Authenticating Computation on Groups: New Homomorphic Primitives and Applications

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

[1] University of Catania, Italy, {catalano, opuglisi}@dmi.unict.it
[2] Scuola Superiore di Catania, University of Catania, Italy, a.marcedone@studium.unict.it

Abstract In this paper we introduce new primitives to authenticate computation on data expressed as elements in (cryptographic) groups. As for the case of homomorphic authenticators, our primitives allow to verify the correctness of the computation *without* having to know of the original data set. More precisely, our contributions are two-fold.
First, we introduce the notion of *linearly homomorphic authenticated encryption with public verifiability* and show how to instantiate this primitive (in the random oracle model) to support Paillier's ciphertexts. This immediately yields a very simple and efficient (publicly) verifiable computation mechanism for encrypted (outsourced) data based on Paillier's cryptosystem.
As a second result, we show how to construct linearly homomorphic signature schemes to sign elements in bilinear groups (LHSG for short). Such type of signatures are very similar to (linearly homomorphic) structure preserving ones, but they allow for more flexibility, as the signature is explicitly allowed to contain components which *are not* group elements. In this sense our contributions are as follows. First we show a very simple construction of LHSG that is secure against weak random message attack (RMA). Next we give evidence that RMA secure LHSG are interesting on 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. Finally, we present a generic transform that converts RMA-secure LHSG into ones that achieve full security guarantees.

1 Introduction

Homomorphic signatures allow to validate computation over authenticated data. More precisely, a signer holding a dataset $\{m_i\}_{i=1,\ldots,t}$ can produce corresponding signatures $\sigma_i = \textbf{Sign}(\text{sk}, m_i)$ and store the signed dataset can on a remote server. Later the server can (publicly) compute $m = f(m_1, \ldots, m_t)$ together with a (succinct) valid signature σ on it. A keynote feature of homomorphic signature is that the validity of σ can be verified *without* needing to know the original messages $m_1, \ldots, m_n$. Because of this flexibility homomorphic signatures have been investigated in several settings and flavors. Examples include homomorphic signatures for linear and polynomial functions [16,15], redactable signatures [40], transitive signatures and more [46,49]. In spite of this popularity, very few realizations of the primitive encompass the very natural case where the computation one wants to authenticate involves elements belonging to typical cryptographic groups (such as, for instance, groups of points over certain classes of elliptic curves, or groups of residues modulo a composite integer).

Our Contribution. In this paper we put forward new tools that allow to authenticate computation on elements in (cryptographic) groups. In this sense our contributions are two-fold. First, we define a new primitive that we call *Linearly Homomorphic Authenticated Encryption with Public Verifiability* (*LAEPuV* for short). Informally, this primitive allows to authenticate computation on (outsourced) encrypted data, with the additional benefit that the correctness of the computation can be publicly verified. The natural application of this primitive is the increasingly relevant

⋆ Part of the work done while visiting Aarhus University.

scenario where a user wants to store (encrypted) data on the cloud in a way such that she can later delegate the cloud to perform computation on this data. Similarly to homomorphic signatures, LAEPuV allows to verify the correctness of the computation *without* needing to download the original ciphertexts locally.

We show an (efficient) realization of the primitive (in the random oracle model) supporting Paillier's ciphertexts. At an intuitive level our construction works by combining Paillier's encryption scheme with some appropriate additively homomorphic signature scheme. Slightly more in detail, the idea is as follows. One first decrypts a "masking" of the ciphertext C and then signs the masked plaintext using the linearly homomorphic signature. Thus we use the homomorphic signature to authenticate computations on ciphertexts by basically authenticating (similar) computations on the masked plaintexts. The additional advantage of this approach is that it allows to authenticate computation on Paillier's ciphertexts while preserving the possibility to re-randomize the ciphertexts. This means, in particular, that our scheme allows to authenticate computation also on randomized versions of the original ciphertexts[1].

This result allows to implement a very simple and efficient (publicly) verifiable computation mechanism for encrypted (outsourced) data based on Paillier's cryptosystem [47]. Previous (efficient) solutions for this problem rely on linearly homomorphic structure preserving signatures [45] and, as such, only supported cryptosystems defined over pairing-friendly groups. Since, no (linearly homomorphic) encryption scheme supporting exponentially large message spaces is known to exist in such groups, our construction appears to be the first one achieving this level of flexibility.

As a second contribution, we show how to construct a very simple linearly homomorphic signature scheme to sign elements in bilinear groups (LHSG for short). Such type of signatures are very similar to (linearly homomorphic) structure preserving ones, but they allow for more flexibility, as the signature is explicitly allowed to contain components which *are not* group elements (and thus signatures are not necessarily required to comply with the Groth-Sahai famework). More in detail, our scheme is proved secure against random message attack (RMA)[2] under a variant of the Computational Diffie-Hellman assumption introduced by Kunz-Jacques and Pointcheval in [44]. In this sense, our construction is less general (but also conceptually simpler) than the linearly homomorphic structure preserving signature recently given in [45][3].

Interestingly, we show that this simple tool 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, on-line/off-line homomorphic signature could bring similar efficiency benefits to protocols relying on homomorphic signatures. For instance, they could be used to improve the overall efficiency of linear network coding routing mechanisms employing homomorphic signatures to fight pollution attacks[4].

[1] We stress however that this does not buy us privacy with respect to the functionality, i.e. the derived (authenticated) ciphertexts are not necessarily indistinguishable from freshly generated (authenticated) ones.

[2] 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

[3] Also, the scheme from [45] allows to sign vectors of arbitrary dimension, while ours supports vectors composed by one single component only)

[4] This is because the sender could preprocess many off-line computations at night or when the network traffic is low and then use the efficient online signing procedure to perform better when the traffic is high.

We show that RMA-secure LHSG naturally fit this more demanding on-line/off-line scenario. Specifically, we prove that if one combines a RMA-secure LHSG with (vector) 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 [14]. 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 secure homomorphic and network coding signatures offering online/offline efficiency tradeoffs both for the message and the file identifier.

To complete the picture, we provide an efficient and *generic* methodology to convert RMA-secure LHSG into ones that achieve full security[5]. We stress that while similar transforms were known for structure preserving signatures (e.g. [26]), to our knowledge this is the first such transform for the case of *linearly homomorphic* signatures in general.

1.1 Other Related Work

Authenticated Encryption. Authenticated Encryption (AE) allows to simultaneously achieve privacy and authentication. Infact AE is considered to be the standard for symmetric encryption, and many useful applications are based on this primitive using a standard strategy called *generic composition*[6] [50,42,31], or using a different paradigm called *encryption with redundency* introduced by An and Bellare in [8]. Bellare and Namprempre in [13] formalize definition of AE and focus on its security aspects. More closely related to our setting is the notion of homomorphic authenticated encryption recently proposed by Joo and Yun in [41]. With respect to ours, their definitions encompass a wider class of functionalities, however their proposed schemes are not practical and do not consider public verifiability.

Linearly homomorphic signatures The concept of homomorphic signature scheme was originally introduced in 1992 by Desmedt [27], and then refined by Johnson, Molnar, Song, Wagner in 2002 [40]. Linearly homomorphic signatures were introduced in 2009 by Boneh *et al.* [14] as a way to prevent pollution attacks in network coding. Following [14] many other works further explored the notion of homomorphic signatures by proposing new frameworks and realizations [34,10,16,15,22,11,23,30,12,21]. In the symmetric setting constructions of homomorphic message authentication codes have been proposed by [14,35,20].

Recently Libert *et al.* [45] introduced and realized the notion of *Linearly Homomorphic Structure Preserving signatures* (LHSPS for short). Structure Preserving cryptography provides a simple and elegant methodology to compose algebraic tools within the framework of Groth Sahai proof systems [38]. 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) [5,3,4,1,2,17,24,25,32,37,39], commitments [36,6] and encryption schemes [18].

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

[5] for lack of space, the description is deferred to appendix C.

[6] Very informally, usign the *generic composition* paridigm there are 3 ways to obtain an AE sheme: Encrypt and MAC, MAC then Encrypt, Encrypt then MAC

(generically) yield efficient simulation sound trapdoor commitment schemes [33], which in turn imply non malleable trapdoor commitments [28] to group elements.

On-Line/Off-Line Signatures. On-Line/Off-Line digital signature were introduced by Even, Goldreich and Micali in [29]. 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 [29] or using chameleon hash [48].
In [19] 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 $x \overset{\$}{\leftarrow} S$ 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 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 $a, b \overset{\$}{\leftarrow} \mathbb{Z}_p$ 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 [44] 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 $a, b \overset{\$}{\leftarrow} \mathbb{Z}_p$) outputs h, h^{ab} $(h \neq 1)$ with more than negligible probability.

Finally we recall the Decisional Composite Residuosity Assumption, introduced by Paillier in [47].

Definition 3 (DCRA). We say that the Decisional composite residuosity assumption (DCRA) holds if there exists no PPT $\mathcal{A}$ that can distinguish between a random element from $\mathbb{Z}^*_{N^2}$ and one from the set $\{z^N | z \in \mathbb{Z}^*_{N^2}\}$ (i.e. the set of the N-th residues modulo N^2), when N is the product of two random primes of proper size.

3 (Publicly) Verifiable delegation of computation on outsourced cipertext.

In this section, we first describe a new primitive that we call Linearly Homomorphic Authenticated Encryption with Public Verifiability (LAEPuV). This is done by essentially adapting the general definition of Joo and Yun [41] of homomorphic authenticated encryption to the linear case and adding the useful requirement of public verifiability.
Next, we describe an instantiation of this primitive supporting Paillier's scheme as the underlying encryption mechanism.

Definition 4 (LAEPuV). A LAEPuV scheme is a tuple of 5 PPT algorithms (**AKeyGen**, **AEncrypt**, **ADecrypt**, **AVerify**, **AEval**) such that:

- **AKeyGen**$(1^\lambda, k)$ takes as input the security parameter λ, and an upper bound k for the number of messages encrypted in each dataset. It outputs a secret key sk and a public key vk (used for function evaluation and verification); the public key implicitly defines a message space $\mathcal{M}$ which is also a group, a file identifier space $\mathcal{D}$ and a ciphertext space $\mathcal{C}$.
- **AEncrypt**$(\mathrm{sk}, \mathrm{fid}, i, m)$ is a probabilistic algorithm which 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 ciphertext c.
- **AVerify**$(\mathrm{vk}, \mathrm{fid}, c, f)$ takes as input the pubic key vk, a ciphertext $c \in \mathcal{C}$, an identifier fid $\in \mathcal{D}$ and $f \in \mathcal{F}$. It return 1 (accepts) or 0 (rejects).
- **ADecrypt**$(\mathrm{sk}, \mathrm{fid}, c, f)$ takes as input the secret key sk, a ciphertext $c \in \mathcal{C}$, an identifier fid $\in \mathcal{D}$ and $f \in \mathcal{F}$ and outputs $m \in \mathcal{M}$ or $\perp$ (if c is not considered valid).
- **AEval**$(\mathrm{vk}, f, \mathrm{fid}, \{c_i\}_{i=1\ldots k})$ takes as input the public key vk, an admissible function f in its vector form $(\alpha_1, \ldots, \alpha_k)$, an identifier fid, a set of k ciphertexts $\{c_i\}_{i=1\ldots k}$ and outputs a ciphertext $c \in \mathcal{C}$. 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 don't explicitly account this to avoid heavy notation.

The correctness conditions of our scheme are the following:

- For any $(\mathrm{sk}, \mathrm{vk}) \leftarrow$ **AKeyGen**$(1^\lambda, k)$ honestly generated keypair, any $m \in \mathcal{M}$, any dataset identifier fid and any $i \in \{1, \ldots, k\}$, with overwhelming probability

$$\mathbf{ADecrypt}(\mathrm{sk}, \mathrm{fid}, \mathbf{AEncrypt}(\mathrm{sk}, \mathrm{fid}, i, m), e_i) = m$$

  where e_i is the i-th vector of the standard basis of $\mathbb{Z}^k$.
- For any $(\mathrm{sk}, \mathrm{vk}) \leftarrow$ **AKeyGen**$(1^\lambda, k)$ honestly generated keypair, any $c \in C$

$$\mathbf{AVerify}(\mathrm{vk}, \mathrm{fid}, c, f) = 1 \iff \exists m \in \mathcal{M} : \mathbf{ADecrypt}(\mathrm{sk}, \mathrm{fid}, c, f) = m$$

- Let $(\mathrm{sk}, \mathrm{vk}) \leftarrow$ **AKeyGen**$(1^\lambda, k)$ be an honestly generated keypair, fid any dataset identifier, $c_1, \ldots, c_k \in \mathcal{C}$ any tuple of ciphertexts such that $m_i =$ **ADecrypt**$(\mathrm{sk}, \mathrm{fid}, c_i, f_i)$. Then, for any admissible function $f = (\alpha_1, \ldots, \alpha_k) \in \mathbb{Z}^k$, with overwhelming probability

$$\mathbf{ADecrypt}(\mathrm{sk}, \mathrm{fid}, \mathbf{AEval}(\mathrm{vk}, f, \mathrm{fid}, \{c_i\}_{i=1\ldots k}), \sum_{i=0}^{k} \alpha_i f_i) = f(m_1, \ldots, m_k)$$

For lack of space, security definitions are given in appendix A.1

3.1 A LAEPuV instantiation supporting Paillier's encryption

Let (**HKeyGen**, **HSign**, **HVerify**, **HEval**) be a linearly homomorphic signature scheme for groups of the form $\mathbb{Z}_N$ (where N is the product of two distinct (safe) primes) unforgeable against a chosen message attack. Moreover, let $\mathcal{H}$ be a family of collision resistant hash functions (whose images can be interpreted as elements of $Z^*_{N^2}$). Then we can construct a LAEPuV scheme as follows.

AKeyGen$(1^\lambda, k)$: Choose two primes p, q of size $\lambda/2$, set $N \leftarrow pq$ and choose a random element $g \in \mathbb{Z}^*_{N^2}$ of order N. Run[7] **HKeyGen**$(1^\lambda, k, N)$ to obtain a signing key sk' and a verification key vk'. Pick an hash function $H \leftarrow \mathcal{H}$. Return $\mathrm{vk} \leftarrow (\mathrm{vk}', g, N, H)$ as the public verification key and $\mathrm{sk} = (\mathrm{sk}', p, q)$ as the secret signing key.

AEncrypt$(\mathrm{sk}, m, \mathrm{fid}, i)$: Choose random $\beta \leftarrow \mathbb{Z}^*_{N^2}$, compute $C \leftarrow g^m \beta^N \mod N^2$. Set $R \leftarrow H(\mathrm{fid}||i)$, and use the factorization of N to compute $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}^*_N$ such that $g^a b^N = RC \mod N^2$. Compute $\sigma \leftarrow$ **HSign**(sk', fid, i, a) and return $c = (C, a, b, \sigma)$.

AVerify$(\mathrm{vk}, \mathrm{fid}, c, f)$: Parse $c = (C, a, b, \sigma)$ and $\mathrm{vk} \leftarrow (\mathrm{vk}', g, N, H)$, then check that:

$$\textbf{HVerify}(\mathrm{vk}, \mathrm{fid}, a, f) = 1$$

$$g^a b^N = C \prod_{i=1}^{k} H(\mathrm{fid}||i)^{f_i} \mod N^2$$

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

ADecrypt$(\mathrm{sk}, \mathrm{fid}, c, f)$: If **AVerify**$(\mathrm{vk}, \mathrm{fid}, c, f) = 0$, return $\perp$. Otherwise, use the factorization of N to compute (m, β) such that $g^m \beta^N = C \mod N^2$ and return m.

AEval$(\mathrm{vk}, \alpha, \mathrm{fid}, c_1, \ldots, c_k)$: Parse $\alpha = (\alpha_1, \ldots, \alpha_k)$ and $c_i = (C_i, a_i, b_i, \sigma_i)$, set

$$C \leftarrow \prod_{i=i}^{k} C_i^{\alpha_i} \mod N^2, \quad a \leftarrow \sum_{i=i}^{k} a_i \alpha_i \mod N,$$

$$b \leftarrow \prod_{i=i}^{k} b_i^{\alpha_i} \mod N^2, \quad \sigma \leftarrow \textbf{HEval}(vk', \mathrm{fid}, f, \{\sigma_i\}_{i=1,\ldots,k})$$

and return $c = (C, a, b, \sigma)$.

Remark 5. As a concrete instantiation of the previous scheme, it's possible to use a simple variant of the (Strong) RSA based signature of [23] as linearly homomorphic signature scheme on $\mathbb{Z}_N$. A complete description of this signature is presented in appendix D.

Scheme security

Theorem 6. *In the random oracle model, if the DCRA holds,* (**HKeyGen**, **HSign**, **HVerify**, **HEval**) *is a linearly homomorphic signature scheme over* $\mathbb{Z}_N$ *unforgeable (against a chosen message attack) and* H *is a random oracle, then the scheme described above is LH-IND-CCA secure according to definition 20.*

[7] Since the signature scheme must support $\mathbb{Z}_N$ as the message space, we give it to the **HKeyGen** algorithm as an additional argument. Note that (in general) this signature algorithm may not use the factorization of N as part of its private key.

Theorem 7. *If $\Sigma = (\mathbf{HKeyGen}, \mathbf{HSign}, \mathbf{HVerify}, \mathbf{HEval})$ is a linearly homomorphic signature scheme over $\mathbb{Z}_N$ unforgeable (against a chosen message attack), then the scheme described above is LH-Uf-CCA secure according to definition 21.*

For lack of space we defer those proofs to appendixes B.1 and B.2

4 Linearly homomorphic signature scheme to sign elements in bilinear groups

Following [45,30,7], our definition of linearly homomorphic signature scheme to sign elements in bilinear groups is essentially equivalent to the one of linearly homomorphic signature scheme (in its strongest variant). To adapt it to our results, we assume that the message space is some bilinear 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 $\in \{0,1\}^*$, 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 $f \in \mathcal{F}$.

We defer the formal definition to appendix A.2

4.1 A random message secure construction

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 remark 9 we show how to to derive a fully secure scheme using the conversion methodology described in appendix C[8]. Our construction uses as underlying building block a generic signature scheme.
Let $\mathbb{G}$, $\mathbb{G}_T$ be groups of prime order p such that $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear map and $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$ a standard signature with message space $\mathcal{M}$. The scheme works as follows:

HKeyGen$(1^\lambda, 1, k)$: Choose a random generator $g \in \mathbb{G}$ and run **KeyGen**(1^λ) to obtain a signing key sk_1 and a verification key vk_1. Pick random $w \overset{\$}{\leftarrow} \mathbb{Z}_p$ and set $W \leftarrow g^w$. Select random group elements $h_1, \ldots, h_k, \overset{\$}{\leftarrow} \mathbb{G}$.
Set $\mathrm{vk} \leftarrow (\mathrm{vk}_1, g, W, h_1, \ldots, h_k)$ as the public verification key and $\mathrm{sk} = (\mathrm{sk}_1, w)$ as the secret signing key.

HSign$(\mathrm{sk}, m, \mathrm{fid}, i)$: 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 defined below) and works according to the type of fid it is given in input):

If fid $\notin \mathcal{L}$, then choose $r \overset{\$}{\leftarrow} \mathbb{Z}_p$, set $\sigma \leftarrow g^r$, $\tau \leftarrow \mathbf{Sign}(\mathrm{sk}, \mathrm{fid}, \sigma)$

Else if fid $\in \mathcal{L}$, then retrieve the associated r, σ, τ from memory.
Then set $M \leftarrow m^w, V \leftarrow (h_i M)^r$ (if a signature for the same fid and the same index i was already issued, then abort). Finally output $\pi \leftarrow (\sigma, \tau, V, M)$ 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 $\mathbb{Z}^n$).

[8] More precisely the scheme proposed in remark 9 is a slightly optimized version of what one would get by naively converting our random message secure scheme. See remark 9 for details.

HVerify(vk, π, m, fid, f): Parse the signature π as (σ, τ, V, M) and f as $(f_1, \ldots, f_k)$. Then check that:

$$\mathbf{Verify}(\mathrm{vk}, \tau, (\mathrm{fid}, \sigma)) = 1$$

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

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

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

HEval (vk, $\alpha, \pi_1, \ldots, \pi_k$): Parse α as $(\alpha_1, \ldots, \alpha_k)$ and π_i as $(\sigma_i, \tau_i, V_i, M_i)$, $\forall i = 1, \ldots, k$. Then, compute $V \leftarrow \prod_{i=1}^{k} V_i^{\alpha_i}$, $M \leftarrow \prod_{i=1}^{k} M_i^{\alpha_i}$ and output $\pi = (\sigma_1, \tau_1, V, M)$ (or $\perp$ if the σ_i are not all equal).

The security of the scheme follows from the following theorem (whose proof is deferred to appendix B.3)

Theorem 8. *If the 2-3CDH assumption holds and $\mathcal{S}$ is a signature scheme unforgeable under adaptive chosen message attack then the scheme described above is a LHSG scheme secure against a random message attack according to definition* 23.

Remark 9. Combining the LHSG described above and theorem 29 we can easily construct an LHSG secure under CMA. Moreover in this particular case it's possible to have just one signature on the fid by computing $\tau = \mathbf{Sign}(\mathrm{sk}, (\mathrm{fid}, \sigma_1, \sigma_2))$ instead of $\tau_1 = \mathbf{Sign}(\mathrm{sk}, (\mathrm{fid}, \sigma_1))$ and $\tau_2 = \mathbf{Sign}(\mathrm{sk}, (\mathrm{fid}, \sigma_2))$ (and therefore using a signature scheme where the message space is $\mathcal{M} = \{0, 1\}^* \times \mathbb{G}^2$). The security proof for this modified scheme is easily adapted from the general one and is therefore omitted.

Remark 10. If the practical application allows the fid to be a group element and not simply a string, we can replace the signature $\mathcal{S}$ with a Structure preserving Signature satisfying the same hypothesis of theorem 8 to obtain the first example of a linearly homomorphic structure preserving signature scheme (LHSPS) where all parts of the signature are actually elements of the group (as opposed to [45], where the fid is inherently used as a bit string). In addition, if the identifier can be chosen at random by the signer and not by the adversary, we can even define σ to be the identifier itself and thus further improve efficiency. In practical instantiation it's possible to use the SPS of [4].

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

In this section, we show a general construction to build an (efficient) on-line/off-line homomorphic (and network coding) signature scheme by combining a LHSG unforgeable against a random message attack (like the one described in section 4.1) with a certain class of sigma protocols. The intuitive idea is that in order to sign a certain message m, one can choose a Σ-Protocol whose challenge space contains m, then sign the first message of the Σ-Protocol with a standard signature (this can be done off-line) and use its knowledge of the witness of the protocol to later compute the response (third message) of the protocol associated to the challenge m. This is secure because, if an adversary was able to produce a second signature with respect to the same first message, by the special soundness of the Σ-Protocol, he would be able to recover the witness itself. We show

how, if both the signature scheme and the Σ-Protocol have specific homomorphic properties, this construction can be extended to build (linearly) homomorphic signatures as well.
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). First, we adapt the definition of linearly homomorphic signature (LHSG) to the On-line/Off-line case. Then, we precisely define the properties required by the sigma protocol, and as a last step we will describe and prove the security of our construction.

5.1 Linearly Homomorphic On-line/Off-line signatures

First, we remark that the only difference between a LHSG and a LHOOS is in the signing algorithm. When signing m the latter can use some data prepared in advance (by running a dedicated algorithm **OffSign**) to speed up the signature process. The definitions of unforgeability are therefore analogous to the ones of traditional LHSG schemes and are omitted to avoid repetition[9].

Definition 11 (LHOOS). A Linearly Homomorphic On-line/Off-line signature scheme is a tuple of PPT algorithms (**KeyGen**, **OffSign**, **OnSign**, **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 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 $\mathcal{D}$ and a signature space Σ.
- **OffSign**(sk) takes as input the secret key and outputs some information I.
- **OnSign**$(\mathrm{sk}, \mathrm{fid}, I, \mathbf{m}, i)$ 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 an instance of I output by **OffSign**. This algorithm must ensure that all the signatures issued for the same fid are computed using the same information I (i.e. by associating each fid with one specific I and storing these couples on a table). It outputs a signature σ.
- **Verify** $(\mathrm{vk}, \sigma, \mathbf{m}, \mathrm{fid}, f)$ takes as input the public key vk, a signature $\sigma \in \Sigma$, a message $\mathbf{m} \in \mathcal{M}$, a dataset identifier fid $\in \mathcal{D}$ and a function $f \in \mathbb{Z}^k$; it outputs 1 (accept) or 0 (reject).
- **Eval**$(\mathrm{vk}, \mathrm{fid}, f, \{\sigma_i\}_{i=1\ldots k})$ takes as input the public key vk, a dataset identifier fid, an admissible function f in its vector form $(\alpha_1, \ldots, \alpha_k)$, a set of k signatures $\{\sigma_i\}_{i=1\ldots k}$ 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 don't to explicitly account this to avoid heavy notation.

The correctness conditions of our scheme are the following:

- Let $(\mathrm{sk}, \mathrm{vk}) \leftarrow$ **KeyGen**$(1^\lambda, n, k)$ be an honestly generated keypair, $\mathbf{m} \in \mathcal{M}$, fid any dataset identifier and $i \in 1, \ldots, k$. If $\sigma \leftarrow$ **Sign**$(\mathrm{sk}, \mathrm{fid}, \mathbf{OffSign}(sk), \mathbf{m}, i)$, then with overwhelming probability
$$\mathbf{Verify}(\mathrm{vk}, \sigma, \mathbf{m}, \mathrm{fid}, e_i) = 1,$$
where e_i is the ith vector of the standard basis of $\mathbb{Z}^k$.

[9] We stress, however, that those definitions are stronger than the ones traditionally introduced for network coding (i.e. the adversary is more powerful and there are more types of forgeries), and therefore our efficient instantiation perfectly integrates in that framework.

– Let $(\mathrm{sk}, \mathrm{vk}) \leftarrow \mathbf{KeyGen}(1^\lambda, n, k)$ be an honestly generated keypair, $\mathbf{m}_1, \ldots, \mathbf{m}_k \in \mathcal{M}$ any tuple of messages signed (or derived from messages originally signed) w.r.t the same fid (and therefore using the same offline information I), and let $\sigma_1, \ldots, \sigma_k \in \Sigma$, $f_1, \ldots, f_k \in \mathcal{F}$ such that for all $i \in \{1, \ldots, k\}$, $\mathbf{Verify}(\mathrm{vk}, \sigma_i, \mathbf{m}_i, \mathrm{fid}, f_i) = 1$. Then, for any admissible function $f = (\alpha_1, \ldots, \alpha_k) \in \mathbb{Z}^k$, with overwhelming probability

$$\mathbf{Verify}(\mathrm{vk}, \mathbf{Eval}(\mathrm{vk}, \mathrm{fid}, f, \{\sigma_i\}_{i=1\ldots k}), f(\mathbf{m}_1, \ldots, \mathbf{m}_k), \mathrm{fid}, \sum_{i=0}^{k} \alpha_i f_i) = 1$$

Remark 12. As for the case of LHSG, relaxing the requirements for the fid (i.e. assuming that it can be chosen offline independently of the message or that it can even be completely random) typically improves efficiency. See remark 17 for an example of how this idea applies to our instantiation.

5.2 Vector and Homomorphic Σ-protocols

Briefly speaking, a Σ-Protocol can be described as a tuple of four algorithms ($\mathbf{\Sigma}$**-Setup**, $\mathbf{\Sigma}$**-Com**, $\mathbf{\Sigma}$**-Resp**, $\mathbf{\Sigma}$**-Verify**), where the first one generates a statement/witness couple, $\mathbf{\Sigma}$**-Com** and $\mathbf{\Sigma}$**-Resp** generate the first and third message of the protocol, and $\mathbf{\Sigma}$**-Verify** is used by the verifier to decide on the validity of the proof (a more formal and detailed description is given in appendix A.4). This notion can be extended to the vector case[10]. For this purpose we adapt the notion of Homomorphic Identification Protocol originally introduced in [9] to the Sigma protocol framework.

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 natural 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 rounds 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 $\mathbf{\Sigma_n}$**-Ext** such that $\forall\, \boldsymbol{x} \in L^n$, $\forall\, R, \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{c'}, \boldsymbol{s'}$ such that $(c, s) \neq (c', s')$, $\mathbf{\Sigma_n}\text{-}\mathbf{Verify}(\boldsymbol{x}, R, \boldsymbol{c}, \boldsymbol{s}) = 1$ and $\mathbf{\Sigma_n}\text{-}\mathbf{Verify}(\boldsymbol{x}, R, \boldsymbol{c'}, \boldsymbol{s'}) = 1$, outputs $(x, w) \leftarrow \mathbf{\Sigma_n}\text{-}\mathbf{Ext}(\boldsymbol{x}, R, \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{c'}, \boldsymbol{s'})$ where x is one of the components of $\boldsymbol{x}$ and $(x, w) \in \mathcal{R}$.
Another important requirement for our construction to work is the following property.

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

– The outputs of the $\mathbf{\Sigma}$**-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 $\mathbf{\Sigma}\text{-}\mathbf{Verify}(x, R_i, c_i, s_i) = 1$ for all i, then

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

Although it is given for the standard case, this property can easily be extended to vector Σ-Protocols: in particular, the group $\mathbb{G}_2$ can be seen as the group of vectors of elements taken from another group $\mathbb{G}$.

[10] The intuition is that it should be more efficient to run a vector Σ-Protocol once than a standard Σ-Protocol multiple times in parallel)

To sum up, we define a class of vector Σ-Protocols having all the properties required by our construction:

Definition 14 (1-n (vector) Σ-Protocol). Let $(\mathbb{G}_1, \circ_1)$, $(\mathbb{G}_2, \circ_2)$ be two computational groups. A 1-n vector sigma protocol consists of four PPT algorithm $\Sigma_n = (\mathbf{\Sigma_n\text{-}Setup}, \mathbf{\Sigma_n\text{-}Com}, \mathbf{\Sigma_n\text{-}Resp}, \mathbf{\Sigma_n\text{-}Verify})$ defined as follows:

$\mathbf{\Sigma_n\text{-}Setup}(1^\lambda, n, \mathcal{R}^n) \to (\mathbf{x}, \mathbf{w})$. It takes as input a security parameter λ, a vector size n and a relation $\mathcal{R}^n$ over a language L^n. It returns a vector of statements and witnesses $(x_1, \ldots, x_n, w_1, \ldots, w_n)$. The challenge space is required to be $\mathbf{ChSp} \subseteq \mathbb{G}_2^n$.

$\mathbf{\Sigma_n\text{-}Com}(\mathbf{x}) \to (R, r)$. 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 $R \in \mathbb{G}_1$.

$\mathbf{\Sigma_n\text{-}Resp}(\mathbf{x}, \mathbf{w}, r, \mathbf{c}) \to s$. It's a deterministic 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.

$\mathbf{\Sigma_n\text{-}Verify}(\mathbf{x}, R, \mathbf{c}, s) \to \{0, 1\}$. 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).

We require this protocol to be group homomorphic and to satisfy the completeness and special honest verifier zero knowledge properties. Moreover, the protocol must guarantee either the vector special soundness outlined above or a stronger soundness property that we define below.

Roughly speaking, this property requires that the extractor, upon receiving the witnesses for all but one statements of the vector $\boldsymbol{x}$, has to come up with a witness for the remaining one.

Definition 15 (Strong (Vector) Special Soundness). Let $\Sigma = (\mathbf{\Sigma\text{-}Setup}, \mathbf{\Sigma\text{-}Com}, \mathbf{\Sigma\text{-}Resp}, \mathbf{\Sigma\text{-}Verify})$ be a 1-n Σ-Protocol for a relation $\mathcal{R}^n$. We say that Σ has the *Strong Special Soundness* property if there exist an efficient extractor algorithm $\mathbf{\Sigma_n\text{-}Ext}$ 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 $c_{j^*} \neq c'_{j^*}$, $\mathbf{\Sigma_n\text{-}Verify}(\boldsymbol{x}, R, \boldsymbol{c}, \boldsymbol{s}) = 1$ and $\mathbf{\Sigma_n\text{-}Verify}(\boldsymbol{x}, R, \boldsymbol{c'}, \boldsymbol{s'}) = 1$, outputs $w_{j^*} \leftarrow \mathbf{\Sigma_n\text{-}Ext}(\boldsymbol{x}, R, \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{c'}, \boldsymbol{s'}, \{w_j\}_{j \neq j^*})$ such that $(x_{j^*}, w_{j^*}) \in \mathcal{R}$.

In appendix A.5 we show that a simple variant of the well known identification protocol by Schnorr is a 1-n Σ-Protocol (with Strong Vector Special Soundness).

5.3 A Linearly Homomorphic On-Line/Off-Line Signature

Suppose $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{Eval})$ is a randomly secure LHSG (even one that only allows to sign scalars), $\Sigma_n = (\mathbf{\Sigma_n\text{-}Setup}\ , \mathbf{\Sigma_n\text{-}Com}\ , \mathbf{\Sigma_n\text{-}Resp}\ , \mathbf{\Sigma_n\text{-}Verify}\ )$ is a 1-n Σ-Protocol and $\mathcal{H} = (\mathbf{CHGen}, \mathbf{CHEval}, \mathbf{CHFindColl})$ defines a family of chameleon hash functions[11]. Moreover, suppose that the LHSG's message space is the same as the group $\mathbb{G}_1$ of the outputs of $\mathbf{\Sigma_n\text{-}Com}$. Our generic construction uses the challange space of the Σ-Protocol as a message space and works as follows:

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

[11] a formal definition of chameleon hash functions can be found in appendix A.3

OFFSign (sk): This algorithm runs the $\mathbf{\Sigma_n}$**-Com** algorithm k times to obtain $(R_i, r_i) \leftarrow$$\mathbf{\Sigma_n}$**-Com** $(\mathbf{x})$, chooses a random string fid' from the dataset identifiers' space and randomness ρ' and sets $\overline{\mathrm{fid}} \leftarrow$ **CHEval**$(\mathrm{hk}, \mathrm{fid}', \rho')$. Then it signs each R_i using the LHSG signing algorithm $\overline{\sigma_i} \leftarrow$ **Sign**$(sk_1, R_i, \overline{\mathrm{fid}}, i)$ and outputs $I_{\mathrm{fid}'} = \{(i, r_i, R_i, \overline{\sigma_i}, \mathrm{fid}', \rho')\}_{i=1,\ldots,k}$.

ONSign $(\mathrm{vk}, \mathrm{sk}, \mathbf{m}, \mathrm{fid}, I_{\mathrm{fid}'}, i)$: It parses $I_{\mathrm{fid}'}$ as $\{(i, r_i, R_i, \overline{\sigma_i}, \mathrm{fid}', \rho')\}_{i=1,\ldots,k}$, computes $s \leftarrow$$\mathbf{\Sigma_n}$**-Resp** $(\mathbf{x}, \mathbf{w}, r_i, \mathbf{m})$, $\rho \leftarrow$ **CHFindColl**$(\mathrm{ck}, \mathrm{fid}', \rho', \mathrm{fid})$ and outputs $\sigma \leftarrow (R_i, \overline{\sigma}_i, s, \rho)$. As explained in the definition, this algorithm must ensure that all the messages signed with respect to the same fid are computed from the same information $I_{\mathrm{fid}'}$

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

$$\mathbf{Verify}(\mathrm{vk}_1, \overline{\sigma}, R, \mathbf{CHEval}(\mathrm{fid}, \rho), f) = 1 \quad \text{and} \quad \mathbf{\Sigma_n}\text{-}\mathbf{Verify}(\mathbf{x}, R, \mathbf{m}, s) = 1.$$

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

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

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

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

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

Remark 16. The construction presented above applies to any LHSG. However, if the LHGS itself is obtained as described in section 4.1, the use of the chameleon hash function could be avoided by substituting the signature scheme $\mathcal{S}$ used for the fid with an on-line/off-line one. This improves efficiency.

Remark 17. Loosening the requirements on the fid can also help improving the performance of the scheme. For example, if the fid is allowed to be chosen randomly by the signer and not by the adversary, the use of the chameleon hash function can be avoided. Moreover, in this case one could use a LHSG that achieves better efficiency by choosing the fid itself (an example is described in remark 10).

Theorem 18. *If* $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{Eval})$ *is a randomly secure LHSG,* $\Sigma_n = ($$\mathbf{\Sigma_n}$**-Setup** ,$\mathbf{\Sigma_n}$**-Com** ,$\mathbf{\Sigma_n}$**-Resp** ,$\mathbf{\Sigma_n}$**-Verify** $)$ *is a* 1-*n* *Σ-Protocol for a non trivial relation* $\mathcal{R}^n$*, and* $\mathcal{H}$ *implements a family of chameleon hash functions then the LHOOS described above is secure against a chosen message attack according to definition 25.*

For lack of space we defer this proof to appendix B.4.

The security obtained by this construction can be strengthened by assuming additional properties on the underlying LHSG scheme: if $\mathcal{S}$ is strongly secure against a random message attack, then we can prove that the resulting construction is strongly secure (against a CMA) as well.

Theorem 19. *If* $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{Eval})$ *is a LHSG scheme strongly unforgeable against a random message attack and* $\Sigma_n = (\mathbf{\Sigma_n}\text{-}\mathbf{Setup}, \mathbf{\Sigma_n}\text{-}\mathbf{Com}, \mathbf{\Sigma_n}\text{-}\mathbf{Resp}, \mathbf{\Sigma_n}\text{-}\mathbf{Verify})$ *is a* 1-*n* *Σ-Protocol for a non trivial relation* $\mathcal{R}^n$*, then the on-line/off-line scheme described above is* ***strongly*** *unforgeable against chosen message attacks.*

The proof is straightforward and similar to the previous one and therefore is omitted.

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, 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/.
6. 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.
7. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 1–20. Springer, March 2012.
8. Jee Hea An and Mihir Bellare. Does encryption with redundancy provide authenticity? In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 512–528. Springer, May 2001.
9. Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 319–333. Springer, December 2009.
10. 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.
11. 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.
12. 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.
13. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, October 2008.
14. 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.
15. 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.
16. 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.
17. 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.

18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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.
26. 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.
27. Yvo Desmedt. Computer security by redefining what a computer is. NSPW, 1993.
28. 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.
29. Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.
30. 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.
31. A. Freier, P. Karlton, and P. Kocher. The ssl protocol: Version 3.0,, 1996.
32. 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/.
33. 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.
34. 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.
35. 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.
36. J. Groth. Homomorphic trapdoor commitments to group elements. Cryptology ePrint Archive, Report 2009/007, 2009. http://eprint.iacr.org/.

37. 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.
38. 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.
39. 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.
40. 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.
41. Chihong Joo and Aaram Yun. Homomorphic authenticated encryption secure against chosen-ciphertext attack. Cryptology ePrint Archive, Report 2013/726, 2013. http://eprint.iacr.org/.
42. S. Kent. Ip encapsulating security payload (esp). RFC 4303, December 2005.
43. Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *ISOC Network and Distributed System Security Symposium – NDSS 2000*. The Internet Society, February 2000.
44. 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.
45. 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.
46. Silvio Micali and Ronald L. Rivest. Transitive signature schemes. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 236–243. Springer, February 2002.
47. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, May 1999.
48. 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.
49. Xun Yi. Directed transitive signature scheme. In Masayuki Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 129–144. Springer, February 2007.
50. T. Ylonen and C. Lonvick. The secure shell (ssh) transport layer protocol. RFC 4253, January 2006.

A Postponed Definitions

A.1 Security definitions for LAEPuV schemes

Here we define a linearly homomorphic version of the IND-CCA security game for public key encryption. Although this might seem surprising at first, as CCA encryption is usually deployed to prevent malleability of the ciphertexts, it is possible to give a meaningful definition also in the context of homomorphic encryption. Namely, since we want to allow the ciphertexts to be manipulated only up to a certain extent (i.e. linear operations where the function applied is publicly declared), the best thing that we can do is explicitly disallow the decryption queries on a ciphertext legitimately derived from the challange cyphertext (as they could reveal information about the hidden bit the adversary is trying to guess).
Again, our definition is adapted to the linear case from [41].

Definition 20 (Linearly homomorphic indistinguishability under chosen chiperthext attack). Let $\mathcal{H} = (\mathbf{AKeyGen}, \mathbf{AEncrypt}, \mathbf{ADecrypt}, \mathbf{AVerify}, \mathbf{AEval})$ a LAEPuV scheme. Linearly Homomorphic IND-CCA is defined by the following game between a challenger and an adversary $\mathcal{A}$:

$\mathbf{LH\text{-}IND\text{-}CCA}_{\mathcal{H},\mathcal{A}}(1^\lambda, k)$:

- **Setup** The challenger runs $(\mathrm{sk}, \mathrm{vk}) \leftarrow \mathbf{AKeyGen}(1^\lambda, k)$. Then it initializes an empty set S and gives vk to the adversary $\mathcal{A}$.

- **Queries I** $\mathcal{A}$ can ask a polynomial number of encryption and decryption queries. Firsts are of the form (fid, m_i, i) (where fid is a dataset identifier, $m_i \in \mathcal{M}$ is a message and $i \in \{1, \ldots, k\}$ is an index). The challenger computes $c_i \leftarrow \mathbf{AEncrypt}(\mathrm{sk}, \mathrm{fid}, i, m)$, gives c_i to $\mathcal{A}$ and updates the set $S \leftarrow S \cup \{(\mathrm{fid}, m_i, i, c_i)\}$. No two queries differing only on the m_i component can be asked by the adversary (if this happens, the answer to the second query is $\perp$). Decryption queries instead are of the form (fid, c_i, f) and $\mathcal{A}$ gets the corresponding output of $\mathbf{ADecrypt}(\mathrm{sk}, \mathrm{fid}, c, f)$ (It can be $\perp$ if c is a not valid ciphertext).

- **Challenge** $\mathcal{A}$ produces a challenge tuple $(\mathrm{fid}^*, i^*, m_0, m_1)$ (as in the previous phase, if a query of the form $(\mathrm{fid}^*, i^*, \cdot)$ has already been answered, the challenger returns $\perp$). The challenger chooses a random bit $b \xleftarrow{\$} \{0, 1\}$ and gives $c^* \leftarrow \mathbf{AEncrypt}(\mathrm{sk}, \mathrm{fid}, i, m_b)$ to $\mathcal{A}$. Then it updates the set $S \leftarrow S \cup \{(\mathrm{fid}, m_b, i, c_i)\}$.

- **Queries II** This phase is carried out as the previous one, the only difference being that the decryption queries made w.r.t. fid^* and a function f where $f_{i^*} \neq 0$ are answered with $\perp$.

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

The advantage of $\mathcal{A}$ in the LH-IND-CCA game is defined as

$$\mathbf{Adv}_{\mathcal{H}}^{\text{LH-IND-CCA}}(\mathcal{A}) \overset{\text{def}}{=} \left| \Pr\left[\text{LH-IND-CCA}_{\mathcal{H},\mathcal{A}}(1^\lambda) = 1\right] - \frac{1}{2} \right|$$

We say that a LAEPuV scheme is secure against a LH-IND-CCA attack if $\mathbf{Adv}_{\mathcal{H}}^{\text{LH-IND-CCA}}(\mathcal{A}) = negl(\lambda)$ for any PPT adversary $\mathcal{A}$.

Definition 21 (Linearly Homomorphic Unforgeability under chosen ciphertext attack (LH-Uf-CCA)). A LAEPuV scheme is Linearly Homomorphic Unforgeable against chosen ciphertext attack if the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible in the security parameter λ:

- **Setup** The challenger runs $(\text{sk}, \text{vk}) \leftarrow \mathbf{AKeyGen}(1^\lambda, k)$. Then it initializes an empty set Q and gives vk to the adversary $\mathcal{A}$.

- **Queries** $\mathcal{A}$ can ask a polynomial number of encryption and decryption queries. First are of the form (fid, m_i, i) (where fid is a dataset identifier, $m_i \in \mathcal{M}$ is a message and $i \in \{1, \dots, k\}$ is an index). The challenger computes $c_i \leftarrow \mathbf{AEncrypt}(\text{sk}, \text{fid}, i, m)$, gives c_i to $\mathcal{A}$ and updates the set $Q \leftarrow Q \cup \{(\text{fid}, m_i, c_i, e_i)\}$. No two queries differing only on the $\mathbf{m}_i$ component can be asked by the adversary (if this happens, the answer to the second query is $\perp$). Decryption queries instead are of the form (fid, c_i, f) and $\mathcal{A}$ gets the corresponding output of $\mathbf{ADecrypt}(\text{sk}, \text{fid}, c, f)$ (It can be $\perp$ if c is a not valid ciphertext).

- **Forgery** $\mathcal{A}$ outputs (fid^*, c^*, f^*), where c^* is a ciphertext, fid^* a file identifier and f^* an admissible function.

Let $Q_{\text{fid}^*} = \{(\text{fid}^*, m_i, c_i, f_i)\}_{i=1,\dots,s} \subseteq Q$ be the set of entries in Q for which $\text{fid} = \text{fid}^*$.
The Adversary wins the game if $\mathbf{ADecrypt}(\text{vk}, \text{fid}^*, c^*, f^*) = m^* \neq \perp$ and one of the following conditions hold:

1 Q_{fid^*} is empty
2 f^* (interpreted as a vector) is in the span of $\{f_1, \dots, f_s\}$ but, for any $\alpha_1, \dots, \alpha_s$ 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 $\{f_1, \dots, f_s\}$

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

A.2 Linearly homomorphic signature scheme to sign elements in bilinear groups (LHSG)

Definition 22 (LHSG). A Linearly homomorphic signature scheme to sign elements in bilinear groups is a tuple of 4 PPT algorithms (**KeyGen**, **Sign**, **Verify**, **Eval**) such that:

- $\mathbf{KeyGen}(1^\lambda, n, k)$ 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 Σ, for some $n_d \in poly(\lambda)$.
- $\mathbf{Sign}(\text{sk}, m, \text{fid}, i)$ takes as input the secret key, an element $m \in \mathcal{M}$, a dataset identifier fid, an index $i \in \{1, \dots, k\}$ and outputs a signature σ.

- **Verify** (vk, σ, m, fid, f) takes as input the public 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** $(\mathrm{vk}, \mathrm{fid}, f, \{\sigma_i\}_{i=1\ldots k})$ takes as input the public key vk, a dataset identifier fid, an admissible function f in its vector form $(\alpha_1, \ldots, \alpha_k)$, a set of k signatures $\{\sigma_i\}_{i=1\ldots k}$ 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 don't explicitly account this to avoid heavy notation.

The correctness conditions of our scheme are the following:

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

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

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

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

Security Roughly speaking, a LHSG 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 23 (Random message attack security). An LHSG is unforgeable against a random message attack if for all n, k 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 $i \in \{1, \ldots, k\}$. 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 $(h, (\mathrm{fid}, m, \sigma, e_i))$ 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 $i, j \in \{1, \ldots, k\}$ $(i \neq j)$. 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 $\mathbf{Adv}^{LHSG-RMA}(\mathcal{A})$ of $\mathcal{A}$ as the probability that $\mathcal{A}$ wins the game.

Definition 24 (Known Random Message Security). . We extend the definition 23 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. 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 $m \overset{\$}{\leftarrow} \mathcal{M}$ 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 25 (Chosen message attack security). An LHSG is unforgeable against chosen message attack if for all n, k 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 (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 of 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 fid = 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 $\textbf{Adv}^{LHSG-CMA}(\mathcal{A})$ of $\mathcal{A}$ the probability that $\mathcal{A}$ wins the game.

A.3 Chameleon Hash Functions

Chameleon hash functions were originally introduced in [43]. Very briefly, a chameleon hash function is a special type of hash function that is collision resistant only without a secret trapdoor information. Each function has an additional argument normally chosen at random when using the hash function in the standard way and used by the owner of the trapdoor key to change the input of the function to any other string without affecting the output. The formal definition can be given as follows.

Definition 26. A chameleon hash family consists of a set $\mathcal{H}$ of functions and a triple of efficient PPT algorithms (**CHGen**, **CHEval**, **CHFindColl**) such that:

– Each function $h \in \mathcal{H}$ is defined as $h : \{0,1\}^* \times \Gamma \to \{0,1\}^l$ (where Γ is a sufficiently large set used as randomness space) and is identified by a string hk.
– $\textbf{CHGen}(1^k)$ is a randomized algorithm that takes as input a security parameter k and outputs a couple of strings (hk, ck) whose length is polinomially related to k. The string hk identifies a function $h_{\text{hk}} \in \mathcal{H}$ and is used as the public hashing key, while ck is the secret chameleon key (used for finding collisions).
– $\textbf{CHEval}(\text{hk}, m, \rho)$ efficiently computes $h_{\text{hk}}(m, \rho)$

– **CHFindColl**(ck, m_1, ρ_1, m_2) computes, using the secret key ck, a value ρ_2 such that $h_{\text{hk}}(m_1, \rho_1) = h_{\text{hk}}(m_2, \rho_2)$, where (hk, ck) is in the image of **CHGen**.

Moreover, the following additional properties are required:

Collision Resistance There exists no PPT adversary $\mathcal{A}$ that on input hk (such that (hk, ck) $\leftarrow$ **CHGen**(1^k)) can find m_1, m_2, ρ_1, ρ_2 satisfying $h_{\text{hk}}(m_1, \rho_1) = h_{\text{hk}}(m_2, \rho_2)$ with more than negligible probability (The probability is taken over the internal random choices of $\mathcal{A}$ and the couples returned by **CHGen**).

Uniform distribution For all $m_1, m_2 \in \mathcal{M}$ and for all pairs (hk, ck) output by **CHGen**, if ρ_1 is chosen uniformly at random, then the probability distribution induced over $\rho_2 \overset{\$}{\leftarrow}$ **CHFindColl**(ck, m_1, ρ_1, m_2) is again the uniform distribution.

A.4 *Σ*-Protocols

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 Σ = (**Σ-Setup**, **Σ-Com**, **Σ-Resp**, **Σ-Verify**) defined as follows:

Σ-Setup$(1^\lambda, \mathcal{R}) \to (x, w)$ It takes as input a security parameter λ and a relation $\mathcal{R}$. It returns a statement x and a witness w such that $(x, w) \in \mathcal{R}$.

Σ-Com$(x) \to (R, r)$ 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.

Σ-Resp$(x, w, r, c) \to s$ Is a deterministic 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.

Σ-Verify$(x, R, c, s) \to \{0,1\}$ Is the verification algorithm that on input the message R, a challenge $c \in$ **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 $(R, r) \leftarrow$ **Σ-Com**(x, r), any $c \in$ **ChSp** and $s \leftarrow$ **Σ-Resp**(x, w, r, c), it holds that **Σ-Verify**$(x, R, c, s) = 1$ with overwhelming probability.

Special Soundness There exists a PPT extractor algorithm **Σ-Ext** such that $\forall x \in L, \forall R, c, s, c', s'$ such that $(c, s) \neq (c', s')$, **Σ-Verify**$(x, R, c, s) = 1$ and **Σ-Verify**$(x, R, c', s') = 1$, outputs $w' \leftarrow$ **Σ-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 $\forall x \in L, \forall c \in$ **ChSp**, $S(x, c)$ generates a pair (R, s) such that **Σ-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.

A.5 Example of Vector Σ Protocols

Here we describe a 1-n vector Σ-Protocol satisfying the Strong Vector Special Soundness Property.

Definition 27 (Schnorr 1-n Σ-Protocol). Let $\mathbb{G}$ a group of prime order p and $\mathcal{R}$ the DL relation on $\mathbb{G}$, DL$= \{(x, w)|x = (p, g, h), \quad h = g^w\}$. 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 (Strong) Schnorr (Vector) 1-n Σ-Protocol consists of four PPT algorithm $\Sigma_n = (\mathbf{\Sigma_n}$**-Setup**, $\mathbf{\Sigma_n}$**-Com**, $\mathbf{\Sigma_n}$**-Resp**, $\mathbf{\Sigma_n}$**-Verify**) defined as follows:

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

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

$\mathbf{\Sigma_n}$**-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 $s \leftarrow r + \sum_{j=1}^n c_j w_j$.

$\mathbf{\Sigma_n}$**-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.

B Postponed Proofs

B.1 Proof of theorem 6

Proof. We reduce the security of the scheme to the one of the DCRA: we use an adversary $\mathcal{A}$ that wins the LH-IND-CCA game to build a distinguisher $\mathcal{D}$ against the DCRA with advantage $\mathbf{Adv}_\mathcal{D} > \mathbf{Adv}_\mathcal{A}$. The distinguisher $\mathcal{D}$ receives in input (y, N) and runs the simulation as follows:

Key generation phase The distinguisher chooses $g \in \mathbb{Z}_{N^2}^*$ as a random element of order N, runs $(\mathrm{sk}', \mathrm{vk}') \leftarrow$ **HKeyGen**$(1^\lambda, k, N)$ and gives (vk', g, N) to $\mathcal{A}$ (the function H is substituted with a random oracle).

Queries The adaptively chosen queries asked by $\mathcal{A}$ are handled as follows.

Random Oracle Queries $\mathcal{D}$ guesses in advance on which couple (fid^*, i^*) the adversary will ask its challenge (since $\mathcal{A}$ is polynomial, $\mathcal{D}$ will be right with non negligible probability). It chooses u^*, v^* at random and sets $R^* \leftarrow g^{u^*} {v^*}^N y^{-1} \mod N^2$ as the output of $H(\mathrm{fid}^*||i^*)$. For any other oracle query (fid, i) it chooses random u, v and sets $R = g^u v^N \mod N^2$ as the output. It stores (fid, i, u, v) in a table T (and, if the same query is asked more than once, the same answer computed from the table T is returned).

Encryption Queries On input a query (fid, i, m), $\mathcal{D}$ retrieves the associated u, v from the table T (if the query (fid, i) has not been already asked, $\mathcal{D}$ simulates it and populates the table T accordingly). Then it computes $C \leftarrow g^m \beta^N$ for a random β, $a \leftarrow u + m$, $b \leftarrow \beta v$, $\sigma \leftarrow$ **Sign**$(\mathrm{sk}', \mathrm{fid}, i, a)$ and returns $c = (C, a, b, \sigma)$ to $\mathcal{A}$

Decryption Queries On input a triple (fid, c, f), where $c = (C, a, b, \sigma)$ is a ciphertext, fid an identifier and f an admissible function, $\mathcal{D}$ verifies the signature on a and that the equation

$$g^a b^N = C \prod_{i=1}^{k} H(\text{fid}||i)^{f_i} \mod N^2$$

holds (the oracle queries appearing in the above equation with non-zero exponents must exist in T, otherwise $\mathcal{D}$ can just assign to these queries a value that does not satisfy the equation and return $\perp$ to $\mathcal{A}$). If this is not the case, it returns $\perp$ to $\mathcal{A}$. Otherwise, it retrieves the couples (u_i, v_i) associated with each query (fid, i) from the table T. Then it computes $m \leftarrow a - \sum_{i=1}^{k} f_i u_i$ and returns m to $\mathcal{A}$. It is easy to see that if c is a valid ciphertext, than the message m is a correct decryption.

Challenge query On input $(\text{fid}^*, i^*, m_0, m_1)$, if $\mathcal{D}$ guessed the right fid and index, it chooses a bit z, computes $C \leftarrow g^{m_z} \beta^N y$ for a random β, $a \leftarrow u + m_z$, $b \leftarrow \beta v$, $\sigma \leftarrow \textbf{Sign}(\text{sk}', \text{fid}, i, a)$ and returns $c^* = (C, a, b, \sigma)$. Otherwise, the simulation is aborted.

Queries II after the challenge phase another queries phase takes place and all queries are handled as before, the only exception being that decryption queries involving identifier fid^* and a function f whose i^*-th component is non-zero are answered with $\perp$.

Output phase When $\mathcal{A}$ outputs a bit z', $\mathcal{D}$ guesses that y is a residue if $z = z'$ and that y is not a residue if $z \neq z'$ or if $\mathcal{A}$ aborts at any time.

First of all, one can notice that all the answers to the queries made by $\mathcal{A}$ (apart from the challenge query) are distributed as in the real case. Moreover, this is also the case for the challenge query provided y is an N-th residue, while c^* contains no information about the bit z in the other case. This is because, if we write y as $g^{y_1} y_2^N$, from c^* an unconditionally powerful adversary could deduce 3 *dependent* equations in the 3 variables y_1, m_z, u^*, which makes their simultaneous solution undetermined.
Therefore, in the case where y is not a residue $\mathcal{A}$ is playing the proper security game and $\mathcal{D}$ wins as long as $\mathcal{A}$ is successful, which happens with probability $1/2 + \textbf{Adv}_A$, and in the case where y is not a residue $\mathcal{A}$ can only guess at random, which makes $\mathcal{D}$ successful with probability at most $1/2$. In sum, since we are assuming the DCRA problem to be hard, it must be that

$$negl(n) > \textbf{Adv}_D \geq |1/2 + \textbf{Adv}_A - 1/2| \geq \textbf{Adv}_A$$

which is our thesis.

B.2 Proof of theorem 7

Proof. The reduction is very simple because, since we are not breaking any assumption related to the modulus N, its factors can be known by the simulator $\mathcal{D}$. $\mathcal{D}$ receives in input a public key vk' for Σ, and prepares the public key for $\mathcal{A}$ as in the real case, with the only difference that it uses vk' instead of running the key generation algorithm of Σ. The encryption and decryption queries are handled as in the real case, with the only exception that the σ component of each ciphertext is requested by $\mathcal{D}$ to its signing oracle instead of being computed by $\mathcal{D}$ itself. When $\mathcal{A}$ outputs a forged ciphertext $c^* = (C^*, a^*, b^*, \sigma^*)$ w.r.t. an identifier fid and a funcion f^*, its σ^* component must be a valid forgery for Σ. If this was not the case, it would imply that, called $c_i = (C_i, a_i, b_i, \sigma_i)$ the signatures returned by the simulator to $\mathcal{A}$ in response to its query (fid, i, m_i),

$$\left(\frac{b^*}{\prod_{i=1}^k b_i^{f_i^*}}\right)^N = \frac{C^*}{\prod_{i=1}^k C_i^{f_i^*}}.$$

Therefore this would not be a forgery as C^* would be in the same residuosity class (and hence contain the same plaintext) of the ciphertext obtained by computing the function f^* on the honestly generated ciphertexts $c_1, \ldots, c_n$.

B.3 Proof of theorem 8

Proof. Proving correctness is straightforward, given the bilinear property of the pairing function. For what concerns security, we split the proof in 3 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 (or the security of the underlying signature scheme $\mathcal{S}$). In particular, let $m^*, \pi^* = (\mathrm{fid}^*, \sigma^*, \tau^*, V^*, M^*), f^*$ 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 $Q_{fid^*} = \{(h_\eta, (fid^*, m_\eta, \pi_\eta, f_\eta))\}_{\eta=1,\ldots,\nu} \subseteq Q$ be the set of signatures seen by $\mathcal{A}$ for which the file identifier is fid^*, where $\pi_\eta = (\mathrm{fid}_\eta, \sigma_\eta, \tau_\eta, V_\eta, M_\eta)$. Then (at least) one of the following conditions hold:

Case 1: Q_{fid^*} is empty, or $(\mathrm{fid}^*, \sigma^*) \neq (\mathrm{fid}_\eta, \sigma_\eta)$ for all $\eta = 1, \ldots, \nu$ (note that, by construction, all the signatures in Q_{fid^*} share the same σ^* component).
Case 2: $(\mathrm{fid}^*, \sigma^*) = (\mathrm{fid}_\eta, \sigma_\eta)$ 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 3: $(\mathrm{fid}^*, \sigma^*) = (\mathrm{fid}_\eta, \sigma_\eta)$ 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 1/3.

$\boxed{\textbf{Case 1.}}$ In this case it is possible to reduce the security to the one of the underlying signature scheme $\mathcal{S}$. The simulator is quite simple: it uses its signing oracle for $\mathcal{S}$ to compute the component of each signature authenticating the fid (i.e. τ), and can easily compute the remaining parts of each signature by creating the rest of the secret and public key as in the real case. When $\mathcal{A}$ outputs a forgery, by definition of this case the simulator can output $((\mathrm{fid}^*, \sigma^*), \tau^*)$ as a forgery for $\mathcal{S}$.
$\boxed{\textbf{Case 2.}}$ First of all one can notice that, because the forgery must satisfy (in particular) the third and fourth verification equations, it must be that

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

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 $\overline{m}, \overline{\pi}$ such couple). So it must be that:

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

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 $V^* = \overline{V}$).
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[12].

Key Generation $\mathcal{B}$ initializes an empty table T (as described in the description of the security game), runs $(\mathrm{sk}_1, \mathrm{vk}_1) \leftarrow \mathbf{KeyGen}(1^\lambda)$ and sets $W \leftarrow g^w$ (so w is implicitly part of the secret key). 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 $\delta_1, \ldots, \delta_k \xleftarrow{\$} \mathbb{Z}_p$. Finally it gives $(\mathrm{vk}_1, g, W, h_1, \ldots, h_k)$ to $\mathcal{A}$.

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

if fid$'$ $\notin T$, it sets

$$\sigma = g^r,$$

$$\tau \leftarrow \mathbf{Sign}(\mathrm{sk}, \mathrm{fid}', \sigma),$$

and stores this data in memory.

if fid$'$ $\in T$, it retrieves the corresponding (σ, τ) from memory.

Then it sets $V \leftarrow \sigma^{\delta_i}$ and $M \leftarrow W^{b_i} = m_i^w$. By inspection, one can check that τ, M and V are correctly distributed as in the real case.

To answer the other queries with dataset identifier fid $\neq$ fid$'$ w.r.t. index i it does the following.

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

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

$$\tau \leftarrow \mathbf{Sign}(\mathrm{sk}, \mathrm{fid}, \sigma),$$

and stores this data in memory.

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

Then it sets $m \leftarrow g^{b_i}$ for a random $b_i \xleftarrow{\$} \mathbb{Z}_p$, $V \leftarrow (h_i M_i)^r$, $M \leftarrow W^{b_i}$.
In both cases, 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.

[12] 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.

Forgery Assume that the adversary $\mathcal{A}$ produced a forgery π^* for the function $f^* = (\alpha_1^*, \ldots, \alpha_k^*)$ w.r.t fid*. If fid$^* \neq$ fid$'$, then it aborts. Otherwise it proceeds as follows.
Considering the signature $\overline{\pi} = (\overline{\text{fid}}, \overline{\sigma}, \overline{\tau}, \overline{V}, \overline{M})$ 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 $\left(\frac{m^*}{\prod_{i=1}^{k} m_i^{\alpha_i^*}}, \frac{V^*}{\overline{V}}\right)$; in-fact the elements of the couple are not trivial by the definition of this subcase.

$\boxed{\textbf{Case 3.}}$ In this case we can use exactly the same simulation of case 2, 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.

B.4 Proof of theorem 18

Here we present a proof sketch in the simplified case where each signing query is immediately followed by the corresponding reveal query. In this setting we can just assume that the evaluation queries are computed by the adversary itself and that for each encryption query it gets the corresponding message/signature couple. A more detailed proof for the general case is deferred to a full version of the paper.
Proof. 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 message attacks, and $\mathcal{H}$ is collision resistant, it must be that $\prod_{i=1}^{k} R_i^{\alpha_i} = R^*$ and fid* = 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 $\mathcal{R}^n$. We will assume first that the underlying Σ-Protocol has the vector special soundness, and explain later how to modify the proof in the case where the strong vector special soundness holds. $\mathcal{B}$ 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 $(\text{vk}_1, \text{sk}_1) \leftarrow \textbf{KeyGen}(1^\lambda, k, n)$ and gives to $\mathcal{A}$ vk $= (\text{vk}_1, \boldsymbol{x})$. 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 $\overline{\sigma_i} \leftarrow \textbf{Sign}(\text{sk}_1, \text{fid}, R_i, i)$ 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 $\overline{\mathbf{s}} = \mathbf{Combine}\left(\mathbf{x}, \alpha^*, \{(R_i, \mathbf{m}_i, s_i)\}_{i=1,\dots,k}\right)$.
$\mathcal{B}$ uses the extractor $(x, w) \leftarrow$**Σ-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 $\boldsymbol{x}$.

In the case where the strong vector special soundness holds, since $\mathbf{m}^* \neq \mathbf{m}_1^{\alpha_1^*} \circ_2 \cdots \circ_2 \mathbf{m}_k^{\alpha_k^*}$, there exists and index j such that the j-th components of the two sides of the equation are different. $\mathcal{B}$ can guess the index j in advance and prepare the public key by generating $n-1$ couples $(x_i, w_i) \in L$ for $i \in \{1, \dots, n\} \setminus j$, setting $x_j \leftarrow x$ (where x is the statement $\mathcal{B}$ received in input) and $\boldsymbol{x} \leftarrow (x_1, \dots, x_n)$. The rest of the public key and the other phases of the simulation are carried as in the previous case. When $\mathcal{A}$ provides a forgery, assuming $\mathcal{B}$ guessed the correct index (this will happen with non negligible probability, otherwise $\mathcal{B}$ aborts), $\mathcal{B}$ can use the extractor for the strong vector special soundness to get the missing witness $w_j \leftarrow$**Σ-Ext** $(\mathbf{x}, R, \mathbf{m}^*, s^*, \overline{\mathbf{m}}, \overline{\mathbf{s}}, \{w_i\}_{i \neq j})$.

C From random message security to chosen message security

In this section we present a general transform to construct an LHSG 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 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.

Remark 28. We stress that while we present our theorems in the context of linearly homomorphic signatures (LHSG), if they are applied to linearly homomorphic structure preserving signatures, the structure preserving property is preserved.

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

- **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 $(\mathrm{sk}_1, \mathrm{vk}_1) \leftarrow \mathbf{HKeyGen}(1^\lambda, n, k)$ and $(\mathrm{sk}_2, \mathrm{vk}_2) \leftarrow \mathbf{HKeyGen}(1^\lambda, n, k)$.
  It outputs $\mathrm{sk} = (\mathrm{sk}_1, \mathrm{sk}_2)$ as the secret signing key and $\mathrm{vk} = (\mathrm{vk}_1, \mathrm{vk}_2)$ as the public verification key. The message space $\mathcal{M}$ is the same of $\mathcal{S}$.
- **TSign**$(\mathrm{sk}, \mathbf{m}, \mathrm{fid}, i)$ It chooses random $\mathbf{m}_1 = (m_{1,1}, \dots, m_{1,n}) \xleftarrow{\$} \mathcal{M}$ and computes $\mathbf{m}_2 \leftarrow \left(\frac{m_1}{m_{1,1}}, \dots, \frac{m_n}{m_{1,n}}\right)$ (where $\mathbf{m} = (m_1, \dots, m_n)$).
  Then it computes $\sigma_1 \leftarrow \mathbf{HSign}(\mathrm{sk}_1, \mathbf{m}_1, i, \mathrm{fid})$, $\sigma_2 \leftarrow \mathbf{HSign}(\mathrm{sk}_2, \mathbf{m}_2, i, \mathrm{fid})$ and outputs $\sigma = (\mathrm{fid}, \mathbf{m}_1, \sigma_1, \sigma_2)$.
- **TVerify**$(\mathrm{vk}, \sigma, \mathbf{m}, \mathrm{fid}, f)$ parses σ as $(\mathrm{fid}, \mathbf{m}_1, \sigma_1, \sigma_2)$, computes $\mathbf{m}_2 \leftarrow \left(\frac{m_1}{m_{1,1}}, \dots, \frac{m_n}{m_{1,n}}\right)$ and checks that the following equations hold:

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

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

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

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

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

and returns

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

Theorem 29. *Suppose $\mathcal{S}$ is a LHSG 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 LHSG secure against a chosen message attack.*

Proof. We prove the theorem by reducing the security of $\mathcal{T}$ to the one of $\mathcal{S}$, and showing how to build a simulator $\mathcal{B}$ that uses an adversary $\mathcal{A}$ against $\mathcal{T}$ to break the RMA security of $\mathcal{S}$.
First of all one can notice that, by construction, if $(\mathbf{m}^*, \pi^* = (\mathrm{fid}^*, \mathbf{m}_1^*, \pi_1^*, \pi_2^*), f^*)$ is a forgery for $\mathcal{T}$ then at least one between $(\mathbf{m}_1^*, \pi_1^*, f^*)$ (case 1) and $(\mathbf{m}^*/\mathbf{m}_1^*, \pi_2^*, f^*)$ (case 2) is a forgery for the corresponding instance of $\mathcal{S}$.
The simulator $\mathcal{B}$ works as follows:
It receives a public key vk' for an instance of $\mathcal{S}$ 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[13] $\mathbf{m}_2$, computes $\pi_2 \leftarrow \mathbf{Sign}(\mathrm{sk}_2, \mathbf{m}_2, \mathrm{fid}_2, i)$ and returns h to $\mathcal{A}$. The handle h, the messages $\mathbf{m}$ and $\mathbf{m}_2$, the signature π_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 $\pi^{(h)} \leftarrow$ **Eval**$(vk, \mathrm{fid}, f, \{\pi^{(h_i)}\}_{i=1,\ldots,k})$, 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 $\mathbf{m}^{(h_i)}, \mathbf{m}_2^{(h_i)}, \pi^{(h_i)}$ 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.

[13] 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.

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 $(\text{fid}, \mathbf{m}_1, \pi_1, f)$. Since the adversary expects to receive a valid signature for a certain message $\mathbf{m}$ (that the simulator knows since it is stored in its own table T together with the handle h, the message $\mathbf{m}_2$ and signature π_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[14] (in the unknowns $\mathbf{m}_2^{(1)}, \ldots, \mathbf{m}_2^{(k)}$) and computing new signatures[15] 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 $\mathbf{m}^*, \pi^* = (\text{fid}^*, \mathbf{m}_1^*, \pi_1^*, \pi_2^*), f^*$ as a valid forgery and that $\mathcal{B}$'s guess was correct. Then $\mathcal{B}$ can return $(\text{fid}^*, \mathbf{m}_1^*, \pi_1^*, f^*)$ as a valid forgery against $\mathcal{S}$ 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 30. *If $\mathcal{S}$ is a LHSG secure against known random message attack the scheme $\mathcal{T}$ described above is a LHSG secure against a chosen message attack.*

D A simple variant of Catalano *et al.* signature.

In [23] Catalano et al. present a Network Coding signature scheme based on the strong RSA assumption. Here we describe a simple variant of the scheme that allows a user to sign messages in a bigger space.

KeyGen$(1^k, N)$ Let N product of two arbitrary safe primes each one of length $k'/2$. The **KeyGen** algorithm chooses two random (safe) primes $\hat{p}, \hat{q}$ of length $k/2$ each such that $gcd(N, \phi(\hat{N})) = 1$[16] where $\hat{N} = \hat{p}\hat{q}$ and proceeds by choosing $g, g_1, h_1, \ldots, h_m$ at random (in $\mathbb{Z}_{\hat{N}}^*$). Moreover it chooses some (efficiently computable) injective function $H : \{0,1\}^* \to \{0,1\}^\ell$ that maps to primes of length $\ell < k'/2$. The public key is set as $(N, H, \hat{N}, g, g_1 h_1, \ldots, h_m)$, while the secret key is $(\hat{p}, \hat{q})$.

[14] 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

[15] 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.

[16] Supposing $\hat{p}$ and $\hat{q}$ are two safe prime of equal lenght $k/2$ we can write $\hat{p} = 2\hat{p}' + 1$ and $\hat{q} = 2\hat{q}' + 1$ so $\phi(\hat{N}) = 4\hat{p}'\hat{q}'$ where $|\hat{p}'| = |\hat{q}'| \approx \frac{k}{2} - 1$. So, fixed N, it's enough choose $\hat{p}, \hat{q}$ of length at least $k'/2 + 2$ to verify $gcd(N, \phi(\hat{N})) = 1$

Sign(sk, fid, M, i) Let e_i the i-th vector of the canonical basis on $\mathbb{Z}^m$. The signing algorithm proceeds as follows. First it maps the random identifier fid to prime: $e \leftarrow H(\text{fid})$. It chooses random elements $s \in \mathbb{Z}_{eN}$ and uses its knowledge of $\hat{p}$ and $\hat{q}$ to solve the following equation

$$x^{eN} = g^s h_i g_1^M \bmod \hat{N}$$

We denote with $\sigma = (e, s, \text{fid}, x)$ the signature for the message M w.r.t. the function e_i and the identifier fid.

Verify(vk, $\sigma, M, \mathbf{f}$) To verify a signature σ for a message M w.r.t. an identifier fid and a function $\mathbf{f}$, the verification algorithm proceeds as follows

- Compute $e \leftarrow H(\text{fid})$
- Check that M, s are in $\mathbb{Z}_{eN}$.
- Define $\mathbf{f}' = \frac{\mathbf{f} - \mathbf{f} \bmod eN}{eN}$ and $\hat{x} = \frac{x}{\prod_{j=1}^{m} h_j^{f'_j}}$
- Finally check that the equation

  $$\hat{x}^{eN} = g^s \prod_{j=1}^{m} h_j^{f_j} g_1^M$$

  is satisfied
- If all the checks above are satisfied, output 1, otherwise 0.

Combine(vk, fid, $\hat{\mathbf{f}}, \sigma_1, \ldots, \sigma_m$) To combine signatures σ_i sharing the same fid it works as follows. Let $\hat{\mathbf{f}} = (\alpha_1, \ldots, \alpha_m)$. It sets $s = \sum_{i=1}^{m} \alpha_i s_i \bmod eN$, $s' = (\sum_{i=1}^{m} \alpha_i s_i - s)/(eN)$. It outputs the signature $\sigma = (e, s, \text{fid}, x)$ which is obtained by computing

$$x = \frac{\prod_{i=1}^{m} x_i^{\alpha_i}}{g^{s'}} \bmod \hat{N}.$$

Security theorem follows very easily from the proof of the original signature scheme in [23].

Theorem 31. *Under the Strong-RSA assumption, the scheme described above is an unforgeable signature scheme under chosen messages attack according to [15] definition.*