Efficient Network Coding Signatures in the Standard Model

Dario Catalano[1], Dario Fiore[2⋆] and Bogdan Warinschi[3]

[1] Dipartimento di Matematica e Informatica,
Università di Catania, Italy.
catalano@dmi.unict.it
[2] Department of Computer Science, New York University, USA
fiore@cs.nyu.edu
[3] Dept. Computer Science, University of Bristol, UK
bogdan@cs.bris.ac.uk

Abstract. Network Coding is a routing technique where each node may actively modify the received packets before transmitting them. While this departure from passive networks improves throughput and resilience to packet loss it renders transmission susceptible to *pollution attacks* where nodes can misbehave and change in a malicious way the messages transmitted. Nodes cannot use standard signature schemes to authenticate the modified packets: this would require knowledge of the original sender's signing key. Network coding signature schemes offer a cryptographic solution to this problem. Very roughly, such signatures allow signing vector spaces (or rather bases of such spaces). Furthermore, these signatures are homomorphic: given signatures on a set of vectors it is possible to create signatures for any linear combination of these vectors. Designing such schemes is a difficult task, and the few existent constructions either rely on random oracles or are rather inefficient. In this paper we introduce two new network coding signature schemes. Both of our schemes are provably secure in the standard model, rely on standard assumptions, *and* are in the same efficiency class with previous solutions based on random oracles.

1 Introduction

Network Coding [1, 22] is an elegant and novel routing approach that is alternative to traditional routing where each node simply stores and forwards the incoming packets. The main difference is that in Network Coding intermediate nodes can modify data packets in transit, still allowing the final recipients to obtain the original information.

More specifically, we consider a network setting where a *source* node wants to transmit a piece of information (a file) to a set of *target* nodes. The source node splits the file into m network packets and sends them to its neighboring nodes. An intermediate node who receives a set of packets from its incoming links, modifies them and sends the resulting packets into the network through its outgoing edges. In *Linear Network Coding* packets are seen as vectors in a linear space over some field and the modifications by the intermediate nodes are linear combinations of these vectors. Such linear combinations can be performed by using ad-hoc coefficients (e.g., fixed by the application or defined by a central authority), or random coefficients chosen by the intermediate nodes in a suitable domain. The latter case is referred as *Random (Linear) Network Coding*. In addition to offering a more decentralized approach, random network coding has been shown to perform almost as well as network coding with ad-hoc coefficients [11, 15, 17]. One important aspect of linear network coding is that it enables target nodes to recover the original information with high probability if they receive sufficiently many correct packets. Interestingly, the target nodes can do so without

⋆ Work done while at École Normale Supérieure.

knowledge of the coefficients chosen by the intermediate nodes. We give a more detailed description of these techniques in Section 2.2.

The original motivation for network coding was to increase throughput in decentralized networks and indeed, the technique performs well in wireless/ad-hoc network topologies where a centralized control may not be available. For example, it has been suggested as a good means to improve file sharing in peer-to-peer networks [21], and digital content distribution over the Internet [14].
The main issue of (random) linear network coding is its susceptibility to *pollution attacks* in which malicious nodes may inject into the network invalid packets to prevent the target nodes from reconstructing the original information. Notice that such invalid packets could also be generated by network failures and not necessarily by malicious nodes. If we consider the specific setting of linear network coding, then an invalid packet is a vector outside the space, i.e., not in the span of the initial m vectors sent by the source node. The main problem here is that intermediate nodes can later use the invalid incoming vectors in the linear combinations, thus generating more invalid packets. This means that errors may dramatically propagate through the network, and adversaries might easily mount a Denial of Service attack to prevent the file from being reconstructed by only injecting *a few* invalid packets.
To solve this issue, two main approaches have been proposed. One is information-theoretic and uses error-correction techniques [16, 17, 19]. Unfortunately, this introduces redundant information that badly affects the communication efficiency. The other approach (the one considered in our work) relies on computational assumptions and uses cryptographic techniques. Here, the main idea to mitigate pollution attacks is to provide a way to authenticate valid vectors. However, standard authentication techniques, such as MACs or digital signatures, do not trivially solve the problem as we want to grant the intermediate nodes some malleability on the received vectors.
The main tool that has been proposed to achieve this goal are *network coding signature* schemes [7]. In a few words, a network coding signature allows to sign a linear subspace $\mathcal{W} \subset \mathbb{F}^N$ in such a way that a signature σ on $\mathcal{W}$ is verified only by those vectors $w \in \mathcal{W}$.
These schemes can be constructed either from *homomorphic hash functions*, or from *homomorphic signatures*. Very briefly, a homomorphic hash function H satisfies the property that for any vectors a, b and scalar coefficients α and β, it holds that $H(\alpha a + \beta b) = H(a)^\alpha H(b)^\beta$. Constructions based on homomorphic hashing [21, 15, 7, 13] are less recent and their security can be based on well-established assumptions in the standard model, such as solving discrete log or factoring. Unfortunately, as main drawback, the public key and the authentication information that has to be sent along with the packets are linear in the size m of the vector space and thus defeats the purpose of increasing the throughput. Furthermore, the sender has to know the entire file before sending the first packet (which is undesirable for example in the ubiquitous streaming applications).
On the other hand, solutions based on homomorphic signatures [7, 13, 3, 10] are more communication-efficient, even though they are computationally a bit more expensive than those built from homomorphic hashing. In a nutshell, a homomorphic signature is a special type of signature scheme that enjoys a linear homomorphic property: for any vectors a, b and scalar coefficients α and β, it holds that $\mathsf{Sign}(\alpha a + \beta b) = \mathsf{Sign}(a)^\alpha \mathsf{Sign}(b)^\beta$. More formally, this means that the scheme is equipped with a $\mathsf{Combine}$ algorithm that given μ signatures $\sigma_1, \ldots, \sigma_\mu$ on vectors $w_1, \ldots, w_\mu$ respectively, and scalar coefficients $\alpha_1, \ldots, \alpha_\mu$, it can compute a signature σ which is valid with respect to the vector $w = \sum_{i=1}^{\mu} \alpha_i \cdot w_i$. Importantly, the combination operation does not require the secret key.
The security notion for this primitive requires that an adversary who receives signatures on a set of vectors $w_1, \ldots, w_m$ should be able to generate only signatures on vectors that lie in the linear

span of $(w_1, \ldots, w_m)$. It should be clear at this point how this primitive can be used to secure the network coding-based application (see Section 2.4 for a detailed description) and, more generally, enable authenticated computation of linear functions of signed data [2].

1.1 Related Work

Since our work focuses on homomorphic network coding signatures, in this section we describe the most significant works in this topic. The notion of homomorphic signature was first introduced by Johnson, Molnar, Song and Wagner in a more general setting [20] and only recently adapted to the particular application for network coding by Boneh, Freeman, Katz and Waters [7]. In their work, Boneh *et al.* propose an efficient construction over bilinear groups and prove its security from the CDH assumption in the random oracle model. One year later, Gennaro, Katz, Krawkzyk and Rabin [13] proposed another implementation of homomorphic network coding signatures based on RSA in the random oracle model. Moreover, as an additional contribution, they showed that even if the homomorphic signature works over a large finite field (or over the integers), it is possible to use small coefficients in the linear combinations, and this significantly improves the efficiency at the intermediate nodes in the network coding application. In [9] Boneh and Freeman give the construction of a homomorphic network coding signature based on lattices. As a new property, their scheme allows to authenticate vectors defined over binary fields, and is based on the problem of finding short vectors in integer lattices. The security of this construction relies on the random oracle heuristic. In addition, the same paper shows a scheme in the standard model, but this scheme is only k-time secure (a signing key can be used to issue only k signatures, where k is fixed in advance). In a subsequent work [8], Boneh and Freeman proposed the notion of homomorphic signatures for polynomial functions. While all previous works considered schemes whose homomorphic property allows to compute only linear functions on the signed data, the scheme in [8] is capable to evaluate multivariate polynomials. Their construction uses ideal lattices and its security is proven in the random oracle model.

The problems associated to the use of the random oracles are well-known and significant research effort is invested in devising implementations that do not rely on this heuristic. For network coding such constructions proved elusive – and we are only aware of two such proposals [3, 10][4].

In [3] Attrapadung and Libert give an implementation over bilinear groups of composite order, using the dual system techniques of Waters [23] to carry on the security proof. Unfortunately the scheme relies on the setting of composite order groups and is thus highly inefficient. Furthermore, even if the scheme were to be converted to gropus of prime order (as suggested, but not fully described in [3]), the efficiency gap between the resulting construction and those in the random oracle solutions is still significant.

The most recent proposal is by Catalano, Fiore and Warinschi who propose a homomorphic network coding signature as an application of the notion of Adaptive Pseudo-Free groups [10]. In particular, the concrete implementation is secure in the standard model under the Strong RSA assumption. While from the point of view of computation the efficiency of this scheme is not far from that of the random oracle construction of Gennaro *et al.* which also works in the RSA group, the signature's size in [10] is much worse than that in [13], as it is very affected by the large random exponent s (that is 1346 bits long if one considers 80 bits of security).

[4] We mention that the random oracle based solution given in [7] might be turned into a scheme secure in the standard model if one is willing to give up the homomorphic property. This makes the resulting solution much less interesting in practice as the signer would need to sign all the vectors in the given subspace at once.

1.2 Our contribution

In this work we design two new homomorphic network coding signatures with security proofs in the standard model. Our realizations outperform in efficiency the two currently known constructions in the standard model [3, 10] and achieve computational and communication efficiency comparable to those of the random oracle implementations [7, 13].

In a bit more details, our first scheme works over asymmetric bilinear groups of prime order p, and it is proven secure under the q-Strong Diffie Hellman assumption (q-SDH for short) introduced by Boneh and Boyen [6]. The construction adapts ideas from the signature by Hofheinz and Kiltz [18] which in turn is based on the concept of Programmable Hash Functions. There a signature is a random $r \in \mathbb{Z}_p$ and a group element X that is a solution of $X^{z+r} = H(M)$, where z is the secret key, and H is the programmable hash function. To obtain a solution for signing vector spaces along the same lines, we developed some non-trivial extensions which roughly speaking deal with the fact that in our case the same random exponent has to be reused for several signatures. In our construction, a signature on a vector $w = (u, v) \in \mathbb{F}_p^{m+n}$ consists of a random element $s \in \mathbb{Z}_p$ and the solution X to the following equation:

$$X^{z+\mathsf{fid}} = h^s h_1^{u_1} \cdots h_m^{u_m} g_1^{v_1} \cdots g_n^{v_n}$$

where $\mathsf{fid} \in \mathbb{Z}_p$ represents the random file identifier and z is the secret key. We can therefore achieve rather short signatures: one group element plus an element of $\mathbb{Z}_p$, that is, about 512 bits for 128 bits of security.

Our second realization works over $\mathbb{Z}_N^*$ where N is the product of two safe primes pq. The scheme can be seen as an optimization of the construction by Catalano-Fiore-Warinschi where the random exponent s now can be taken as small as $2k$ bits (where k denotes the desired bit security). The signature on a vector $w = (u, v) \in \mathbb{F}^{m+n}$ is a random integer $s \in \mathbb{Z}_e$ and the solution x to the equation

$$x^e = g^s h_1^{u_1} \cdots h_m^{u_m} g_1^{v_1} \cdots g_n^{v_n} \bmod N$$

where e is a random prime representing the file identifier, and $g, h_1, \ldots, h_m, g_1, \ldots, g_n \in \mathbb{Z}_N^*$ are in the public key. As an additional improvement, we show how to do linear combinations $\pmod e$, allowing for the signature scheme to be used in networks with paths of any lengths. This was not the case in [10] and [13] where the parameters have to be set according to a bound L on the maximum length of a path between the source and the target nodes in the network.

A more detailed efficiency analysis of our schemes as well as comparisons with previous solutions, are given in Section 5.

Concurrent work In a concurrent and independent work Freeman has proposed a semi-generic transformation for building linearly-homomorphic signatures from standard signature schemes [12]. This transformation yields new linearly homomorphic signature schemes that are secure in the standard model under a new security notion (introduced in [12]) which is slightly stronger than the one considered in our work. Our schemes (that are different from the ones obtained in [12]) enjoy better efficiency, and it is of future interest to check whether they can satisfy the stronger notion of security.

2 Background and Definitions

In what follows we will denote with $k \in \mathbb{N}$ a security parameter. We say that a function $\epsilon : \mathbb{N} \to \mathbb{R}^+$ is negligible if and only if for every positive polynomial $p(k)$ there exists a $k_0 \in \mathbb{N}$ such that for all $k > k_0$: $\epsilon(k) < 1/p(k)$. If S is a set, we denote with $x \xleftarrow{\$} S$ the process of selecting x uniformly at random in S. Let $\mathcal{A}$ be a probabilistic algorithm. We denote with $x \xleftarrow{\$} \mathcal{A}(\cdot)$ the process of running $\mathcal{A}$ on some appropriate input and assigning its output to x.

2.1 Computational Assumptions

An integer N is called *RSA modulus* if it is the product of two distinct prime numbers pq. The Strong RSA Assumption was introduced by Baric and Pfitzmann in [4]. Informally, the assumption states that given a public RSA modulus N, and a random value $z \in \mathbb{Z}_N$, any PPT adversary cannot compute an e-th root of z for an $e \neq 1$ of its choice.

Definition 1 (Strong RSA Assumption). *Let N be a random RSA modulus of length k where $k \in \mathbb{N}$ is the security parameter, and z be a random element in $\mathbb{Z}_N$. Then we say that the Strong RSA assumption holds if for any PPT adversary $\mathcal{A}$ the probability*

$$Pr[(y, e) \leftarrow \mathcal{A}(N, z) : y^e = z \bmod N \wedge e \neq 1]$$

is negligible in k.

Let $\mathbb{G}, \mathbb{G}'$ and $\mathbb{G}_T$ be bilinear groups of prime order p such that $e : \mathbb{G} \times \mathbb{G}' \to \mathbb{G}_T$ is a bilinear map. The q-Strong Diffie-Hellman Assumption (q-SDH for short) was introduced by Boneh and Boyen in [5] and it is defined as follows.

Definition 2 (q-SDH Assumption). *Let $k \in \mathbb{N}$ be the security parameter, $p > 2^k$ be a prime, and $\mathbb{G}, \mathbb{G}', \mathbb{G}_T$ be bilinear groups of the same order p such that g and g' are the generators of $\mathbb{G}$ and $\mathbb{G}'$ respectively. Then we say that the q-SDH Assumption holds in $\mathbb{G}, \mathbb{G}', \mathbb{G}_T$ if for any PPT algorithm $\mathcal{A}$ and any $q = poly(k)$, the following probability (taken over the random choice of x and the random coins of $\mathcal{A}$) is negligible in k*

$$\Pr[\mathcal{A}(g, g^x, g^{x^2}, \cdots, g^{x^q}, g', (g')^x) = (c, g^{1/(x+c)})]$$

2.2 Background on Linear Network Coding

In linear network coding [1, 22] a file to be transmitted is viewed as a set of n-dimensional vectors $(v^{(1)}, \ldots, v^{(m)})$ defined over the integers or over some finite field. To transmit a file $\mathcal{V} = (v^{(1)}, \ldots, v^{(m)})$ the source node creates m augmented vectors $(w^{(1)}, \ldots, w^{(m)})$ where each $w^{(i)}$ is obtained by prepending to $v^{(i)}$ a vector $u^{(i)}$ of length m, i.e., $w^{(i)} = (u^{(i)}, v^{(i)})$. Precisely, $(u^{(1)}, \ldots, u^{(m)})$ represents the canonical basis of $\mathbb{Z}^m$, that is $u^{(i)}$ is the i-th unitary vector, with 1 in position i and 0 elsewhere. This way, the vectors $w^{(1)}, \ldots, w^{(m)}$ form a basis of a subspace $\mathcal{W} \subset \mathbb{F}^{m+n}$. Vectors $w^{(i)}$ of the above form are called *properly augmented vectors* while $(w^{(1)}, \ldots, w^{(m)})$ is a *properly augmented basis.*

In this setting, the *source* node sends these vectors as packets in the network. Whenever a node in the network receives $(w^{(1)}, \ldots, w^{(\mu)})$ on its μ incoming edges, it computes a linear combination $\hat{w}$ of the received vectors and transmits $\hat{w}$ in the network through its outgoing edges. The coefficients

used in the linear combination can be fixed by the application, established by a central authority, or they can be randomly chosen by each node. The latter is the case considered in our work and it is called "random network coding". As shown in [11, 15, 17], random network coding performs almost as well as linear network coding with ad-hoc coefficients. To recover the original file a node must receive m (valid) vectors $\hat{w}^{(1)}, \ldots, \hat{w}^{(m)}$ of the form described before, i.e., $\hat{w}^{(i)} = (\hat{u}^{(i)}, \hat{v}^{(i)})$. In particular, in order for the file to be reconstructed, the vectors $(\hat{u}^{(1)}, \ldots, \hat{u}^{(m)})$ need to be linearly independent. Let denote with $\hat{U}$ the matrix whose rows are the vectors $(\hat{u}^{(1)}, \ldots, \hat{u}^{(m)})$ and with $\hat{\mathcal{V}}$ the matrix whose rows are the vectors $(\hat{v}^{(1)}, \ldots, \hat{v}^{(m)})$. Then, the original file can be retrieved by computing

$$\mathcal{V} = \hat{U}^{-1} \cdot \hat{\mathcal{V}}.$$

Although the above described approach solves the problem of recovering the information in network coding, as we mentioned in the introduction, the main issue in this approach is that it is susceptible to *pollution attacks* where malicious nodes may inject invalid packets in the network so that the reconstruction of the original file becomes impossible. This is particularly sensitive also because a single error introduced by a (malicious) node can be propagated by honest nodes.

Before describing solutions, we observe how two trivial approaches do not solve the problem. First, the source node cannot simply sign the transmitted packets as the receivers are likely to get modified versions of them (by the effect of the linear combinations). Second, the source could sign the entire file. This would prevent the receivers to accept incorrect files, but it does not provide an efficient way for the receivers to recover the correct file as malicious nodes can still inject invalid packets to mount a DoS attack.

To mitigate the effect of pollution attacks two main approaches have been proposed. They can be divided into two categories: *information-theoretic* and *computational*.

Information theoretic approaches [16, 17, 19] use error-correction techniques to introduce redundancy in the transmitted vectors so that it is possible to reconstruct the original file as long as the number of compromised vectors is not too big. These methods have the advantage of not relying on computational assumptions, but, unfortunately, they introduce a significant overhead in the communication.

On the other hand, approaches based on computational assumptions use cryptographic techniques to provide a way for honest nodes to verify that the received packets are correct. The main tool to achieve this goal are *network coding signature schemes*. Roughly speaking, the basic requirement of such schemes is that they allow to efficiently check if a given vector is valid, i.e., it has been generated as linear combination of initial (valid) vectors $w^{(1)}, \ldots, w^{(m)}$. Two classes of network coding signatures are known: those based on homomorphic hashing [21, 15, 7], and those using homomorphic signatures [20, 7, 13, 10].

In our work, we focus on the second class of schemes, that is homomorphic network coding signatures. We give relevant definitions in the following section.

2.3 Network Coding Signatures

In this section we give the definition of a network coding signature scheme and its security notion, as done by Boneh *et al.* in [7]. As we mentioned before, a network coding signature scheme allows to sign a subspace $\mathcal{W} \subset \mathbb{F}^N$ so that any vector $w \in \mathcal{W}$ is accepted, whereas vectors $w \notin \mathcal{W}$ are rejected. In particular, in our work we focus on subspaces $\mathcal{W}$ that are described by a properly augmented basis.

We assume that a file is associated with a file identifier fid that is chosen by the source node before the transmission. In general, such fid can be the filename. Though, in our systems we need such file identifiers to be randomly chosen by the source node. Thus we think of fid as an element of an efficiently samplable set $\mathcal{I}$.

Definition 3 (Network Coding Signatures). *A* network coding signature *is defined by a triple of algorithms (*$\mathsf{NetKG}, \mathsf{NetSign}, \mathsf{NetVer}$*) such that:*

$\mathsf{NetKG}(1^k, m, n)$ *On input the security parameter k and two integers m, n, this algorithm outputs $(\mathsf{vk}, \mathsf{sk})$ where sk is the secret signing key and vk is the public verification key. m defines the dimension of the vector spaces while n is an upper bound to the size of the signed vectors. We assume that the public key implicitly defines the field $\mathbb{F}$ over which vectors and linear combinations are defined.*

$\mathsf{NetSign}(\mathsf{sk}, \mathsf{fid}, \mathcal{W})$ *The signing algorithm takes as input the secret key sk, a random file identifier fid and a properly augmented basis of a m-dimensional subspace $\mathcal{W} \subset \mathbb{F}^{m+\ell}$ (with $1 \leq \ell \leq n$), and it outputs a signature σ.*

$\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}, w, \sigma)$ *Given the public key vk, a file identifier fid, a vector $w \in \mathbb{F}^{m+\ell}$ (for $1 \leq \ell \leq n$) and a signature σ, the algorithm outputs 0 (reject) or 1 (accept).*

For correctness, we require that for all honestly generated key pairs $(\mathsf{vk}, \mathsf{sk})$, all identifiers $\mathsf{fid} \in \mathcal{I}$, all $1 \leq \ell \leq n$, and all $\mathcal{W} \subset \mathbb{F}^{m+\ell}$, if $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{fid}, \mathcal{W})$ then $\mathsf{Ver}(\mathsf{vk}, \mathsf{fid}, w, \sigma) = 1 \ \forall w \in \mathcal{W}$.

Security of Network Coding Signatures. The security notion of network coding signatures is defined by the following game between a challenger and an adversary $\mathcal{A}$:

Setup. The adversary chooses positive integers m, n and gives them to the challenger. The challenger runs $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{NetKG}(1^k, m, n)$ and gives vk to $\mathcal{A}$.

Signing queries. The adversary can ask signatures on vector spaces $\mathcal{W}_i \subset \mathbb{F}^{m+\ell}$ (with $\ell \leq n$) of its choice, specified by giving to the challenger a properly augmented basis describing $\mathcal{W}_i$. The challenger chooses a random file identifier fid_i, runs $\sigma_i \xleftarrow{\$} \mathsf{NetSign}(\mathsf{sk}, \mathsf{fid}_i, \mathcal{W}_i)$ and returns σ_i to $\mathcal{A}$.

Forgery. The adversary outputs a tuple $(\mathsf{fid}^*, w^*, \sigma^*)$.

We say that the adversary wins this game if $\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}^*, v^*, \sigma^*) = 1$ and either one of the following cases holds: (1) $\mathsf{fid}^* \neq \mathsf{fid}_i$ for all i (*type-I forgery*); (2) $\mathsf{fid}^* = \mathsf{fid}_i$ for some i, but $w^* \notin \mathcal{W}_i$ (*type-II forgery*).

We define the advantage of $\mathcal{A}$ into breaking a network coding signature scheme, $\mathbf{Adv}^{NC}(\mathcal{A})$, as the probability that $\mathcal{A}$ wins the above security game, and we say that a network coding signature is secure if for any PPT $\mathcal{A}$, $\mathbf{Adv}^{NC}(\mathcal{A})$ is at most negligible in the security parameter.

Finally, we give the formal definition of *homomorphic network coding signature.*

Definition 4 (Homomorphic Network Coding Signatures). *A* homomorphic network coding signature scheme *is defined by a 4-tuple of algorithms (*$\mathsf{NetKG}, \mathsf{NetSign}, \mathsf{NetVer}, \mathsf{Combine}$*) such that:*

$\mathsf{NetKG}(1^k, m, n)$ *On input the security parameter k and two integers $m, n \geq 1$, this algorithm outputs $(\mathsf{vk}, \mathsf{sk})$ where sk is the secret signing key and vk is the public verification key. Here, m defines the dimension of the vector spaces and $n+m$ is an upper bound to the size of the signed vectors. We assume that the public key implicitly defines the field $\mathbb{F}$ over which vectors and linear combinations are defined, and that it contains the description of an efficiently samplable distribution for fid.*

$\mathsf{NetSign}(\mathsf{sk}, \mathsf{fid}, w)$ *The signing algorithm takes as input the secret key* sk*, a file identifier in the support of* fid *and a vector* $w \in \mathbb{F}^{\ell+m}$ *(with* $1 \leq \ell \leq n$*) and outputs a signature* σ*.*

$\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}, w, \sigma)$ *Given the public key* vk*, a file identifier* fid*, a vector* $w \in \mathbb{F}^{\ell}$ *and a signature* σ*, the algorithm outputs 0 (reject) or 1 (accept).*

$\mathsf{Combine}(\mathsf{vk}, \mathsf{fid}, \{(w^{(i)}, \alpha_i, \sigma_i)\}_{i=1}^{\mu})$ *This algorithm takes as input the public key* vk*, a file identifier* fid*, and a set of tuples* $(w^{(i)}, \alpha_i, \sigma_i)$ *where* σ_i *is a signature,* $w^{(i)} \in \mathbb{F}^{\ell}$ *is a vector and* $\alpha_i \in \mathbb{F}$ *is a scalar. This algorithm outputs a new signature* σ *such that: if each* σ_i *is a valid signature on vector* $w^{(i)}$*, then* σ *is a valid signature for* w *obtained from the linear combination* $\sum_{i=1}^{\mu} \alpha_i \cdot w^{(i)}$*.*

For correctness, we require that for all $m, n \geq 1$, all honestly generated pairs of keys $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{NetKG}(1^k, m, n)$ the following hold:

- For all $\mathsf{fid} \in \mathcal{I}$ and all $w \in \mathbb{F}^{m+\ell}$, if $\sigma \xleftarrow{\$} \mathsf{NetSign}(\mathsf{sk}, \mathsf{fid}, w)$, then $\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}, w, \sigma) = 1$.
- For all $\mathsf{fid} \in \mathcal{I}$, any $\mu > 0$, and all sets of triples $\{(w^{(i)}, \alpha_i, \sigma_i)\}_{i=1}^{\mu}$, if $\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}, w^{(i)}, \sigma_i) = 1$ for all i, then it must be the case that

$$\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}, \sum \alpha_i w^{(i)}, \mathsf{Combine}(\mathsf{vk}, \mathsf{fid}, \{(w^{(i)}, \alpha_i, \sigma_i)\}_{i=1}^{\mu}) = 1.$$

As noticed by Boneh et al. [7], homomorphic network coding signatures are a special case of network coding signatures.

2.4 An Efficient Linear Network Coding Scheme

In this section we specify the linear network coding scheme considered in our work. Basically, it is the random network coding solution described in the previous section except that we consider some optimizations recently proposed by Gennaro *et al.* in [13]. The scheme works as follows.

The application specifies four global parameters $m, n, M, p' \in \mathbb{N}$ such that $m, n \geq 1$, and p' is a prime. In this setting, a file $\mathcal{V}$ to be transmitted is always encoded as a set of m vectors $(v^{(1)}, \ldots, v^{(m)})$ where each $v^{(i)}$ takes values in $\mathbb{F}_M^{\ell}$ where M is a bound on the initial magnitude of each coordinate and $\ell \leq n$. Since m is fixed in advance by the application, at the time of the transmission, once the size of the file $\mathcal{V}$ is known, the total length of information in every vector $v^{(i)}$ is determined. Thus, ℓ can be chosen accordingly as any number between 1 and n. The freedom in choosing ℓ is important as different choices have different impact on the efficiency of the scheme: a smaller ℓ saves bandwidth, while a larger ℓ saves computation (see [13] for more details). The parameter p' specifies the domain $P = \{0, \ldots, p' - 1\}$ from which the network nodes sample the coefficients for the linear combination. Linear combinations can then be performed either over the integers, or modulo some large prime p (which is specified by the application or by the signature scheme). Gennaro et *al.* show that taking a small p' (e.g., $p' = 257$) allows to improve the performances of the network coding scheme as well as to keep a good decoding probability. In particular, they show that this holds in both cases when the linear combinations are done over the integers, or over some large prime $p > M$. Precisely, in the latter case, the performances remain better (than the case when coefficients are chosen in $\mathbb{F}_p$) as long as the bit-size of p' is negligible compared to the bit-size k of the prime p.

Global application parameters: $m, n, M, p' \in \mathbb{N}$ as specified above.

Key Generation: Each source node generates a pair of keys $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{NetKG}(1^k, m, n)$ of a homomorphic network coding signature scheme.

File transmission: On input a file $\mathcal{V}$ represented by m vectors $v^{(1)}, \ldots, v^{(m)} \in \mathbb{F}_M^\ell$ (with $\ell \leq n$), the source node generates augmented vectors $w^{(1)}, \ldots, w^{(m)}$, i.e., $w^{(i)} = (u^{(i)}, v^{(i)})$ where $u^{(i)}$ is the i-th unity vector. Next, it chooses a random file identifier $\mathsf{fid} \xleftarrow{\$} \mathcal{I}$ (recall that $\mathcal{I}$ is specified by vk), and for $i = 1$ to m, it generates $\sigma_i \xleftarrow{\$} \mathsf{NetSign}(\mathsf{sk}, \mathsf{fid}, w^{(i)})$. Finally, it sends the tuples $(\mathsf{fid}, w^{(i)}, \sigma_i)$ on its outgoing edges.

Intermediate nodes: When a node receives μ vectors $w^{(1)}, \ldots, w^{(\mu)}$ and signatures $\sigma_1, \ldots, \sigma_\mu$, all corresponding to file fid, it proceeds as follows. First, it checks that $\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}, w^{(i)}, \sigma_i) = 1$, for $i = 1$ to μ. It discards all the vectors (and signatures) that did not pass the check. For the remaining vectors (for simplicity, let they be $w^{(1)}, \ldots, w^{(\mu)}$), the node chooses $\alpha_1, \ldots, \alpha_\mu \xleftarrow{\$} P$, and computes: $w = \sum_{i=1}^{\mu} \alpha_i \cdot w^{(i)}$, $\sigma \leftarrow \mathsf{Combine}(\mathsf{vk}, \mathsf{fid}, \{(w^{(i)}, \alpha_i, \sigma_i)\}_{i=1}^{\mu})$. Finally, the node sends $(\mathsf{fid}, w, \sigma)$ on its outgoing edges.

Target node: Once a node obtains m linearly independent vectors $w^{(1)}, \ldots, w^{(m)}$ together with the respective signatures and the same file identifier fid, it first checks that they are all valid, i.e., it verifies that $\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}, w^{(i)}, \sigma_i) = 1$, $\forall i = 1, \ldots, m$. Given m valid vectors, the node can reconstruct the original file $(v^{(1)}, \ldots, v^m)$ as described in Section 2.2.

3 A construction based on SDH

In this section we propose the construction of a network coding homomorphic signature based on the Strong Diffie-Hellman assumption.

Recall that we are in the setting of the linear network coding application described in the previous section. A file $\mathcal{V}$ is represented as a set of m vectors $(v^{(1)}, \ldots, v^{(m)})$ such that each $v^{(i)} \in \mathbb{F}_p^\ell$ where p is a (publicly known) prime specified by the key generation algorithm and $\ell \leq n$. Notice that all the operations with the vectors are thus defined over the finite field $\mathbb{F}_p$, i.e., $\bmod p$. Moreover, the space for file identifiers is the set $\mathbb{Z}_p^*$ where p is the same prime specified in the key generation.

Below we give a precise description of the scheme's algorithms[5]:

$\mathsf{NetKG}(1^k, n, m)$**:** Let $\mathbb{G}, \mathbb{G}', \mathbb{G}_T$ be bilinear groups of prime order p such that $e : \mathbb{G} \times \mathbb{G}' \to \mathbb{G}_T$ is a bilinear map and $g \in \mathbb{G}, g' \in \mathbb{G}'$ are two generators. Pick a random $z \xleftarrow{\$} \mathbb{Z}_p$ and set $Z = (g')^z$. Choose random elements $h, h_1, \ldots, h_m, g_1, \ldots, g_n \xleftarrow{\$} \mathbb{G}$. Output the public verification key $\mathsf{vk} = (p, g, g', Z, h, h_1, \ldots, h_m, g_1, \ldots, g_n)$ and the secret key $\mathsf{sk} = z$.

$\mathsf{NetSign}(\mathsf{sk}, \mathsf{fid}, w)$**:** Let $w = (u, v) \in \mathbb{F}_p^{m+n}$ be a properly augmented vector, and let fid be randomly chosen in $\mathbb{Z}_p^*$. The signing algorithm proceeds as follows.

Pick a random $s \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$X = \left(h^s \prod_{i=1}^{m} h_i^{u_i} \prod_{i=1}^{n} g_i^{v_i} \right)^{\frac{1}{z+\mathsf{fid}}}$$

Finally, output $\sigma = (X, s)$.

$\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}, w, \sigma)$ Let $\sigma = (X, s) \in \mathbb{G} \times \mathbb{Z}_p$. This algorithm checks whether σ is a valid signature on a vector $w = (u, v)$ w.r.t. the file identifier fid.

[5] For ease of exposition, in our description we assume that the vectors w have the maximum length $m + n$. In fact, in our scheme any shorter vector with $\ell < n$ can be augmented by appending $n - \ell$ zeros.

If the following equation holds, then output 1, otherwise output 0:

$$e(X, Z \cdot (g')^{\mathsf{fid}}) = e(h^s \prod_{i=1}^{m} h_i^{u_i} \prod_{i=1}^{n} g_i^{v_i}, g').$$

$\mathsf{Combine}(\mathsf{vk}, \mathsf{fid}, \{w^{(i)}, \alpha_i, \sigma_i\}_{i=1}^{\mu})$**:** Recall that for all $i \in \{1, \ldots, \mu\}$ $w^{(i)} = (u^{(i)}, v^{(i)})$ where $u^{(i)} \in \mathbb{F}_p^m$ and $v^{(i)} \in \mathbb{F}_p^n$, and that $\alpha_i \in \mathbb{F}_p$ is a randomly chosen coefficient. Moreover, recall that in our application this algorithm is run when every σ_i has been verified as a valid signature on $w^{(i)}$ w.r.t. fid.

The algorithm computes

$$X = \prod_{i=1}^{\mu} (X_i)^{\alpha_i}, \quad s = \sum_{i=1}^{\mu} \alpha_i \cdot s_i \bmod p$$

and outputs $\sigma = (X, s)$.

3.1 Efficiency

A signature consists of one element of $\mathbb{G}$ and one element of $\mathbb{Z}_p$. Signing costs a multi-exponentiation in $\mathbb{G}$, whereas verification needs to compute two pairings, one exponentiation in $\mathbb{G}'$, i.e., $(g')^{\mathsf{fid}}$, and one multi-exponentiation.

3.2 Proof of security

The following theorem proves that the scheme described before is a secure homomorphic network coding signature.

Theorem 1. *If the q-SDH assumption holds in $(p, \mathbb{G}, \mathbb{G}', \mathbb{G}_T)$ for any polynomial q, then the scheme described above is a secure network coding signature.*

As usual, the proof proceeds by contradiction. Namely, assume there exists a PPT adversary $\mathcal{A}$ such that $\mathbf{Adv}^{NC}(\mathcal{A}) = \epsilon$ is non-negligible. Then we show that such $\mathcal{A}$ can be used to build an efficient algorithm $\mathcal{B}$ that solves the q-SDH assumption, where q is an upper bound to the number of signatures asked by $\mathcal{A}$ (which is polynomial in the security parameter). More specifically, we show that we can classify different types of adversaries according to the type of forgery they produce. For each of these cases we will show a different simulation.

Let q be the number of queries made by $\mathcal{A}$, and $\mathsf{fid}_1, \ldots, \mathsf{fid}_q$ be all the random file identifiers generated by the Challenger during the security game. Let $(\mathsf{fid}^*, w^*, \sigma^*)$ be the forgery returned by the adversary $\mathcal{A}$ such that $\mathcal{A}$ wins. By definition it holds $\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}^*, w^*, \sigma^*) = 1$ and the forgery is either of type-I or of type-II. Namely:

Type-I: $\mathsf{fid}^* \neq \mathsf{fid}_i, \forall i = 1, \ldots, q$;

Type-II: $\exists j \in \{1, \ldots, q\} : \mathsf{fid}^* = \mathsf{fid}_j$ and $w^* \notin \mathcal{W}_j$, where $\mathcal{W}_j$ is the vector space asked by $\mathcal{A}$ in the j-th query.

Notice that at least one of these cases has to occur with probability at least $1/2$. For our proof, we will describe two different simulators $\mathcal{B}_1$ and $\mathcal{B}_2$ that work in Type-I and Type-II case respectively. Then, we can put together the two simulations by defining our main algorithm $\mathcal{B}$ so that $\mathcal{B}$ flips a

coin $b \xleftarrow{\$} \{1,2\}$ and runs $\mathcal{B}_b$. If $\mathcal{B}_1$ and $\mathcal{B}_2$ have advantage ϵ_1 and ϵ_2 respectively, then the advantage of $\mathcal{B}$ is $\geq min(\frac{\epsilon_1}{2}, \frac{\epsilon_2}{2})$.

To prove the theorem we prove the following two lemmata.

Lemma 1. *If there exists a PPT adversary $\mathcal{A}$ such that $\mathbf{Adv}^{NC}(\mathcal{A}) \geq \epsilon$ and $\mathcal{A}$ returns a Type-I forgery, then there exists a PPT algorithm $\mathcal{B}_1$ that breaks the q-SDH assumption with advantage at least $\epsilon/2(m+n)$.*

Proof. Among Type-I forgeries we identify two sub-cases that are defined as follows. Let $\sigma^* = (X^*, s^*)$ and $w^* = (u^*, v^*)$. Let $\alpha_0, \alpha_1, \ldots, \alpha_m, y_1, \ldots, y_n \in \mathbb{Z}_p$ be such that for all $i = 1$ to m: $h_i = h^{\alpha_i/\alpha_0}$ and for all $i = 1$ to n: $g_i = h^{y_i/\alpha_0}$. Consider the values (s^*, u^*, v^*) from the forgery and let $\gamma = (\alpha_0 s^* + \sum_{k=1}^{m} \alpha_k u_k^* + \sum_{k=1}^{n} y_k v_k^*)$. We distinguish between the following two types of forgeries:

Type-I.a: $(\mathsf{fid}^*, \sigma^*, w^*)$ such that $\gamma = 0$. We show that this case can be reduced to solving discrete log (which in turn implies a solver for q-SDH).

Type-I.b: $(\mathsf{fid}^*, \sigma^*, w^*)$ such that $\gamma \neq 0$. This case can be directly reduced to the q-SDH assumption.

Our algorithm $\mathcal{B}_1$ takes as input a tuple $(g, g^z, g^{z^2}, \ldots, g^{z^q}, g', (g')^z)$ and works as follows. First, it flips a coin $c \xleftarrow{\$} \{0,1\}$. If $c = 0$, it guesses that the adversary will produce a Type-I.a forgery. Otherwise, if $c = 1$, it guesses that the adversary will return a forgery of Type-I.b. Notice that with probability at least $1/2$ the guess is correct.

Type-I.a simulation. If $\mathcal{B}_1$ guessed on a Type-I.a forgery, it runs a simulation that uses $\mathcal{A}$ to solve the discrete logarithm problem. More precisely, given the pair $(g, g^z) \in \mathbb{G}^2$ from the q-SDH instance, $\mathcal{B}_1$ will use $\mathcal{A}$ to find z.

Recall that by definition of forgery $w^* \neq 0^{m+n}$, namely there exists an index $\nu \in \{1, \ldots, m+n\}$ such that $w_\nu^* \neq 0$. $\mathcal{B}_1$ guesses such ν at the beginning of the simulation by choosing it at random in $\{1, \ldots, m+n\}$, and it creates the public key as follows. It chooses integers $\alpha_0, \alpha_1, \ldots \alpha_{m+n} \xleftarrow{\$} \mathbb{Z}_p$, and it sets $h = g^{\alpha_0}$. If $1 \leq \nu \leq m$, then $\mathcal{B}_1$ sets $h_i = g^{\alpha_i}$ for all $i = 1$ to m, $i \neq \nu$, $h_\nu = g^z$, and $g_i = g^{\alpha_{m+i}}$ for all $i = 1$ to n.

Otherwise, if $m < \nu \leq m+n$, it sets $g_{\nu-m} = g^z$, $h_i = g^{\alpha_i}$ for all $i = 1$ to m, and $g_i = g^{\alpha_{m+i}}$ for all $i = 1$ to n, $i \neq \nu - m$. To complete the key generation, $\mathcal{B}_1$ chooses a random $z' \xleftarrow{\$} \mathbb{Z}_p$, and it computes $Z = (g')^{z'}$.

Let $\mathsf{vk} = (p, g, g', Z, h, h_1, \ldots, h_m, g_1, \ldots, g_n)$ be the public key computed as above that is returned to the adversary. It is easy to see that vk is correctly distributed. Moreover, since $\mathcal{B}_1$ knows z', it can easily answer all the signing queries.

So, assume that at the end of the game the adversary produced a forgery such that $\gamma = 0$, that is: $\alpha_0 s^* + \sum_{k=1}^{m+n} \alpha_k w_k^* = 0$. If $\mathcal{B}_1$'s guess about the index ν such that $w_\nu^* \neq 0$ was right (this is true with probability $1/(m+n)$), then the sum contains the element $\alpha_\nu \cdot w_\nu^* \neq 0$, where $\alpha_\nu = z$. Thus, it is straightforward to see how $\mathcal{B}_1$ can extract z and then use it to solve q-SDH.

Type-I.b simulation. In this case $\mathcal{B}_1$ has guessed that $\mathcal{A}$ will return a forgery of Type-I.b, i.e., such that $\gamma \neq 0$. The simulator proceeds as follows.

Key Generation: Recall that q is an upper bound for the number of signing queries asked by $\mathcal{A}$. $\mathcal{B}_1$ chooses $\mathsf{fid}_1, \ldots, \mathsf{fid}_q \xleftarrow{\$} \mathbb{Z}_p^*$ at random. It defines the polynomial

$$P(z) = \prod_{i=1}^{q}(z + \mathsf{fid}_i) = \sum_{i=0}^{q} p_i z^i$$

and computes its coefficients $p_0, \ldots, p_q$. It sets $Z = (g')^z$, and computes $S = \prod_{i=0}^{q}(g^{z^i})^{p_i} = g^{P(z)}$. Next, for all $i = 1$ to m, it picks a random $\alpha_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $h_i = S^{\alpha_i}$. For all $i = 1$ to n, it picks a random $y_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $g_i = S^{y_i}$. Finally, h is computed as S^{α_0} for a randomly chosen $\alpha_0 \xleftarrow{\$} \mathbb{Z}_p$. $\mathcal{B}_1$ gives $\mathsf{vk} = (g, g', Z, h, h_1, \ldots, h_m, g_1, \ldots, g_n)$ to $\mathcal{A}$. Notice that such vk has the same distribution as that generated by the real key generation algorithm.

Signing queries: We describe how $\mathcal{B}_1$ answers all signing queries. For all $j \in \{1, \ldots, q\}$, consider the j-th query asking a signature for the vector space $\mathcal{W} = (w^{(1)}, \ldots, w^{(m)})$. Recall that these are properly augmented vectors where the first m components of $w^{(i)}$ are the i-th unity vector, i.e., $w^{(i)} = (u^{(i)}, v^{(i)})$. $\mathcal{B}_1$ uses the j-th file identifier fid_j, and defines the polynomial

$$P_j(z) = \frac{P(z)}{z + \mathsf{fid}_j} = \prod_{i=1, i\neq j}^{q}(z + \mathsf{fid}_i) = \sum_{i=0}^{q-1} \delta_i z^i$$

It computes $S_j = \prod_{i=1}^{q-1}(g^{z^i})^{\delta_i} = S^{\frac{1}{z+\mathsf{fid}_j}}$. Next, for all $i = 1$ to m, it picks a random $s_i \xleftarrow{\$} \mathbb{Z}_p$ and it computes

$$X_i = S_j^{\alpha_0 s_i + \alpha_i + \sum_{k=1}^{n} y_k v_k^{(i)}} = \left(S^{\alpha_0 s_i + \alpha_i + \sum_{k=1}^{n} y_k v_k^{(i)}}\right)^{\frac{1}{z+\mathsf{fid}_j}} = \left(h^{s_i} \cdot h_i \prod_{k=1}^{n} g_k^{v_k^{(i)}}\right)^{\frac{1}{z+\mathsf{fid}_j}}$$

Finally, $\mathcal{B}_1$ returns fid_j and $\{\sigma_i = (X_i, s_i)\}_{i=1}^{m}$ to the adversary. As one can easily check, these are valid signatures that are distributed like real ones.

Forgery: At the end of the game the adversary is supposed to output a forgery $(\mathsf{fid}^*, w^*, \sigma^*)$ such that $\mathsf{fid}^* \neq \mathsf{fid}_j$ for all $j \in \{1, \ldots, q\}$, and $\gamma \neq 0$. Moreover, $\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}^*, w^*, \sigma^*) = 1$, which means that

$$e(X^*, Z \cdot (g')^{\mathsf{fid}^*}) = e(h^{s^*} \prod_{i=1}^{m} h_i^{u_i^*} \prod_{i=1}^{n} g_i^{v_i^*}, g')$$

that is

$$(X^*)^{z+\mathsf{fid}^*} = S^{\alpha_0 s^* + \sum_{k=1}^{m} \alpha_k u_k^* + \sum_{k=1}^{n} y_k v_k^*} = g^{P(z)(\alpha_0 s^* + \sum_{k=1}^{m} \alpha_k u_k^* + \sum_{k=1}^{n} y_k v_k^*)}$$

Since $\mathsf{fid}^* \neq \mathsf{fid}_j, \forall j = 1, \ldots, q$, it holds $(z + \mathsf{fid}^*) \nmid P(z)$. Hence, there exists a polynomial $\eta(z) = \sum_{i=0}^{q-1} \eta_i z^i$ and a value $\eta' \in \mathbb{Z}_p$ (that are efficiently computable) such that $P(z) = \eta(z)(z + \mathsf{fid}^*) + \eta'$. If we let $\gamma = (\alpha_0 s^* + \sum_{k=1}^{m} \alpha_k u_k^* + \sum_{k=1}^{n} y_k v_k^*)$, then we can write $X^* = g^{(\eta(z) + \frac{\eta'}{z+\mathsf{fid}^*})\gamma}$. Since $\gamma \neq 0$, $\mathcal{B}_1$ can finally compute

$$g^{\frac{1}{z+\mathsf{fid}^*}} = \left[(X^*)^{1/\gamma} \cdot \left(\prod_{i=1}^{Q-1}(g^{z^i})^{\eta_i}\right)^{-1}\right]^{1/\eta'}$$

and return $(g^{\frac{1}{z+\mathsf{fid}^*}}, \mathsf{fid}^*)$ as a solution for the q-SDH problem.

To conclude the proof let us analyze $\mathcal{B}_1$'s probability of success. It is straightforward to see that if the adversary has advantage ϵ into forging the signature scheme, then $\mathcal{B}_1$ has probability at least $\epsilon/2(m+n)$ of solving Q-SDH. □

Lemma 2. *If there exists an adversary $\mathcal{A}$ that returns a Type-II forgery and that has advantage ϵ against the security of the above scheme, then there exists an algorithm $\mathcal{B}_1$ that breaks the q-SDH assumption with advantage at least ϵ/qn.*

Proof. By definition, if the adversary outputs a forgery $(\mathsf{fid}^*, w^*, \sigma^*)$ such that $\mathsf{fid}^* = \mathsf{fid}_j$, then it must be $w^* \notin \mathcal{W}$, where $\mathcal{W}$ is the vector subspace for which the adversary asked a signature in the j-th query (i.e., the one to which the challenger assigned identifier fid_j) and that is described by the properly augmented basis $(w^{(1)}, \ldots, w^{(m)})$. For all possible coefficients $\eta \in \mathbb{F}_p^m$, it holds $w^* \neq \sum_{i=1}^m \eta_i \cdot w^{(i)}$. In particular, this must hold also for the coefficients $u^* \in \mathbb{F}_p^m$. In this case, notice that

$$(w^* - \sum_{i=1}^{m} w^{(i)} u_i^*) = (0, \ldots, 0, z_1, \ldots, z_n) \neq 0^{m+n}.$$

The first m components are 0 because $w^{(i)}$ contains the unity vector $u^{(i)}$. Thus, since it is not zero, there must exist at least a $\nu \in \{1, \ldots, n\}$ such that $z_\nu \neq 0$.

Let α_ν be such that $g_\nu = h^{\alpha_\nu}$, and let $s_1, \ldots, s_m$ be the random exponents chosen by the Challenger to produce the m signatures in the j-th query. If we consider the value $\gamma = s^* - \sum_{i=1}^m s_i u_i^* + \alpha_\nu z_\nu$, then we can distinguish between two distinct types of adversaries:

Type-II.a: $\mathcal{A}$ produces queries and a forgery such that $\gamma = 0$. If this is the case, then we show that this can be easily reduced to solving discrete logarithm.

Type-II.b: $\mathcal{A}$ produces queries and a forgery such that $\gamma \neq 0$. For this case we show a reduction to the q-SDH problem.

Notice that the value γ is determined by the material chosen by the adversary during the game (in particular, the forgery and the j-th vector space) conditioned on the random choices of the Challenger. At least one of these cases occurs with probability at least $1/2$.

Our algorithm $\mathcal{B}_2$ takes as input a tuple $(g, g^z, g^{z^2}, \ldots, g^{z^q}, g', (g')^z)$ and works as follows. First, it flips a coin $c \xleftarrow{\$} \{0, 1\}$. If $c = 0$, it guesses that the adversary will produce a Type-II.a forgery. Otherwise, if $c = 1$, it guesses that the adversary will return a Type-II.b forgery. This guess will be correct with probability at least $1/2$.

Type-II.a simulation. If $\mathcal{B}_2$ guessed that the forgery will be of Type-II.a, then it takes the pair $(g, g^z) \in \mathbb{G}^2$ from the q-SDH instance, and it proceeds as follows. It first chooses a random index $\nu \xleftarrow{\$} \{1, \ldots, n\}$, it picks $\alpha_0 \xleftarrow{\$} \mathbb{Z}_p$ at random, and it sets $h = g^{\alpha_0}$ and $g_\nu = g^{z/\alpha_0}$. The remaining part of the public key is generated as in the real key generation algorithm. In particular, $\mathcal{B}_2$ will choose a random $z' \xleftarrow{\$} \mathbb{Z}_p$ and set $Z = (g')^{z'}$. This way, $\mathcal{B}_2$ can answer all the signing queries.

To conclude the simulation, assume that $\mathcal{B}_2$'s guesses about $\mathcal{A}$ returning a Type-II.a forgery, and the index ν such that $z_\nu \neq 0$ were both right. If this is the case, then for the value γ defined above it holds $\gamma = s^* - \sum_{i=1}^m s_i u_i^* + \alpha_\nu z_\nu = 0$. It is trivial to see how $\mathcal{B}_2$ can extract $\alpha_\nu = z$ from the above equation.

Type-II.b simulation. If $\mathcal{B}_2$ has guessed that the forgery will be of Type-II.b (i.e., such that $\gamma \neq 0$), then it runs the following simulation. First, it picks random indices $j \xleftarrow{\$} \{1, \ldots, q\}$ and

$\nu \xleftarrow{\$} \{1,\ldots,n\}$ as its guesses for the query j such that $\mathsf{fid}^* = \mathsf{fid}_j$ and the index ν such that $z_\nu \neq 0$ respectively. Then, it proceeds as follows.

Key Generation: $\mathcal{B}_2$ chooses $\mathsf{fid}_1,\ldots,\mathsf{fid}_q \xleftarrow{\$} \mathbb{Z}_p$ at random and defines the polynomials

$$P(z) = \prod_{i=1}^{q}(z+\mathsf{fid}_i) = \sum_{i=0}^{q} p_i z^i, \quad P^*(z) = \frac{P(z)}{z+\mathsf{fid}_j} = \prod_{i=1,i\neq j}^{q}(z+\mathsf{fid}_i) = \sum_{i=0}^{q-1}\delta_i z^i$$

It sets $Z = (g')^z$, and it computes $S = g^{P(z)} = \prod_{i=0}^{q}(g^{z^i})^{p_i}$ and $S^* = g^{P^*(z)} = \prod_{i=0}^{q-1}(g^{z^i})^{\delta_i}$. Next, it computes $h = (S^*)^{\alpha_0}$ and $g_\nu = h^{\alpha_\nu}$ for randomly chosen $\alpha_0, \alpha_\nu \xleftarrow{\$} \mathbb{Z}_p$. For all $i = 1$ to m, it picks random $\omega_i, \beta_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $h_i = S^{\omega_i} h^{-\beta_i}$. For all $i = 1$ to n, $i \neq \nu$, it picks a random $y_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $g_i = S^{y_i}$. Finally, $\mathcal{B}_2$ outputs $\mathsf{vk} = (p, g, g', Z, h, h_1, \ldots, h_m, g_1, \ldots, g_n)$. Notice that vk is perfectly distributed like in the real case.

Signing queries: for all $k \in \{1,\ldots,q\}$ let $\mathcal{W} = (w^{(1)},\ldots,w^{(m)})$ be the vector space asked in the k-th query (where each $w^{(i)}$ is a properly augmented vector $(u^{(i)}, v^{(i)})$).

For all $k \neq j$ the signatures are simulated as follows. $\mathcal{B}_2$ uses the k-th file identifier fid_k, and defines the polynomials

$$P_k(z) = \frac{P(z)}{z+\mathsf{fid}_k} = \prod_{i=1,i\neq k}^{q}(z+\mathsf{fid}_i) = \sum_{i=0}^{q-1}\theta_i z^i,$$

$$P_{j,k}(z) = \frac{P(z)}{(z+\mathsf{fid}_k)(z+\mathsf{fid}_j)} = \prod_{i=1,i\neq k,j}^{q}(z+\mathsf{fid}_i) = \sum_{i=0}^{q-2}\phi_i z^i$$

It computes $S_k = \prod_{i=0}^{q-1}(g^{z^i})^{\theta_i} = S^{\frac{1}{z+\mathsf{fid}_k}}$ and $S_{j,k} = \prod_{i=0}^{q-2}(g^{z^i})^{\phi_i} = (S^*)^{\frac{1}{z+\mathsf{fid}_k}}$. Next, for all $i = 1$ to m, it picks a random $s_i \xleftarrow{\$} \mathbb{Z}_p$ and it computes

$$X_i = S_{j,k}^{\alpha_0 s_i + \alpha_0\alpha_\nu v_\nu^{(i)} - \alpha_0\beta_i} S_k^{\omega_i + \sum_{l=1,l\neq\nu}^{n} y_l v_l^{(i)}} = \left(h^{s_i} \cdot h_i \prod_{l=1}^{n} g_l^{v_l^{(i)}}\right)^{\frac{1}{z+\mathsf{fid}_k}}$$

To answer the j-th query, $\mathcal{B}_2$ uses a different approach. For all $i = 1$ to m, $\mathcal{B}_2$ sets $s_i = \beta_i - \alpha_\nu v_\nu^{(i)} \bmod p$ and it computes

$$X_i = (S^*)^{\omega_i + \sum_{l=1,l\neq\nu}^{n} y_l v_l^{(i)}} = \left((S^*)^{\alpha_0(s_i + \alpha_\nu v_\nu^{(i)}) - \alpha_0\beta_i}\right)^{\frac{1}{z+\mathsf{fid}_k}} \left(S^{\omega_i} \prod_{l=1,l\neq\nu}^{n} g_l^{v_l^{(i)}}\right)^{\frac{1}{z+\mathsf{fid}_k}}$$

It is not hard to check that this is a valid signature.

Finally, $\mathcal{B}_2$ returns fid_k and $\{\sigma_i = (X_i, s_i)\}_{i=1}^{m}$ to the adversary.

Forgery: At the end of the game the adversary is supposed to output a forgery $(\mathsf{fid}^*, w^*, \sigma^*)$ such that $\mathsf{fid}^* = \mathsf{fid}_j$ and $\gamma \neq 0$. From the validity of σ^* we have that

$$e(X^*, Z \cdot (g')^{\mathsf{fid}^*}) = e(h^{s^*} \prod_{i=1}^{m} h_i^{u_i^*} \prod_{i=1}^{n} g_i^{v_i^*}, g')$$

Let

$$\Psi = \frac{X^*}{\prod_{i=1}^{m} X_i^{u_i^*}} = \left(h^{s^* - \sum_{i=1}^{m} s_i u_i^*} \prod_{k=1}^{n} g_k^{z_k} \right)^{\frac{1}{z+\mathsf{fid}^*}} = \left(g^{P^*(z)\alpha_0(s^* - \sum_{i=1}^{m} s_i u_i^* + \alpha_\nu z_\nu)} \right)^{\frac{1}{z+\mathsf{fid}^*}} g^{P^*(z)(\sum_{k=1,k\neq\nu}^{n} y_k z_k)}$$

Let $\gamma' = \alpha_0(s^* - \sum_{i=1}^{m} s_i u_i^* + \alpha_\nu z_\nu) = \alpha_0 \cdot \gamma$, where γ is defined as above.
Notice that $(z + \mathsf{fid}^*) \nmid P^*(z)$. Thus, there exists a polynomial $\eta(z) = \sum_{i=0}^{q-1} \eta_i z^i$ and a value $\eta' \in \mathbb{Z}_p$ (that are efficiently computable) such that $P^*(z) = \eta(z)(z + \mathsf{fid}^*) + \eta'$. We can write:

$$\Psi \cdot g^{P^*(z)(-\sum_{k=1,k\neq\nu}^{n} y_k z_k)} = g^{(\eta(z) + \frac{\eta'}{z+\mathsf{fid}^*})\gamma'}$$

$\mathcal{B}_2$ can finally compute:

$$g^{\frac{1}{z+\mathsf{fid}^*}} = \left[\left(\Psi \cdot g^{P^*(z)(-\sum_{k=1,k\neq\nu}^{n} y_k z_k)} \right)^{1/\gamma'} \cdot \left(\prod_{i=1}^{q-1} (g^{z^i})^{\eta_i} \right)^{-1} \right]^{1/\eta'}$$

and it returns $(g^{\frac{1}{z+\mathsf{fid}^*}}, \mathsf{fid}^*)$ as a solution for the q-SDH problem.

To conclude the proof, we analyze $\mathcal{B}_2$'s probability of solving q-SDH. If the adversary has advantage at least ϵ into forging the signature scheme (in the Type-II case), then our algorithm $\mathcal{B}_2$ has advantage at least $\epsilon/2qn$ of solving the q-SDH problem. □

4 A (Strong) RSA based realization

In this section we describe our strong-RSA based implementation. We stress that the file to be signed is encoded as a set of vectors $(v^{(1)}, \ldots, v^{(m)})$ of ℓ components each where $\ell \leq n$ for some pre-specified bound n. Before being signed and transmitted, such vectors will be prepended with m unitary vectors $u^{(i)}$ (each having m components). We denote with $w^{(i)}$ the resulting vectors. Our implementation uses a parameter λ to specify the space $\mathcal{I}$ for the file identifiers. If M is the bound on the initial magnitude of each vector component, then $2^\lambda > M$ and $\mathcal{I}$ is the set of prime numbers of (exactly) $\lambda + 1$ bits, greater than 2^λ.

Finally, we notice that in this scheme the exact finite field over which are done the linear combinations is different for each file. In particular, it will be $\mathbb{F}_e$ where $e = \mathsf{fid}$ (e is a prime number) is the file identifier chosen by the sender. More precisely, this means that whenever a vector space $\mathcal{W}$ has to be signed, a file identifier $\mathsf{fid} = e$ is chosen (as a sufficiently large prime) and it is associated to $\mathcal{W}$. Thus, linear combinations are done $\bmod e$ and $w \notin \mathcal{W}$ implies that w cannot be written as a linear combination mod e of vectors of $\mathcal{W}$.

A precise description of our network coding scheme $\mathsf{NetPFSig} = (\mathsf{NetKG}, \mathsf{NetSign}, \mathsf{NetVer}, \mathsf{Combine})$ follows.

$\mathsf{NetKG}(1^k, \lambda, m, n)$ The NetKG algorithm chooses two random (safe) primes p, q of length $k/2$ each. It sets $N = pq$ and proceeds by choosing $g, g_1, \ldots, g_n, h_1, \ldots, h_m$ at random (in $\mathbb{Z}_N^*$). In addition to k, here we assume an additional security parameter λ which specifies the space $\mathcal{I}$ of file identifiers as described before. The public key is set as $(N, g, g_1, \ldots, g_n, h_1, \ldots, h_m)$, while the secret key is (p, q).

$\mathsf{NetSign}(\mathsf{sk}, \mathsf{fid}, w)$ The signing algorithm proceeds as follows. Let $w = (u, v) \in \mathbb{F}_M^{m+n}$ and let fid be a random file identifier, which is a prime number of the form specified before. For ease of exposition, let $e = \mathsf{fid}$. The signer chooses a random element $s \in \mathbb{Z}_e$ and uses its knowledge of p and q to solve the following equation

$$x^e = g^s \prod_{j=1}^{m} h_j^{u_j} \prod_{j=1}^{n} g_j^{v_j} \bmod N$$

Finally, it outputs the signature $\sigma = (s, x)$.

$\mathsf{NetVer}(\mathsf{vk}, \mathsf{fid}, w, \sigma)$ To verify a signature $\sigma = (s, x)$ on a vector w, the verification algorithm proceeds as follows. Let $e = \mathsf{fid}$.

- Check that e is an odd number of the right size (i.e. $\lambda + 1$ bits).
- Check that all the u's, v's and s are in $\mathbb{Z}_e$.
- Check that the equation $x^e = g^s \prod_{j=1}^{m} h_j^{u_j} \prod_{j=1}^{n} g_j^{v_j} \bmod N$ is satisfied by the given x.
- If all the checks above are satisfied, output 1, otherwise 0.

$\mathsf{Combine}(\mathsf{vk}, \mathsf{fid}, \{w^{(i)}, \alpha_i, \sigma_i\}_{i=1}^{\mu}$ To combine signatures σ_i, corresponding to vectors $w^{(i)}$ sharing the same fid, the algorithm proceeds as follows.

- It computes

$$w = \sum_{i=1}^{\mu} \alpha_i \cdot w^{(i)} \bmod e,\ w' = (\sum_{i=1}^{\mu} \alpha_i \cdot w^{(i)} - w)/e,\ s = \sum_{i=1}^{\mu} \alpha_i s_i \bmod e,\ s' = (\sum_{i=1}^{\mu} \alpha_i s_i - s)/e$$

Let $w' = (u', v')$. It outputs $\sigma = (s, x)$ where x is obtained by computing:

$$x = \frac{\prod_{i=1}^{\mu} x_i^{\alpha_i}}{g^{s'} \prod_{j=1}^{m} h_j^{u'_j} \prod_{j=1}^{n} g_j^{v'_j}} \bmod N$$

To complete the description of the scheme we show its correctness. In particular, while the correctness of the signatures returned by the signing algorithm can be easily checked by inspection, we pause to show that also the signatures obtained from the $\mathsf{Combine}$ algorithm are correct. Assume that for $i = 1$ to μ, $\sigma_i = (x_i, s_i)$ is a valid signature on the vector $w^{(i)} = (u^{(i)}, v^{(i)})$, and let α_i be the integer coefficients of the linear combination. Let $\sigma = (x, s)$ be the signature as computed by $\mathsf{Combine}(\mathsf{vk}, \mathsf{fid}, \{w^{(i)}, \alpha_i, \sigma_i\}_{i=1}^{\mu}$. We have that:

$$x^e = \frac{\prod_{i=1}^{\mu} (x_i^e)^{\alpha_i}}{(g^{s'} \prod_{j=1}^{m} h_j^{u'_j} \prod_{j=1}^{n} g_j^{v'_j})^e} \tag{1}$$

$$= \frac{g^{\sum_{i=1}^{\mu} s_i \alpha_i} \prod_{j=1}^{m} h_j^{\sum_{i=1}^{\mu} u_j^{(i)} \alpha_i} \prod_{j=1}^{n} g_j^{\sum_{i=1}^{\mu} v_j^{(i)} \alpha_i}}{(g^{s'} \prod_{j=1}^{m} h_j^{u'_j} \prod_{j=1}^{n} g_j^{v'_j})^e} \tag{2}$$

$$= g^{(\sum_{i=1}^{\mu} s_i \alpha_i - s'e)} \prod_{j=1}^{m} h_j^{(\sum_{i=1}^{\mu} u_j^{(i)} \alpha_i - u'_j e)} \prod_{j=1}^{n} g_j^{(\sum_{i=1}^{\mu} v_j^{(i)} \alpha_i - v'_j e)} \tag{3}$$

$$= g^s \prod_{j=1}^{m} h_j^{u_j} \prod_{j=1}^{n} g_j^{v_j} \tag{4}$$

which shows correctness as desired. Above, equation (2) is justified by that each σ_i is valid, and equation (4) follows from the definition of s' and $w' = (u', v')$ as computed in the $\mathsf{Combine}$ algorithm.

4.1 Efficiency

Each signature consists of an element of $\mathbb{Z}_N$ and one integer of λ bits. Signing costs one full exponentiation and one multi-exponentiation in $\mathbb{Z}_N$ with λ-bits exponents, plus the sampling of a random prime number (which is dominated by the cost of prime verification). The verification needs an exponentiation with a $(\lambda+1)$-bits prime, x^e, and one multi-exponentiation with λ-bits exponents.

4.2 Proof of security

Theorem 2. *Under the Strong-RSA assumption, the scheme described above is a secure homomorphic network coding signature.*

Proof. Let $\mathcal{A}$ be an efficient adversary against the security of the scheme. This means that, with non negligible probability, $\mathcal{A}$ is able to produce a valid forgery $\sigma^* = (s^*, x^*)$ which is valid for a vector w^* and a file identifier fid^*. We show how to build an efficient adversary $\mathcal{B}$ that "uses" it to break the strong RSA assumption (for the case when the challenge is a quadratic residue). Let t be the maximum number of signatures asked by $\mathcal{A}$. We denote with σ_j the signature produced for vector space $\mathcal{W}_j$ and $e_j = \mathsf{fid}_j$ be the used file identifier. If one considers e^* and the set $\{e_1, \ldots, e_t\}$ it is possible distinguish two types of forgeries:

Type I the adversary outputs a signature containing an e^* such that $e* \nmid \prod_{i=1}^{t} e_i$,
Type II the adversary outputs a signature containing an e^* such that $e* \mid \prod_{i=1}^{t} e_i$.

At the beginning of the game we guess on the type of forgery will be provided by $\mathcal{A}$ in order to set up an appropriate simulation accordingly. This guess will be right with probability at least $1/2$.

Type I. $\mathcal{B}$ takes as input (N, τ) where N is the product of two safe primes p, q (where $p = 2p' + 1$ and $q = 2q' + 1$) and $\tau \in QR_N$. The goal here is to find an e-th root y of τ for e of $\mathcal{B}$'s choice.

In the following we describe the simulator $\mathcal{B}$ during the three phases of the simulation.

Key Generation $\mathcal{B}$ chooses randomly chooses t random file identifiers $e_1, \ldots, e_t$ of the appropriate length and it generates the public key as follows:

- pick random $\alpha_0, \alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \xleftarrow{\$} \{1, \ldots, N^2\}$
- let $E = \prod_{i=1}^{t} e_i$ and set $g = \tau^{E\alpha_0}$, $g_i = g^{\alpha_i}$ for all $i = 1$ to n and $h_i = g^{\beta_i}$ for all $i = 1$ to m.

Finally $\mathcal{B}$ gives $\mathsf{vk} = (N, g, h_1 \ldots, h_m, g_1, \ldots, g_n)$ to $\mathcal{A}$

For all $1 \leq i \leq n$ let $\alpha_i = b_i p'q' + c_i$ where $0 \leq c_i < p'q'$. Since each α_i is chosen from a suitably large interval, the distributions of each $(\alpha_i \bmod p'q')$ is statistically indistinguishable from the uniform distribution over $\mathbb{Z}_{p'q'}$. So $g_1, g_2, \ldots, g_n$ are distributed like random quadratic residues of $\mathbb{Z}_N^*$. Moreover the conditional distribution of b_i given c_i is statistically indistinguishable from the uniform distribution over $\{0, \ldots, \lfloor N^2/p'q' \rfloor\}$. The same argument applies to g and all the h_i's.

Signing queries At this stage $\mathcal{A}$ is allowed to adaptively query signatures on vector spaces $\mathcal{W}_j$ (meeting the requirement specified above) of its own choice. In particular, each $\mathcal{W}_j$ is described by a properly augmented basis and it has to be answered with m signatures, a signature for each vector of the basis. Therefore, $\mathcal{B}$ has to give all these signatures to $\mathcal{A}$.

For ease of exposition, let us slightly abuse notation by denoting with $\mathcal{W}$, for all $k \in \{1, \ldots, t\}$, the k-th queried vector space. By these positions each signature query is managed as follows.

Let $\mathcal{W} = (w^{(1)}, \ldots, w^{(m)})$ such that for $i = 1$ to m $w^{(i)} = (u^{(i)}, v^{(i)})$ where $u^{(i)}$ is the i-th unitary vector. The value s_i is chosen at random in $\mathbb{Z}_{e_k}$. Next, $\mathcal{B}$ computes the solution of each $x_i^{e_k} = g^{s_i} \cdot \prod_{j=1}^{m} h_j^{u_j^{(i)}} \cdot \prod_{j=1}^{n} g_j^{v_j^{(i)}}$ as follows:

- let $E_k = \prod_{j=1, j\neq k}^{t} e_j$
- $\forall i = 1, \ldots, m : x_i = (\tau^{E_k})^{\alpha_0 s_i + \sum_{j=1}^{m} u_j^{(i)} \beta_j + \sum_{j=1}^{n} v_j^{(i)} \alpha_j}$

Finally $\mathcal{B}$ gives $(e_k, \{s_i, x_i)\}_{i=1}^{m})$ to $\mathcal{A}$. It is easy to see that all the (s_i, x_i) are valid signatures for the respective vectors $w^{(i)}$, and that they are distributed as in the real case.

Challenge Once the previous phase is over, $\mathcal{A}$ is supposed to output a forgery (e^*, w^*, σ^*) where $\sigma^* = (s^*, x^*)$ is valid with respect to the vector w^* and the identifier e^*. By definition of valid forgery it has to be the case that

$$x^{e^*} = g^{s^*} \cdot \prod_{j=1}^{m} h_j^{u_j^*} \cdot \prod_{j=1}^{n} g_j^{v_j^*} = \tau^{E(\alpha_0 s^* + \sum_{j=1}^{m} u_j^* \beta_j + \sum_{j=1}^{n} v_j^* \alpha_j)}.$$

Let $E' = E(\alpha_0 s^* + \sum_{j=1}^{m} u_j^* \beta_j + \sum_{j=1}^{n} v_j^* \alpha_j)$ and $d = gcd(e^*, E')$. Provided that $e^* \nmid E'$, $\mathcal{B}$ can use standard techniques (i.e. Shamir's trick) to extract an (e^*/d)-th root y of τ and thus it can output $(e^*/d, y)$ to break Strong-RSA.

To conclude this part of the proof we are left with the task of showing that $e^* \nmid E'$ with non-negligible probability. Since all the e exponents are primes and we are assuming a Type I forgery, it has to be the case that $e^* \nmid E$. So, it remains to show that $e^* \nmid (\alpha_0 s^* + \sum_{j=1}^{m} u_j^* \beta_j + \sum_{j=1}^{n} v_j^* \alpha_j)$ with non-negligible probability.

As pointed out before, we set $\alpha_i = b_i p' q' + c_i$. Since each b_i is information theoretically hidden to $\mathcal{A}$, e^* might depend only on the c_i's (the same holds for the β_i's). Moreover, as $e^* \nmid p'q'$ the probability that $e^* \mid (\alpha_0 s^* + \sum_{j=1}^{m} u_j^* \beta_j + \sum_{j=1}^{n} v_j^* \alpha_j)$, or equivalently $(\alpha_0 s^* + \sum_{j=1}^{m} u_j^* \beta_j + \sum_{j=1}^{n} v_j^* \alpha_j) = 0 \bmod e^*$, is close to $1/e^*$. This means that $e^* \nmid E'$ with probability close to $1 - 1/e^*$.

Type II. This encompasses the case when $\mathcal{A}$ "reuses" some previously seen exponent when producing its forgery. This is because, being all the e_i's primes, the fact that $e* \mid \prod_{i=1}^{t} e_i$, implies that $e^* = e_k$ for some k. Again, let e^*, w^*, σ^* be the forgery provided by the adversary, where $\sigma^* = (s^*, x^*)$ and $w^* = (u^*, v^*)$. Since in this case we assume that $e^* = e_k$, i.e., $\mathsf{fid}^* = \mathsf{fid}_k$, it has to be the case that w^* is not in $\mathcal{W}_k$ (type-II forgery).

For ease of exposition, let us abuse notation and denote with $\mathcal{W} = (w^{(1)}, \ldots, w^{(m)})$ the k-th vector space $\mathcal{W}_k$ asked by $\mathcal{A}$. Since $w^* \notin \mathcal{W}$, it has to be $w^* - \sum_{i=1}^{m} w^{(i)} u_i^* \neq 0^{m+n} \bmod e^*$. In particular, this means that there must exist an index $\nu \in \{1, \ldots, n\}$ such that $z_\nu = v_\nu^* - \sum_{i=1}^{m} v_\nu^{(i)} u_i^* \neq 0 \bmod e^*$. In what follows we will require the simulator to guess both such index k and position ν. Thus, $\mathcal{B}$'s guess will be correct with probability $1/tn$. Now, if we consider the forgery provided by the adversary and the values $x_1, \ldots, x_m$ obtained from the signatures of the vector space $\mathcal{W}_k$ generated by the simulator, we distinguish two additional sub-cases:

(a) $x^* = \prod_{j=1}^{m} x_j^{u_j^*} \bmod N$

(b) $x^* \neq \prod_{j=1}^{m} x_j^{u_j^*} \bmod N$

We provide different simulations for the two cases. In particular we describe Type-II.b first.

Type-II.b We describe a simulator $\mathcal{B}$ that solves Strong RSA for the case of Type-II.b forgeries.

Key Generation. $\mathcal{B}$ chooses $e_1, \ldots, e_t$ at random. Next, $\mathcal{B}$ picks random $y_1, \ldots, y_n \xleftarrow{\$} QR_N$, $\alpha_\nu, \omega_1, \ldots, \omega_m, \beta_1, \ldots, \beta_m \xleftarrow{\$} \{1, \ldots, 2N\}$.
Let $E = \prod_{i=1}^{t} e_i$ and $E_k = \prod_{i=1, i\neq k}^{t} e_i$. $\mathcal{B}$ proceeds by creating the public key as follows. $g = \tau^{E_k}$, $g_\nu = g^{\alpha_\nu}$, $g_i = y_i^E \ \forall i = 1, \ldots, n$ and $i \neq \nu$, $h_i = g^{e_k\omega_i - \beta_i} \ \forall i = 1, \ldots, m$. Finally it gives the public key to $\mathcal{A}$. It is easy to get convinced that the distribution of the so generated public key is statistically close to that of a "true" public key.

Signing queries $\mathcal{B}$ answers $\mathcal{A}$'s signature queries as follows.
Let $\mathcal{W}$ be the i-th queried vector space. For all $i \in \{1, \ldots, t\}\backslash\{k\}$ let e_i be the used file identifier. For $j = 1$ to m, $\mathcal{B}$ chooses random $s_j \in \mathbb{Z}_{e_i}$ and sets

$$x_j = (\tau^{\prod_{l\neq k, i} e_l})^{s_j + e_k\omega_j - \beta_j + \alpha_\nu v_\nu^{(j)}} \left(\prod_{l=1, l\neq\nu}^{m} y_l^{v_l^{(j)}} \right)^{E_i}.$$

It is easy to verify that x_j is such that $x_j^{e_i} = g^{s_j} \cdot \prod_{l=1}^{m} h_l^{u_l^{(j)}} \cdot \prod_{l=1}^{n} g_l^{v_l^{(j)}}$.

For $i = k$ a different machinery is required. Let $\mathcal{W} = (w^{(1)}, \ldots, w^{(m)})$ be the k-th queried vector space where $w^{(i)} = (u^{(i)}, v^{(i)})$. For $j = 1$ to m, $\mathcal{B}$ sets $s_j = \beta_j - \alpha_\nu v_\nu^{(j)} \bmod e_i$, $\omega_j = (\beta_j - \alpha_\nu v_\nu^{(j)} - s_j)/e_k$ and computes:

$$x_j = \tau^{E_k\omega_j} \cdot \left(\prod_{l\neq\nu} y_l^{v^{(j)}{}_l} \right)^{E_k} = \sqrt[e_k]{g^{s_j} \cdot h_j \cdot \prod_{l=1}^{n} g_l^{v_l^{(j)}}}.$$

Finally $\mathcal{B}$ provides all the m signatures (s_j, x_j) generated above to $\mathcal{A}$. Notice that such signatures follows a distribution which is statistically close with respect to that that would have been produced by a genuine signer.

Challenge In this phase $\mathcal{A}$ will output a type-II forgery (defined by (e^*, w^*, s^*, x^*)). We show that $\mathcal{B}$ can extract an e^*-th root of τ as follows.
First, let

$$\begin{aligned} x_1^{e_k} &= g^{s_1} \prod_{j=1}^{m} h_j^{u_j^{(1)}} \prod_{j=1}^{n} g_j^{v_j^{(1)}} \\ &\vdots \\ x_m^{e_k} &= g^{s_m} \prod_{j=1}^{m} h_j^{u_j^{(m)}} \prod_{j=1}^{n} g_j^{v_j^{(m)}} \end{aligned} \tag{5}$$

denote the verification equations arising from $\mathcal{B}$'s signatures on vector space $\mathcal{W} = \mathcal{W}_k$. Combining them with the received forgery one gets:

$$\begin{aligned} \left(\frac{x^*}{\prod_{j=1}^{m} x_j^{u_j^*}} \right)^{e^*} &= g^{(s^* - \sum_{l=1}^{m} u_l^* s_l)} \prod_{j=1}^{m} h_j^{(u_j^* - \sum_{l=1}^{m} u_l^* u_j^{(l)})} \prod_{j=1}^{n} g_j^{(v_j^* - \sum_{l=1}^{m} u_l^* v_j^{(l)})} \\ &= (\tau^{E_k})^{(s^* - \sum_{l=1}^{m} u_l^* s_l) + \alpha_\nu z_\nu} \left(\prod_{j=1, j\neq\nu}^{n} y_j^{z_j} \right)^E \end{aligned}$$

where the second equality comes from the fact that, for simplicity, we set $z_j = v_j^* - \sum_{l=1}^m u_l^* v_j^{(l)} \bmod e^*$, and $\prod_{j=1}^m h_j^{(u_j^* - \sum_{l=1}^m u_l^* u_j^{(l)})} = 1$. Thus we can rewrite the equation above as

$$\left[\left(\frac{x^*}{\prod_{j=1}^m x_j^{u_j^*}} \right) \left(\prod_{j=1, j\neq\nu}^n y_j^{-z_j} \right)^{E_k} \right]^{e^*} = \tau^{E_k(s^* - \sum_{l=1}^m u_l^* s_l + \alpha_\nu z_\nu)}.$$

Let $E' = E_k(s^* - \sum_{l=1}^m u_l^* s_l + \alpha_\nu z_\nu)$. In order to extract a root of τ we have to show that $e^* \nmid E'$ with non-negligible probability. Observe that $e^* \nmid E_k$ and that $\alpha_\nu = bp'q' + c$ where $b \in \{0, 1\}$ (with probability close to 1) and b is information theoretically hidden to the adversary. We show that $\Pr[e^* \nmid (s^* - \sum_{l=1}^m u_l^* s_l + \alpha_\nu z_\nu)]$ is at least $1/2$. To see this, assume by contradiction that $\Pr[e^* \mid (s^* - \sum_{l=1}^m u_l^* s_l + \alpha_\nu z_\nu)]$ is non-negligibly higher than $1/2$. Then it must be that $e^* \mid (s^* - \sum_{l=1}^m u_l^* s_l + cz_\nu)$ and $e^* | (s^* - \sum_{l=1}^m u_l^* s_l + (\phi(N) + c)z_\nu)$, which implies that $e^* | z_\nu \phi(N)$. Since $z_\nu \in \mathbb{Z}_{e^*}$ and e^* is prime, e^* must be a non trivial factor of $\phi(N)$.
Therefore, $e^* \nmid E'$ with probability at least $1/2$ and in this case $\mathcal{B}$ can use standard techniques (i.e., Shamir's trick) to extract an (e^*/d)-th root y of τ where $d = gcd(e^*, E')$.

Type-II.a For the case of Type-II.a forgeries the simulator performs basically the same Setup and Signing queries phases as in the Type-I simulation. The only difference here is that in the setup we set $g_\nu = \tau^E$. Once a forgery is provided (e^*, w^*, s^*, x^*) being it a Type-II.a one, we have that

$$g^{s^*} \prod_{j=1}^m h_j^{u_j^*} \prod_{j=1}^n g_j^{v_j^*} = g^{\sum_{l=1}^m s_l u_l^*} \prod_{j=1}^m h_j^{\sum_{l=1}^m u_j^{(l)} u_l^*} \prod_{j=1}^n g_j^{\sum_{l=1}^m v_j^{(l)} u_l^*}$$

This leads to the following

$$\tau^{E(\alpha_0(s^* - \sum_{j=1}^m s_j u_j^*) + \sum_{j=1, j\neq\nu}^n \alpha_j z_j + z_\nu)} = 1 \bmod N$$

where, again, z_i is $v_i^* - \sum_{j=1}^m v_i^{(j)} u_j^* \bmod e^*$. Let $\gamma = (\alpha_0(s^* - \sum_{j=1}^m s_j u_j^*) + \sum_{j=1, j\neq\nu}^n \alpha_j z_j + z_\nu)$. Notice that each $\alpha_j = b_j p'q' + c_j$ where b_j is information theoretically hidden to the adversary and that $z_\nu \neq 0 \bmod e^*$ (this is the simulator's guess). Therefore, with non-negligible probability we have an integer $E\gamma \neq 0$ such that $E\gamma = 0 \bmod \phi(N)$, that allows to factor and thus to trivially solve the Strong RSA problem. □

5 Efficiency and Comparisons

In this section we discuss the efficiency of our two constructions compared to that of other known homomorphic network coding signatures. As we already mentioned, there are no that many schemes in the literature realizing this primitive: a few constructions [7, 13, 9, 8] rely on random oracles, and a couple of more recent schemes [3, 10, 12] are proven secure in the standard model. We should also mention that there are other schemes in the standard model based on homomorphic hashing. However these are less appealing in practice mainly because the basis vectors have to be signed all at once, which means that in the network coding application the source node must know the entire file before sending the first packet, which is not desirable in several applications (think of a source node which is a sensor collecting data in some time interval). Moreover, the authentication information to be sent along with the packets is quite long.

Therefore, we compare our constructions with the schemes in the standard model, and later in this section we will briefly discuss a comparison with the random oracle based ones.

In the scheme by Attrapadung and Libert [3] a signature consists of three group elements where the bilinear groups have composite order N, with N product of three primes. To compute a signature, the scheme needs to perform two multi-exponentiations and one exponentiation, whereas the verification time is dominated by the computation of four pairings in such composite order groups. Even if one applies standard techniques to convert the scheme in prime order groups (as suggested in [3]), the overhead would still remain significant.

In [12] Freeman proposes a general framework, that can be seen as a generalization of the Attrapadung and Libert methodology, for converting signature schemes with certain properties into linearly homomorphic ones. The main appeal of Freeman's construction are two. First, his model allows for a stronger adversary than that considered here. Moreover, the proposed approach is general enough to work with several currently known signature schemes. However all the resulting (linearly homomorphic) signatures are less efficient than those given in this paper.

In the scheme by Catalano, Fiore and Warinschi [10] each signature consists of an element of $\mathbb{Z}_N^*$ and an integer s of $\lambda_s = 3k+|N|$ bits, where k is the security parameter and $|N|$ is the bit size of the RSA modulus N (which is related to k). Signing and verifying both need one multi-exponentiation (where all exponents have size λ, except one of size λ_s) and one exponentiation. Since in this scheme the linear combinations are done over the integers, it can support only a limited number of linear combinations, that in the network coding application translates to supporting only networks with paths of predetermined bounded length. Technically, the reason of such bound is that the vector coordinates cannot be let grow more than the size of the prime e.

In this scenario, our solution based on q-SDH seems the most efficient one both in terms of bandwidth and computation. In fact, recall that in our case a signature is one group element plus one element of $\mathbb{Z}_p$: 512 bits in total, if one considers $k = 128$ bits of security and asymmetric pairings. The operations for signing and verifying are similar in all the schemes, but our SDH construction has the advantage that such operations can be performed over prime order groups. Our RSA realization, can be seen as a significant optimization of the Catalano-Fiore-Warinschi's scheme [10]. The improvements are mainly two. First, our scheme allows for a much smaller exponent s. In fact, in our case s can be of λ bits, that is even more than 10 times shorter than in [10], if one considers 128 bits of security. Intuitively, the reason of using a large s in [10] is that in the real scheme s is truly random, while in the simulation it is used to hide some information of $2k + |N|$ bits, which decreases its entropy down to k bits. So, there s is taken sufficiently large so as to keep it within negligible statistical distance from a uniform value of λ_s bits. In our case, s is in $\mathbb{Z}_e$, and we take advantage of modular reduction to obtain a uniformly distributed s also in the simulation. Notice that having such a short s saves in both bandwidth and computation. Second, our idea of computing all the linear combinations $\pmod e$ avoids the problem that the vector coordinates may grow beyond e. In this way we can support networks with paths of any lengths, which was not the case in the previous RSA-based schemes [10] and [13].

Finally, we consider the schemes in the random oracle model that work over similar algebraic settings, i.e., bilinear groups [7] and RSA [13]. Compared to them, our solutions are (not surprisingly) slightly worse. The main difference is the size of the public key that in our case is linear in $m + n$, whereas in [7, 13] it is constant (because $O(m + m)$ group elements are generated on-the-fly using the random oracle). On the other hand, the size of a signature and the time needed to

sign and verify are somewhat comparable. In this sense, we believe that our solutions offer a good compromise if one does not want to rely on the random oracle heuristic.

Acknowledgements The work described in this paper has been supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. The authors would like to thank Dennis Hofheinz and Eike Kiltz for helpful discussions in the early stage of this work.

References

1. R. Ahlswede, Ning-Cai, S. Li, and R.W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.
2. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat, and Brent Waters. Computing on authenticated data. To appear at TCC 2012. Also in `http://eprint.iacr.org/2011/096`.
3. 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, Taormina, Italy, March 6–9, 2011. Springer, Berlin, Germany.
4. Niko Bari and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494, Konstanz, Germany, May 11–15, 1997. Springer, Berlin, Germany.
5. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.
6. Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.
7. 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, Irvine, CA, USA, March 18–20, 2009. Springer, Berlin, Germany.
8. 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, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.
9. 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, Taormina, Italy, March 6–9, 2011. Springer, Berlin, Germany.
10. 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, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.
11. P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In *41st Allerton Conference on Communication, Control and Computing*, 2003.
12. David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. PKC 2012. Cryptology ePrint Archive `http://eprint.iacr.org/2012/060`.
13. 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, Paris, France, May 26–28, 2010. Springer, Berlin, Germany.
14. C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *Proc. of IEEE INFOCOM 2005*, pages 2235–2245, 2005.
15. T. Ho, R. Koetter, M. Médard, D. Karger, and M. Effros. The benefit of coding over routing in a randomized setting. In *Proc. of International Symposium on Information Theory (ISIT)*, page 442, 2003.
16. T. Ho, B. Leong, R. Koetter, M. Médard, M. Effros, and D. Karger. Byzantine modification detection in multicast networks using randomized network coding. In *Proc. of International Symposium on Information Theory (ISIT)*, pages 144–152, 2004.

17. T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52:4413–4430, 2006.
18. Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 21–38, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Berlin, Germany.
19. S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Médard, and M. Effros. Resilient network coding in the presence of byzantine adversaries. *IEEE Transactions on Information Theory*, 54:2596–2603, 2008.
20. 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, San Jose, CA, USA, February 18–22, 2002. Springer, Berlin, Germany.
21. M. Krohn, M. Freedman, and D. Mazieres. On the-fly verification of rateless erasure codes for efficient content distribution. In *2004 IEEE Symposium on Security and Privacy*, pages 226–240, Berkeley, California, USA, May 9–12, 2004. IEEE Computer Society Press.
22. Shuo-Yen Robert-Li, Raymond Y. Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003.
23. Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Berlin, Germany.