Secure Network Coding Over the Integers*

Rosario Gennaro[1], Jonathan Katz[2**], Hugo Krawczyk[1], and Tal Rabin[1]

[1] IBM T.J. Watson Research Center, Hawthorne, NY
{rosario,talr}@us.ibm.com, hugo@ee.technion.ac.il
[2] Department of Computer Science, University of Maryland
jkatz@cs.umd.edu

Abstract. Network coding offers the potential to increase throughput and improve robustness without any centralized control. Unfortunately, network coding is highly susceptible to "pollution attacks" in which malicious nodes modify packets improperly so as to prevent message recovery at the recipient(s); such attacks cannot be prevented using standard end-to-end cryptographic authentication because network coding mandates that intermediate nodes modify data packets in transit.

Specialized "network coding signatures" addressing this problem have been developed in recent years using homomorphic hashing and homomorphic signatures. We contribute to this area in several ways:

- We show the first homomorphic signature scheme based on the RSA assumption (in the random oracle model).
- We give a homomorphic hashing scheme that is more efficient than existing schemes, and which leads to network coding signatures based on the hardness of factoring (in the standard model).
- We describe variants of existing schemes that reduce the communication overhead for moderate-size networks, and improve computational efficiency (in some cases quite dramatically – e.g., we achieve a 20-fold speedup in signature generation at intermediate nodes).

Underlying our techniques is a modified approach to random linear network coding where instead of working in a vector space over a *field*, we work in a module over the *integers* (with small coefficients).

1 Introduction

Network coding [2, 18] offers an alternative, decentralized approach to traditional multicast routing. We consider a network setting where a *source node* has a file that it wants to distribute to a set of *target nodes*. The source partitions the file into m packets which it transmits to its neighboring nodes. Further transmission happens through intermediate nodes who receive packets via incoming links and produce modified packets sent over outgoing links. These outgoing packets are

* This work was supported by the US Army Research Laboratory and the UK Ministry of Defence under agreement number W911NF-06-3-0001.

** Work done while visiting IBM, and supported also by NSF CAREER award #0447075, NSF grant #0627306, and the US DoD/ARO MURI program.

computed as *linear combinations* of incoming packets, where packets are viewed as vectors in a vector space over some field. (See further discussion in Section 2.1.) We focus on the case of *random linear network coding* [10, 13], where scalars are chosen by each intermediate node *at random* from the underlying field. This strategy induces a fully decentralized solution to the routing problem since nodes do not need to coordinate their actions.

Target nodes reconstruct the original file sent by the source using the packets they receive. This can be done if the intermediate nodes *augment* each vector they send with m additional *coding coordinates* that encode the linear combination that resulted in that vector. A target that receives a set of augmented vectors for which the coding coordinates induce a full rank matrix can recover the file sent by the source via simple matrix inversion. (See Section 2.1.) A fundamental question is: what is the *decoding probability* at the targets; i.e., what is the probability with which a target is able to reconstruct the original file? The network coding literature shows that small-size fields (e.g., $\mathbb{F}_{2^8}$) provide good decoding probability for sufficiently connected networks.

Although network coding can increase throughput and reliability relative to alternative techniques, it is susceptible to *pollution attacks* in which malicious nodes inject invalid packets that prevent reconstruction of the file at the targets. (An invalid packet is any packet that is not in the linear span of the original augmented vectors sent by the source.) Due to the way vectors are propagated and combined in the network, a single invalid packet injected by an attacker can invalidate many more packets further downstream. This constitutes a serious denial of service attack which can be mounted effortlessly.

Two naive solutions to this problem are easily seen to be inapplicable. Having the source sign the file prevents a target node from reconstructing an incorrect file, but does not enable the target to efficiently reconstruct the correct file in the first place. (Moreover, it does not provide any way for intermediate nodes to drop invalid packets they receive.) Having the source sign each augmented vector it sends (using a standard signature scheme) is also of no help, since intermediate nodes are *supposed* to modify vectors in transit. Prior work has shown, however, that dedicated *network coding signatures* can be used to address pollution attacks. Such signatures have been based on two primitives: *homomorphic hash functions* [17, 21] or *homomorphic signatures* [16, 7, 6]. In both cases, homomorphic properties ensure that the signature (or hashing) operation on a linear combination of vectors results in a corresponding homomorphic combination of signatures (or hash values). See Section 2.2 for further details.

Constructions of homomorphic hash functions are well known, and can be implemented over any prime-order group where the discrete logarithm problem is hard. Building homomorphic signatures is more challenging. So far the only known construction is based on bilinear groups [6] and involves costly pairing operations. In particular, network coding signatures based on homomorphic signatures are computationally more expensive than those built from homomorphic hashing. However, the latter are less communication-efficient since they require each packet transmitted to be sent along with some "authentication data" whose

length is proportional to m (the number of file vectors). One drawback of both approaches is that they replace the small fields used in "standard" network coding with very large fields appropriate for cryptography. For example, instead of using vectors over an 8-bit field as in traditional network coding, the cryptographic approaches use vectors over a 160-bit field instead. This increases both the communication and computational overhead.

Our Contributions. We present new and improved network coding signatures. First, we show the first homomorphic signature scheme based on the RSA assumption in the random oracle model.[1] In particular, it offers more efficient processing at the intermediate nodes as compared to the scheme of [6] that is based on bilinear groups and pairings. The bandwidth overhead is also lower for networks of moderate size (e.g., where the maximum path length between source and target nodes is 20–30 hops).

We also present a new homomorphic hashing scheme which is quite efficient. Treating each information vector v as a single (large) integer, we define our hash function simply as $H_N(v) = 2^v \bmod N$ for a composite N. This hash function is homomorphic over the integers and can be proven collision resistant based on the hardness of factoring. This constructions leads to a network coding signature scheme based on the factoring assumption and without random oracles.

A core technique we use for both the above constructions is to apply network coding in a module over the *integers* rather than in a vector space over a *field* as is traditionally done. By working over the integers we enable the homomorphic properties of the above two schemes (where the group order is unknown), and furthermore can work with *small coefficients* (that need not be cryptographically large). This has the immediate effect of improving the computation at intermediate nodes, and it also reduces the total bandwidth overhead for networks with moderate-length paths between source and targets.

We must analyze how this change from working over a field to working over the integers affects the decoding probability. We show that if the integer coefficients are taken from a set $Q = \{0, \ldots, q-1\}$ for prime q, then the decoding probability is at least as good as working over the field $\mathbb{F}_q$; thus we conclude that using 8-bit coefficients is good enough for most applications.

The ability to perform with network coding with small integer coefficients allows us also to improve the performance of existing schemes. We show that by choosing coefficients from a small set Q as above (but still performing computations modulo the large prime p as required by prior schemes) we can significantly improve performance: e.g., we obtain roughly a 20-fold improvement in signature generation time at intermediate nodes and a reduction in the communication overhead as well.

[1] Yu et al. [20] recently proposed an RSA-based homomorphic signature scheme, but their scheme is essentially flawed (e.g., no signature, even one produced by an honest source, ever passes verification). The problem is that Yu et al. *incorrectly* assume (cf. equations (11) and (12) in Section III-B of [20]) that for integers A, b, d, a prime e, and RSA composite N, it holds that $(A^b \bmod e)^d \bmod N = (A \bmod e)^{bd} \bmod N$.

Organization. Section 2 reviews network coding and existing network coding signature schemes. In Section 3, we discuss network coding over the integers and show how this translates into performance improvements for existing network coding signature schemes. We present our RSA-based homomorphic signature scheme in Section 4, and our factoring-based homomorphic hashing scheme in Section 5.

2 Background

2.1 Network Coding

We present a high-level description of *linear* network coding (the only type with which we are concerned in this work); for further details see [12]. In this setting, we have a network with a distinguished node S, called the *source*, and a subset of nodes known as *targets*. The objective is for S to transmit a *file* $\bar{F}$ to all the target nodes, where $\bar{F}$ is represented as a matrix containing the m (row) vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)} \in \mathbb{F}^n$ over some finite field $\mathbb{F}$.

The source first creates m *augmented vectors* $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$ defined as

$$\bar{w}^{(i)} = (\overbrace{\underbrace{0, \ldots, 0, 1}_{i}, 0, \ldots, 0}^{m} \parallel \bar{v}^{(i)}) \in \mathbb{F}^{m+n};$$

i.e., each original vector $\bar{v}^{(i)}$ of the file is pre-pended with the vector of length m containing a single '1' in the ith position. These augmented vectors are sent by the source to its neighboring nodes.

Each (well-behaved) intermediate node I in the network processes packets (i.e., incoming vectors) as follows. Upon receiving packets $w^{(1)}, \ldots, w^{(\ell)} \in \mathbb{F}^{m+n}$ on its ℓ incoming communication edges, I computes a packet w for each of its outgoing links as a linear combination of the packets that it received. That is, each outgoing packet w transmitted by I takes the form $w = \sum_{i=1}^{\ell} \alpha_i w^{(i)}$, where $\alpha_i \in \mathbb{F}$. We say a vector w transmitted in the network (in the scenario above) is *valid* if it lies in the linear span of the original augmented vectors $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$. It is easy to see that if all nodes follow the protocol honestly, then every packet transmitted in the network is valid.

Different strategies for choosing the coefficients α_i yield different variants of network coding. When the $\{\alpha_i\}$ are chosen randomly and independently by each intermediate node, for each of its outgoing communication links, the resulting scheme is referred to as *random* linear network coding [8, 10, 13]. When analyzing efficiency, we assume random linear network coding is used; our constructions, however, ensure security regardless of how the coefficients are chosen.

To recover the original file, a target node must receive m (valid) vectors $\{w^{(i)} = (u^{(i)} \| v^{(i)})\}_{i=1}^{m}$ for which $u^{(1)}, \ldots, u^{(m)}$ are linearly independent. If we define a matrix U whose rows are the vectors $u^{(1)}, \ldots, u^{(m)}$ and a matrix V

whose rows are the vectors $v^{(1)}, \ldots, v^{(m)}$, the original file can be recovered as

$$\bar{F} = U^{-1}V. \tag{1}$$

Assuming the coefficients are chosen randomly and independently by the intermediate nodes, the *decoding probability* — i.e., the probability with which a given target node will be able to recover the file (or, equivalently, the probability with which a given target node will receive m linearly independent vectors, in the sense required above) — is determined by the network topology and the size of the field $\mathbb{F}$. To minimize the communication overhead (due to the first m coordinates of every transmitted vector), it is desirable to keep $|\mathbb{F}|$ as small as possible; on the other hand, choosing $|\mathbb{F}|$ too small would reduce the decoding probability too much. For typical networks encountered in practice, taking $|\mathbb{F}| \approx 256$ has been shown to give a decoding probability of better than 99%.

2.2 Network Coding Signatures

We have already discussed the problem of pollution attacks, and why standard cryptographic mechanisms are incapable of preventing them. Early efforts to deal with pollution attacks focused on *information-theoretic* solutions [11, 14, 15] that use error-correction techniques to ensure that targets can reconstruct the file as long as the ratio of valid to invalid vectors they receive is sufficiently high. Unfortunately, these techniques (inherently) impose limitations on the number of nodes the adversary can corrupt, the number of packets that can be modified, and/or the number of links on which the adversary can eavesdrop. Researchers have more recently turned to *cryptographic* approaches that place no restrictions on the adversary (other than assuming that the adversary is computationally bounded) [17, 7, 21, 6]. These approaches give *network coding signature schemes* that allow anyone holding the public key[2] of the source to determine whether a given vector is valid. This allows target nodes to reject invalid vectors before reconstructing the file; it also allows intermediate nodes to filter out invalid vectors when generating their outgoing messages. For formal definitions of network coding signatures and their security requirements, see [6].

Two classes of network coding signature schemes are known: those based on *homomorphic hashing*, and those using *homomorphic signatures*. We describe these now at a high level.

Schemes based on homomorphic hashing [17, 21, 6]. A homomorphic hash function H is a collision-resistant hash function with the property that for any vectors a, b and scalars α, β it holds that $H(\alpha a + \beta b) = H(a)^{\alpha} H(b)^{\beta}$. Collision resistance implies (via standard arguments) that if one knows vectors a, b, c for which $H(c) = H(a)^{\alpha} H(b)^{\beta}$ then it must be the case that $c = \alpha a + \beta b$.

A concrete example [17] of a homomorphic hash function is given by what we call the *exponential homomorphic hash (EHH)* scheme. Let $\mathbb{G}$ be a cyclic group

[2] A symmetric-key analogue is also possible [9, 1], but this allows only a (single) target to verify validity of vectors.

of order p, and let the public key contain random generators $g_1, \ldots, g_n \in \mathbb{G}$. Define a function $\mathbf{H}$ on vectors $v = (v_1, \ldots, v_n) \in \mathbb{Z}_p^n$ as

$$\mathbf{H}(v) = \prod_{j=1}^{n} g_j^{v_j}. \tag{2}$$

The homomorphic property is easily verified, and collision resistance is implied by the discrete logarithm assumption in $\mathbb{G}$.

Homomorphic hash functions can be used for network coding as follows. For each original vector $\bar{v}^{(i)}$, the source S computes $h_i = H(\bar{v}^{(i)})$; it then signs $(h_1, \ldots, h_m)$ (together with a unique file identifier fid) using a standard signature scheme. The $\{h_i\}$ and their signature are then appended to every packet sent in the network.[3] A node can determine whether a vector $w = (u \parallel v)$ is valid by checking the signature on the $\{h_i\}$ (and the fid), and then verifying whether $\prod_{i=1}^{m} h_i^{u_i} \stackrel{?}{=} H(v)$. In particular, for the EHH scheme $h_i = \mathbf{H}(\bar{v}^{(i)})$ and verification takes the form:

$$\prod_{i=1}^{m} h_i^{u_i} \stackrel{?}{=} \mathbf{H}(v) \stackrel{\text{def}}{=} \prod_{j=1}^{n} g_j^{v_j}. \tag{3}$$

The resulting network coding signature scheme can be proven secure without random oracles based on the discrete logarithm assumption [17, 6].

When using homomorphic hashing, the only change in the processing done by intermediate nodes is to verify the hash and forward the authentication information. However, the linear network coding operations performed by intermediate nodes are now done over the (large) field $\mathbb{F} = \mathbb{Z}_p$.

Homomorphic signature schemes [16, 7, 6]. Here, the full signature (and not just the hash) is homomorphic. Namely, the signature scheme has the property that for any vectors a, b and scalars α, β, it holds that $\mathsf{Sign}(\alpha a + \beta b) = \mathsf{Sign}(a)^{\alpha}\mathsf{Sign}(b)^{\beta}$. The security property, roughly speaking, is that given the signatures of vectors $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$ it is *only* feasible to generate signatures on vectors in the linear span of $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$. The application to network coding is immediate: The source S signs each augmented vector $\bar{w}^{(i)}$ and transmits each $\bar{w}^{(i)}$ together with its signature $\mathsf{Sign}(\bar{w}^{(i)})$. An intermediate node I that receives a set of incoming vectors with their corresponding signatures will (i) verify the signatures (discarding any vector whose signature is invalid) and (ii) compute (using the homomorphic property) a valid signature on each outgoing vector that it generates. Thus, in addition to the normal network coding processing, intermediate nodes must now generate a signature on each outgoing packet. On the other hand, the per-packet communication overhead due to the signature is now *constant* rather than linear in m as in the case of homomorphic hashing.

A concrete example of a homomorphic signature scheme (the *BFKW scheme*) was given by Boneh et al. [6]; the scheme can be proven secure based on the CDH assumption in the random oracle model. We provide a description here for future reference. To begin, the source S establishes a public key as follows:

[3] In some settings, there may be alternate ways to distribute the $\{h_i\}$ authentically.

1. Generate $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e)$ where $\mathbb{G}, \mathbb{G}_T$ are groups of prime order p, and $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ is a bilinear map. Choose random $g_1, \ldots, g_n, h \in \mathbb{G}$.
2. Choose $s \leftarrow \mathbb{Z}_p$, and set $f := h^s$.
3. Let $H : \{0,1\}^* \times \mathbb{Z} \to \mathbb{G}$ be a hash function, modeled as a random oracle.
4. Output the public key $PK = (\mathcal{G}, H, g_1, \ldots, g_n, h, f)$ and the private key s.

To sign a vector $w = (u \parallel v) \in \mathbb{Z}_p^{m+n}$ associated with the file identifier fid, the source S computes the signature

$$\sigma := \left(\prod_{i=1}^{m} H(\mathsf{fid}, i)^{u_i} \prod_{j=1}^{n} g_j^{v_j} \right)^s .$$

(Note that the above can be viewed as applying a cryptographic operation [that depends on the secret key] to a homomorphic hash of w.) An intermediate node who knows PK can verify validity of a vector $w = (u \parallel v)$ with associated signature σ by checking whether

$$e(\sigma, h) \stackrel{?}{=} e\left(\prod_{i=1}^{m} H(\mathsf{fid}, i)^{u_i} \prod_{j=1}^{n} g_j^{v_j},\ f \right). \tag{4}$$

Upon receiving vectors $w^{(1)}, \ldots, w^{(\ell)}$ with valid signatures $\sigma_1, \ldots, \sigma_\ell$, an intermediate node can generate a valid signature on any linear combination $w = \sum_i \alpha_i w^{(i)}$ by computing $\sigma := \prod_{i=1}^{\ell} \sigma_i^{\alpha_i}$.

3 Network Coding Over the Integers

In this section we describe the idea of implementing network coding *over the integers* rather than over a finite field. This approach is essential for the cryptographic schemes we propose in the following sections, and also results in efficiency improvements for existing schemes as we describe here.

Let us first motivate this departure from traditional network coding. Examining the signature schemes described in the previous section, one can see that they result in significant performance penalties relative to basic (insecure) network coding, in terms of both communication and computation. The increase in communication is due to the fact that instead of working over a small (e.g., 8-bit) field as in basic network coding, the cryptographic schemes work modulo a 160-bit prime p. Each file vector is thus augmented by 160-bit coordinates, rather than 8-bit coordinates as in basic network coding — a 20-fold increase in the communication overhead. This also impacts computation; for example, the time required to verify signatures when using the EHH scheme (cf. Eq. (3)) is proportional to the bit-length of the exponents (i.e., the coefficients $\{u_i\}$). A similar effect can be observed in the time required to compute signatures at intermediate nodes when using the BFKW scheme (cf. Eq. (4)).

To alleviate these performance costs, our approach will be to choose small integer coefficients as opposed to 160-bit scalars as in previous schemes. In more detail: We now view the file $\bar{F}$ transmitted by the source S as a sequence of vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$ with *integer* coordinates. (At this point we do not specify the dimension of these vectors or the range of the coordinates – these details will depend on the specific cryptographic scheme used). These vectors are augmented with unit vectors $\bar{u}^{(1)}, \ldots, \bar{u}^{(m)}$ as described in Section 2.1. Intermediate nodes will again compute outgoing packets as random linear combinations of incoming vectors, except that now these combinations are taken over the integers and the coefficients α_i are chosen uniformly from $Q = \{0, \ldots, q-1\}$ for some small prime q (e.g., $q = 257$). (A hybrid approach using small integer coefficients but with linear combinations performed modulo a large prime is studied in Section 3.1. In no case are the computations done modulo q.) We stress that the coordinates of the file vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$ need not lie in Q.

Recall from Section 2.1 that the usefulness of random linear network coding depends on the decoding probability, namely, the probability with which a recipient can correctly reconstruct the file transmitted by the source. Technically, this is the probability that the recipient collects m vectors whose u-portions form an invertible matrix U (see Eq. (1)). For the setting described above, where operations are performed over the integers, we must re-analyze the decoding probability since existing bounds hold only when network coding is performed over a finite field. Fortunately, we show that working over the integers can only improve the decoding probability in a sense we make formal now.

Lemma 1. *Fix q prime. For any network, the decoding probability when network coding is performed over the integers with intermediate nodes choosing coefficients uniformly from $Q = \{0, \ldots, q-1\}$, is at least the decoding probability when network coding is performed modulo q (with intermediate nodes choosing coefficients uniformly from Q).*

Proof. Fix a sequence of coefficients $\alpha_i \in Q$ chosen by the intermediate nodes during a run of the network coding protocol when operations are performed modulo q. Assume these coefficients lead to successful recovery of the file in this case. This means that the target receives m vectors such that the u-portions of these vectors give a matrix U with $\det(U) \neq 0 \pmod{q}$. Note that $\det(U) \bmod q$ is unchanged if no modular reductions are performed in the network, but instead all modular reductions are 'delayed' and performed only by the target. But if U is an integer matrix, $\det(U) \neq 0 \bmod q$ implies $\det(U) \neq 0$ over the integers; thus, successful recovery would occur for these same coefficients if all operations were performed over the integers. □

The lemma implies that in order to get a good decoding probability when working over the integers, it suffices to choose q such that the decoding probability when working modulo q is sufficiently good. This puts us back in the setting of standard network coding, where the required size of the underlying field is well-studied. Appropriate choice of q depends on the network topology, required fault tolerance, etc., but in most practical applications an 8-bit q suffices.

In fact, we expect that working over the integers with coefficients chosen from $Q = \{0, \ldots, q-1\}$ will induce a decoding probability that is *noticeably better* than working over a field of size q. If so, one could further save in bandwidth and computation by reducing the size of q. Another variant to investigate is choosing coefficients from the set $\{-q/2, \ldots, q/2\}$.

Coordinate growth. When we work over the integers without any modular reduction, the size of the coordinates of the vectors transmitted in the network increases with each traversed hop. Specifically, each hop through some node increases the maximal coordinate of some vector in the network by a factor of at most $\min\{mq, \ell q\}$, where ℓ is the in-degree of that node. (Note that even if $\ell > m$, the incoming vectors contain a set of at most m linearly independent vectors.) So, after L hops the first m coordinates each have magnitude at most $(mq)^L$ (since the initial m coordinates in the augmented vectors sent by the source are 0/1-valued), while the remaining coordinates have magnitude at most $M(mq)^L$, where M is the maximal size of coordinates in the original file vectors $\bar{v}^{(i)}$. As we will see, by working over the integers we obtain bandwidth improvements (in typical networks) in spite of this coordinate growth.

We remark also that an attacker can generate large *valid* packets by choosing large coefficients, thus countering some of the bandwidth gains achieved by having honest nodes use small coefficients. (Note, however, that nodes may be able to reject suspiciously large packets; e.g., those that deviate significantly from the average packet size received at the node or packets whose coefficients exceed an upper bound derived from the distance between the node and the source.) Network coding signatures cannot and do not prevent all forms of denial of service; their purpose is to prevent pollution attacks that are easy for an attacker to carry out yet have devastating effect.

3.1 Improvements to Existing Schemes

We consider here a "hybrid" variant where intermediate nodes choose small integer coefficients but operations are performed modulo a large prime p. This approach will allow us to significantly improve the performance of the schemes described in Section 2.2, while keeping their security guarantees intact.

In the schemes described in Section 2.2, network coding is done modulo a large prime p. That is, the original vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$ transmitted by the source are in $\mathbb{Z}_p^n$; the coefficients for the linear combinations are chosen at random from $\mathbb{Z}_p$; and the linear combinations are performed modulo p. Here we suggest to keep these schemes unchanged except that the random coefficients chosen by each intermediate node will be taken from the set $Q = \{0, ..., q-1\}$ for some small prime q (we stress that linear combinations are still computed modulo p).

We first analyze the effect of this change on the decoding probability, showing that the decoding probability remains high as long as (1) p is a random k-bit prime, and (2) m and the maximal path length L from the source to the target are negligible relative to 2^k (the latter is the case in our applications where k is typically 160 or larger).

Lemma 2. *Fix q prime. For any network, the decoding probability of the "hybrid" scheme described above (where intermediate coefficients are chosen at random from $Q = \{0, \ldots, q-1\}$ and the linear combinations are performed modulo a random k-bit prime p) is at least the decoding probability when network coding is performed modulo q (with intermediate nodes choosing coefficients uniformly from Q), up to an $O((Lm \log mq)/2^k)$ additive term.*

Proof. As in the case of Lemma 1, we may assume that all linear combinations in the network are performed over the integers, and all modular reductions are performed only at the end by the target node. Fix some set of coefficients, chosen by all intermediate nodes, for which reconstruction of the file (when operations are performed mod q) succeeds. Letting U^* denote the integer matrix computed at the target, this means that $\det(U^*) \neq 0 \bmod q$ which, in turn, implies $\det(U^*) \neq 0$ (over the integers). We now show that except with probability $O(Lm \log mq/2^k)$ over choice of p, it also holds that $\det(U^*) \neq 0 \bmod p$.

Let d denote the bit-length of $\det(U^*)$. The number of primes of length k dividing $\det(U^*)$ is at most d/k, and the number of primes of length k is $O(2^k/k)$. Thus the probability that p divides $\det(U^*)$ is at most $O(d/2^k)$. It remains to bound $d = O(\log |\det(U^*)|)$.

The matrix U^* is composed of the u-portion of vectors received by the target. As seen before, the u-coordinates of such vectors have magnitude at most $(mq)^L$, where L is the maximal path length from the source to the target. So U^* is an $m \times m$ matrix with each entry having magnitude at most $(mq)^L$. Thus, $\det(U^*) \leq m!(mq)^{Lm} \leq (mq)^{m(L+1)}$ and $d = O(Lm \log mq)$. □

We proceed to examine how using small integer coefficients can improve the performance of the network coding signature schemes discussed in Section 2.2.

Saving bandwidth. In the two schemes reviewed in Section 2.2, all vectors transmitted in the network are pre-pended with a u-portion consisting of m coordinates each 160 bits in length. (For simplicity, we assume here that p is a 160-bit prime.) Using our approach, all vectors are pre-pended with a u-portion consisting of m integer coordinates each of whose length is at most 160 bits (since we are still performing reduction modulo p). On average, however, the length of these coordinates can be much smaller.[4] For example, assume the maximum path length is 16 hops and u-coordinates increase by at most 10 bits per hop (this is the case, e.g., if $\ell = 4$ and $q = 253$). After the first hop the u-coordinates are at most 10-bits long; after the second hop they are at most 20-bits long, etc. Thus, in the worst-case we use (on average over all hops) 80 bits per coordinate which reduces the bandwidth of the u-components by a factor of two as compared to the case when intermediate nodes choose coefficients from $\mathbb{Z}_p$. Better improvements are obtained when average path lengths are shorter; even when average path lengths are longer, our approach can never perform worse than the basic approach.

[4] Note that the coordinates of the v-portion of the vectors are not affected by the use of small coefficients; in both cases these are always 160-bit values.

Saving computation. Reducing the bit-length of the u-coordinates yields computational savings as well, due to the use of shorter exponents during verification (cf. Eqs. (3) and (4)). A major improvement is also obtained in the computation required by intermediate nodes in generating the signatures of their outgoing vectors when using the BFKW scheme, exactly due to the use of small coefficients. This gives a 20-fold improvement for this operation, regardless of the average path length in the network. See also the following remark.

Remark 1. Signature verification can be done on an opportunistic basis by intermediate nodes (e.g., for a random subset of vectors). In contrast, signature computation must be done by *all* intermediate nodes for each outgoing packet.

4 An RSA-Based Network Coding Signature Scheme

In this section we present an RSA-based network coding signature scheme that enjoys a proof of security in the random oracle model under the RSA assumption, and relies on the ability to perform random linear network coding over the integers as described in Section 3. The scheme is similar to the BFKW scheme and adapts ideas from [3, 4] in the same way the BFKW scheme borrows from [19]. We construct a homomorphic signature scheme by applying a multiplicatively homomorphic signature to a homomorphic hash of the vector being signed. The homomorphic hash we use is similar to the EHH scheme except that we work modulo an RSA composite rather than modulo a prime. We take as our multiplicatively homomorphic signature the "textbook RSA" scheme where a signature on x is just $x^d \bmod N$. The resulting scheme is presented in Section 4.1.

In order to use the resulting scheme for network coding, it is essential that the linear operations being performed by the nodes "work" relative to an unknown modulus (that arises in our case because $\phi(N)$ is unknown). To achieve this, we have intermediate nodes perform network coding over the integers. We describe this in detail in Section 4.2.

4.1 An RSA-Based Homomorphic Signature Scheme

We start by defining an RSA-based homomorphic signature scheme denoted Bsig.

• Public and secret keys: Let N be a product of two safe primes; in particular, the subgroup of quadratic residues $\mathcal{QR}_N$ is cyclic and random elements of $\mathcal{QR}_N$ are generators of this subgroup with overwhelming probability. The public key is $(N, e, g_1, \ldots, g_n)$ and the secret key is d, where $ed = 1 \bmod \phi(N)$ and $g_1, \ldots, g_n$ are random generators of $\mathcal{QR}_N$.

• Signature generation: The signature on $v = (v_1, \ldots, v_n) \in \mathbb{Z}^n$ is given by

$$\mathsf{Bsig}(v) = \left(\prod_{i=1}^{n} g_i^{v_i} \right)^d \bmod N. \tag{5}$$

Verification is done in the obvious way. It is easy to see that this scheme is homomorphic: for any $v, v' \in \mathbb{Z}^n$ and $\alpha, \beta \in \mathbb{Z}$, we have $\mathsf{Bsig}(\alpha v + \beta v') = (\mathsf{Bsig}(v))^\alpha \cdot (\mathsf{Bsig}(v'))^\beta$.

4.2 An RSA-based Network Coding Signature Scheme

Here we describe how the above scheme Bsig can be extended to give a network coding signature scheme Nsig. We first review the underlying network coding being performed, focusing on details not already covered in Section 3. The file held by the source S is a sequence of vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$, where each $\bar{v}^{(i)} \in \mathbb{Z}^n$ for some value n. Note that once the size of the file and the number m of vectors is fixed, a lower bound $|v|$ on the bit-length of each of the vectors $\bar{v}^{(i)}$ is determined, and n can take on any value between 1 and $|v|$. As we will see, smaller values of n reduce communication while larger values of n reduce computation (very often $n = 1$ will provide the most practical trade-off).

As usual, before sending the $\bar{v}^{(i)}$ vectors to the network, the source pre-pends them with unit vectors $\bar{u}^{(i)}$ thus producing $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)} \in \mathbb{Z}^{m+n}$. Everything else is carried out as already described in Section 3: in particular, intermediate nodes generate random linear combinations (over the integers) of incoming packets, using coefficients chosen uniformly from $Q = \{0, ..., q-1\}$ for prime q.

Let L be an upper bound on the path length from the source to any target. (Looking ahead, the Nsig scheme defined below may reject packets that traverse more than L hops.) Given L we define a bound $B = (mq)^L$ which represents the largest possible value of a u-coordinate in any (honestly generated) vector; cf. Section 3. If M denotes an upper bound on the magnitude of the coordinates of the initial vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$, then the maximal magnitude of any coordinate in an honestly generated vector is $B^* = BM$.

We now introduce our scheme Nsig.

• Parameters: m, n, M, B, and B^*.

• Public and secret keys: The public key $(N, e, g_1, \ldots, g_n)$ and the secret key d are as in Bsig, except that e is chosen to be prime with $e > mB^*$ (for efficiency reasons e can be chosen to have low Hamming weight). In addition, the scheme uses a public hash function $H : \{0,1\}^* \to \mathcal{QR}_N$ that will be modeled as a random oracle.

• Signature generation by source S: On input a file given by m vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)} \in \mathbb{Z}^n$, the source S generates the augmented vectors $\bar{w}^{(i)} = \bar{u}^{(i)} \parallel \bar{v}^{(i)} \in \mathbb{Z}^{m+n}$ in the usual way. S chooses random $\mathsf{fid} \in \{0,1\}^k$, and computes $h_i = H(i, \mathsf{fid})$ for $i = 1, \ldots, m$. The signature on each vector $w = (u_1, \ldots, u_m, v_1, \ldots, v_n)$ is:

$$\mathsf{Nsig}(w) = \left(\prod_{i=1}^{m} h_i^{u_i} \prod_{j=1}^{n} g_j^{v_j} \right)^d \bmod N. \tag{6}$$

S transmits each $\bar{w}^{(i)}$ along with its signature and fid.

• Signature verification: Given $w = u \parallel v = (u_1, \ldots, u_m, v_1, \ldots, v_n) \in \mathbb{Z}^{m+n}$, a file identifier fid, and a signature σ, verification is done as follows. Reject immediately if any of the u-coordinates is negative or larger than B, or any of the v-coordinates is negative or larger than B^*. Otherwise, compute $h_i = H(i, \mathsf{fid})$ for $i = 1, \ldots, m$ and accept the signature if and only if

$$\sigma^e \stackrel{?}{=} \prod_{i=1}^{m} h_i^{u_i} \prod_{j=1}^{n} g_j^{v_j} \bmod N. \tag{7}$$

(An optimized batch verification procedure for testing multiple incoming vectors is presented at the end of this subsection.)

• Signature combination at intermediate nodes. Upon receiving $w^{(1)}, \ldots, w^{(\ell)}$ associated with the same fid and with valid signatures $\sigma_1, \ldots, \sigma_\ell$, an intermediate node proceeds as follows. It first discards any $w^{(i)}$ having a u-coordinate larger than B/mq or a v-coordinates larger than B^*/mq.[5] For simplicity we continue to denote the non-discarded vectors by $w^{(1)}, \ldots, w^{(\ell)}$. The intermediate node then chooses random coefficients $\alpha_1, \ldots, \alpha_\ell \in Q$, sets $w = \sum_{i=1}^{\ell} \alpha_i w^{(i)}$, and computes the signature on w as:

$$\sigma = \prod_{i=1}^{\ell} \sigma_i^{\alpha_i} \bmod N. \tag{8}$$

We prove security of this scheme in the following subsection. First, we compare the performance of our scheme to the original BFKW scheme and the variant BFKW scheme (using small integer coefficients) described in Section 3.1.

Bandwidth. The lengths of the coordinates of the vectors w transmitted in the network increase by at most $s = \log(mq)$ bits for each traversed hop. Thus, after t hops each u-coordinate has bit-length at most ts. If $s = 10$, for example, then it will take 32 hops before the total communication overhead due to the u-coordinates exceeds that of the original BFKW scheme (where u-coordinates are always of size 160 bits). For most networks, where maximum path-lengths are expected to be much less than 32 hops, Nsig therefore incurs lower overhead. Comparing Nsig to the variant BFKW scheme described earlier in this work, we see that the two schemes have the same overhead until coordinates reach 160 bits; after that the variant BFKW scheme performs better (since in Nsig coordinates keep growing while in BFKW they do not). This, however, does not take into account the fact that in Nsig the v-coordinates also increase while in BFKW they do not. Fortunately, we can choose n to be small (e.g., $n = 1$), thus making this overhead insignificant (see more below regarding the choice of n).

Computation. The most critical operation is signature generation at intermediate nodes (see Remark 1 in Section 3.1). In Nsig this operation is extremely

[5] These bounds are more restrictive than those required by signature verification, and are intended to ensure that signature verification will succeed at the *next* hop.

efficient since the exponents α_i in Eq. (8) are small (say, 8 bits each). Thus, we can expect this operation to be roughly 20 times faster in Nsig than in the original BFKW scheme. (The variant BFKW scheme is expected to perform about as well as Nsig.) Verification is more expensive. Looking at Eq. (7), we see that verification in Nsig requires an exponentiation using $(|v| + (m+n)\log B)$-bit exponents and a $(\log m + \log B + |v|/n)$-bit exponent (i.e., the bit-length of e). Since the impact of n is more significant with regard to bandwidth than computation, in most cases it makes sense to choose $n = 1$. The resulting cost of verification is still better than that of the original BFKW scheme due to the pairing operation and the cost of hashing onto the bilinear groups in the latter. The cost of a hashing operation in this case is equivalent to a full exponentiation and it is needed for computing each of the m values $h_i = H(\mathsf{fid}, i)$ (and also to compute the generators $g_1, \ldots, g_n$ in the case that one implements BFKW with fixed-size public key). In contrast, in the case of Nsig the computational cost of the hashing operations is negligible. Moreover, if one uses $n = 1$ the resultant public key has a single generator while in BFKW one needs $|v|/160$ of them (e.g., for a 4 Kbyte $|v|$, BFKW requires 200 generators). The cost of computation can be further improved by resorting to a batch verification of incoming vectors as described next.

Batch verification. The most expensive operation in the Nsig scheme is the verification of incoming signatures. Here we show that instead of verifying each incoming signature it suffices to *verify just one outgoing signature*. The probability that the verification of this outgoing vector succeeds but one of the incoming vectors was invalid is at most $1/q$. This is fine in most cases since even if a node forwards an invalid vector this will be caught with high probability by subsequent nodes (i.e., the probability that t consecutive honest nodes do not discover a forgery is at most $1/q^t$). To achieve this optimization we modify the actions of intermediate nodes as follows.

Upon receiving $w^{(1)}, \ldots, w^{(\ell)}$ associated with the same fid and with alleged signatures $\sigma_1, \ldots, \sigma_\ell$, intermediate node I discards any vector that has too large coordinates as described above. Then, I generates one outgoing vector as usual, i.e., chooses random coefficients $\alpha_1, \ldots, \alpha_\ell \in Q$ and sets $w = \sum_{i=1}^{\ell} \alpha_i w^{(i)}$. It then sets $\sigma = \prod_{i=1}^{\ell} \sigma_i^{\alpha_i} \bmod N$ and verifies (using Eq. (7)) that $\mathsf{Nsig}(w)$ equals σ or $-\sigma$. If this verification succeeds, then no further verifications are needed. That is, I outputs w on one of its outgoing edges and proceeds to compute other outgoing vectors as in the case that all incoming vectors and their signatures were valid (with the usual random linear combinations and using Eq. (8) to generate outgoing signatures but without additional verifications). In this way, the number of signature verifications at any intermediate node is 1 regardless of the number of incoming or outgoing vectors. (Note: If the above verification of outgoing w fails, I may decide to discard its incoming vectors or test each one separately to find the valid ones – the important point is that under normal operation, i.e., without adversarial activity, a single verification suffices).

A proof of correctness of the above batch verification technique follows [5] and is presented in the full version.

4.3 Proof of Security

We now prove security of Nsig relative to the definition given in [6].

Theorem 1. *Under the RSA assumption,* Nsig *is a secure network coding signature scheme when the hash function H is modeled as a random oracle.*

Proof. Given a forger $\mathcal{F}$ attacking Nsig, we build an algorithm $\mathcal{S}$ that solves the RSA problem. Here $\mathcal{S}$ stands for *simulator* and also for *source* since $\mathcal{S}$ will be simulating the actions of the source being attacked by $\mathcal{F}$.

Algorithm $\mathcal{S}$ receives input N, e, C where N, e are distributed as in an Nsig public key and $C \in_R \mathcal{QR}_N$. Its goal is to output $C^{1/e} \bmod N$. (Note that if $\mathcal{S}$ computes $C^{1/e} \bmod N$ for $C \in_R \mathcal{QR}_N$ with non-negligible probability, this contradicts the standard RSA assumption where C is chosen uniformly from $\mathbb{Z}_N^*$.) Algorithm $\mathcal{S}$ begins by choosing $i_0 \in_R \{1, \ldots, n\}$ and then setting $g_{i_0} := C$. For $i \neq i_0$, algorithm $\mathcal{S}$ chooses $r_i \in_R \mathcal{QR}_N$ and sets $g_i := r_i^e \bmod N$. Then $\mathcal{S}$ calls $\mathcal{F}$ on the public key $(N, e, g_1, \ldots, g_n)$.

$\mathcal{F}$ chooses a file, represented as a set of vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$, and requests a signature on it. In response, algorithm $\mathcal{S}$ chooses $\sigma_1, \ldots, \sigma_m \in_R \mathcal{QR}_N$ and $\mathsf{fid} \in \{0,1\}^k$, and then sets (using the programmability of the random oracle H)

$$h_i \stackrel{\text{def}}{=} H(i, \mathsf{fid}) := \sigma_i^e \prod_{j=1}^{n} g_j^{-\bar{v}_j^{(i)}} \bmod N. \tag{9}$$

(If fid was used previously to sign another file, $\mathcal{S}$ aborts. This occurs with negligible probability and we ignore it from here on.) Finally, $\mathcal{S}$ gives to $\mathcal{F}$ the signature σ_i on the augmented vector $\bar{w}^{(i)} = \bar{u}^{(i)} \parallel \bar{v}^{(i)}$ (for $i = 1, \ldots, m$), along with fid. It is easy to see that signatures are distributed exactly in the real experiment.

Say $\mathcal{F}$ outputs a forgery, i.e., a file id fid^*, a vector $w^* \notin \text{span}\{\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}\}$ (where $\{\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}\}$ is the unique set of augmented vectors signed using fid^*), and a valid signature $\sigma^* = \mathsf{Nsig}(w^*)$ on w^*. We show how $\mathcal{S}$ can use this to solve its given RSA instance.

Denote $w^* = u^* \parallel v^* = (u_1^*, \ldots, u_m^*, v_1^*, \ldots, v_n^*)$, and define the vector

$$z^* = w^* - \sum_{i=1}^{m} u_i^* \bar{w}^{(i)}. \tag{10}$$

Note $z = (0, \ldots, 0, z_1, \ldots, z_n)$; that is, its first m coordinates are all zero. Moreover, since $w^* \notin \text{span}\{\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}\}$ at least one of the values z_i is non-zero. With probability at least $1/n$ we thus have $z_{i_0} \neq 0$, and we assume this to be the case from now on.

By definition of z^* and the homomorphic property of Nsig we have:

$$\begin{aligned} \mathsf{Nsig}(z^*) &= \mathsf{Nsig}(w^* - \sum_{i=1}^{m} u_i^* \bar{w}^{(i)}) \\ &= \mathsf{Nsig}(w^*) \prod_{i=1}^{m} \mathsf{Nsig}(\bar{w}^{(i)})^{-u_i^*} = \sigma^* \prod_{i=1}^{m} \sigma_i^{-u_i^*} \bmod N. \end{aligned} \tag{11}$$

On the other hand, we can also represent $\mathsf{Nsig}(z^*)$ as

$$\begin{aligned}\mathsf{Nsig}(z^*) &= \left(\prod_{i=1}^{m} h_i^0 \prod_{i=1}^{n} g_i^{z_i}\right)^{1/e} \\ &= (C^{z_{i_0}})^{1/e} \prod_{i \neq i_0} (g_i^{z_i})^{1/e} = (C^{z_{i_0}})^{1/e} \prod_{i \neq i_0} r_i^{z_i} \bmod N. \end{aligned} \tag{12}$$

Combining Eqs. (11) and (12) we get that

$$(C^{z_{i_0}})^{1/e} = \sigma^* \prod_{i=1}^{m} \sigma_i^{-u_i^*} \prod_{i \neq i_0} r_i^{-z_i} \bmod N,$$

from which $\mathcal{S}$ can compute a value x such that $x^e = C^{z_{i_0}} \bmod N$. Using a standard trick, $\mathcal{S}$ can then compute $C^{1/e} \bmod N$ provided that $\gcd(z_{i_0}, e) = 1$. But this is the case since $e > mBM$ is prime and

$$-mBM \;\leq\; v_{i_0}^* - \textstyle\sum_{i=1}^{m} u_i^* \bar{w}_{i_0}^{(i)} \;=\; z_{i_0} \;=\; v_{i_0}^* - \textstyle\sum_{i=1}^{m} u_i^* \bar{w}_{i_0}^{(i)} \;\leq\; MB.$$

(Since w^* passes verification we have $0 \leq u_i^* \leq B$ and $0 \leq v_i^* \leq MB$; it always holds that $0 \leq \bar{w}_j^{(i)} \leq M$.) □

Remark 2. The above proof uses the fact that e is larger than the coordinates of valid vectors. Indeed, if coordinates larger than e are allowed then given a valid vector $w = u||v$ with signature σ an attacker can output the forged signature $\sigma' = \sigma \cdot g_1$ on the vector $w' = u||v'$ with $v' = v + (e, 0, \ldots, 0)$.

5 Homomorphic Hashing Modulo a Composite

Network coding signatures based on homomorphic hashing can offer significant computational advantages relative to constructions based on homomorphic signature schemes since, when using the former, a node that chooses not to verify an incoming vector (cf. Remark 1) need not perform any cryptographic operations. On the other hand, constructions based on homomorphic hashing consume more bandwidth since nodes now need to obtain the (authenticated) hash values of the original file vectors. If delivery of the hash values to nodes can be done in some out-of-band fashion, however, this drawback is mitigated.

Here we introduce a homomorphic hashing scheme, denoted $\mathbf{H}_N$, that is similar to the EHH scheme described in Section 2.2 but where operations are performed modulo a composite N. This results in the homomorphic properties holding over a group of unknown order; hence this scheme can only be applied when the underlying network coding is done over the integers. $\mathbf{H}_N$ has better computational efficiency than the EHH scheme (over prime-order groups) from Section 2.2; although $\mathbf{H}_N$ produces larger hash values, it requires a smaller public key than the EHH scheme.

In this section we once again assume linear network coding being performed over the integers as described in Section 3. Namely, the file to be transmitted is represented by vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)} \in \mathbb{Z}^n$, and intermediate nodes choose coefficients uniformly from a set $Q = \{0, \ldots, q-1\}$ (for some small prime q) and compute all linear combinations without any modular reduction.

Let N be the product of two safe primes so that the group $\mathcal{QR}_N$ of quadratic residues modulo N is cyclic, and let $g_1, \ldots, g_n$ be generators of $\mathcal{QR}_N$. For $v = (v_1, \ldots, v_n) \in \mathbb{Z}^n$, define

$$\mathbf{H}_N(v) = \prod_{j=1}^{n} g_j^{v_j} \bmod N.$$

This is a homomorphic hash function that is collision resistant if factoring N is hard (proof omitted). Thus, $\mathbf{H}_N$ can serve as a basis for a network coding signature scheme as discussed in Section 2.2. In particular, a node receiving a vector $w = (u_1, \ldots, u_m, v_1, \ldots, v_n)$ can verify it by checking whether

$$\prod_{i=1}^{m} h_i^{u_i} \stackrel{?}{=} \mathbf{H}_N(v) \stackrel{\text{def}}{=} \prod_{j=1}^{n} g_j^{v_j} \bmod N. \tag{13}$$

where $h_i = \mathbf{H}_N(\bar{v}^{(i)}), i = 1, \ldots, m$. Below we show a batch verification optimization that allows an intermediate vector to verify *all* of its incoming vectors with a *single* application of Eq. (13).

Bandwidth considerations for this scheme, which uses integer coefficients that grow over time, are similar to those of the RSA-based scheme from Section 4. An additional benefit of $\mathbf{H}_N$ is that there is no need to determine an *a priori* bound on these coefficients. As in the case of the RSA-based scheme from Section 4, one way to limit the effect of coordinate growth on the total communication is to set $n = 1$. As we now discuss, this not only reduces bandwidth overhead but also improves computational performance significantly.

Fix $n = 1$ so that each block of information $\bar{v}^{(i)}$ is a *single* (long) integer. Choosing N appropriately[6], we can take 2 as a generator of $\mathcal{QR}_N$, thus obtaining:

$$\mathbf{H}_N(v) = 2^v \bmod N.$$

This achieves the most salient advantage of the $\mathbf{H}_N$ scheme: *fast exponentiation*. Another advantage of this homomorphic hash is that it considerably improves the size of the public parameters relative to the EHH scheme. To see this, observe that in the EHH scheme the total length of the set of generators $g_1, \ldots, g_n$ included in the public parameters is (at least) $n \log p$ which is (at least) as large as each information vector $\bar{v}^{(i)}$. Moreover, the number of generators is usually very large; e.g., for vectors $\bar{v}^{(i)}$ of size 4KB and 160-bit p the EHH scheme

[6] Choose $N = p_1 p_2$ $p_1 = 2p'_1 + 1, p_2 = 2p'_2 + 1$, and p_1, p_2, p'_1, p'_2 all prime; $p'_1, p'_2 = 3 \bmod 8$; and $p_1, p_2 = 7 \bmod 8$.

needs 200 random generators.[7] In the case of $\mathbf{H}_N$, on the other hand, only one generator is needed and furthermore this generator can be fixed to 2; the public parameters need only include N.

Batch verification. The use of $\mathbf{H}_N$ for network coding can be further optimized by using batch verification at intermediate nodes similarly to the procedure described in Section 4.2 for the Nsig signature. Specifically, instead of verifying each incoming vector using Eq. (13), an intermediate node can just generate one outgoing vector as usual (i.e., as a random linear combination over $\{0, \ldots, q-1\}$ of the incoming vectors) and then apply Eq. (13) to the resultant vector. It can be shown that the probability that this single verification passes but one of the incoming vectors was invalid (not in the span of $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$) is at most $1/q$. The probability that t consecutive nodes will be foiled to accept invalid vectors is at most $1/q^t$. Thus a single verification per intermediate node suffices regardless of the number of incoming or outgoing vectors.

In all, we have shown that the homomorphic hashing scheme $\mathbf{H}_N$ leads to a computationally efficient network coding signature scheme whose security can be proven based on the factoring assumption in the standard model (and assuming the security of the signature scheme used to sign the hash values $h_1, \ldots, h_n$).

Acknowledgments Research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the US Government, the UK Ministry of Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints of this work for Government purposes, notwithstanding any copyright notation hereon.

References

1. S. Agrawal and D. Boneh. Homomorphic MACs: MAC-based integrity for network coding. In *Applied Cryptography and Network Security (ACNS)*, 2009.
2. R. Ahlswede, N. Cai, S. Li, and R. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.
3. G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song. Provable data possession at untrusted stores. In *ACM Conference on Computer and Communications Security*, pages 598–609, 2007.
4. G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In *Advances in Cryptology — Asiacrypt 2010*, volume 5912 of *LNCS*, pages 319–333. Springer, 2010.
5. M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology — EUROCRYPT 1998*, volume 1403 of *Springer LNCS*, pages 236–250, 1998.

[7] The size of the public parameters can be reduced by using a hash function to compute the generators "on the fly." Besides necessitating the use of the random oracle model, this also introduces additional computational overhead.

6. D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *Public Key Cryptography (PKC)*, 2009.
7. D. Charles, K. Jain, and K. Lauter. Signatures for network coding. In *40th Annual Conference on Information Sciences and Systems (CISS '06)*, 2006. To appear in *International Journal of Information and Coding Theory*.
8. P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In *41st Allerton Conference on Communication, Control, and Computing*, 2003.
9. C. Gkantsidis and P. Rodriguez. Cooperative security for network coding file distribution. In *Proc. of IEEE INFOCOM 2006*, pages 1–13, 2006.
10. T. Ho, R. Koetter, M. Médard, D. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *Proc. of International Symposium on Information Theory (ISIT)*, 2003.
11. 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. Intl. Symposium on Information Theory (ISIT)*, pages 144–152, 2004.
12. T. Ho and D. Lun. *Network Coding: An Introduction*. Cambridge University Press, 2008.
13. 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 Trans. Inform. Theory*, 52(10):4413–4430, 2006.
14. S. Jaggi. *Design and Analysis of Network Codes*. PhD thesis, California Institute of Technology, 2006.
15. 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 Trans. on Information Theory*, 54(6):2596–2603, 2008.
16. R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *Proc. of CT-RSA 2002*, volume 2271 of *Springer LNCS*, pages 244–262, 2002.
17. M. Krohn, M. Freedman, and D. Mazieres. On the-fly verification of rateless erasure codes for efficient content distribution. In *Proc. IEEE Symposium on Security & Privacy*, pages 226–240, 2004.
18. S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Trans. Inform. Theory*, 49(2):371–381, 2003.
19. H. Shacham and B. Waters. Compact proofs of retrievability. In *Advances in Cryptology — Asiacrypt 2008*, volume 5350 of *LNCS*, pages 90–107. Springer, 2008.
20. Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan. An efficient signature-based scheme for securing network coding against pollution attacks. In *INFOCOM*, 2008.
21. F. Zhao, T. Kalker, M. Médard, and K. Han. Signatures for content distribution with network coding. In *Proc. Intl. Symp. on Information Theory (ISIT)*, 2007.