Preventing Pollution Attacks in Multi-Source Network Coding

Shweta Agrawal[1⋆] Dan Boneh[2⋆⋆] Xavier Boyen[2⋆⋆⋆] and David Freeman[3†]

[1] University of Texas at Austin, shweta.a@gmail.com.
[2] Stanford University, {dabo,xb}@cs.stanford.edu
[3] CWI and Universiteit Leiden, freeman@cwi.nl

Abstract. Network coding is a method for achieving channel capacity in networks. The key idea is to allow network routers to linearly mix packets as they traverse the network so that recipients receive linear combinations of packets. Network coded systems are vulnerable to pollution attacks where a single malicious node floods the network with bad packets and prevents the receiver from decoding correctly. Cryptographic defenses to these problems are based on homomorphic signatures and MACs. These proposals, however, cannot handle mixing of packets from multiple sources, which is needed to achieve the full benefits of network coding. In this paper we address integrity of multi-source mixing. We propose a security model for this setting and provide a generic construction.

1 Introduction

Network coding [2, 17] is an elegant technique that replaces the traditional "store and forward" paradigm of network routing by a method that allows routers to transform the received data before re-transmission. It has been established that for certain classes of networks, *random linear* coding is sufficient to improve throughput [11]. In addition, linear network codes offer robustness and adaptability and have many practical applications (in wireless and sensor networks, for example) [10]. Due to these advantages, network coding has become very popular.

On the other hand, networks using network coding are exposed to problems that traditional networks do not face. A particularly important instance of this is the *pollution* problem: if some routers in the network are malicious and forward invalid combinations of received packets, then these invalid packets get mixed with valid packets downstream and quickly pollute the whole network. In addition, the receiver who obtains multiple packets has no way of ascertaining which of these are valid and should be used for decoding. Indeed, using even one invalid packet during the decoding process causes all the messages to be decoded wrongly. For a detailed discussion of pollution attacks, we refer the reader to [4, 19, 12].

To prevent the network from being flooded with invalid packets, it is desirable to have "hop-by-hop containment." This means that even if a bad packet gets injected into the network, it is detected and discarded at the very first hop. Thus, it can be dropped before it is combined with any other packets, preventing its pollution from spreading.

Hop-by-hop containment cannot be achieved by standard signatures or MACs. As pointed out in [1], signing the message packets does not help since recipients do not have the original message packets and therefore cannot verify the signature. Nor does signing the entire message prior to transmission work, because it forces the recipient to decode exponentially many subsets of received packets to find a decoded message with a consistent signature. Thus, new integrity mechanisms are needed to mitigate pollution attacks.

⋆ Supported by DARPA IAMANET.
⋆⋆ Supported by DARPA IAMANET and NSF.
⋆⋆⋆ Supported by DARPA IAMANET.
† Supported by an NSF International Research Fellowship, with additional support from the Office of Multidisciplinary Activities in the NSF Directorate for Mathematical and Physical Sciences

Previous Work. Security of network coding has been considered from both the information theoretic and cryptographic perspectives. In the former, the adversary is modelled as having control over a limited number of links in the network. Such approaches, though useful for wireline networks, have limited application in wireless networks. For a detailed discussion of these techniques, see for example [6, 9, 13, 14]. Cryptographic techniques have also been proposed, for example in [7, 18, 19, 4]. These authors construct digital signatures for signing a linear subspace. If V is a subspace and σ its signature, then there is a verification algorithm which accepts the pair $(\mathbf{v}, \sigma)$ for all $\mathbf{v} \in V$, but it is difficult to construct a vector $\mathbf{y} \notin V$ for which the pair $(\mathbf{y}, \sigma)$ verifies. An alternative approach is to use a MAC (instead of a signature) for integrity of a linear subspace; see [1] for a construction.

While the signature and MAC schemes in [7, 18, 19, 4, 1] are elegant, they are quite limited: they only allow routers to combine vectors from a single sender. (Furthermore, the constructions of [7, 18, 19] require a new public key to be generated for each file, thus compromising efficiency.) Traditional network coding assumes a network where there are many senders each simultaneously sending messages and network routers linearly combine vectors from multiple senders. This setting is essential in showing that network coding can improve the efficiency of 802.11 wireless networks [15].

Our Contribution. Our goal is to construct a signature mechanism that provides integrity when network routers combine packets from many sources. This problem is considerably harder than the single source problem. First, defining security is more difficult. It is necessary to model "insider" attacks where the attacker controls network routers as well as some senders. The attacker's goal is to generate valid signatures on mixed packets; after decoding these packets the recipient believes that an honest sender sent a message M^* that was never sent by the honest sender.

More precisely, if there are s senders in the network, we allow the attacker to control $s-1$ of them. Furthermore, the attacker can mount a chosen message attack on the single honest sender. The attacker's goal is to generate a mixed packet with a valid signature that after decoding corresponds to an existential forgery on the single honest sender.

In Section 3 we show that a natural generalization of the single-sender security model in [4] to the multi-sender setting results in a model that cannot be satisfied. We do this by constructing a generic attack against an abstract multi-source network coding signature scheme. In Section 4 we present a security model that captures the constraints of the multi-sender problem. Our model retains the desirable properties of the single-source model, such as hop-by-hop containment of forged packets, and is achievable.

In Section 5 we present a construction satisfying our security model. We give a generic construction from a new primitive called a *vector hash*, and we show how to instantiate the construction based on the discrete logarithm assumption. In Section 6 we prove a lower bound that shows that our model necessitates a relatively space-inefficient construction; our discrete log scheme (asymptotically) achieves this lower bound.

2 Network Coding

We refer the reader to [17] for a detailed introduction to network coding. Here, we present a brief overview for completeness. We model a network as a directed graph consisting of a set of vertices (or *nodes*) V and a set of edges E. We assume the graph is connected. A node that only transmits data is called a *source node*. We start with the basic model, in which one source wishes to transmit one file F through the network. The source interprets the data in F as a set of m vectors $\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_m$ in an n-dimensional vector space over a finite field $\mathbb{F}_p$. (The prime p and the dimensions n and m are fixed parameters in the system.) We sometimes refer to individual vectors as *blocks* or *packets*. The source then appends a unit vector of length m to the vectors $\hat{\mathbf{v}}_i$ to create m *augmented vectors* $\mathbf{v}_1, \ldots, \mathbf{v}_m$ given by

$$\mathbf{v}_i = (—\hat{\mathbf{v}}_i—, \overbrace{\underbrace{0, \ldots, 0, 1}_{i}, 0, \ldots, 0}^{m}) \in \mathbb{F}_p^{n+m}.$$

The augmented vectors comprise the data to be transmitted through the network. We call the first n entries of the vector $\mathbf{v}_i$ the *data component* and the last m entries the *augmentation component*.

The "coding" part of network coding works as follows: an intermediate node in the network receives some set of vectors $\mathbf{w}_1, \ldots, \mathbf{w}_\ell$, chooses ℓ random elements $\overline{\beta}_i \in \mathbb{F}_p$, and transmits the vector $\mathbf{y} = \sum_{i=1}^{\ell} \overline{\beta}_i \mathbf{w}_i$ along its outgoing edges. The key property of the augmentation is that the augmentation component contains exactly the linear combination coefficients used to construct $\mathbf{y}$. That is, we know that $\mathbf{y} = \sum_{i=1}^{m} y_{n+i} \mathbf{v}_i$ even though the intermediate node may never see the $\mathbf{v}_i$. This property allows any node that receives a set of m linearly independent vectors $\mathbf{y}_1, \ldots, \mathbf{y}_m$ to recover the original $\mathbf{v}_i$. Specifically, if we let D be $m \times n$ matrix whose ith row consists of the data component of $\mathbf{y}_i$, and A be the $m \times m$ matrix whose ith row consists of the augmentation component of $\mathbf{y}_i$, then the rows of $A^{-1}D$ are exactly the initial vectors $\hat{\mathbf{v}}_i$.

Since network coding consists of linearly combining vectors, the subspace spanned by the (augmented) vectors of a file remains invariant under network operations. Hence we can equivalently consider a file to be represented by the subspace spanned by the vectors that comprise it.

Notation: We use n to denote the dimension of the data space and m to denote the dimension of a vector subspace that represents a single file. The number of files in the system is usually denoted by f. For $\mathbf{v} \in \mathbb{F}_p^{n+\ell}$, we will use $\hat{\mathbf{v}}$ to denote the data component of $\mathbf{v}$, i.e., the first n coordinates of $\mathbf{v}$, and $\overline{\beta}_{\mathbf{v}}$ to denote the augmentation component of $\mathbf{v}$, i.e., the remaining ℓ coordinates. When we use a vector space V as input to or output of an algorithm we assume that V is described by an explicit basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_\ell\}$. A basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_\ell\}$ is said to be *properly augmented* if for $i = 1, \ldots, \ell$, the augmentation component $\overline{\beta}_{\mathbf{v}_i}$ is the unit vector $\mathbf{e}_i$ with a 1 in the ith position.

We will refer to the augmented vectors that the source wishes to transmit as *primitive* vectors. Here, "primitive" alludes to the fact that these vectors have not been mixed with any other; their augmentation components are unit vectors. *Aggregate* vectors, on the other hand, refer to vectors that have been formed as a result of linearly combining primitive or other aggregate vectors.

2.1 Multiple sources

We now consider the situation where the network has multiple sources, each of which can transmit multiple files into the network. In this description, we are assuming that all nodes in the network are honest. In principle the network coding setup is the same as in the single-source situation described above, but there is some more bookkeeping to do. The complication arises from the fact that the intermediate nodes wish to combine vectors from files produced by different sources, but each source knows nothing of what the other sources are doing.

In the single source case, each file is associated with a file identifier id. The identifier allows the receiver to group together packets that belong to the same file. This prevents, for example, delayed honest packets from a previous file transmission from being decoded along with the current file's vectors. Hence each vector (primitive or aggregate) that traverses the system carries with it the identifier of the file it belongs to.

In the multi-source case, the file identifier id plays an even more crucial role — it allows the intermediate nodes to combine vectors arising from different files. In this scenario, an aggregate vector may be associated with multiple files, and the identifier attached to an aggregate vector $\mathbf{v}$ must carry with it the identifiers of all of the files whose vectors went into making $\mathbf{v}$. Upon receiving

two vectors, where each vector contains a (probably different) list of identifiers $\overline{\mathsf{id}}$, an intermediate node will need to "merge" the lists of identifiers to a common list and adjust the two vectors' augmentation components so that they can be linearly combined.

For example, suppose a node receives two vectors $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{F}_p^{n+m}$ with identifiers id_1 and id_2, respectively. Splitting $\mathbf{v}_i$ into its data and augmentation components, we write $\mathbf{v}_i = (\hat{\mathbf{v}}_i, \mathbf{a}_i)$. If $\mathsf{id}_1 = \mathsf{id}_2$ then the vectors come from the same file and the situation is analogous to the single source case (hence no additional adjustment is needed). However, if $\mathsf{id}_1 \neq \mathsf{id}_2$ then the vectors came from different files and we have to introduce additional augmentation before we can linearly combine the vectors. In this case we define $\mathbf{v}_1' = (\hat{\mathbf{v}}_1, \mathbf{a}_1, \mathbf{0})$ and $\mathbf{v}_2' = (\hat{\mathbf{v}}_2, \mathbf{0}, \mathbf{a}_2) \in \mathbb{F}_p^{n+2m}$, where $\mathbf{0}$ denotes a length-m zero vector. Thus when we compute a linear combination $\mathbf{v} = a\mathbf{v}_1' + b\mathbf{v}_2'$, the data components are mixed together but the augmentation coefficients remain separate. We can then use the identifier $\overline{\mathsf{id}} = (\mathsf{id}_1, \mathsf{id}_2)$ to indicate which set of augmentation coefficients correspond to which file.

More generally, we define an algorithm Merge that merges the lists of identifiers contained in aggregate vectors and adjusts the vectors' augmentations. This algorithm is intrinsic to the multiple-source setting: the algorithm does not itself linearly combine vectors, but rather it *prepares* aggregate vectors (coming from different sources, made up of different files) to be mixed together. If $\mathbf{v} \in \mathbb{F}_p^{n+mf}$ is an aggregate vector, we continue to call the first n entries of $\mathbf{v}$ the data component; we call the rest of $\mathbf{v}$ the augmentation component, and we divide the augmentation component into f *augmentation blocks* of length m.

Algorithm 1 (Merge).
Input: lists of identifiers $\overline{\mathsf{id}}_1, \overline{\mathsf{id}}_2$ of lengths f_1, f_2, respectively, and vectors $\mathbf{w}_i \in \mathbb{F}_p^{n+mf_i}$ for $i = 1, 2$.
Output: two vectors $\mathbf{w}_1', \mathbf{w}_2' \in \mathbb{F}_p^{n+mf'}$ and a list of identifiers $\overline{\mathsf{id}}'$ of length f', or the symbol $\perp$.

1. If either $\overline{\mathsf{id}}_i$ contains a repeated entry, output $\perp$.
2. Let $\overline{\mathsf{id}}'$ be the list whose entries are the union of the elements of $\overline{\mathsf{id}}_1$ and $\overline{\mathsf{id}}_2$, ordered in some pre-determined way (e.g. lexicographically). Let f' be the length of $\overline{\mathsf{id}}'$.
3. For i in $1, 2$, define $\mathbf{w}_i' \in \mathbb{F}_p^{n+mf'}$ by setting the data component of $\mathbf{w}_i'$ equal to the data component of $\mathbf{w}_i$, and for j in $1, \ldots, f'$, setting the jth augmentation block of $\mathbf{w}_i'$ as follows:
   - If the jth element of $\overline{\mathsf{id}}'$ is the kth element of $\overline{\mathsf{id}}_i$, the jth augmentation block of $\mathbf{w}_i'$ is equal to the kth augmentation block of $\mathbf{w}_i$.
   - If the jth element of $\overline{\mathsf{id}}'$ is not an element of $\overline{\mathsf{id}}_i$, the jth augmentation block of $\mathbf{w}_i'$ is $\mathbf{0}$.
4. Output the list $\overline{\mathsf{id}}'$ and the vectors $\mathbf{w}_1', \mathbf{w}_2'$.

The intermediate node can now compute a random linear combination $\mathbf{y}$ of the $\mathbf{w}_1'$ and $\mathbf{w}_2'$ and use the list $\overline{\mathsf{id}}'$ as the identifier component of the signature on $\mathbf{y}$. (In the example above we executed this algorithm on two vectors each with an identifier list of length $f_i = 1$.)

We also define an algorithm called MergeSpaces that uses the Merge algorithm to combine two files described as vector spaces.

Algorithm 2 (MergeSpaces).
Input: disjoint lists of identifiers $\overline{\mathsf{id}}_1, \overline{\mathsf{id}}_2$ and two vector spaces $V = \operatorname{span}(\mathbf{v}_1, \ldots, \mathbf{v}_k) \subset \mathbb{F}_p^{n+k}$ and $W = \operatorname{span}(\mathbf{w}_1, \ldots, \mathbf{w}_\ell) \subset \mathbb{F}_p^{n+\ell}$.
Output: a subspace $Z \subset \mathbb{F}_p^{n+k+\ell}$ and an identifier $\overline{\mathsf{id}}'$.

1. Let B be the set of nonzero vectors produced by

$$\mathsf{Merge}(\overline{\mathsf{id}}_1, \overline{\mathsf{id}}_2, \mathbf{v}_1, \mathbf{0}), \ldots, \mathsf{Merge}(\overline{\mathsf{id}}_1, \overline{\mathsf{id}}_2, \mathbf{v}_k, \mathbf{0}), \mathsf{Merge}(\overline{\mathsf{id}}_1, \overline{\mathsf{id}}_2, \mathbf{0}, \mathbf{w}_1), \ldots, \mathsf{Merge}(\overline{\mathsf{id}}_1, \overline{\mathsf{id}}_2, \mathbf{0}, \mathbf{w}_\ell).$$

2. Let $\overline{\mathsf{id}}'$ be the identifier output by any of the calls to Merge in Step (1).
3. Output $Z = \text{span}(B)$ and $\overline{\mathsf{id}}'$.

By applying MergeSpaces repeatedly using concatenated lists of identifiers, the algorithm generalizes to take any number of vector spaces and identifiers as input.

3 Signatures and File Identifiers

For single sources, a network coding signature scheme consists of three algorithms, Setup, Sign, and Verify, whose functionality correspond to the usual notions for a signature scheme. In this setting, the Sign algorithm produces signatures on a *vector space*, and the Verify algorithm checks whether the signature is valid on a given *vector*. In addition, both Sign and Verify take as additional input a file identifier id, which binds a signature to a file. Informally, the correctness condition is that if σ is a signature on a vector space V with identifier id, then for all $\mathbf{v} \in V$, $\mathsf{Verify}(\mathsf{id}, \mathbf{v}, \sigma)$ outputs "accept." (For formal definitions, see [4, Section 3.1].)

For multiple sources, we need to add an additional algorithm Combine that will be used by intermediate routers to produce signatures on vectors that are linear combinations of vectors from different files. More precisely, Combine takes as input two tuples $(\mathbf{v}_i, \overline{\mathsf{id}}_i, \sigma_i, a_i)$ for $i = 1, 2$, where $\mathbf{v}_i$ are vectors, $\overline{\mathsf{id}}_i$ are (lists of) identifiers, σ_i are signatures, and a_i are network coding coefficients. The algorithm outputs a signature σ'. The correctness condition is that if σ_i is a valid signature on $\mathbf{v}_i$ with identifier $\overline{\mathsf{id}}_i$ for $i = 1, 2$, then σ' is a valid signature on $a_1\mathbf{v}'_1 + a_2\mathbf{v}'_2$ with identifier $\overline{\mathsf{id}}'$, where $\mathbf{v}'_1, \mathbf{v}'_2, \overline{\mathsf{id}}'$ are output by $\mathsf{Merge}(\overline{\mathsf{id}}_1, \overline{\mathsf{id}}_2, \mathbf{v}_1, \mathbf{v}_2)$.

In the single-source setting, the Sign algorithm takes id as input. Thus, a vector $\mathbf{v}$ carries a pair (id, σ) where id is the file identifier chosen arbitrarily, and σ is generated by Sign. In the multi-source case however, allowing senders to pick file identifiers gives them the ability to frame other users in the system, so that receiver Bob can be made to believe that user Alice sent him a packet which, in fact, Alice did not. In most network coded systems with multiple senders, such as BitTorrent [8], insider attacks form the real threat, so this attack has significant practical implications. Fortunately, this attack can be thwarted by enforcing that the file identifiers be *cryptographically verifiable*. In the subsequent sections, we will formalize these notions. We first describe the attack, and then use the intuition gained from the attack to construct a framework that can circumvent it.

3.1 Generic Attack (for arbitrary file identifiers)

Here we construct an attack against an abstract multi-source network coding signature scheme that consists of the algorithms Setup, Sign, Combine, Verify discussed above. We make no assumptions about these algorithms beyond their functionality. We show that it is impossible to achieve hop-by-hop containment if the identifier id is chosen arbitrarily by the sender and is given as input to the Sign algorithm. We construct a generic attack in which an intermediate node is fooled into accepting invalid packets as valid. As mentioned before, the attack is an "insider" attack where one of the senders is malicious. The malicious sender can assign two different vector spaces the same id and sign both using his secret key. An intermediate node has no hope of ever detecting this, since two packets constructed using these two vector spaces are both individually valid, but they are not pairwise valid, and can cause the receiver to incorrectly decode an honest user's message. We make this formal below.

We explain the attack with subspace dimension $m = 1$; the attack easily generalizes to arbitrary m. In our system, the honest sender is Alice, the receiver is Bob, and the malicious user is Mallet.

Honest User Alice. Alice wishes to send a file described as a single nonzero vector $\hat{\mathbf{v}}_1 \in \mathbb{F}_p^n$. She sets $\mathbf{v}_1 = (\hat{\mathbf{v}}_1, 1)$, chooses a file identifier id_a and uses her secret key sk_a to create a signature τ_1 on the one-dimensional subspace $V_1 \subset \mathbb{F}_p^{n+1}$ spanned by $\mathbf{v}_1$, with identifier id_a. Then she transmits the packet $P_1 = (\mathbf{v}_1, \mathsf{id}_a, \tau_1)$.

Malicious User Mallet. Mallet receives P_1 and does the following:

1. Generate a key pair $(\mathsf{sk}_m, \mathsf{pk}_m)$.
2. Pick two vectors $\hat{\mathbf{v}}_2, \hat{\mathbf{v}}_3 \in \mathbb{F}_p^n$ such that the set $\{\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \hat{\mathbf{v}}_3\}$ are linearly independent. Let V_2, V_3 be the subspaces of $\mathbb{F}_p^{n+1}$ spanned by $\mathbf{v}_2 = (\hat{\mathbf{v}}_2, 1)$ and $\mathbf{v}_3 = (\hat{\mathbf{v}}_3, 1)$, respectively.
3. Choose an identifier $\mathsf{id}_m \neq \mathsf{id}_a$, and use the key sk_m to compute signatures τ_2, τ_3 on subspaces V_2, V_3 with identifier id_m. Create the packets $P_2 = (\mathbf{v}_2, \mathsf{id}_m, \tau_2)$, $P_3 = (\mathbf{v}_3, \mathsf{id}_m, \tau_3)$.
4. Run Merge on $(\mathbf{v}_1, \mathbf{v}_2)$ and $(\mathsf{id}_a, \mathsf{id}_m)$ to obtain $\overline{\mathsf{id}} = (\mathsf{id}_a, \mathsf{id}_m)$ and vectors $\mathbf{v}_1' = (\hat{\mathbf{v}}_1, 1, 0)$, $\mathbf{v}_2' = (\hat{\mathbf{v}}_2, 0, 1)$.
5. Run $\mathsf{Combine}\big((\mathbf{v}_1, \mathsf{id}_a, \tau_1, 1), (\mathbf{v}_2, \mathsf{id}_m, \tau_2, 1)\big)$ to produce a signature τ_4 on the vector $\mathbf{v}_4 = \mathbf{v}_1' + \mathbf{v}_2' = (\hat{\mathbf{v}}_1 + \hat{\mathbf{v}}_2, 1, 1) \in \mathbb{F}_p^{n+2}$. Let $P_4 = (\mathbf{v}_4, \overline{\mathsf{id}}, \tau_4)$.
6. Send P_3 and P_4 to Bob.

Receiver Bob. Bob receives P_3 and P_4, each of which pass the verification test (by the correctness of Sign and $\mathsf{Combine}$). Bob then tries to decode the received data to recover Alice's file.
The identifier $\overline{\mathsf{id}} = (\mathsf{id}_a, \mathsf{id}_m)$ indicates that $\mathbf{v}^* = \mathbf{v}_4 - (\hat{\mathbf{v}}_3, 0, 1)$ is a vector sent by Alice, since the augmentation component of $\mathbf{v}^*$ is $(1, 0)$. However, the data part of $\mathbf{v}^*$ is $\hat{\mathbf{v}}_1 + \hat{\mathbf{v}}_2 - \hat{\mathbf{v}}_3$, which cannot be in the subspace spanned by $\hat{\mathbf{v}}_1$ since $\{\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \hat{\mathbf{v}}_3\}$ are linearly independent. Thus $\mathbf{v}^*$ is an invalid vector accepted by Bob.

In the above attack, Mallet was able to frame Alice by secretly reusing id_m for two different vector spaces. We see from this attack that arbitrary file identifiers provide a malicious insider too much power. It is thus necessary to tie the identifiers *cryptographically* to the files they represent, in a way that is verifiable at every node in the network. In particular, the Sign algorithm should output both an identifier id and a signature σ. To verify the identifier we use an algorithm IdTest that takes as input a public key pk, a vector $\mathbf{y}$, and a list of identifiers $\overline{\mathsf{id}}$, and outputs "accept" if $\mathbf{y}$ is in the subspace V identified by $\overline{\mathsf{id}}$. To avoid the above attack, the following tasks must be infeasible for Mallet:

1. Given a public key pk_a, find an identifier id_a and a vector $\mathbf{y}$ such that $\mathsf{IdTest}(\mathsf{pk}_a, \mathbf{y}, \overline{\mathsf{id}}_a)$ outputs "accept."
2. Given a vector space V, a public key pk_a, and $(\mathsf{id}_a, \sigma) := \mathsf{Sign}(\mathsf{sk}_a, V)$ (where sk_a is the secret key corresponding to pk_a), find a $\mathbf{y} \notin V$ such that $\mathsf{IdTest}(\mathsf{pk}_a, \mathbf{y}, \mathsf{id}_a)$ outputs "accept."

If Mallet can succeed at either task, then Bob is convinced that the vector $\mathbf{y}$ belongs to a file sent by Alice, when in fact it does not. (Indeed, in the first case Alice didn't even send a file!)

These two tasks are quite familiar: they are analogous to the two ways of breaking a single-source network coding signature scheme [4, Section 3.1]. This analysis leads to our key observation: **the file identifier produced by Sign must itself be a vector space signature**. It follows that all the security properties of the system are carried in the identifier id, so we can set the "signature" part σ equal to id or eliminate it entirely. We formalize these ideas in the following section.

Remark 3. One can show that allowing the use of arbitrary file identifiers not only makes hop-by-hop containment impossible, but also forces the receiver to solve the clique problem for proper decoding. Specifically, there is a formal reduction from the clique problem to decoding in multi-source network coding; details are in Appendix C.

4 Network Coding Signatures

We formally define the multi-source network coding signature scheme. Here the Sign algorithm generates an element σ that is used both as a signature and a file identifier. The Verify algorithm implements the functionality of the IdTest algorithm in the previous section and allows every node to validate the identifier/signature of an incoming packet. Since signatures and identifiers play the same role, the $\mathsf{Combine}$ algorithm provides the same functionality as the Merge algorithm of Section 2, while also keeping track of the public keys involved.

Definition 4. A multi-source network coding signature scheme is a tuple of five PPT algorithms, $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Verify}$, with the following properties:

$\mathsf{Setup}(1^\lambda, n, m)$**:** On input the unary representation of a security parameter 1^λ, a data space dimension n, and a subspace dimension m, outputs a description of system parameters params. This description includes the prime p used to define the field over which vector spaces are defined, as well as n and m.

$\mathsf{KeyGen}(\mathsf{params})$**:** Outputs a randomly generated user key pair $(\mathsf{sk}, \mathsf{pk})$.

$\mathsf{Sign}(\mathsf{params}, \mathsf{sk}, V)$**:** On input a secret key sk and a subspace $V \subset \mathbb{F}_p^{n+m}$, outputs an signature σ.

$\mathsf{Combine}\big(\mathsf{params}, (\mathbf{v}_1, \vec{\sigma}_1, \overline{\mathsf{pk}}_1, a_1), (\mathbf{v}_2, \vec{\sigma}_2, \overline{\mathsf{pk}}_2, a_2)\big)$**:** Takes as input two vectors $\mathbf{v}_1 \in \mathbb{F}_p^{n+mf_1}$ and $\mathbf{v}_2 \in \mathbb{F}_p^{n+mf_2}$, two lists of signatures $\vec{\sigma}_1, \vec{\sigma}_2$, two lists of public keys $\overline{\mathsf{pk}}_1, \overline{\mathsf{pk}}_2$, and two coefficients $a_1, a_2 \in \mathbb{F}_p$. The algorithm outputs a list of signatures $\vec{\sigma}$ and a list of public keys $\overline{\mathsf{pk}}$.

$\mathsf{Verify}\big(\mathsf{params}, \overline{\mathsf{pk}}, \mathbf{v}, \vec{\sigma}\big)$**:** On input a list of public keys $\overline{\mathsf{pk}}$, a vector $\mathbf{v} \in \mathbb{F}_p^{n+mf}$, and a list of signatures $\vec{\sigma}$, outputs $\top$ (for accept) or $\bot$ (for reject).

Correctness. We require that for any set of system parameters determined by $\mathsf{Setup}(1^\lambda, n, m)$, the following hold:

1. For primitive signatures: Consider a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{params})$ and a vector space $V \subset \mathbb{F}_p^{n+m}$. Let σ be the output of $\mathsf{Sign}(\mathsf{params}, \mathsf{sk}, V)$. Let $\overline{\mathsf{pk}} = \{\mathsf{pk}\}$ and $\vec{\sigma} = \{\sigma\}$. Then for all $\mathbf{v} \in V$, we require that
$$\mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}, \mathbf{v}, \vec{\sigma}) = \top.$$
2. Recursively, for combined signatures: Consider two lists of public keys $\overline{\mathsf{pk}}_1, \overline{\mathsf{pk}}_2$, two vectors $\mathbf{v}_1, \mathbf{v}_2$, two lists of signatures $\vec{\sigma}_1, \vec{\sigma}_2$ such that
$$\mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}_1, \mathbf{v}_1, \vec{\sigma}_1) = \mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}_2, \mathbf{v}_2, \vec{\sigma}_2) = \top.$$
Let $\mathbf{v}_1', \mathbf{v}_2', \vec{\sigma}'$ be the output of $\mathsf{Merge}(\mathbf{v}_1, \mathbf{v}_2, \vec{\sigma}_1, \vec{\sigma}_2)$. For any $a_1, a_2 \in \mathbb{F}_p$, we require that if $\vec{\sigma}, \overline{\mathsf{pk}}$ is the output of the $\mathsf{Combine}$ algorithm on inputs $(\mathbf{v}_1, \vec{\sigma}_1, \overline{\mathsf{pk}}_1, a_1), (\mathbf{v}_2, \vec{\sigma}_2, \overline{\mathsf{pk}}_2, a_2)$, then:
   (a) $\vec{\sigma}' = \vec{\sigma}$,
   (b) For j in $1, \ldots, f = |\vec{\sigma}|$, if the jth element of $\vec{\sigma}$ is the kth element of $\vec{\sigma}_i$ for $i \in \{1, 2\}$, then the jth element of $\overline{\mathsf{pk}}$ is the kth element of $\overline{\mathsf{pk}}_i$.
   (c) $\mathsf{Verify}\big(\mathsf{params}, \overline{\mathsf{pk}}', a_1\mathbf{v}_1' + a_2\mathbf{v}_2', \vec{\sigma}\big) = \top$.

In the second correctness condition, (a) tells us that identifiers and signature play the same role, while (b) requires that the list of public keys produced by $\mathsf{Combine}$ corresponds (in a natural way) to the list of identifiers produced by Merge.

4.1 Security

The security game captures the fact that if the system is secure, even an attacker who controls all sources but one and is given a chosen message oracle for the honest source cannot create an existential forgery on the honest source. The game between a challenger and an adversary $\mathcal{A}$ with respect to a signature scheme $\mathcal{S}$ proceeds as follows.

Init. The challenger runs $\mathsf{Setup}(1^\lambda, n, m)$ to obtain system parameters params and runs $\mathsf{KeyGen}(\mathsf{params})$ to obtain sk^* and pk^*. It sends pk^* and params to $\mathcal{A}$. It keeps sk^* to itself.

Signature queries. $\mathcal{A}$ adaptively requests signatures for vector spaces $V_1, \ldots, V_\ell \subset \mathbb{F}_p^{n+m}$. The challenger responds by computing $\mathsf{Sign}(\mathsf{params}, \mathsf{sk}^*, V_i)$ for $i = 1, \ldots, \ell$ and sends the resulting signatures to $\mathcal{A}$.

Forgery attempt. $\mathcal{A}$ eventually outputs a 4-tuple $(\overline{\mathsf{pk}}^\dagger, \mathbf{v}^\dagger, \vec{\sigma}^\dagger, W^\dagger)$, where $\overline{\mathsf{pk}}^\dagger$ is a list of f (not necessarily distinct) public keys $\overline{\mathsf{pk}}^\dagger = (\mathsf{pk}_1, \ldots, \mathsf{pk}_f)$ that contains the challenge public key pk^*, $\mathbf{v}^\dagger$ is a nonzero vector in $\mathbb{F}_p^{n+mf}$, $\vec{\sigma}^\dagger$ is list of f signatures, and $W^\dagger = \mathrm{span}\{\mathbf{w}_1, \ldots, \mathbf{w}_t\} \subset \mathbb{F}_p^{n+t}$ for some t.

Adjudication. Let $\vec{\sigma}^\dagger = (\sigma_1, \ldots, \sigma_f)$ be the list of (distinct) identifiers output by $\mathcal{A}$, where, w.l.o.g. we assume the first k components $\sigma_1, \ldots, \sigma_k$ are returned as the signatures for the chosen message queries $V_1, \ldots, V_k$, $k \leq \ell$. Let $\vec{\sigma}_w$ be the last $f - k$ elements of $\vec{\sigma}^\dagger$. Let V^* be the vector space output by $\mathsf{MergeSpaces}(V_1, \ldots, V_k, W^\dagger, \sigma_1, \ldots, \sigma_k, \vec{\sigma}_w)$.

The forger wins the game if $\mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}^\dagger, \mathbf{v}^\dagger, \vec{\sigma}^\dagger) = \top$ and at least one of the following two conditions holds:

1. There exists i in $1, \ldots, f$ such that the ith component of $\mathsf{pk}^\dagger$ is equal to pk^*, but σ_i is not any of the signatures obtained in response to chosen message queries.
2. For $i = 1, \ldots, t$, we have $\mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}_w, \mathbf{w}_i, \vec{\sigma}_w) = \top$, but $\mathbf{v}^\dagger \notin V^*$.

Definition 5. The *advantage* NC-Adv$[\mathcal{A}, \mathcal{S}]$ of $\mathcal{A}$ is defined to be the probability that $\mathcal{A}$ wins the security game. A multi-source network coding scheme $\mathcal{S}$ is *secure* if for all probabilistic, polynomial-time adversaries $\mathcal{A}$ the advantage NC-Adv$[\mathcal{A}, \mathcal{S}]$ is negligible in the security parameter λ.

4.2 Discussion of the security model

In the security game, the attacker requests signatures for files $V_1, \ldots, V_k$ and creates his own file $W^\dagger$. Intuitively, $W^\dagger$ corresponds to the vector space (the set of files) whose data the adversary mixes with the honest user's data in order to frame the honest user. Winning condition (1) implies that the attacker can create a valid fake signature for one of the files that he requests signatures for, i.e., for a file signed with sk^*. Winning condition (2) implies that the attacker can produce a fake file $W^\dagger$ whose basis vectors pass the verification test, and a vector $\mathbf{v}^\dagger$ that passes the verification test but lives outside the subspace V^* that is the span of network coding combinations of the files he requested and created. A receiver that decodes the basis vectors of $W^\dagger$ together with the vector $v^\dagger$ will be fooled into accepting a vector from the user with public key pk^* that this user never sent.

Implied properties. The security model implies that even given the secret key sk, no PPT adversary can construct distinct vector spaces $V_1, V_2 \in \mathbb{F}_p^{n+m}$ such that

$$\mathsf{Sign}(\mathsf{params}, \mathsf{sk}, V_1) = \mathsf{Sign}(\mathsf{params}, \mathsf{sk}, V_2)$$

Note, however, that this is no ordinary collision resistance property. During signature verification the vector space V is not available and therefore the Verify algorithm must validate the signature given only $\mathbf{y} \in V$.

This collision resistance property is crucial during decoding. The decoder collects all incoming packets with a specific identifier into a full rank matrix and runs the decoding procedure. Collision resistance ensures that all packets with the same signature belong to the same vector space.

To see that this collision resistance property follows from our definition, it is not difficult to give a generic attack that works on any scheme for which this property is not satisfied. The attack, in fact, is essentially the same as the attack presented in Section 3.1.

The vector space $W^\dagger$. Recall that the forgery attempt by the adversary consists of the 4-tuple $(\overline{\mathsf{pk}}^\dagger, \mathbf{v}^\dagger, \vec{\sigma}^\dagger, W^\dagger)$ where $\overline{\mathsf{pk}}^\dagger$ is a vector of public keys containing the challenge public key pk^*. The other public keys in the vector $\overline{\mathsf{pk}}^\dagger$ are invented by the adversary and it is therefore possible that the adversary knows the corresponding private keys.

The vector $\mathbf{v}^\dagger$ and the signature $\vec{\sigma}^\dagger$ are the adversary's existential forgery. Suppose that $(\mathbf{v}^\dagger, \vec{\sigma}^\dagger)$ verify as a valid vector-signature pair with respect to $\overline{\mathsf{pk}}^\dagger$. We require the adversary to output the vector space $W^\dagger$ to prove that he is capable of exploiting $\mathbf{v}^\dagger$ to fool a recipient to incorrectly accept a vector from the single honest sender. To fool the recipient, the attacker can generate valid vector-signature pairs for all basis vectors of $W^\dagger$ using the secret keys at his disposal. Since all these vectors have valid signatures, a recipient might try to decode the basis of $W^\dagger$ along with the vector $\mathbf{v}^\dagger$. If $\mathbf{v}^\dagger \notin V^*$, after decoding this set of vectors (i.e. after subtracting from $\mathbf{v}^\dagger$ the projection of $\mathbf{v}^\dagger$ onto $W^\dagger$), the recipient obtains a vector $\mathbf{u}$ that he believes came from the honest sender, but which the honest sender never sent since $\mathbf{u}$ is not in $\mathsf{MergeSpaces}(V_1, \ldots, V_k, \sigma_1, \ldots, \sigma_k)$.

Hence, if the attacker is capable of producing a forgery for which condition (2) of adjudication holds, then an adversary can fool a recipient by sending it a sequence of properly signed vectors. We would like to require that for a secure signature scheme it should be impossible to produce a valid forgery where $\mathbf{v}^\dagger \notin V^\dagger$. Unfortunately, this strong requirement appears to be unsatisfiable. We therefore weaken it to require that $\mathbf{v}^\dagger \notin V^*$ only when there is a possibility that the vectors in $W^\dagger$ will be jointly decoded with $\mathbf{v}^\dagger$, namely when $\mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}_w, \mathbf{w}_i, \vec{\sigma}_w) = \top$ for all basis vectors $\mathbf{w}_i$ of $W^\dagger$. This is an acceptable weakening of the security requirement since the decoder will never group together vectors that have different identifiers. In Section 5 we show that the resulting definition is satisfiable.

We note that requiring the adversary to output $W^\dagger$ is analogous to the security model of aggregate signatures [5] where the attacker outputs an aggregate signature from f public keys, where $f - 1$ of them are invented by the attacker. Moreover, the attacker must output the list of $f - 1$ messages that went into the aggregate forgery, for each of the public keys the attacker invented. Our vector space $W^\dagger$ plays the same role as the $f - 1$ messages in the aggregate forgery.

5 Construction of a Multi-source Signature Scheme

In this section, we construct an explicit multi-source network coding signature scheme satisfying Definition 4. In order to give a generic construction, we first define an auxiliary primitive called *vector hash*.

5.1 Vector Hashes

A vector hash consists of three algorithms, $\mathsf{Setup}, \mathsf{Hash}, \mathsf{Test}$, with the following properties:

$\mathsf{HashSetup}(1^\lambda, n)$**:** Input: unary representation of a security parameter λ and dimension of the data space n. Output: public parameters pp.

$\mathsf{Hash}(\mathsf{pp}, \mathbf{v})$**:** Input: public parameters pp and a vector $\mathbf{v} \in \mathbb{F}_p^n$. Output: hash h of the vector $\mathbf{v}$. We require that this algorithm be deterministic.

$\mathsf{Test}(\mathsf{pp}, \mathbf{y}, \overline{\beta}, \mathbf{h})$**:** Input: Public parameters pp, a vector $\mathbf{y} \in \mathbb{F}_p^n$, a vector of coefficients $\overline{\beta} \in \mathbb{F}_p^m$ and a vector of m hash values $\mathbf{h}$. Output: $\top$ (true) or $\bot$ (false).

Let $\mathbf{h}$ be a set of hashes of a basis $\mathbf{v}_1, \ldots, \mathbf{v}_m$ of a vector space V. Intuitively, we want the Test algorithm to tell us whether $\mathbf{y}$ was constructed correctly from the basis, i.e., whether $\mathbf{y} = \sum \beta_i \mathbf{v}_i$. This means that Test should output $\top$ whenever $\mathbf{y}$ is constructed correctly, and it should be difficult for an adversary to find a vector $\mathbf{y} \notin V$ and a $\overline{\beta}$ such that Test outputs $\top$. We now formalize these correctness and security conditions.

Correctness. For correctness, we require the following for all public parameters $\mathsf{pp} \leftarrow \mathsf{HashSetup}(1^\lambda)$:

1. For all $\mathbf{v} \in \mathbb{F}_p^n$, if $h \leftarrow \mathsf{Hash}(\mathsf{pp}, \mathbf{v})$ then we have $\mathsf{Test}(\mathsf{pp}, \mathbf{v}, 1, h) = \top$.
2. Let $\mathbf{v} \in \mathbb{F}_p^n$, let $\overline{\beta} \in \mathbb{F}_p^\ell$ for some ℓ, and let $\mathbf{h}$ be a list of hashes of length ℓ. Fix $i \in \{0, \ldots, \ell\}$, let $\overline{\beta}' \in \mathbb{F}_p^{m+1}$ be the vector $\overline{\beta}$ with a zero inserted between the ith and $(i+1)$th place, and let $\mathbf{h}'$ be the vector $\mathbf{h}$ with any hash value inserted between the ith and $(i+1)$th place. We require that if $\mathsf{Test}(\mathsf{pp}, \mathbf{v}, \overline{\beta}, \mathbf{h}) = \top$, then $\mathsf{Test}(\mathsf{pp}, \mathbf{v}, \overline{\beta}', \mathbf{h}') = \top$.
3. Let $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{F}_p^n$, let $\overline{\beta}_1, \overline{\beta}_2 \in \mathbb{F}_p^\ell$ for some ℓ, let $\mathbf{h}$ be a list of hashes of length ℓ. Let $a, b \in \mathbb{F}_p$, let $\mathbf{y} = a\mathbf{v}_1 + b\mathbf{v}_2$, and $\overline{\beta} = a\overline{\beta}_1 + b\overline{\beta}_2$. We require that if $\mathsf{Test}(\mathsf{pp}, \mathbf{v}_i, \overline{\beta}_i, \mathbf{h}) = \top$ for $i = 1, 2$ then $\mathsf{Test}(\mathsf{pp}, \mathbf{y}, \overline{\beta}, \mathbf{h}) = \top$.

Security. Let $\mathcal{VH} = (\mathsf{HashSetup}, \mathsf{Hash}, \mathsf{Test})$ be a vector hash. Let $\mathcal{A}$ be a PPT algorithm that takes as input public parameters $\mathsf{pp} \leftarrow \mathsf{HashSetup}(1^\lambda, n)$ and outputs a vector $\mathbf{v}^* \in \mathbb{F}_p^n$, an m-dimensional vector space $V \subset \mathbb{F}_p^n$ (for some m) represented as basis vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$, an m-tuple of coefficients $\overline{\beta}$, and a vector of hashes $\mathbf{h} = (h_1, \ldots, h_m)$.

Definition 6. With notation as above, we say that $\mathcal{A}$ *breaks* the vector hash scheme $\mathcal{VH}$ if $\mathbf{v}^* \notin V$, $\mathsf{Test}(\mathsf{pp}, \mathbf{v}^*, \overline{\beta}, \mathbf{h}) = \top$, and $\mathsf{Test}(\mathsf{pp}, \hat{\mathbf{v}}_i, \mathbf{e}_i, h_i) = \top$ for $i = 1, \ldots, m$. We define the *advantage* Hash-Adv$[\mathcal{A}, \mathcal{VH}]$ of $\mathcal{A}$ to be the probability that $\mathcal{A}$ breaks $\mathcal{VH}$. We say that a vector hash $\mathcal{VH}$ is *secure* if for all PPT algorithms $\mathcal{A}$ the advantage Hash-Adv$[\mathcal{A}, \mathcal{VH}]$ is negligible in the security parameter λ.

In Appendix A we give an example vector hash using a finite cyclic group $\mathbb{G}$ of order p. This vector hash is secure if the discrete logarithm problem is infeasible in $\mathbb{G}$.

5.2 The Construction

For this construction, we use as a black box a vector hash as defined in Section 5.1.

Signature scheme $\mathcal{NS}$: Let $\mathcal{VH} = (\mathsf{HashSetup}_h, \mathsf{Hash}_h, \mathsf{Test}_h)$ be a vector hash and let $\mathcal{S} = (\mathsf{Setup}_s, \mathsf{KeyGen}_s, \mathsf{Sign}_s, \mathsf{Verify}_s)$ be a signature system for signing messages in $\{0,1\}^*$. Our network coding signature scheme is as follows:

$\mathsf{Setup}(1^\lambda, n, m)$**:** Run $\mathsf{HashSetup}_h(1^\lambda, n)$ to obtain hash parameters and $\mathsf{Setup}_s(1^\lambda)$ to obtain signature parameters. Let params contain m, n, and the outputs of these algorithms.

$\mathsf{KeyGen}(\mathsf{params})$**:** Run KeyGen_s to obtain public key pk and the private key sk. Output $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{params}, \mathsf{sk}, \{\mathbf{v}_1, \ldots, \mathbf{v}_m\})$**:** For $i = 1, \ldots, m$, set $h_i := \mathsf{Hash}_h(\mathsf{params}, \hat{\mathbf{v}}_i)$. Set $\mathbf{h} = (h_1, \ldots, h_m)$ and $\eta := \mathsf{Sign}_s(\mathsf{sk}, \mathbf{h})$. Let $\sigma := \big(\mathbf{h}, \eta\big)$. Output σ.

$\mathsf{Combine}\big(\mathsf{params}, (\mathbf{v}_1, \vec{\sigma}_1, \overline{\mathsf{pk}}_1, a_1),\ (\mathbf{v}_2, \vec{\sigma}_2, \overline{\mathsf{pk}}_2, a_2)\big)$**:** Let $\vec{\sigma}', \mathbf{v}_1', \mathbf{v}_2' := \mathsf{Merge}(\vec{\sigma}_1, \vec{\sigma}_2, \mathbf{v}_1, \mathbf{v}_2)$.

1. To create a list $\overline{\mathsf{pk}}'$, do:
   For j in $1, \ldots, k = |\vec{\sigma}|$, if the jth element of $\vec{\sigma}$ is the kth element of $\vec{\sigma}_i$ for $i \in \{1, 2\}$, then the jth element of $\overline{\mathsf{pk}}'$ is the kth element of $\overline{\mathsf{pk}}_i$.
2. Output $\vec{\sigma}'$ and $\overline{\mathsf{pk}}'$.

$\mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}, \mathbf{y}, \vec{\sigma})$**:** Interpret $\vec{\sigma}$ as a list of f signatures where each $\sigma_i = (\mathbf{h}_i, \eta_i)$. Write $\mathbf{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_f)$. Do the following:

1. For i in $1, \ldots, f$, compute $\mathsf{Verify}_s(\mathsf{pk}_i, \mathbf{h}_i, \eta_i)$.
2. Compute $\mathsf{Test}_h(\mathsf{params}, \hat{\mathbf{y}}, \overline{\beta}_{\mathbf{y}}, \mathbf{H})$.

If all steps output $\top$, output $\top$; else output $\bot$.

The only difference between the $\mathsf{Combine}$ algorithm in our signature scheme and the Merge algorithm of Section 2.1 is that the $\mathsf{Combine}$ algorithm also keeps track of the public keys associated with the signatures.

Instead of sending a separate hash signature η_i in each σ_i, we can aggregate these signatures together for space efficiency. We do not use aggregate signature schemes in our description for compactness and clarity. If the system is implemented using the vector hash $\mathcal{VH}$-DL of Appendix A and aggregate signatures of length $\log_2 p$ bits, then a signature on f files is of length $(fm+1)\log_2 p$ bits. We will see in Section 6 below that for large values of f and m this length is optimal.

Correctness. We demonstrate the correctness conditions of Definition 4.

1. For primitive signatures: Consider a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{params})$ and a vector space $V \subset \mathbb{F}_p^{n+m}$ described by a properly augmented basis $\mathbf{v}_1, \ldots, \mathbf{v}_m$. Let σ be the output of $\mathsf{Sign}(\mathsf{params}, \mathsf{sk}, V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\})$. Interpret the signature σ as $\sigma = (\mathbf{h}, \eta)$.
   For primitive signatures, there is only one file $f = 1$. We examine each step of Verify in turn:
   1. Since $\eta = \mathsf{Sign}_s(\mathsf{sk}, \mathbf{h})$, we have $\mathsf{Verify}_s(\mathsf{pk}, \mathbf{h}, \eta) = \top$ by correctness of $\mathcal{S}$.
   2. Since $h_i = \mathsf{Hash}_h(\mathsf{params}, \hat{\mathbf{v}}_i)$, and $\beta_{\mathbf{v}_i}$ is the unit vector $\mathbf{e}_i$ since we are using a properly augmented basis, correctness conditions (1) and (3) of $\mathcal{VH}$ imply that $\mathsf{Test}_h(\mathsf{params}, \hat{\mathbf{v}}_i, \beta_{\mathbf{v}_i}, \mathbf{h}) = \top$.

   It follows that every basis vector $\mathbf{v}_i$ passes the signature verification test, i.e.,

   $$\mathsf{Verify}(\mathsf{params}, \mathsf{pk}, \mathbf{v}_i, \sigma) = \top.$$

2. Recursively, for combined signatures: Consider two lists of public keys $\overline{\mathsf{pk}}_1, \overline{\mathsf{pk}}_2$, two augmented vectors $\mathbf{v}_1, \mathbf{v}_2$, two lists of signatures $\vec{\sigma}_1, \vec{\sigma}_2$ such that

   $$\mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}_1, \mathbf{v}_1, \vec{\sigma}_1) = \mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}_2, \mathbf{v}_2, \vec{\sigma}_2) = \top. \tag{5.1}$$

   Let $\mathbf{v}_1', \mathbf{v}_2', \vec{\sigma}'$ be the output of $\mathsf{Merge}(\mathbf{v}_1, \mathbf{v}_2, \vec{\sigma}_1, \vec{\sigma}_2)$ and $f = |\vec{\sigma}|$. Let $\mathbf{H}_i$ be the list of all the hash elements in $\vec{\sigma}_i$ for $i = 1, 2$. Let $a_1, a_2 \in \mathbb{F}_p$ be network combination coefficients, and let $\mathbf{y} = a_1\mathbf{v}_1' + a_2\mathbf{v}_2'$. Let $\vec{\sigma}, \overline{\mathsf{pk}}$ be the output of the $\mathsf{Combine}$ algorithm on inputs $(\mathbf{v}_1, \vec{\sigma}_1, \overline{\mathsf{pk}}_1, a_1), (\mathbf{v}_2, \vec{\sigma}_2, \overline{\mathsf{pk}}_2, a_2)$.

   Conditions (a) and (b) are now immediate. For (c), we note that in our scheme, $\sigma_j' = (\mathbf{h}_j, \eta_j)$ for $j = 1, \ldots, f$. Let $\mathbf{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_f)$. We examine each step of the Verify algorithm:

1. By the assumption (5.1) and the way we have set up the correspondence between indices of $\overline{\mathsf{pk}}$ and $\vec{\sigma}$, we have $\mathsf{Verify}_s(\mathsf{pk}_j, \mathbf{h}_j, \eta_j) = \top$ for j in $1, \ldots, f$.
2. By assumption (5.1) we know that $\mathsf{Test}_h(\mathsf{params}, \hat{\mathbf{v}}_i, \overline{\beta}_{\mathbf{v}_i}, \mathbf{H}_i) = \top$ for $i = 1, 2$. By correctness property (2) of $\mathcal{VH}$, we have $\mathsf{Test}_h(\mathsf{params}, \hat{\mathbf{v}}'_i, \overline{\beta}_{\mathbf{v}'_i}, \mathbf{H}) = \top$ for $i = 1, 2$. Then, by correctness property (3) of $\mathcal{VH}$, we have $\mathsf{Test}_h(\mathsf{params}, \hat{\mathbf{y}}, \overline{\beta}_{\mathbf{y}}, \mathbf{H}) = \top$.

Thus, we have that $\mathsf{Verify}\big(\mathsf{params}, \overline{\mathsf{pk}}', \mathbf{y}, \vec{\sigma}\big) = \top$.

Security. Our security theorem is as follows; the proof is in Appendix B.

Theorem 7. *The network coding signature scheme $\mathcal{NS}$ is secure assuming that $\mathcal{VH}$ is a secure vector hash, and assuming $\mathcal{S}$ is a secure signature scheme.*

6 A Lower Bound on File Identifier Size

In Section 3 we showed that a cryptographically verifiable file identifier must have the properties of a vector space signature. In particular, this implies that if the identifier satisfies certain additional (mild) assumptions, then we can use the argument of [4, Section 6] to show that any secure identifier for an m-dimensional subspace of $\mathbb{F}_p^N$ must be of length at least $m \log_2 p$ bits.

Recall that in our generic setup of Section 3, where file identifiers are distinct from signatures, the algorithm IdTest determines whether an identifier id is valid for the vector $\mathbf{v}$. (Here id can identify either a single file or a collection of files.) The correctness requirement is that if id is associated to the vector space V, then $\mathsf{IdTest}(\mathsf{pk}, \mathbf{v}, \mathsf{id}) = \top$ for all $\mathbf{v} \in V$. A slightly stronger requirement is that the function $\mathsf{IdTest}(\mathsf{pk}, \cdot, \cdot)$ is *additive*. A function $g : \mathbb{F}_p^N \times \{0,1\}^* \to \{\top, \bot\}$ is *additive* if for all vectors $\mathbf{v}_1, \mathbf{v}_2$ and all x such that $g(\mathbf{v}_1, x) = g(\mathbf{v}_2, x) = \top$, we have $g(\mathbf{v}_1 + \mathbf{v}_2, x) = \top$. The function $\mathsf{Verify}(\mathsf{params}, \mathsf{pk}, \cdot, \cdot)$ of our construction in Section 5 using any vector hash is additive; indeed, this seems to be a natural property of network coding signature schemes.

We also assume that for fixed $m < N$, file identifiers for m-dimensional subspaces $V \subset \mathbb{F}_p^N$ are all of the same size. This is a mild assumption that serves chiefly to facilitate our exposition. We derive our lower bound from the following theorem, whose statement and proof are straightforward adaptations of [4, Theorem 9].

Theorem 8. *Let ℓ, m, N be integers with $0 < m < N$ and let p be a prime. Let f be a deterministic function that maps m-dimensional vector subspaces $V \subset \mathbb{F}_p^N$ to identifiers $\mathsf{id} \in \{0,1\}^\ell$, and let $g : \mathbb{F}_p^N \times \{0,1\}^\ell \to \{\top, \bot\}$ be an additive function. If $\ell \leq m \log_2 p - 4m/p - 1$, then at least half of all m-dimensional subspaces $V \subset \mathbb{F}_p^N$ have the property that $g(\mathbf{v}, f(V)) = \top$ for all $\mathbf{v} \in V$.*

In the application of the theorem, we fix a challenge secret key sk and public key pk as well as any randomness used in the system. The function f gives the identifier output by $\mathsf{Sign}(\mathsf{params}, \mathsf{sk}, V)$, while the function g computes $\mathsf{IdTest}(\mathsf{pk}, \mathbf{v}, \mathsf{id})$. An adversary attacks the system by choosing a random subspace V, obtaining an identifier id for V, and producing a vector $\mathbf{y} \notin V$. With probability at least $1/2$, we have $\mathsf{IdTest}(\mathsf{pk}, \mathbf{y}, \mathsf{id}) = \top$. Thus if $\mathbf{y}$ is used in a network coding system, honest users will believe that $\mathbf{y}$ is associated with the file identified by id when in fact it is not.

A similar argument applies to lists of identifiers produced as a result of the Merge algorithm, since the list of identifiers produced must satisfy the correctness properties with respect to the subspace Z produced by $\mathsf{MergeSpaces}$ on the original files. As a result, we conclude that not only must the identifier of a single file be large, but the identifiers of multiple files in a secure multi-source network coding system must grow linearly with the number of files in the system. Unfortunately, the lower bound is large, and limits the practicality of security in this setting.

References

1. S. Agrawal and D. Boneh. Homomorphic MACs: MAC-based integrity for network coding. In *Proceedings of ACNS '09*, volume 5536 of *LNCS*. Springer, 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. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in Cryptology — Crypto '94*, volume 839 of *LNCS*, pages 216–233. Springer, 1994.
4. D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *Public-Key Cryptography — PKC '09*, volume 5443 of *LNCS*, pages 68–87. Springer, 2009.
5. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology — Eurocrypt 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, 2003.
6. N. Cai and R. Yeung. Secure network coding. *Proceedings of the 2002 IEEE International Symposium on Information Theory.*, 2002.
7. D. Charles, K. Jain, and K. Lauter. Signatures for network coding. In *40th Annual Conference on Information Sciences and Systems (CISS '06)*, 2006.
8. B. Cohen. Incentives build robustness in BitTorrent, 2003. Available at `http://www.bittorrent.org/bittorrentecon.pdf`.
9. J. Feldman, T. Malkin, C. Stein, and R. Servedio. On the capacity of secure network coding. *Proc. 42nd Annual Allerton Conference on Communication, Control, and Computing*, 2004.
10. C. Fragouli, J.-Y. Le Boudec, and J. Widmer. Network coding: an instant primer. *SIGCOMM Comput. Commun. Rev.*, 36(1):63–68, 2006.
11. C. Fragouli and E. Soljanin. *Network Coding Fundamentals*. Now Publishers Inc., Hanover, MA, USA, 2007.
12. K. Han, T. Ho, R. Koetter, M. Médard, and F. Zhao. On network coding for security. In *Military Communications Conference (Milcom)*, 2007.
13. T. Ho, B. Leong, R. Koetter, M. Médard, M. Effros, and D. Karger. Byzantine modification detection in multicast networks using randomized network coding. In *Proceedings of the 2004 IEEE International Symposium on Information Theory (ISIT)*, June 2004.
14. 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.
15. S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. XORs in the air: practical wireless network coding. *IEEE/ACM Trans. Netw.*, 16(3):497–510, 2008.
16. J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC Press, 2007.
17. R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, 11:782–795, 2003.
18. 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.
19. F. Zhao, T. Kalker, M. Médard, and K. Han. Signatures for content distribution with network coding. In *Proc. Intl. Symp. Info. Theory (ISIT)*, 2007.

A Example Vector Space Hash:

We now construct an example of a vector hash and prove that it is correct and secure. Security is based on the discrete log assumption.

A.1 Vector Hash $\mathcal{VH}$-DL:

Our vector hash is based on the construction of Krohn et al. [18] and is as follows:

Hash-Setup$(1^\lambda, n)$**:** Input: security parameter λ (in unary) and dimension of the data space n.

1. Choose a group $\mathbb{G}$ of prime order $p > 2^\lambda$.
2. Choose generators $g_i \xleftarrow{R} \mathbb{G} \setminus \{1\}$ for $i = 1, \ldots, n$.
3. Output $\mathsf{pp} := p$, $(g_1, \ldots, g_n)$, and description of $\mathbb{G}$.

Hash$(\mathsf{pp}, \mathbf{v})$**:** Input: public parameters pp and a vector $\mathbf{v} \in \mathbb{F}_p^n$.

1. Output the hash $h := \prod_{j=1}^n g_j^{v_j}$.

Test$(\mathsf{pp}, \mathbf{y}, \overline{\beta}, \mathbf{h})$**:** Input: public parameters pp, a vector $\mathbf{y} \in \mathbb{F}_p^n$, a vector of coding coefficients $\overline{\beta}$, and a vector of hash values $\mathbf{h}$.

1. If $|\overline{\beta}| \neq |\mathbf{h}|$, output $\perp$.
2. If $|\overline{\beta}| = |\mathbf{h}| = m$ and $\left(\prod_{j=1}^n g_j^{y_j}\right) = \left(\prod_{i=1}^m \mathbf{h}_i^{\beta_i}\right)$, output $\top$; otherwise output $\perp$.

A.2 Correctness and Security.

The correctness conditions are easily verified; we note that (3) follows from the fact that the exponents of the two sides of the verification equation are linear in $\mathbf{y}$ and $\overline{\beta}$, respectively. For security we have the following:

Theorem 9. *The vector hash $\mathcal{VH}$-DL is secure assuming the discrete logarithm problem in $\mathbb{G}$ is hard.*

In particular, let $\mathcal{A}$ be a PPT adversary that breaks vector hash $\mathcal{VH}$-DL. Then there exists a polynomial-time algorithm $\mathcal{B}$ that computes discrete logarithms in $\mathbb{G}$ such that

$$\text{Hash-Adv}[\mathcal{A}, \mathcal{VH}\text{-DL}] \leq 2 \cdot \text{DL-Adv}[\mathcal{B}, \mathbb{G}],$$

Proof. Given the output $(\mathbf{v}^*, V, \overline{\beta}, \mathbf{h})$ of algorithm $\mathcal{A}$ that breaks the vector hash, we can produce distinct vectors $\mathbf{v}, \mathbf{w} \in \mathbb{F}_p^n$ such that $\prod_{j=1}^n g_j^{v_j} = \prod_{j=1}^n g_j^{w_j}$. By standard arguments [3], an algorithm $\mathcal{A}$ that produces such a collision with probability ε can be used to compute discrete logarithms in $\mathbb{G}$ with probability at least $\varepsilon/2$. For further details, see [18, §VI]. □

B Proof of Security for Scheme $\mathcal{NS}$:

Theorem 10. *The network coding signature scheme $\mathcal{NS}$ is secure assuming that $\mathcal{VH}$ is a secure vector hash, and assuming $\mathcal{S}$ is a secure signature scheme.*

In particular, let $\mathcal{A}$ be a polynomial-time adversary as in Definition 4.1. Then there exists a polynomial-time adversary $\mathcal{B}_1$ that forges signatures for $\mathcal{S}$ and a polynomial-time algorithm $\mathcal{B}_2$ that breaks the vector space hash $\mathcal{VH}$, such that

$$\text{NC-Adv}[\mathcal{A}, \mathcal{NS}] \leq \text{Sig-Adv}[\mathcal{B}_1, \mathcal{S}] + \text{Hash-Adv}[\mathcal{B}_2, \mathcal{VH}],$$

where $\text{Sig-Adv}[\mathcal{B}_1, \mathcal{S}]$ *is the probability that $\mathcal{B}_1$ wins the security game for the standard signature scheme $\mathcal{S}$ (see [16, §12.2]).*

Proof. We use the notation of Section 4.1. Let $\mathcal{A}$ be an adversary that attacks the network coding signature scheme $\mathcal{NS}$ instantiated with the vector hash $\mathcal{VH}$ and the signature scheme $\mathcal{S}$. Let $(\overline{\mathsf{pk}}^\dagger, \mathbf{v}^\dagger, \vec{\sigma}^\dagger, W^\dagger)$ be the output of $\mathcal{A}$. As before, let $\vec{\sigma}^\dagger = (\vec{\sigma}_1, \ldots, \vec{\sigma}_f)$ be the list of (unique) signatures/identifiers in the composite signature $\vec{\sigma}^\dagger$, where w.l.o.g., we assume that the first k components $\sigma_1, \ldots, \sigma_k$ are the identifiers returned in the signatures obtained from the chosen message queries $V_1, \ldots, V_k$. Let $\overline{\mathsf{pk}}^\dagger = (\mathsf{pk}_1, \ldots, \mathsf{pk}_f)$. Each identifier σ_i is of the form $(\mathbf{h}_i, \eta_i)$, where $\mathbf{h}_i$ is a list of m hashes in the range of Hash_h and η_i is a signature in the range of Sign_s. Let $\mathbf{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_f)$.

Suppose $\mathcal{A}$ wins the security game. The fact that $\mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}^\dagger, \mathbf{v}^\dagger, \vec{\sigma}^\dagger) = \top$ implies that

$$\mathsf{Verify}_s(\mathsf{pk}_i, \mathbf{h}_i, \eta_i) = \top \quad \text{for all } i, \text{ and} \tag{B.1}$$

$$\mathsf{Test}_h(\mathsf{params}, \hat{\mathbf{v}}^\dagger, \beta_{\mathbf{v}^\dagger}, \mathbf{H}) = \top. \tag{B.2}$$

Now suppose winning condition (1) holds. Let i be such that $\mathsf{pk}_i = \mathsf{pk}^*$. If $\sigma_i = (\mathbf{h}_i, \eta_i)$ is not in any of the signatures obtained in response to the chosen message queries, then the message $\mathbf{h}_i$ was never queried to the Sign_s algorithm. Condition (B.1) thus implies that we have forged an $\mathcal{S}$ signature η_i on the message $\mathbf{h}_i$.

On the other hand, suppose winning condition (2) holds. Let V^* be the vector space output by $\mathsf{MergeSpaces}$ in the adjudication step, i.e. $V^* = \mathsf{MergeSpaces}(V_1, \ldots, V_k, W^\dagger, \sigma_1, \ldots, \sigma_k, \vec{\sigma}_w)$, where $\sigma_w = (\mathbf{h}_w, \eta_w)$ is the signature corresponding to space $W^\dagger$. We write $\mathbf{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_k, \mathbf{h}_w)$.

Let $\mathbf{w}_1, \ldots, \mathbf{w}_t$ be the basis vectors of $W^\dagger$ output by the adversary. By correctness condition (3) of $\mathcal{VH}$, we can assume without loss of generality that the basis vectors $\mathbf{w}_i$ are properly augmented. Since the winning condition implies that $\mathsf{Verify}(\mathsf{params}, \overline{\mathsf{pk}}_w, \mathbf{w}_i, \vec{\sigma}_w) = \top$ for all i, we have that $\mathsf{Test}_h(\mathsf{params}, \hat{\mathbf{w}}_i, \mathbf{e}_i, \mathbf{h}_w) = \top$ for all i.

Since the signature scheme is correct, for each space V_i, $i = 1, \ldots, k$, with signature $\sigma_i = (\mathbf{h}_i, \eta_i)$ we have $\mathsf{Verify}(\mathsf{params}, \mathsf{pk}^*, \mathbf{v}_{ij}, \sigma_i) = \top$, where pk^* is the challenge public key and $\mathbf{v}_{ij}$ is the jth basis vector of vector space V_i. Since we can again assume without loss of generality that the basis vectors $\mathbf{v}_{ij}$ are properly augmented, this implies that $\mathsf{Test}_h(\mathsf{params}, \hat{\mathbf{v}}_{ij}, \mathbf{e}_j, \mathbf{h}_i) = \top$ for all vectors $\mathbf{v}_{ij}$, $i \in [f], j \in [m]$.

It now follows from our description of the Merge algorithm that the basis of V^* output by $\mathsf{MergeSpaces}$ is also properly augmented. Let $U \subset \mathbb{F}_p^n$ be the subspace spanned by the data components of vectors in V^*, and let $\{\mathbf{u}_\alpha\}$ be the basis of U consisting of the data components of the $\mathbf{v}_{ij}$ and $\mathbf{w}_i$. Correctness property (2) of $\mathcal{VH}$ now implies that $\mathsf{Test}_h(\mathsf{params}, \mathbf{u}_\alpha, \mathbf{e}_\alpha, \mathbf{H}) = \top$ for all α. Since $\mathbf{v}^\dagger \notin V^*$, the data component $\hat{\mathbf{v}}^\dagger$ is not in U. It now follows from Definition 6 and equation (B.2) that we have broken $\mathcal{VH}$. □

C Reduction from clique

Here we show that if nodes are allowed to pick their own file identifiers in multi-source network coding, as in the model of Section 3, then the decoder is essentially forced to solve an instance of the clique problem in order to decode.

We introduce the notion of *consistency of vectors*. Two vectors are said to be inconsistent if and only if they were generated using different basis vectors for the same slot (i.e., same index of same file). Two vectors that are not inconsistent are called consistent. It is evident that only a pairwise consistent set of vectors should be used for decoding.

The difficulty stems with the fact that pairwise consistency is not a transitive relation. Consider for example three vectors, whose "slots" are as follows: $A = (\alpha, 0)$, $B = (0, \beta)$, and $C = (\gamma, \beta)$. In this example, A and B are consistent, and so are B and C, but A and C are clearly inconsistent.

Intuitively speaking, it is this property that forces the decoder to solve an instance of the clique problem in order to find a large enough set of vectors that are all pairwise consistent, in order to avoid the attack described in section 3.1. Below we formally show this necessity.

C.1 Unconstrained multi-source network coding signature

First, we define the "natural" notion of unconstrained multi-source network coding signature, in which the sources are now allowed to choose the file ids.

Definition 11. An *unconstrained multi-source network coding signature scheme* (unMulNCS) is defined by a set of probabilistic, polynomial-time algorithms,

$$(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{VerifySingle}, \mathsf{ConsistencyCheck}),$$

where $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Combine}$ have the same functionality as in Section 3.1, $\mathsf{VerifySingle}$ is the same as the Verify algorithm in Section 3.1, and $\mathsf{ConsistencyCheck}$ is defined as follows:

$\mathsf{ConsistencyCheck}(\{\overline{\mathbf{y}_i}\}_1^{fm})$.
Input: A set of fm vectors $\mathbf{y}_i$.
Output: 0 (reject) if vectors are inconsistent or 1 (accept) if vectors are consistent.

The addition of the new algorithm $\mathsf{ConsistencyCheck}$ is necessary to foil the attack described in section 3.1. This algorithm checks that the set of vectors that are used for decoding are pairwise consistent. If each received vector is represented by a node on a graph and edge between vectors indicate that they are consistent, it is evident that to find a set of m consistent vectors, it is enough to solve the m-clique problem on this graph. We show next that it is also *necessary* to solve the clique problem for decoding, i.e. clique reduces to decoding in this setting.

For simplicity, we consider a reduction from the computational rather than decisional version of the clique problem (thus NP-hard rather than NP-complete).

C.2 Consistent decoding

As described earlier, the senders in the multi-source network inject a set of vectors, say m, into the network, which are linearly combined together along the way. The receiver receives a set of thus aggregated vectors, where each vector contains data, an augmentation header, and a digital signature. The decoding problem entails choosing a set of m legitimate and linearly independent vectors, which are then used to solve a linear system (via matrix inversion) and thus determine the original message vectors sent by the senders.

Hence, the multi-source network coding decoding problem DECODE(S, m) takes as input a pair (S, m) where S is a set of vectors that correspond to vectors received by the receiver, and an integer $m < |S|$ that corresponds to the total number of message vectors injected into the network by the senders. The problem is to output a set of m vectors which are the original message vectors sent by the senders.

As described above, this is a two step process, the first of which involves choosing a set $S' \subset S$, where $|S'| = m$, such that all vectors in S' can be used in the matrix inversion step to recover the original message vectors. As demonstrated by the attack, the m vectors chosen for matrix inversion need to be 'consistent' in that they are all created using the same set of basis vectors. We call this problem of choosing m consistent vectors as CHOOSE-CONSISTENT(S, m).

The (computational) CLIQUE(G, m) problem takes as input graph G and integer $m \leq |G|$ and outputs a clique in G of size m.

Consider that we have access to a network coding decoder oracle. We use this oracle to solve the clique problem.

Theorem 12. *Consistent decoding in the unMulNCS setting is NP-hard.*

Specifically, we prove the following claim, which immediately implies the theorem:

Claim: CLIQUE$(G = (V, E), \frac{|V|+2}{2}) \leq$ CHOOSE-CONSISTENT$(2|V|, |V| + 2)$

Proof. Consider a graph $G = (V, E)$ with $|V| = k$. We want to find a clique of size $\frac{k+2}{2}$ in it. We convert the given instance of $CLIQUE(G, \frac{k+2}{2})$ into a network coding instance in the following manner.

First, we assign an arbitrary ordering to the nodes of G, say $v_1, \ldots, v_k$. We will now associate a unique vector of length k to each node in V. We denote the vector associated with node v_i as u_i and the j^{th} co-ordinate of u_i as $u_{i,j}$ for $j \in \{1, \ldots, k\}$. We proceed as follows:

for $i = 1 \ldots k$ **do**
  for $j = 1 \ldots k$ **do**
    if $j = i$ **then**
      set $u_{i,j} = 1$
    end if
    if $j \neq i$ **then**
      if $(v_i, v_j) \in E$ **then**
        set $u_{i,j} = 1$
      else
        set $u_{i,j} = 0$
      end if
    end if
  end for
end for

This procedure ensures that for each pair of nodes (v_i, v_j), the associated vectors (u_i, u_j) are compatible if and only if there is an edge between v_i and v_j in G.

Next, consider the set of vectors $u_1, \ldots, u_k$ produced by the above procedure. For each $i \leq k$, we "clone" the vector u_i into two "extended" copies u'_i and u''_i of dimension $k + 2$, such that

$$u'_{i,j} = u''_{i,j} = u_{i,j} \quad \text{for } 1 \leq j \leq k \ , \qquad u'_{i,k+1} = u''_{i,k+2} = 1 \ , \qquad u'_{i,k+2} = u''_{i,k+1} = 0 \ .$$

We associate u'_i and u''_i to the same vertex of G as u_i. This duplication process ensures that the two extended vectors u'_i and u''_i are not identical, yet match u_i in the first k coordinates, and are compatible with each other. This ensures that u'_i and u''_i have the same set of compatibilities with u'_j and u''_j, for $j \neq i$, as the original vector u_i does with u_j.

Once this construction of vectors has been performed, we let S denote the resulting set of $2k$ vectors, namely, $S = \cup_{i=1}^{k} \{u'_i, u''_i\}$. We query the CHOOSE-CONSISTENT oracle with $S, (k + 2)$.

The oracle returns a set S' of $k + 2$ pairwise compatible vectors, where, by construction, two vectors are compatible iff their associated nodes have an edge between them. We map each vector in S' to its corresponding node in G, omitting duplicates (which will arise every time S' contains both u'_i and u''_i for some i). The result is a clique of size at least $\frac{k+2}{2}$ in G, as required. □