Secure Network Coding Over the Integers

Rosario Gennaro[*] Jonathan Katz[†] Hugo Krawczyk[*] Tal Rabin[*]

Abstract

Network coding has received significant attention in the networking community for its 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 in a way that prevents the reconstruction of information at recipients; such attacks cannot be prevented using standard end-to-end cryptographic authentication because network coding requires that intermediate nodes modify data packets in transit. Specialized solutions to the problem have been developed in recent years based on homomorphic hashing and homomorphic signatures. The latter are more bandwidth-efficient but require more computation; in particular, the only known construction uses bilinear maps.

We contribute to this area in several ways. We present the first homomorphic signature scheme based solely on the RSA assumption (in the random oracle model), and present a homomorphic hashing scheme based on composite moduli that is computationally more efficient than existing schemes (and which leads to secure network coding signatures based solely on the hardness of factoring in the standard model). Both schemes use shorter public keys than previous schemes. In addition, we show variants of existing schemes that reduce the communication overhead significantly for moderate-size networks, and which improve computational efficiency in some cases quite dramatically (e.g., we achieve a 20-fold speedup in the computation of intermediate nodes). At the core of our techniques is a modified approach to network coding where instead of working in a vector space over a field, we work directly over the integers (with small coefficients).

[*]IBM T.J. Watson Research Center, Hawthorne, NY. Email: `rosario@us.ibm.com, hugo@ee.technion.ac.il, talr@us.ibm.com`. Research sponsored by the US Army Research laboratory and the UK Ministry of Defence under agreement number W911NF-06-3-0001.

[†]Dept. of Computer Science, University of Maryland. Work done while visiting IBM. Research supported by NSF CAREER award #0447075, NSF Trusted Computing grant #0627306, the U.S. DoD/ARO MURI program, and the US Army Research Laboratory and the UK Ministry of Defence under agreement number W911NF-06-3-0001.

1 Introduction

Network coding (NC) [2, 15] offers an alternative, decentralized approach to traditional multicast routing. We consider a network setting where a *source node* has a piece of information (a file) that it wants to distribute to a set of *target nodes*. The source partitions the file into m network 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 computed as *linear combinations* of incoming packets, where packets are viewed as vectors in a linear space over some field. We focus on the case of *random linear network coding* [7, 10], where scalars are chosen *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 as each chooses its own linear combinations independently of other nodes.

Target nodes reconstruct the original file sent by the source from the packets they receive. Fortunately, this does not require knowledge by the target of all scalars chosen by intermediate nodes. Instead, it suffices to *augment* each information vector traveling the network with m additional *coding coordinates* that encode in a compact way the history of all linear combinations that resulted in that vector. A target that receives a set of augmented vectors for which the attached coding coordinates induce a full rank matrix can recover the original information sent by the source via a simple matrix inversion operation (see Section 2.1). A fundamental question is what is the *decoding probability* at the targets, namely, the probability with which a target is able to reconstruct the original file (or, equivalently, the probability for the target to collect enough linearly independent coding vectors). The NC literature, and actual applications, show that small-size fields (e.g., of size 256) provide very good decoding probability for networks with sufficient connectivity.

However, while random linear NC can increase throughput and reliability relative to alternative techniques, it suffers from a serious weakness: its susceptibility to *pollution attacks* in which malicious nodes inject into the network invalid packets that prevent the reconstruction of information at the targets. Here, an invalid packet is any packet that is not in the linear span of the original augmented vectors sent by the source. By the way vectors are propagated and combined in the network, a single invalid packet injected by the attacker can invalidate many more packets, eventually preventing the reconstruction of information at the targets. This constitutes a serious denial of service attack which an attacker can mount effortlessly and which in real-life scenarios can be sufficient to outweigh the benefits of network coding.

Clearly, what is needed is a way for intermediate nodes to be able to verify the validity of incoming vectors. Note, however, that since packets are modified by intermediate nodes, a regular signature by the source on the original information is not sufficient. Prior work has shown, however, that dedicated *network coding signatures* can be used to solve this problem. These are based on one of two primitives: eother (collision-resistant) *homomorphic hash functions* [14, 17] or *homomorphic signature schemes* [13, 4, 3] In both cases, the homomorphic properties are used such 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 details, and [3] for a complete survey.

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 [3] and involves costly pairing operations. In particular, NC 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 sent in the network to be sent with some "authentication data" whose length is proportional to m (the number of information vectors). One drawback of both approaches is that they replace the small fields used in NC with very large ones. Thus, instead of using vectors over an 8-bit field as in traditional NC, the cryptographic approaches use vectors over a 160-bit (i.e., cryptographically strong) field instead. This results in a factor of 20 increase in the bandwidth overhead.

Our contributions. We contribute to this area in several ways. We present the first homomorphic signature scheme based on the RSA assumption in the random oracle model.[1] In particular, it is the first homomorphic signature scheme to avoid bilinear groups and pairings, and thus has more efficient processing at the intermediate nodes. Bandwidth overhead is lower than in existing schemes for networks of moderate size (e.g., where the maximum path length between source and target nodes is 20–30 hops). In addition, the scheme uses a public key of constant size.

We also present a new homomorphic hashing scheme that works modulo a composite number N. However, instead of the many random generators used by known schemes, ours uses a single *fixed* generator which can be set to 2, speeding up significantly the exponentiation operation. Concretely, by considering each information vector v transmitted over the network as a single (large) integer, we define our hash function simply as $H_N(v) = 2^v \bmod N$. The hash function is homomorphic over the integers and can be proven collision resistant based on the hardness of factoring. In particular, it leads to a provably secure NC signature scheme based on the factoring assumption and without random oracles.

A core technique we must develop for the above constructions is to apply network coding over the *integers* rather than 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*. As noted earlier, a major disadvantage of prior cryptographic techniques in the context of NC is their use of large (160-bit) coordinates and equally large coefficients for the linear combinations. In contrast, by working over the integers we are able to choose small (e.g., 8-bit) integer coefficients for the linear combinations. This has the immediate effect of improving the computation at intermediate nodes by a factor of 20, and it also reduces the total bandwidth overhead for networks with moderate-length paths between source and targets. A crucial question we need to answer is how this affects the decoding probability. Fortunately, we can 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 a field of size q and therefore 8-bit coefficients are good enough for most applications.

The ability to work with NC with small integer coefficients allows us also to improve the performance of existing schemes. We show that by simply choosing coefficients for the linear combinations in existing schemes from a small set Q as above (but still performing computations modulo the large prime p as required by these schemes) we are able to improve significantly the performance: we obtain a 20-fold improvement on signature generation and a reduction on the communication overhead as well.

We mention that our approach to NC over the integers may also enable constructions of *lattice-based* homomorphic signatures, and we are currently investigating this possibility.

[1]Yu et al. [16] recently proposed an RSA-based homomorphic signature scheme, but their scheme is flawed as verification of their signatures *always fails*. The source of the problem is that Yu et al. *incorrectly* assume (cf. equations (11) and (12) in section III-B) that for integers A, b, d, a prime p, and (independent) RSA composite r, it holds that $((A^b \bmod p)^d \bmod r) = (A \bmod p)^{bd} \bmod r$.

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 [9]. 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 an ordered sequence of m vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)} \in \mathbb{F}^n$ over some finite field $\mathbb{F}$.

Before transmission, the source S 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 transmitting node, for each of its outgoing communication links, the resulting scheme is referred to as *random* linear network coding [5, 7, 10]. For the purposes of analyzing efficiency, we assume in our work that this mechanism is used for choosing the coefficients; 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 $v^{(1)}, \ldots, v^{(m)}$, then the original file can be recovered as

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

where $\bar{V}$ is a matrix whose rows are the original information vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$.

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, as 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 probability of decoding failure of less than 1%.

2.2 Network Coding Signatures

The benefits of network coding can be outweighed by pollution attacks in which intermediate nodes forward *invalid* vectors to their neighbors. Even a *single* invalid vector that reaches a target node may be sufficient to cause incorrect reconstruction of the original file; furthermore, an invalid packet sent by a malicious node will, with high probability, render all subsequent packets downstream invalid as well. Note that the in-network processing required by network coding makes it impossible to apply traditional signatures or message authentication codes to individual packets.

It is not hard to design cryptographic schemes that prevent targets from reconstructing an *incorrect* file: e.g., a standard signature can be appended to the file before transmission, and verified upon reconstruction. This technique, however, is not sufficient to enable the target to reconstruct the *correct* file if multiple invalid vectors are present. Moreover, it does not provide any way for intermediate nodes to drop invalid packets they receive.

Early efforts to deal with pollution attacks focused on *information-theoretic* solutions that use error-correction techniques to ensure that targets can reconstruct the file as long as the ratio of valid to invalid vectors received is sufficiently high [8, 11, 12]. 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.

For the above reasons, researchers have more recently turned to *cryptographic* approaches that place no bounds on the adversary other than the assumption that the adversary is computationally bounded [14, 4, 17, 3]. 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. In particular, this allows target nodes to reject invalid vectors before reconstructing the file; it also allows intermediate nodes to filter out invalid vectors before generating their outgoing messages, thus preventing contamination of honestly generated vectors further downstream. A precise definition of network coding signatures and their security requirements is presented in Appendix A.

Two classes of network coding signature schemes are known: those based on *homomorphic hashing*, and those using *homomorphic signatures*.

Schemes based on homomorphic hashing [14, 17, 3]. 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 [14] of a homomorphic hash function is given by what we call the *exponential homomorphic hash (EHH)* scheme. Let $\mathbb{G}$ be a cyclic group of order p, and let the public key contain a description of $\mathbb{G}$ along with 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

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

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

[2]A symmetric-key analogue is also possible [6, 1], but this allows only a (single) target to verify validity of vectors.

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 $\prod_{i=1}^{m} h_i^{u_i} \overset{?}{=} H(v)$. In particular, for the EHH scheme this verification takes the form:

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

The resultant signature scheme can be proven secure in the standard model (no random oracles) based on the discrete logarithm assumption [14, 3].

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 [13, 4, 3]. 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 signatures on some set 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 then 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 I generates. Thus, in addition to the normal network coding processing, intermediate nodes must now compute 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. [3]; the scheme can be proven secure based on the CDH assumption in the random oracle model. We provide a description here for completeness and further reference. To begin, the source S establishes a public key as follows:

1. Generate $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, e)$ such that $\mathbb{G}, \mathbb{G}_T$ have prime order p, and $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ is a bilinear map. Choose random generators $h, g_1, \ldots, g_n \in \mathbb{G}$.

2. Choose $s \leftarrow \mathbb{Z}_p$, and set $f := h^s$.

3. Let $H : \mathbb{Z} \times \mathbb{Z} \to \mathbb{G}$ 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

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

As discussed previously, S then sends each augmented vector $\bar{w}^{(i)}$ along with its signature.

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

An intermediate node who knows PK can verify validity of a vector $w = (u \parallel v)$ with associated signature σ by computing

$$\gamma_1(PK, \sigma) \overset{\text{def}}{=} e(\sigma, h) \quad \text{and} \quad \gamma_2(PK, \mathsf{fid}, m, w) \overset{\text{def}}{=} e\left(\prod_{i=1}^{m} H(\mathsf{fid}, i)^{u_i} \prod_{j=1}^{n} g_j^{v_j},\ f\right), \tag{4}$$

and then checking whether $\gamma_1(PK, \sigma) \overset{?}{=} \gamma_2(PK, \mathsf{fid}, m, w)$. 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 $w = \sum_i \alpha_1 w^{(i)}$ by computing $\sigma := \prod_{i=1}^{\ell} \sigma_i^{\alpha_i}$.

3 Network Coding over the Integers

In this section we introduce the main technique that underlies the results in this paper: running random linear network coding over the integers with coefficients taken from a set of small (e.g., 8-bit) integers. This departs from the traditional linear coding schemes in that operations will be performed over the integers and will significantly depart from existing network coding signature schemes which use very large coefficients (typically, 160-bit long). This approach allows us to improve significantly on existing network coding signature schemes and, even more importantly, will enable the creation of new schemes, in particular one based solely on RSA.

Examining the two signature schemes from the previous section, one can see that the cryptographic techniques incur significant penalties relative to the basic (insecure) network coding protocols, both in terms of communication and computation. The additional computation is inherent to the cryptographic techniques and results mainly from the exponentiations used in these schemes. Communication increase is due to the fact that instead of working over a small (e.g., 8-bit) field, as in basic network coding, we are now working modulo a 160-bit prime p. This means that each information vector is augmented with m coordinates each of size at least 160 as opposed to m coordinates of size 8 each in basic network coding. This is a 20-fold increase in the communication overhead of the scheme. The size of the scalars used in these schemes also impacts computation. For example, the size of exponents in the left-hand side of Eq. (3) is directly proportional to the size of the u_i coordinates. Even more significant is the negative impact of these larger scalars when computing signature at intermediate nodes in the BFKW scheme of Section 2.2. In the $\mathsf{Combine}$ phase of this scheme a node computes $\prod_{i=1}^{\ell} \sigma_i^{\alpha_i}$ where the cost of exponentiation is directly proportional to the size of scalars α_i each of length 160 bits or more.

To alleviate these performance costs and allow for new schemes, our approach will be to choose small coefficients (say, of size 8 bits) as opposed to the 160-bit scalars of previous schemes. However, this raises the question of what effect these small integer coefficients (and working over the integers) has on the decoding probability of these schemes. Fortunately, we show that this approach improves performance without sacrificing decoding probability and without compromising the security of the resultant cryptographic signatures.

3.1 Network Coding Over the Integers

We modify the traditional random linear network coding schemes to work over the integers (rather than over a field) as follows. The original file $\bar{F}$ transmitted by the source S is encoded 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 subset of integers from which these coordinates are taken – 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 in regular network coding. Linear combinations of incoming vectors at intermediate nodes will use random coefficients α_i from a set of small integers $Q = \{0, \ldots, q-1\}$ for some small prime q, e.g., $q = 257$. These linear combinations will be performed *over the integers* (in particular, this will be the setting for the schemes presented in Section 4 and Section 5)[4]. We stress that these computations are *not* done modulo q. Also, the coordinates in the information vectors $\bar{v}^{(i)}$ do no have to be taken from the set Q, only the scalars for the linear combination do.

Recall (from Section 2.1) that the success of a random linear network coding scheme depends on the decoding probability, namely, the probability with which a target in the network succeeds in correctly reconstructing the vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$ transmitted by the source. Equivalently, this is the probability that the target collects m vectors whose u parts form an invertible matrix U (see Eq. (1)). The network coding literature studies this probability when linear operations are carried out over a field, but what happens to the decoding probability when the linear combinations are performed over the integers (no modular operations)? In this case, we need to consider the probability that the incoming vectors to the target induce a matrix U that is invertible (over the *rationals*) or, equivalently, that the determinant $\det(U)$ (computed over the integers) is non-zero.

The following simple (but significant) lemma shows how the decoding probability for network coding over the integers related to the decoding probability in the usual case, when network coding is performed over some base field:

Lemma 1 *For any network, the decoding probability when coefficients α_i are taken from the set $Q = \{0, \ldots, q-1\}$, for prime q, and all linear combinations are computed over the integers, is the no worse than the decoding probability in the same network using a traditional network coding scheme over a field of size q.*

Proof: Fix a sequence of coefficients α_i chosen by all nodes during a run of the network coding protocol. We need to show that if these α_i's resulted in a matrix U available to a target such that U is invertible modulo q then the same set of α_i's, when used to compute linear combinations over the integers, would result in a matrix U (not reduced modulo q) that is invertible in $\mathbb{Z}$. But this follows directly from the fact that if $\det(U)$ is non-zero modulo q then it is non-zero over the integers and hence invertible. $\square$

The lemma implies that in order to get a good decoding probability when working with linear combinations over the integers, it suffices to choose q such that the decoding probability when performing the linear combinations modulo q is sufficiently good. This puts us back in the standard setting of network coding (without cryptography) where the required size of the underlying field is well-studied. The appropriate size of q depends on the network topology, required fault tolerance, etc. In most practical applications a field of 8 bits (e.g., $\mathbb{Z}_q$ for $q = 251$ or $q = 257$) provides for good decoding probability and hence is sufficient also for our purposes.

In fact, we expect that working over the integers with coefficients from the set $Q = \{0, \ldots, q-1\}$ will induce a decoding probability that is *noticeably better* than working over a field of size q. If so, one could save more in bandwidth and computation by further reducing the size of q (e.g., using a 6-bit q). Another variant to investigate is choosing coefficients from the set $\{-q/2 \ldots q/2\}$ (in this case, one has to adjust the bounds on coordinates tested in the Nsig scheme of Section 4).

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. Specif-

[4]A hybrid approach where we work with small coefficients but modulo a large prime is studied in Section 3.2.

ically, each hop increases the maximal coordinate in a vector by a factor of at most $\min\{mq, \ell q\}$, where ℓ is the in-degree of a node.[5] (Note that ℓ can be larger than m but in this case a node can replace the incoming vectors with an equivalent set of at most m linearly independent vectors.) So, after L hops the first m coordinates each have size 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 size at most $M(mq)^L$, where M is the maximal size of coordinates in the original file vectors $\bar{v}^{(i)}$. The increase in bit-size of coordinates is the logarithm of the above numbers. As we will see, in spite of this growth of coordinates the smaller coefficients will impact favorably the bandwidth overhead of existing signature schemes and the new ones we present.

We note that one way to achieve a bandwidth-efficient implementation of the vectors w transmitted in the network is to represent all coordinates in the vector by the same number of bits (determined by the largest coordinate in the vector) and include this length in the vector representation; when vectors are combined and coordinates grow this length indicator will increase too. As another implementation note, we comment that in our setting Eq. (1) can be computed modulo p for a prime p that is larger than the largest coordinate in any information vector $\bar{v}^{(i)}$; in particular, computing U^{-1} can be done modulo such p.

Note also that an attacker can generate unnecessarily large valid packets using linear combinations with large coefficients. This can counter some of the bandwidth gains achieved by working with small coordinates. In general, network coding signatures do not prevent all forms of denial of service (the very cryptographic operations create such opportunities). The point, however, is to prevent the trivial attack in which arbitrary packets inserted into the network have a catastrophic denial of service effect without the attacker spending any resources. In the particular case of an attacker injecting valid packets with unnecessarily large coordinates, several measures can be taken. For example, a node can discard suspicious packets based on an estimate of the size of packets it is supposed to get (based on other packets it receives, or its position in the network, etc), and it can make sure not to mix packets with small coordinates with those with large ones.

3.2 Improvements to Existing Schemes

Lemma 1 is instrumental in enabling the new schemes presented in Sections 4 and 5. Here, we deal with a "hybrid" variant where we choose small integer coefficients as above but do the linear combination modulo a large prime p. This approach will allows us to immediately achieve some significant performance improvements on the existing schemes described in Section 2.2 while keeping the security guarantees of these schemes intact (i.e., the security proofs of those schemes [3] remain valid).

Recall that in these two signature schemes, network coding works modulo a large prime p, of size at least 160 bits. 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 mod p. Here we suggest to keep these schemes unchanged except that the scalars chosen by each intermediate node for performing linear combinations of incoming vectors will be taken from the set $Q = \{0, ..., q-1\}$, for small prime q ($q \approx 256$) (we stress that in this case the linear combinations are still computed mod p).

Before we see the beneficial effect on performance that small coefficients have, we need to make sure that we have a a good decoding probability with such coefficients *even when computations are carried modulo a large prime p.* Lemma 1 shows that this is the case if we work strictly over

[5] If $w = \sum_{j=1}^{\ell} \alpha_j w^{(j)}$, then $w_i = \sum_{j=1}^{\ell} \alpha_j w_i^{(j)} \leq \ell q \max\{w_i^{(1)}, \ldots, w_i^{(\ell)}\}$.

the integers, but what is the effect on the decoding probability when we carry linear combinations modulo p? The answer is that the statement of the lemma essentially holds in this case too provided that the value m (the number of vectors $\bar{v}^{(i)}$) and the maximal path in the network from source to target, that we denote by L, are both negligible relative to 2^k where k is the length of primes from which p is chosen at random (typically, $k \geq 160$). More formally:

Lemma 2 *For any network, the decoding probability of the above scheme (where coefficients for the linear combinations are chosen at random from the set $Q = \{0, 1, \ldots, q\}$, and the linear combinations are performed modulo a random k-bit prime p) is, up to a negligible additive term $O(Lm \log q/2^k)$, the same or better than the decoding probability of a standard random linear coding scheme over a field of size q on the same network.*

Proof: We extend the proof of Lemma 1 to this setting (the only difference is that we now perform the linear operations modulo p). Let U be the matrix generated at a target and whose invertibility defines the successful reconstruction of information. Since we are now performing linear combinations modulo p then this matrix has entries in $\mathbb{Z}_p$. Let's consider a run that led to the matrix U, but this time we will perform the operations over the integers without the modular reduction. We denote by U^* the corresponding matrix in this modified run. Clearly, we have that $U = U^* \bmod p$ (where the modular congruence is entry-wise). We then have that $\det(U) = 0 \bmod p$ if and only if $\det(U^*) = 0$ or $\det(U^*) \neq 0$ but p divides $\det(U^*)$. The case that $\det(U^*) = 0$ is the same as in Lemma 1. Thus, all we need to show is that the probability that $\det(U^*) \neq 0$ but p divides $\det(U^*)$ is negligible.

Since all computations leading to the integer matrix U^* are independent of p, and p is an independent random prime of length k then we can bound the probability that p divides $\det(U^*)$ as follows. Let's denote by d the bit-length of $\det(U^*)$. The number of primes of length k dividing $\det(U^*)$ is at most d/k, and the total number of primes of length k is (up to constant factors) $2^k/k$. Thus the probability to choose p of length k that divides $\det(U^*)$ is bounded by d/k divided by $2^k/k$, i.e., by $d/2^k$. It remains to show that d is negligible relative to 2^k. For this we proceed to bound the value $d = |\det(U^*)|$.

The matrix U^* is composed of the augmented part u of vectors $w = u \parallel v$ received by the target. Recall from Section 3.1 that u coordinates traversing L hops will be integers $\leq (mq)^L$. (In the actual running of the protocol, if the entries u_i grow above p these will be reduced mod p but here we are considering the entries of U^* where the modular operation is omitted.) Thus, U^* is a $m \times m$ matrix with each entry at most $(mq)^L$ and then $\det(U^*) \leq m!(mq)^{Lm} \leq (mq)^{m(L+1)}$, and we have $d = |\det(U^*)| \leq m(L+1)(\log m + \log q)$ which is negligible relative to 2^k (for any practical parameters m, L we will have $mL << 2^{160}$). □

We proceed to examine how the use of small coefficients benefits the performance of the network coding signature schemes discussed in Section 2.2. In subsequent sections we use the small coefficient technique in essential ways to obtain new signature schemes.

Saving bandwidth. Recall that in the two schemes reviewed in Section 2.2 one prepends to each information vector a coding vector u composed of m coordinates each carrying a 160-bit number (here we use 160 instead of $|p|$ for clarity and concreteness). This is in strong contrast to the 8-bit per coordinate in the traditional (non-cryptographic) linear coding schemes. Using small coefficients as we propose alleviates the communication penalty of the cryptographic schemes. Savings are achieved for any size network, however, the savings are more noticeable in those networks where the typical distance from source to target is relatively small (say 20 nodes or less). For an example, let's assume first a case where one adds 10 bits to the size of each u-coordinate for each node traversed (say, we use $q = 253$ and a node has an average of 4 incoming nodes) and the path from source to target has 20 nodes. After the first node is traversed the u coordinates are 10-bit long, after the second they are 20-bit long, etc. After 16 nodes, the coordinate is 160-bit long. Up to this point we have not applied any mod p to the computation of these u coordinates since they were smaller than p. Whenever these coordinates grow above 160 bits the mod p reduction keeps them up to 160 bits (they do not grow further). In total, for the first 16 hops we spent, on average, 80 bits per coordinate which is half the bandwidth consumed by the 160-bit coordinates in the basic schemes described in Section 2.2. The total number of bits saved is $16 \cdot 80 \cdot m$. If each node added 16 bits (assuming a large in-degree per node) the savings would have been $10 \cdot 80 \cdot m$. In general, if each node adds t bits to the coordinates and we have a path length of $160/t$ or more then the savings in bandwidth over the first $160/t$ hops is $80 \cdot 160/t \cdot m$. For shorter paths the savings in total number of bits is smaller but larger in proportion to the total bandwidth. For network with small (say, 10-hop or less) source-target paths, as in some wireless network settings, the bandwidth benefits can be particularly significant.

Note that the coordinates of the v vectors are not affected by the use of small coefficients, these start at $|p|$-bit each and remain of this size throughout the computation.

Saving computation. The savings in the size of the u_i coordinates described above immediately translate to computational savings in the homomorphic hashing based scheme of Section 2.2 as shorter u_i's imply shorter exponents in the verification equation (3). Similar savings apply to the BFKW verification formula (4) from Section 2.2. However, the most impressive computational savings due to the use of small coefficients are achieved in the Combine operation (i.e., signature generation at intermediate nodes) of the BFKW scheme. This operation, of the form $\prod_{i=1}^{\ell} \sigma_i^{\alpha_i}$ is expensive since the exponents α_i are each of size 160. But when applying our small-coefficient strategy we have the α_i's of size 8-bit *thus inducing a 20-fold (!) computational savings in the most critical operation of the scheme* (see the following remark).

Remark 1 Note that signature verification can be done on an opportunistic basis (e.g., for a random subset of vectors) such that if a node fails to identify an invalid packet, other nodes in the vicinity will do so. In contrast, signature computation must be done by *all* nodes for each outgoing packet and hence it constitutes the computational bottleneck of homomorphic signatures.

4 An RSA-Based Network Coding Signature Scheme

In this section we present our RSA-based network coding signature which builds on the ability to perform random linear coding over the integers as described in Section 3.1. The scheme enjoys a proof of security in the random oracle model under the standard RSA assumption, and constitutes the first homomorphic signature scheme based on traditional cryptography (i.e., without relying on bilinear maps). The scheme performs best in networks where the distance from source to targets is not too large (say, up to 20 or 30 hops), where it offers significant performance benefits relative the bilinear-based homomorphic signature schemes.

The basic idea of the scheme is simple, and similar to the BFKW scheme from Section 2.2. Specifically, the idea is to compose a multiplicatively homomorphic signature that works on fixed-length inputs with a multiplicative hash function applied to the vectors in the underlying linear space. The homomorphic hash is similar to the function $\mathbf{H}$ from Section 2.2 but rather than working modulo a prime one works modulo an RSA composite N (more precisely, this hash function uses generators of the cyclic group QR_N of quadratic residues modulo N). The homomorphic signature on fixed length messages is the plain RSA transformation $x^d \bmod N$. The result of this composition is shown in Eq. (5) below. In order to use this function in the context of linear network coding, one needs that the linear space and linear operations applied to network packets correspond to arithmetic modulo $\phi(N)/4$. This is so since for preserving the homomorphic properties, the linear operations translate into operations in the exponent of the generators g_i whose order is $\phi(N)/4$. This, however, presents a difficulty: intermediate nodes in the network that need to compute these linear operations do not know $\phi(N)$ since revealing this value allows to factor N. To overcome this problem we will perform linear network coding over the integers (without ever reducing these values modulo $\phi(N)/4$) as described in Section 3.1.

Following the description in Section 3.1, we are going to represent the information (or file) held by the source S as 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 $\bar{F}$ to be transmitted and the number m of vectors is set by the application, the total length of the information in each of the vectors $\bar{v}^{(i)}$ is determined (we denote this length by $|v|$). Then the value n can be chosen to be any number between 1 and $|v|$. The freedom in choosing n is important since, as we will see later, there is a performance tradeoff related to n: smaller n's save communication while larger n save computation. As in regular network coding, before sending the $\bar{v}^{(i)}$ vectors to the network, the source pre-pends to them unit vectors $\bar{u}^{(i)}$ thus producing $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)} \in \mathbb{Z}^{m+n}$. The rest is carried out as in Section 3.1; namely, intermediate nodes will perform the random linear combinations of incoming packets over the integers with coefficients chosen from the set $Q = \{0, ..., q-1\}$ for a small prime q (say, $q \approx 256$). From Lemma 1 we know that this strategy leads to good decoding probabilities.

We now describe our basic RSA-based homomorphic signature scheme and then proceed to show how to use it in the network coding setting.

4.1 Basic Homomorphic RSA-Based Signature

We start by defining an RSA-based homomorphic signature that acts on vectors of integers of dimension n; we denote this Basic scheme by Bsig.

• RSA Group: $\mathbb{Z}_N^*$ where N is the product of two primes. We require that QR_N, the subgroup of quadratic residues, be cyclic and that random elements in QR_N be generators of the group with high probability. One way to ensure these properties is to choose the factors of N as safe primes.

• Public key: $(N, e, g_1, \ldots, g_n)$ where N is an RSA composite as above, e a public exponent, and $g_1, \ldots, g_n$ are random generators of QR_N.

• Private signing key: d, such that $ed = 1 \bmod \phi(N)$.

• Signature: Let $v = (v_1, \ldots, v_n) \in \mathbb{Z}^n$, we define

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

It is easy to see that this signature is homomorphic: for any $v, v' \in \mathbb{Z}^n$ and $\alpha, \beta \in \mathbb{Z}$, $\mathsf{Bsig}(\alpha v+\beta v') = (\mathsf{Bsig}(v))^\alpha \cdot (\mathsf{Bsig}(v'))^\beta$.

4.2 The Network Coding Signature Nsig

Here we describe how the above signature Bsig is used as a network coding signature that we denote by Nsig (N for "Network coding").

Bounding coordinates. The Nsig scheme below will have a parameter L such that packets that traverse more than L nodes may be rejected; thus, L will serve as an upper bound on the distance from source to targets (more generally, L can be chosen such that the network coding scheme has good decoding probability when restricted to nodes at distance $\leq L$ from the source). Given L we define a bound $B = (mq)^L$, which represents the largest possible coordinate in any vector u transmitted in the network (as discussed in Section 3.1, this this is the maximal value a u_i coordinate can assume after traversing L or less nodes from the source). We use M to denote an upper bound on each of the coordinates of the initial vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$ transmitted by the source (namely, we assume that for $i = 1, \ldots, m$, $\bar{v}^{(i)} \in \{0, \ldots, M\}^n$). Then, the maximal coordinate in a valid vector v transmitted in the network is BM, which we denote by B^*.

The Nsig Scheme.

• Parameters: m, n, M, and B (as defined above) and $B^* = BM$

• Public key: The source (sender) S in the network coding setting has a public key of the form $(N, e, g_1, \ldots, g_n)$ as defined for Bsig; the exponent e is chosen as specified below and the private key d is set accordingly. In addition, the scheme uses a public (deterministic) hash function H that maps arbitrary strings into the set QR_N.

• Public exponent e: The public RSA exponent e is chosen as a prime larger than mB^*. To optimize performance e can be chosen such that its binary representation contains very few 1's (e.g., $2^{\lceil \log mB^* \rceil} + a$ for small integer a); this also allows for a short representation of e.

• Private signing key: d, such that $ed = 1 \bmod \phi(N)$ ($d \leq \phi(N)$ as in regular RSA).

• Signing data by the source. On input a file represented 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 $\mathbb{Z}^{m+n}$ (where $\bar{u}^{(i)}$ is the i-th unit vector). S chooses a random identifier for the file, which we denote by fid, and uses the hash function H to define m values in QR_N, $h_i = H(i, \mathsf{fid})$, $i = 1, \ldots, m$. We extend the definition of the basic signature Bsig above to vectors in $\mathbb{Z}^{m+n}$ of the form $w = (u_1, \ldots, u_m, v_1, \ldots, v_n)$ and define:

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

where the g_j's are part of the public key and the h_i's are defined above as a function of the file identifier fid (namely, the h_i values are file-specific).

The source S computes for each vector $\bar{w}^{(i)}$ its corresponding signature $\mathsf{Nsig}(\bar{w}^{(i)})$, and transmits $\bar{w}^{(i)}$ and its signature along with the file identifier fid.

• Verification $\mathsf{Vrfy}(w, \sigma, S, \mathsf{fid})$. A node receiving a vector $w = u \parallel v = (u_1, \ldots, u_m, v_1, \ldots, v_n)$ in $\mathbb{Z}^{m+n}$, a file identifier fid and a signature value σ proceeds as follows. If any of the u-coordinates is negative or larger than B, or any of the v-coordinates is negative or larger than B^*, reject w as invalid (i.e., set $\mathsf{Vrfy}(w, \sigma, S, \mathsf{fid}) = 0$). Else retrieve the public key $(N, e, g_1, \ldots, g_n)$ of S, compute

$h_i = H(i, \mathsf{fid})$ for $i = 1, \ldots, m$, and accept w as valid (i.e., set $\mathsf{Vrfy}(w, \sigma, S, \mathsf{fid}) = 1$) if and only if

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

• Signature combination by an intermediate node I. Upon receiving ℓ vectors $w^{(1)}, \ldots, w^{(\ell)}$, corresponding to file fid, and their corresponding signatures $\sigma_1, \ldots, \sigma_\ell$, check that $\mathsf{Vrfy}(w^{(i)}, \sigma_i, S, \mathsf{fid}) = 1, i = 1, \ldots, \ell$. Discard those $w^{(i)}$ that do not pass verification and also any $w^{(i)} = u^{(i)} \parallel v^{(i)}$ for which any of the u coordinates is larger than B/mq or any of the v coordinates is larger than B^*/mq.[6] For the non-discarded vectors (for simplicity we denote them by $w^{(1)}, \ldots, w^{(\ell)}$), I applies the $\mathsf{Combine}(w^{(1)}, \ldots, w^{(\ell)}, \sigma_1, \ldots, \sigma_\ell)$ operation defined as: choose random coefficients $\alpha_1, \ldots, \alpha_\ell \in Q = \{0, \ldots, q-1\}$, set $w = \sum_{i=1}^{\ell} \alpha_i w^{(i)}$, and compute the signature on w as:

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

It is easy to see that the signatures generated by the source on the original vectors $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$ pass verification, and so do the signatures generated by honest intermediate nodes.

We prove security of this scheme in the following subsection.

Performance. The Nsig scheme provides better computational performance than the alternative bilinear-based homomorphic signature schemes thanks to its avoidance of pairing computations and the use of small coefficients. This is particularly the case for the schemes that existed before our paper, such as BFKW [3], though the computational gap is reduced when using our own version of BFKW as described in Section 3.2. Even in this case, the cost of pairing computations (and the cost of hashing into the bilinear groups) make Nsig more efficient. Nsig (with small n) is also more efficient regarding the public key size. Regarding bandwidth, Nsig has some bandwidth advantages relative to the original BFKW, at least for moderate-size networks, say 20 to 30 hops, but our own version of BFKW using small coefficients has less communication overhead at least for distances above 15 from the source. We provide a somewhat more detailed analysis next.

Bandwidth: The coordinates of the vectors w transmitted in the network increase by a number of bits equal to $s = \log(mq)$ for each traversed hop. Thus, after t hops each u coordinate will be of size at most ts (remember that the u vectors start as unit vector, i.e., with 0/1 coordinates). Thus, if we take for example $s = 10$ then it will take 32 hops before the total overhead of the u coordinates exceeds that of the *original* BFKW scheme from Section 2.2 (where u coordinates are always of size 160 bits). Thus, with such s and for networks with total length $L \leq 32$, Nsig performs better in the total overhead due to the u vectors. For $s = 16$, which is probably more than needed for practical networks, Nsig will be better up to $L = 20$. In general for moderate size networks Nsig is likely to incur less overhead in the u coordinates. If we compare Nsig to our own improved variant of the BFKW scheme of Section 3.2, using small coefficients, then the two schemes have the same overhead up to the point that the increasing coordinates reach 160 bits; after that the advantage is on the modified BFKW side (since in Nsig coordinates keep growing while in BFKW they do not). The above, however, does not take into account the fact that in Nsig also the v coordinates

[6]These bounds are slightly more restrictive than those required in the verification procedure Vrfy and are intended to ensure that the vector w generated in this step satisfies the coordinate bounds required by Vrfy.

increase while in BFKW they do not. Here, every hop adds s bits to each of the n coordinates in the v vectors, for a total of ns bits per hop. Fortunately, we can choose n to be as small as 1, thus essentially making this overhead insignificant (see more below on the choice of n).

Computation: The most critical operation from performance point of view is signature generation at intermediate nodes (see Remark 1 in Section 3.2); this operation is *very efficient* in Nsig since the exponents α_i in Eq. (8) are small (8 bits each). This is 20 times faster than the corresponding operation in the original BFKW scheme and similar to the cost of this operation in our improved version of BFKW. The more expensive operation is verification at intermediate nodes given by Eq. (7). Its right hand side has an exponentiation with exponent of size at most $|v| + mb + nb$ (where $b = \log B$ is the maximal number of bits by which coordinates increase during the protocol and $|v|$ denotes the total bitsize of each of the vectors $\bar{v}^{(i)}$). The left side has exponent of size $|e| = |mBM| = \log m + b + |v|/n$. To minimize the total size of exponents one computes the minimum respect to n of $|v|/n + bn$ which is attained at $n = \sqrt{|v|/b}$. However, since the value of n is more significant in the way it impacts bandwidth than computation then in most cases it makes sense to choose $n = 1$ (this induces an increase in the size of n by at most $|v|$ bits; however, since the right hand side of Eq. (7) is already larger than $|v|$ then the additional overhead is relatively small – especially considering that due to its sparsity a longer e only incurs in additional squarings). The resultant cost of verification is still better than BFKW's due to the additional costs of BFKW: the pairing operation and the cost of hashing into the bilinear groups (which has the cost of an additional exponentiation per hash operation and is required for computing the values $H(\mathsf{fid}, i)$ in BFKW). Moreover, if one is to decrease the public key size in BFKW using the hash function H to generate the generators $g_1, \ldots, g_n$ then one has to perform n additional expensive hashing operations. If nodes can cache these generators then the hashing expense is reduced. In the case of Nsig the computational cost of the hashing operations is negligible. Moreover, if one uses $n = 1$ in Nsig 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)

4.3 Proof of Security of the Nsig Scheme

We prove that the Nsig scheme is secure according to the security definition [3] presented in Appendix A; informally, the essence of the model is represented by the following game. The attacker $\mathcal{F}$ (for forger) is provided with a public key for the Nsig scheme chosen by the signer running the key setup algorithm. $\mathcal{F}$ chooses files for signing (possibly adaptively), each of which is represented as a set of vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$. For each such file, the signer returns a random file identifier fid (such that for any two different files the corresponding identifiers are different) and the Nsig signatures on the corresponding augmented vectors $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$. Eventually, $\mathcal{F}$ outputs a forgery, namely, a file id fid^*, a vector w^*, and a signature s^*. $\mathcal{F}$ wins if fid^* is one of the identifiers chosen by the signer, s^* is a valid Nsig signature on w^* under file id fid^*, and w^* is *not* in the linear span of the vectors $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$ corresponding to file fid.

Our analysis models the hash function H as random oracle and is based on the RSA assumption: Given a composite N (product of safe primes) an exponent e as defined for Nsig and $C \in_R \mathbb{Z}_N^*$, it is hard to find $C^d \bmod N$. We note that in the proof we use the *equivalent* formulation of the assumption in which C is chosen at random from QR_N rather than from $\mathbb{Z}_N^*$.

Theorem 3 *Under the RSA Assumption and in the random oracle model, the scheme* Nsig *is a secure (homomorphic) network coding signature.*

Proof: Given a forger $\mathcal{F}$ that breaks, with non-negligible probability, the signature scheme Nsig (with parameters as described above), we build an algorithm $\mathcal{S}$ that inverts RSA on $\mathbb{Z}_N^*$. Here $\mathcal{S}$ stands for Simulator and also for Source since $\mathcal{S}$ will be simulating the actions of the source whose signature $\mathcal{F}$ is attacking.

Algorithm $\mathcal{S}$ receives input N, e, C where N, e are distributed as in a Nsig signature and $C \in_R QR_N$. Its goal is to output $C^d \bmod N$ with non-negligible probability.

$\mathcal{S}$ calls $\mathcal{F}$ on an instance of the Nsig scheme where the N, e values are those that $\mathcal{S}$ received as input and the generators $g_1, , \ldots, g_n$ are chosen by $\mathcal{S}$ as follows. $\mathcal{S}$ chooses $i_0 \in_R \{1, \ldots, n\}$ and sets $g_{i_0} = C$ (where C is the challenge input to $\mathcal{S}$). For $i \neq i_0$, $\mathcal{S}$ chooses $r_i \in_R QR_N$, and sets $g_i = r_i^e$.

For each file, or set $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$, chosen by $\mathcal{F}$, $\mathcal{S}$ (now acting as the data source) chooses a random file identifier fid and sets (using the programmability of the random oracle H) the values $h_i = H(i, \mathsf{fid}), i = 1, \ldots, m$, as follows

$$h_i = H(i, \mathsf{fid}) \overset{\text{def}}{=} s_i^e \prod_{j=1}^{n} g_j^{-\bar{v}_j^{(i)}} \text{ where } s_i \in_R QR_N \tag{9}$$

It is easy to verify, by the above choice of $h_1, \ldots, h_m$, that the signature on the vector $\bar{w}^{(i)} = \bar{u}^{(i)} \parallel \bar{v}^{(i)}$ equals s_i (as chosen by $\mathcal{S}$ in Eq. (9)). $\mathcal{S}$ returns to $\mathcal{F}$ the file identifier fid and the signatures $s_1, \ldots, s_m$ corresponding to vectors $\bar{w}^{(1)}, \ldots, \bar{w}^{(m)}$, respectively.

We also specify that if in the above procedure $\mathcal{S}$ chooses fid for which one of the pairs (i, fid), $i = 1, \ldots, m$, was previously queried from the function H, then S aborts.

By assumption, with non-negligible probability, $\mathcal{F}$ outputs a forgery, i.e., a file id fid^*, a vector $w^* \notin Span(\bar{w}^{(1)}, \ldots, \bar{w}^{(m)})$ (where the $\bar{w}^{(i)}$ are the vectors corresponding to fid^*) and a valid signature $s^* = \mathsf{Nsig}(w^*)$ on w^* (i.e., a valid Nsig signature under the public key values set by $\mathcal{S}$).

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

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

It is easy to see that z has the form $(0, \ldots, 0, z_1, \ldots, z_n)$, namely, all its first m coordinates are zero (remember that the $\bar{u}^{(i)}$ part in the vectors $\bar{w}^{(i)}$ are unit vectors and hence $\sum_{i=1}^{m} u_i^* \bar{u}^{(i)} = (u_1^*, \ldots, u_m^*)$). Moreover, since $w^* \notin Span(\bar{w}^{(1)}, \ldots, \bar{w}^{(m)})$ then it must be that at least one of the values z_i is non-zero (otherwise we would have $z^* = 0$ and then $w^* = \sum_{i=1}^{m} u_i^* \bar{w}^{(i)}$). Thus, with probability at least $1/n$, $z_{i_0} \neq 0$ (remember that $\mathcal{S}$ chose i_0 at random in $\{1, \ldots, n\}$).

We now show how $\mathcal{S}$ can find $C^d \bmod N$ given the above information. Note by the definition of z^* and the homomorphic property of Nsig we have

$$\mathsf{Nsig}(z^*) = \mathsf{Nsig}(w^* - \sum_{i=1}^{m} u_i^* \bar{w}^{(i)}) = \mathsf{Nsig}(w^*) \prod_{i=1}^{m} \mathsf{Nsig}(\bar{w}^{(i)})^{-u_i^*} = s^* \prod_{i=1}^{m} s_i^{-u_i^*} \tag{11}$$

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

$$\mathsf{Nsig}(z^*) = \mathsf{Nsig}(0, \ldots, 0, z_1, \ldots, z_n) = \Big(\prod_{i=1}^{m} h_i^0 \prod_{i=1}^{n} g_i^{z_i} \Big)^d = (C^{z_{i_0}})^d \prod_{i \neq i_0} (g_i^{z_i})^d = (C^{z_{i_0}})^d \prod_{i \neq i_0} r_i^{z_i} \tag{12}$$

Combining (11) and (12) we get that

$$(C^{z_{i_0}})^d = s^* \prod_{i=1}^{m} s_i^{-u_i^*} \prod_{i \neq i_0} r_i^{-z_i} \tag{13}$$

Since $\mathcal{S}$ knows all the values on the right-hand side of (13) it can compute $s' = (C^{z_{i_0}})^d \bmod N$.

Now, using the well-known "Shamir's trick", we can derive the value $s = C^d \bmod N$ from s' provided that $gcd(z_{i_0}, e) = 1$. Indeed, choosing a, b such that $ae + bz_{i_0} = 1$, it can be verified that $s = C^d \bmod N = C^a s'^b$ which $\mathcal{S}$ can compute.

Thus, $\mathcal{S}$ inverts C provided that z_{i_0} is coprime to e. But this is the case since e is prime and $-e < z_{i_0} < e$. Indeed, since w^* passes verification then it must be that $0 \leq u_i^* \leq B, 0 \leq v_i^* \leq MB$. From Eq. (10) it follows, for $j = 1, \ldots n$, that $z_j = v_j^* - \sum_{i=1}^{m} u_i^* \bar{w}_j^{(i)} \geq -\sum_{i=1}^{m} BM \geq -mBM \geq -e$ where the last inequality follows the definition of e. The case $z_j \leq e$ is similar. □

Remark. Note that the above proof uses the fact that we choose the exponent e larger than any coordinate. Interestingly, this is not just an artifact of the proof but essential for security. If coordinates are allowed to grow above e then an attacker can generate a signature for a non-valid vector as follows. Given a valid vector $w = u||v$ with valid signature σ, the attacker can generate a valid signature σ' for $w' = u||v'$ where $v' = v + (e, 0, \ldots, 0)$ by setting $\sigma' = \sigma \cdot g_1$. This will generate a valid signature even though $w' \notin Span(\bar{w}^{(1)}, \ldots, \bar{w}^{(m)})$.

5 Homomorphic Hashing Modulo a Composite

As shown in Section 2.2, a homomorphic collision-resistant hash function that acts on a linear space can be used to build a network coding signature. The idea is that a node that holds the authenticated hash values corresponding to the original information vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$, can validate any incoming (augmented) vector by comparing the hash of this vector with the corresponding (homomorphic) combination of the original hash values as described in Eq. (3). Network coding signatures based on homomorphic hashing, where nodes do not generate signatures on outgoing vectors, offer significant computational advantages over fully homomorphic signature schemes. In particular, a node that chooses not to verify an incoming vector (see Remark 1) does not need to take any cryptographic action. In addition, schemes based solely on homomorphic hashing are often more efficient for signature verification (for example, avoiding a costly pairing computation or a long RSA exponentiation). On the other hand, these schemes require that verifying nodes obtain, for each file, the authenticated hash values of the original vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$ (i.e., m hash values plus the source's signature on them). In settings where delivering these authenticated values to nodes, possibly in some off-line way[7], is practical, homomorphic hash schemes provide for attractive alternatives to network coding signatures based on homomorphic signatures.

In this section we study a homomorphic hashing scheme, denoted $\mathbf{H}_N$, that is similar to the EHH scheme in Section 2.2 but works in a network coding framework similar to the RSA case. That is, the hash function is computed modulo a composite N and the packet vectors (and their linear combinations) are defined over the integers. The resultant scheme is significantly more efficient than the RSA scheme of Section 4 and also computationally more efficient than the EHH scheme

[7]Assuming that the m hash values are much shorter than the m vectors to be transmitted, their delivery can be done by alternative means such as point-to-point or broadcast mechanisms, or at the time when nodes register to receive some content (such in the case of peer-to-peer networks)

(over prime order groups) from Section 2.2. And, while it produces larger hash values, it requires a minimal public key (just an RSA modulus) without the overhead of the many generators of EHH.

We use the linear network coding setting of Section 3.1 (also used in Section 4), namely, working over the integers and with random linear coefficients taken from a set $Q = \{0, \ldots, q-1\}$ for small prime q. In particular, the file to be transmitted is represented by m n-dimensional integer vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)} \in \mathbb{Z}^n$. For concreteness, the reader can think of $n = 1$ and each vector $\bar{v}^{(i)}$ being a large integer (indeed, this will be the most efficient instantiation of our scheme).

Let's consider the following simple adaptation to the composite case of the EHH scheme from Section 2.2. Let N be the product of two safe primes so that the group QR_N of quadratic residues modulo N is cyclic, and let $g_1, \ldots, g_n$ be generators of QR_N. For $v = (v_1, \ldots, v_n) \in \mathbb{Z}^n$ we define (analogously to Eq. (2))

$$\mathbf{H}_N(v) = \prod_{j=1}^{n} g_j^{v_j} \bmod N. \tag{14}$$

For $v \in \mathbb{Z}^n$ this is a homomorphic hash function that is collision resistant if factoring N is hard. 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 as

$$\prod_{i=1}^{m} h_i^{u_i} \stackrel{?}{=} \mathbf{H}_N(v) = \prod_{j=1}^{n} g_j^{v_j} \bmod N \tag{15}$$

where $h_1, \ldots, h_m$ are the hash values of the information vectors $\bar{v}^{(1)}, \ldots, \bar{v}^{(m)}$.

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; one 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 scheme from Section 4, one way to limit the effect of coordinate growth in total communication is by setting $n = 1$. As we see below, not only this reduces bandwidth overhead but also improves computational performance significantly.

Indeed, we are going to represent each block of information $\bar{v}^{(i)}$ (and subsequent vectors v transmitted in the network) as a *single* (long) integer so that $n = 1$. This is possible since the function $g^v \bmod N$ is collision resistant even for very large integers and for a single generator g (security follows from the hardness of factorization since if values v and v' collide then $v - v'$ is a multiple of $\phi(N))/4$ which suffices to factor N). Moreover, by choosing N appropriately[8], we can *fix* the single generator of QR_N to the value 2, thus obtaining:

$$\mathbf{H}_N(v) = 2^v \bmod N, \; v \in \mathbb{Z} \tag{16}$$

Not only does this reduce communication overhead but it provides the most salient advantage of the $\mathbf{H}_N$ scheme, namely, *fast exponentiation.* Compare this to the prime order group scheme of Eq. (2) which uses n random generators as the bases (and due to n being typically large one cannot use multi-base optimizations). This implies a significant speedup of the hash verification at intermediate nodes when using the $\mathbf{H}_N$ scheme in the context of network coding.

Yet another advantage of this homomorphic hash is that it improves considerably the size of public parameters relative to the prime order group implementation from Section 2.2. To see this,

[8]Choose $N = pq$ where $p = 2p' + 1, q = 2q' + 1$ with p, q, p', q' are all primes, and p', q' are congruent to 3 mod 8 and p, q are congruent to 7 mod 8. This ensures that 2 is a quadratic residue in QR_p and since QR_p is of prime order then 2 is also a generator of QR_p. Same holds for QR_q, and therefore 2 is a generator of QR_N.

let's first note that in the prime order case of Eq. (2), the total length of the set of generators $g_1, \ldots, g_n$ is $n|p|$ which is as large as each information vector $\bar{v}^{(i)}$. Indeed, each individual g_i can only hash values smaller than p and it must be chosen at random to ensure that finding collisions is as hard as computing discrete logarithms. Also, the number of generators is usually very large, e.g., for vectors $\bar{v}^{(i)}$ of size 4KB and p of 160 bits one needs 200 random generators. In the case of $\mathbf{H}_N$ the generator is fixed (to 2) so it does not require extra public parameters except for N.

We note that in the prime order case, one can avoid including explicitly the values $g_1, \ldots, g_n$ in the description of the public parameters if these generators are derived using a fixed hash function. This, however, looses one of the security advantages of the homomorphic hash scheme described in Section 2.2, namely, the fact that security could be proven without relying on the random oracle model. Also, note that in implementations that use elliptic curves a hash operation may cost as much as a full exponentiation. Working modulo N dispenses of these extra complexities.

In all we have that the homomorphic hashing scheme from Eq. (16) leads to a computationally efficient and secure network coding signature scheme whose security can be proven solely based on the factoring assumption in the standard model.

Acknowledgments

The views and conclusions contained in this document 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 U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints 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] D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *Public Key Cryptography (PKC)*, 2009.

[4] 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.*

[5] P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In *41st Allerton Conference on Communication, Control, and Computing*, 2003.

[6] C. Gkantsidis and P. Rodriguez. Cooperative security for network coding file distribution. In *Proc. of IEEE INFOCOM 2006*, pages 1–13, 2006.

[7] 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.

[8] 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.

[9] T. Ho and D. Lun. *Network Coding: An Introduction.* Cambridge University Press, 2008.

[10] Tracey Ho, Muriel Médard, Ralf Koetter, David R. Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Trans. Inform. Theory*, 52(10):4413–4430, 2006.

[11] S. Jaggi. *Design and Analysis of Network Codes.* PhD thesis, California Institute of Technology, 2006.

[12] 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.

[13] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *Proc. of CT-RSA 2002*, volume 2271 of *Springer LNCS*, pages 244–262, 2002.

[14] M. Krohn, M. Freedman, and D. Mazieres. On the-fly verification of rateless erasure codes for efficient content distribution. In *Proc. of IEEE Symposium on Security and Privacy*, pages 226–240, 2004.

[15] Shuo-Yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear network coding. *IEEE Trans. Inform. Theory*, 49(2):371–381, 2003.

[16] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan. An efficient signature-based scheme for securing network coding against pollution attacks. In *INFOCOM*, 2008.

[17] Fang Zhao, Ton Kalker, Muriel Médard, and Keesook Han. Signatures for content distribution with network coding. In *Proc. of International Symposium on Information Theory (ISIT)*, 2007.

A Formal Definitions

For convenience, we repeat the definitions from [3]. We first define the general notion of a network coding signature scheme.

Definition 4 *A* network coding signature scheme *is a triple of probabilistic, polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vrfy})$ *with the following functionality:*

- $\mathsf{Setup}(1^k, N)$. *Given a security parameter* 1^k *and an integer* N *(denoting the length of vectors to be signed*[9]*), this algorithm outputs a public key* PK *and a secret key* SK*. We assume* PK *and* SK *contain* N*, and implicitly define a field* $\mathbb{F}$*.*
- $\mathsf{Sign}(SK, \mathsf{fid}, V)$. *Given a secret key* SK*, a file identifier* $\mathsf{fid} \in \{0,1\}^*$*, and an* m*-dimensional subspace* $V \subset \mathbb{F}^N$*, with* $0 < m \leq N$*, described as a set of basis vectors* $v_1, \ldots, v_m$*, this algorithm outputs a signature* σ*.*

[9] In our treatment N corresponds to $m + n$.

- $\mathsf{Vrfy}(PK, \mathsf{fid}, y, \sigma)$. *Given a public key PK, an identifier* $\mathsf{fid} \in \{0,1\}^*$, *a vector $y \in \mathbb{F}^N$, and a signature σ, this algorithm output either 0 (reject) or 1 (accept).*

We require that for all k, N and each (PK, SK) output by $\mathsf{Setup}(1^k, N)$, *for all subspaces $V \subseteq \mathbb{F}^N$, and for all* $\mathsf{fid} \in \{0,1\}^*$, *if* $\sigma \leftarrow \mathsf{Sign}(SK, \mathsf{fid}, V)$ *then* $\mathsf{Vrfy}(PK, \mathsf{fid}, y, \sigma) = 1$ *for any $y \in V$.*

We next define security of such a scheme.

Definition 5 *A network coding signature scheme* $\mathcal{S} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* secure *if the success probability of any probabilistic, polynomial-time adversary $\mathcal{A}$ in the following experiment is negligible in the security parameter k:*

Setup. *The adversary $\mathcal{A}$ sends a positive integer N to the challenger. The challenger runs* $\mathsf{Setup}(1^k, N)$ *to obtain (PK, SK), and sends PK to $\mathcal{A}$.*

Queries. *Proceeding adaptively, $\mathcal{A}$ specifies vector subspaces $V_i \subset \mathbb{F}^N$. The challenger chooses* $\mathsf{fid}_i \in \{0,1\}^k$ *uniformly at random, and sends* fid_i *and* $\sigma_i \leftarrow \mathsf{Sign}(SK, \mathsf{fid}_i, V_i)$ *to $\mathcal{A}$. (We assume all the* $\{\mathsf{fid}_i\}$ *are distinct.)*

Output. *$\mathcal{A}$ outputs an identifier* $\mathsf{fid}^* \in \{0,1\}^k$, *a signature σ^*, and a vector $y^* \in \mathbb{F}^N$.*

For $\mathsf{fid} \in \{0,1\}^k$ *define*

$$V(\mathsf{fid}) = \begin{cases} V_i & \mathsf{fid} = \mathsf{fid}_i \textit{ for some } i \\ \{\mathbf{0}\} & \textit{otherwise} \end{cases}$$

$\mathcal{A}$ succeeds *if* $\mathsf{Vrfy}(PK, \mathsf{fid}^*, y^*, \sigma^*) = 1$ *and* $y^* \notin V(\mathsf{fid}^*)$.

We also define the notion of a *homomorphic* network coding signature scheme, which can be used to construct a network coding signature scheme in the obvious way.

Definition 6 *A* homomorphic network coding signature scheme *is defined by a tuple of probabilistic, polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vrfy})$ *with the following functionality:*

- $\mathsf{Setup}(1^k, N)$. *Given a security parameter 1^k and an integer N (denoting the length of vectors to be signed), this algorithm outputs a public key PK and a secret key SK. We assume PK and SK contain N, and implicitly define a field $\mathbb{F}$.*

- $\mathsf{Sign}(SK, \mathsf{fid}, v)$. *Given a secret key SK, a file identifier* $\mathsf{fid} \in \{0,1\}^*$, *and a vector $v \in \mathbb{F}^N$, this algorithm outputs a signature σ.*

- $\mathsf{Combine}(PK, \mathsf{fid}, \{(\alpha_i, \sigma_i)\}_{i=1}^{\ell})$. *Given a public key PK, a file identifier* fid, *and ℓ pairs consisting of a scalar $\alpha_i \in \mathbb{F}$ and a signature σ_i, the algorithm outputs a signature σ. (Looking ahead, if each σ_i is a valid signature on some v_i, then σ will be a valid signature on $\sum_{i=1}^{\ell} \alpha_i v_i$.)*

- $\mathsf{Vrfy}(PK, \mathsf{fid}, y, \sigma)$. *Given a public key PK, an identifier* $\mathsf{fid} \in \{0,1\}^*$, *a vector $y \in \mathbb{F}^N$, and a signature σ, this algorithm outputs either 0 (reject) or 1 (accept).*

We require that for all k, N and each (PK, SK) output by $\mathsf{Setup}(1^k, N)$, *the following hold:*

- *For all* $\mathsf{fid} \in \{0,1\}^*$ *and all $y \in \mathbb{F}^N$, if* $\sigma \leftarrow \mathsf{Sign}(SK, \mathsf{fid}, y)$ *then* $\mathsf{Vrfy}(PK, \mathsf{fid}, y, \sigma) = 1$.

- *For all* $\mathsf{fid} \in \{0,1\}^*$, *any $\ell > 0$, and all sets of triples $\{(\alpha_i, \sigma_i, v_i)\}_{i=1}^{\ell}$, if it holds that* $\mathsf{Vrfy}(PK, \mathsf{fid}, v_i, \sigma_i) = 1$ *for all i, then it must be the case that*

$$\mathsf{Vrfy}\left(PK,\ \mathsf{fid},\ \textstyle\sum_i \alpha_i v_i,\ \mathsf{Combine}(PK, \mathsf{fid}, \{(\alpha_i, \sigma_i)\})\right) = 1.$$