1\}$. It's the verification algorithm that on input the message R, the challenger $\mathbf{c} \in \mathbf{ChSp}$ and a response s it outputs 1 (accept) or 0 (reject).

A 1-n vector sigma protocol is called group homomorphic if it satisfies the properties in definition 13
Now we present some simple variants of the well known identification protocol by Schnorr and show that it is actually an (homomorphic) vector Σ protocol. Those protocols are 1-n Vector Sigma protocol according to definition 17.

Definition 18 (Schnorr Vector Σ-Protocol). Let $\mathbb{G}$ a group of prime order p and $\mathcal{R}$ the DL relation on $\mathbb{G}$, $\text{DL}= \{(x, w) | x = (p, g, h), \quad h = g^w\}$. Let $\overline{g} \in \mathbb{G}$ a group generator. We define $DL_{\overline{g}} = \{(x, w) | x = \overline{g}^w\}$ the restriction of the DL relation to

$g = \overline{g}$.
The vector Schnorr Σ-Protocol consist of four PPT algorithm $\Sigma_n = (\Sigma_n\textbf{-Setup}, \Sigma_n\textbf{-Com}, \Sigma_n\textbf{-Resp}, \Sigma_n\textbf{-Verify})$ defined as follows:

$\Sigma_n\textbf{-Setup}(1^\lambda, n, \mathcal{R})$ It chooses a random group generator $g \in \mathbb{G}$ and a vector of witnesses $\boldsymbol{w} = (w_1, \ldots, w_n) \xleftarrow{\$} \mathbb{Z}_p^n$. Then it computes the vector of statements $(x_1, \ldots, x_n) \leftarrow (g^{w_1}, \ldots, g^{w_n})$ and sets $\boldsymbol{x} \leftarrow (x_1, \ldots, x_n, g)$. Next it outputs $(\boldsymbol{x}, \boldsymbol{w})$. Obviously the couple $(x_j, w_j) \in DL_g \ \ \forall j = 1, \ldots, n$.

$\Sigma_n\textbf{-Com}(\boldsymbol{x})$ It chooses a random $r \in \mathbb{Z}_p$, sets $R \leftarrow g^r$ and returns (r, R).

$\Sigma_n\textbf{-Resp}(\boldsymbol{x}, \boldsymbol{w}, r, \boldsymbol{c})$ Let $\boldsymbol{c} \in \mathbb{Z}_p^n$ be the second message of the protocol. It computes $\boldsymbol{s} \leftarrow (r + c_1 w_1, \ldots, r + c_n w_n)$ and outputs the vector $\boldsymbol{s}$.

$\Sigma_n\textbf{-Verify}(\boldsymbol{x}, R, \boldsymbol{c})$ It checks that

$$g^{s_j} = R x_j^{c_j} \quad \forall j = 1, \ldots, n.$$

If all the above equations hold outputs 1, else outputs 0.

Theorem 19. *The protocol described in definition 18 it's a* 1*-n group homomorphic vector sigma protocol.*

Combining theorems 19, 14, 12 we obtain a practical instantiation of an on-line/off-line linearly homomorphic signature scheme. In particular, as anticipated in remark 8, the scalar LHSPS scheme presented in section 3.1 allows us to change the fid in the online phase in an efficient way. Namely, suppose one wants to convert a signature issued with respect to fid into one with respect to fid$'$. The signer can do so, by first choosing $h \leftarrow g^q$ for a random $q \xleftarrow{\$} \mathbb{Z}_p$ during the key generation phase (as opposed to randomly picking it from $\mathbb{G}$). Then, once it has a signature w.r.t. fid, it can use q to compute a new value s' such that $g^{H_K(\text{fid}\|\sigma)} h^s = g^{H_K(\text{fid}'\|\sigma)} h^{s'}$, by setting

$$s' \leftarrow s + \frac{H_K(\text{fid}\|\sigma) - H_K(\text{fid}'\|\sigma)}{q}$$

It is easy to see that this modification does not affect the validity of the signature.

To further improve efficiency without affecting the security guarantees of our general construction of LHOOS , we now introduce a modified version of the special soundness property for vector sigma protocols. Roughly speaking, the extractor is now given the witnesses for all but one statements of the vector $\boldsymbol{x}$ and has to come up with a witness for the remaining one.

Definition 20 (Strong (Vector) Special Soundness). Let $\Sigma = (\Sigma\textbf{-Setup}, \Sigma\textbf{-Com}, \Sigma\textbf{-Resp}, \Sigma\textbf{-Verify})$ a vector sigma protocol for a relation $\mathcal{R}^n$. We say that Σ has the *Strong Special Soundness* property if there exist an efficient extractor algorithm $\Sigma_n\textbf{-Ext}$ such that $\forall\, \boldsymbol{x} \in L^n, \forall j^* \in \{1, \ldots, n\}, \forall\ R, \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{c'}, \boldsymbol{s'}$ such that $(c, s) \neq (c', s')$, $\Sigma_n\textbf{-Verify}(\boldsymbol{x}, R, \boldsymbol{c}, \boldsymbol{s}) = 1$ and $\Sigma_n\textbf{-Verify}(\boldsymbol{x}, R, \boldsymbol{c'}, \boldsymbol{s'}) = 1$, outputs $w_{j^*} \leftarrow \Sigma_n\textbf{-Ext}(\boldsymbol{x}, R, \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{c'}, \boldsymbol{s'}, \{w_j\}_{j \neq j^*})$ such that $(x_{j^*}, w_{j^*}) \in \mathcal{R}$.

Remark 21. Theorems 14 and 15 can be easily proved also in the case when the vector sigma protocol Σ^n satisfies this Strong Special Soundness property instead of the standard one presented in section 4.1.
As an example, this is a modified version of Schnorr Sigma protocol that has the Strong (Vector) Special Soundness property. Note how the third message of the protocol consists of a single integer value, as opposed to a vector of n integers in the previous construction.

Definition 22 (Strong Schnorr Vector Σ-Protocol). Let $\mathbb{G}$ a group of prime order p and $\mathcal{R}$ the DL relation on $\mathbb{G}$, DL$= \{(x, w)|x = (p, g, h), \ \ h = g^w\}$. Let $\overline{g} \in \mathbb{G}$ a group generator. We define $DL_{\overline{g}} = \{(x, w)|x = \overline{g}^w\}$ the restriction of the DL relation to $g = \overline{g}$.
The Strong Schnorr Vector Σ-Protocol consists of four PPT algorithm $\Sigma_n = (\Sigma_n$-**Setup**, Σ_n-**Com**, Σ_n-**Resp**, Σ_n-**Verify**) defined as follows:

- Σ_n-**Setup**$(1^\lambda, n, \mathcal{R})$ It chooses a random group generator $g \in \mathbb{G}$ and a vector of witnesses $\boldsymbol{w} = (w_1, \ldots, w_n) \xleftarrow{\$} \mathbb{Z}_p^n$. Then it computes the vector of statements $(x_1, \ldots, x_n) \leftarrow (g^{w_1}, \ldots, g^{w_n})$ and sets $\boldsymbol{x} \leftarrow (x_1, \ldots, x_n, g)$. Then it outputs $(\boldsymbol{x}, \boldsymbol{w})$. Obviously the couple $(x_j, w_j) \in DL_g \ \ \forall j = 1, \ldots, n$.
- Σ_n-**Com**$(\boldsymbol{x})$ It chooses a random $r \in \mathbb{Z}_p$, sets $R \leftarrow g^r$ and returns (r, R).
- Σ_n-**Resp**$(\boldsymbol{x}, \boldsymbol{w}, r, \boldsymbol{c})$ Let $\boldsymbol{c} \in \mathbb{Z}_p^n$ the second message of the protocol. This algorithm outputs $s \leftarrow r + \sum_{j=1}^n c_j w_j$.
- Σ_n-**Verify**$(\boldsymbol{x}, R, \boldsymbol{c})$ It checks that

$$g^s = R \prod_{j=1}^{n} x_j^{c_j}.$$

  If the above equation holds, it outputs 1, else outputs 0.