# Null Keys: Limiting Malicious Attacks Via Null Space Properties of Network Coding

Elias Kehdi, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
{*elias, bli*}*@eecg.toronto.edu*

***Abstract*—The performance of randomized network coding can suffer significantly when malicious nodes corrupt the content of the exchanged blocks. Previous work have introduced error correcting codes by generalizing some well known bounds in coding theory. Such codes are based on introducing redundancy in space domain. Other approaches require the use of homomorphic hashing functions, which are computationally expensive.**

**In this paper, we present a novel and computationally efficient security algorithm, referred to as *Null Keys*, to detect and contain malicious attacks based on the subspace properties of random linear network coding. The participating nodes verify the integrity of a block by checking if it belongs to the subspace spanned by the source blocks. This is possible when every node has a vector orthogonal to all the combinations of the source blocks. These vectors, referred to as null keys, belong to the null space of the source blocks and go through a random combination when distributed by the source. Unlike previous security approaches, our *Null Keys* algorithm allows nodes to rapidly detect corrupted blocks without changing the code or imposing redundancy on the exchanged data. We analytically evaluate the pollution produced by jamming attacks, and demonstrate the effectiveness of *Null Keys* by varying the strength of the malicious nodes. We also show, through extensive simulations, that the *Null Keys* approach is more effective than cooperative security using homomorphic hashing when it comes to limiting the pollution spread.**

## I. Introduction

Network coding allows participating nodes in a network to code incoming data flows rather than simply forwarding them, and its ability to achieve the maximum multicast flow rates in directed networks was first shown in the seminal paper by Alswede *et al.* [1]. Koetter *et al.* [2] have later shown that by coding on a large enough field, linear codes are sufficient to achieve the multicast capacity, and Ho *et al.* [3] have shown that the use of random linear codes — referred to as *random network coding* — is a more practical way to design linear codes to be used. Gkantsidis *et al.* [4] have applied the principles of random network coding to the context of peer-to-peer (P2P) content distribution, and have shown that file downloading times can be reduced.

However, these salient advantages of network coding are only applicable in networks consisting of trustworthy nodes, which is not the case in more realistic scenarios, where protection against malicious attacks remains to be a major challenge. When participating nodes are allowed to code incoming blocks, the network becomes more susceptible to *jamming attacks*. In a jamming attack, first studied in [5] in the context of content distribution with network coding, a malicious node can generate a corrupted block and send it to its downstream nodes, who then unintentionally combine it with other legitimate coded blocks to create a new encoded block. As a result, a single corrupted block pollutes the network and prevents the receivers from decoding, and such pollution can rapidly propagate in the network, leading to substantially degraded performance due to the wasted bandwidth distributing corrupted blocks. Needless to say, there exists a strong motivation to check coded blocks on-the-fly to see if they are corrupted, before using them for encoding.

The proposed solutions to address jamming attacks with network coding fall in two categories: *error correction* and *error detection*. A class of *network error correcting* codes, first introduced by Cai and Yeung [6], aim at correcting corrupted blocks at sink nodes by introducing a level of redundancy. However, encoding and decoding at participating nodes with network error correcting codes proposed in the literature is computationally complex; and since such error correction is performed at receivers, bandwidth consumed by corrupted blocks at relay nodes will not be reclaimed or reduced. It may also be challenging to incorporate a sufficient level of redundancy to guarantee that all errors are corrected in large networks.

In comparison, *error detection* schemes allow intermediate nodes to verify the integrity of the incoming blocks, and to make a local decision on whether or not a block is corrupted. Intuitively, if corrupted blocks are detected before they propagate to downstream nodes, bandwidth will not be wasted on sending them. However, such verifications require hashes that are able to survive random linear combinations, since the received coded blocks are linearly combined with random coefficients without decoding. *Homomorphic hashing* has first been introduced by Krohn *et al.* [7] to allow intermediate nodes to detect corrupted blocks. However, homomorphic hash functions are also computationally complex to compute, and since each node needs to verify all incoming blocks before using them, the performance of the network would be limited by the rate of computationally processing their homomorphic hashes.

In this paper, we propose a novel and computationally simple verification algorithm, referred to as *Null Keys*. Similar to other error detection algorithms based on homomorphic hash functions, the *Null Keys* algorithm allows each node to verify that an incoming block is not corrupted, and as such limit malicious jamming attacks by preventing the propagation of corrupted blocks to downstream nodes. However, unlike

previously proposed algorithms in the literature, the *Null Keys* algorithm allows nodes to rapidly verify incoming blocks without the penalty of computational complexity. Rather than trying to find a suitable existing signature scheme for error detection, the *Null Keys* algorithm is designed specifically for random linear network coding.

The idea in *Null Keys* is based on the randomization and the subspace properties of random network coding. We take advantage of the fact that in random linear network coding, the source blocks form a subspace and any linear combination of these blocks belongs to that same subspace. In our approach, the source provides each node with a vector from the null space of the matrix formed by its blocks. Those vectors, referred to as null keys, map any legitimate coded block (that is not corrupted) to zero. Thus, the verification process is a simple multiplication that checks if the received block belongs to the original subspace. Similar to the source blocks, the null keys go through random linear combinations, which makes it hard for a malicious node to identify them at its neighbors. The null keys can be secured using homomorphic hash functions since they do not impose a significant overhead on the network. The *Null Keys* algorithm does not require any additional coding complexity as in previous approaches on error detection, nor add redundancy to the original blocks, as in previous approaches on error correction. Using analytical and simulation based studies, we compare *Null Keys* with homomorphic hashing, and validate its effectiveness on restricting the pollution caused by malicious jamming attacks.

With respect to computational complexity, for a block of size $m$, the verification process using *Null Keys* requires $O(c \cdot m)$ operations, where $c$ is the minimum cut between a node and the source. In contrast, as shown in [7], the expected per-block cost for a hash verification is $(m\lambda_q/2 + d)\text{MultCost}(p)$, where $\lambda_q$ is a large random prime security parameter and $\text{MultCost}(p)$ is the cost of multiplication in $\mathbb{Z}_p$. Using probabilistic verifications, *cooperative security* [5] reduces the computation complexity of hashing at the cost of lowering the security level. The *Null Keys* algorithm is shown to decrease the percentage of corrupted nodes by around $15\%$ compared to *cooperative security*, in which $20\%$ of the blocks are probabilistically checked.

The remainder of the paper is organized as follows. In Section II, we present related work on error correction and detection schemes. In Section III, we review the properties of subspaces and null spaces. In Section IV, we describe the *Null Keys* algorithm. In Section V, we model the malicious behavior and evaluate our security approach by presenting theoretical analysis of the verification process. In Section VI, we show the simulation results and discuss the ability of *Null Keys* to limit the pollution spread, as compared to cooperative security that uses homomorphic hashing. Finally, we conclude the paper in Section VII.

## II. Related Work

Several approaches were taken to design codes that can correct errors at sink nodes. The codewords are chosen such that the minimum distance between them allows sink nodes to decode the messages even when they are mixed with error blocks. Cai and Yeung were the first to introduce network error correcting codes [6]. Similarly, Zhang defines the minimum rank of network error correction codes based on error space [8]. In [9], Jaggi *et al.* designed their code using binary erasure channel (BEC) codes. Nutman *et al.* also studied causal adversaries in the distributed setting [10]. On the other hand, Koetter and Kschischang [11] designed a coding metric on subspaces and proposed a minimum distance decoder, based on a bivariate linearized polynomial.

However, network error-correction codes introduce overhead and require a large amount of computations in both the encoding and the decoding stages. This can result in an increase in time delay and a reduction of network performance. Hence, the error-correcting codes presented are not practical. In our approach, we do not modify the code and assume all the nodes are receivers. We allow the intermediate nodes to detect corrupted blocks, instead of waiting for sink nodes to verify the received messages. The goal is not to correct the errors but, instead, to limit attacks from malicious nodes by preventing legitimate nodes from using corrupted blocks they receive.

Along the lines of error detection, Krohn *et al.* [7] presented a scheme that allows the nodes to perform on-the-fly verification of the blocks exchanged in the network. The approach is based on a homomorphic collision-resistant hash function (CHRF) that can survive random linear combinations. To improve the verification process, Krohn *et al.* propose to verify the blocks probabilistically and in batches instead of verifying every block. But, the batching puts the downloaders at risk of successful attacks. Another main drawback of this scheme is the large number of hash values required. The size of the hash values is proportional to the number of blocks. To solve this problem, Li *et al.* [12] proposed a new homomorphic hash function based on a modified trap-door one-way permutation. Instead of sending the homomorphic hashes, a random seed is distributed. Gkantsidis *et al.* [5] proposed a cooperative security scheme, where nodes cooperate to protect the network. Cooperative security reduces the time required by the homomorphic hashes, by introducing more overhead in the network imposed by alert messages. The security approaches in [7] and [5], assume the existence of a separate channel or a trusted party. Charles *et al.* [13], on the other hand, introduce a new homomorphic signature scheme based on elliptic curves and does not need any secret channel.

Although homomorphic hashes are suitable for random linear combinations, they are very complex. This lead to a number of subsequent research papers that tried to reduce their complexities [5], [12], [13]. With homomorphic hashing the size of the hash values is proportional to the number of blocks. However, in our approach, the number of null keys introduced is limited to the out-degree of the source node. Moreover, we do not add any redundancy to the source blocks. We do not attempt to find a method that provides a higher level of security than homomorphic hashes, but instead we

aim at finding a simpler approach that guarantees to limit the damage of jamming attacks and stop the malicious nodes from polluting the network.

## III. Overview of Subspaces and Null Spaces

Consider a network where a source $S$ has $r$ blocks, each represented by $d$ elements from the finite field $\mathbb{F}_q$. The source augments block $i$ with $r$ symbols, with one at position $i$ and zero elsewhere, to form the vector $x_i$. We denote by $X$ the $r \times (r+d)$ matrix whose $i^{\text{th}}$ row is $x_i$. These source blocks form a set of $r$ independent vectors that span a subspace $\Pi_X$. Any linear combination of the vectors $\{x_1, .., x_r\}$ belongs to the same subspace $\Pi_X$. In other words, $\Pi_X$ is closed under random linear combinations. The common property shared by all the blocks at the participating nodes is the fact that they all belong to the subspace $\Pi_X$ spanned by the basis vectors $\{x_1, ..., x_r\}$.

The null space of the matrix $X$, denoted as $\Pi_X^{\perp}$, is the set of all vectors $z$ for which $Xz = 0$. It is the *kernel* of the mapping defined by the matrix multiplication $Xz$. For any $m \times n$ matrix $A$, we have

$$\text{rank}(A) + \text{nullity}(A) = n, \tag{1}$$

known as the rank-nullity theorem, where the dimension of the null space of $A$ is called the *nullity* of $A$. Applying the rank-nullity theorem, with network coding, the dimension of $\Pi_X^{\perp}$ is equal to $d$, the original dimension of the source blocks before appending the $r$ symbols. The subspace $\Pi_X^{\perp}$ is spanned by the basis vectors $\{z_1, ..., z_d\}$. We denote by $Z$ the $d \times (r+d)$ matrix whose $i^{\text{th}}$ row is $z_i$. Similarly, the null space is closed under random linear combinations. Hence, Gaussian elimination can be used to find the basis for $\Pi_X^{\perp}$.

With network coding, the blocks exchanged in the network are random linear combinations of $\{x_1, ..., x_r\}$ and belong to $\Pi_X$. Each one of these blocks is orthogonal to any combination of $\{z_1, ..., z_d\}$ which belongs to the subspace $\Pi_X^{\perp}$.

## IV. Null Keys Algorithm

The only property shared by the exchanged blocks is the fact that they belong to the same subspace $\Pi_X$, as referred to in Section III. This vector space is selected by the source node and injected in the network. We propose to verify the integrity of the blocks by checking if they belong to the subspace $\Pi_X$. As mentioned in Section III, all the vectors in $\Pi_X$ are orthogonal to any combination of the basis vectors of the null space $\Pi_X^{\perp}$. In our proposed security scheme, each one of the participating nodes is provided with vectors from $\Pi_X^{\perp}$, referred to as null keys, to verify that the received blocks satisfy the orthogonality condition. As any participating node, the malicious nodes also have access to null keys. If an attacker knows the values of the null keys collected by his neighbor, he can send corrupted blocks that do not belong to $\Pi_X$ but are orthogonal to those null keys, and pollute his neighbor's cache. However, with the path diversity and the distributed randomness, it is extremely hard for a malicious node to identify those null keys. We argue that the randomization and subspace properties of network coding can limit the pollution attacks of the malicious nodes. We evaluate the efficiency of our security scheme through theoretical analysis in Section V, and simulations in Section VI.

In this paper, we use the notations of [2]. We model the network by a directed random graph $G = (V, E)$, where $V$ is the set of nodes, and $E$ is the set of edges. One source $S \in V$ wishes to send its blocks to all the participating nodes $v \in V$. Edges are denoted by $e = (v, v') \in E$, where $v =$head$(e)$ and $v' =$tail$(e)$. The set of edges that end at a vertex $v \in V$ is defined as $\Gamma_I(v) = \{e \in E : \text{head}(e) = v\}$ while the set of edges originating at $v$ is defined as $\Gamma_O(v) = \{e \in E : \text{tail}(e) = v\}$. The in-degree of the vertex $v$ is denoted by $\delta_I(v) = |\Gamma_I(v)|$ and the out-degree of $v$ is denoted by $\delta_O(v) = |\Gamma_O(v)|$. The source node $S$ sends $r$ blocks $\{x_1, ..., x_r\}$ of size $r + d$, that span a $r$-dimensional subspace $\Pi_X$ defined over the finite field $\mathbb{F}_q$.

A received vector at any node $v$ is valid if and only if it belongs to $\Pi_X$. Each node verifies the integrity of the received blocks using the orthogonality principle discussed in Section III. The source provides the participating nodes with keys to perform the verification process. Using the blocks $\{x_1, ..., x_r\}$, it forms the basis vectors $\{z_1, ..., z_d\}$ that span the null space $\Pi_X^{\perp}$ of $X$. Instead of sending one private key to the trustworthy nodes, the source sends a random linear combination of the vectors $\{z_1, ..., z_d\}$ on each of its out-going edges $e \in \Gamma_O(S)$ destined to all the participants including the malicious nodes. Those vectors, referred to as null keys, are used in the verification process.

During the distribution of null keys, the nodes' behavior is not changed. They still transmit random linear combinations of incoming blocks to their neighbors. When a node $v_i$ receives one key from each of its incoming edges $e \in \Gamma_I(S)$, it forms the matrix of null keys $K_i$ and sends a random linear combination of the received null keys on each of its outgoing links $e \in \Gamma_O(S)$. Each node $v_i$ that has formed the matrix $K_i$ can verify the integrity of a received data block $w$ by checking if it satisfies the following condition,

$$K_i w^T = 0. \tag{2}$$

Equation (2) is a simple multiplication that allows the nodes to rapidly perform the verification. However, if the condition is satisfied, it does not imply that the received vector belongs to $\Pi_X$. In case the malicious node knows the values of the null keys collected by its neighbor, it can easily find a corrupted vector that satisfies Equation (2). However, for a malicious node that does not know the content of its neighbor, it is hard to find a corrupted block that can pass the verification process. A detailed explanation and proofs are provided in Section V. Once a bogus block is detected, it is dropped and the malicious node can be isolated from the network. The *Null Keys* algorithm is shown in Algorithm 1, where random coefficients are denoted by $C_i$.

In order to prevent malicious nodes from faking the null keys, the source signs them. Since, as the data blocks, the keys go through random linear combinations, they are protected using homomorphic hashing. As indicated in Section II,

**Algorithm 1** *Null Keys Algorithm.*

**Algorithm** NullKeys($X$)
  **for** all $v_i \in V$
    **if** $v_i$ is $S$
      $X^{\perp} \leftarrow$ nullspace($X$)
      **for** $j = 1$ to $\delta_O(S)$
        $e_j \leftarrow C_j X^{\perp}$
    **else**
      **for** $j = 1$ to $\delta_I(v_i)$
        $K_i \leftarrow [K_i;\ e_j]$
      **for** $j = 1$ to $\delta_O(v_i)$
        $e_j \leftarrow C'_j K_i^{\perp}$

**Algorithm** Verification($v_i,\ w$)
  **if** $K_i w^T$ is equal to 0
    $w$ is safe
  **else**
    $w$ is corrupted

homomorphic hashes provide a high level of security, but, the large number of hash values and computation costs required lead to significant delay and degrade the network performance. However, in our approach, homomorphic hashes are used to sign only the null keys. The source node injects only $\delta_O(S)$ null keys in the network. Hence the number of hash values required is not significant. On the other hand, the computation cost is limited to the in-degree $\delta_I(v)$ of each node $v$, only when distributing the keys. Every node $v$, performs a total number of $\delta_I(v)$ verifications of null keys. Thus, in our security scheme, the signature of the null keys does not impose significant latency or computation cost. The technique proposed by Li *et al.* [12] can be applied. This scheme, based on pseudo-random generators, does not assume the existence of a secret channel.

## V. Security Evaluation and Theoretical Analysis

We adopt some of the notations and lemmas used by Jafarisiavoshani *et al.* [14], who investigated the connection between subspaces and topological properties of network coding. We denote by $P(i)$ the parents of node $v_i$ and by $P^l(i)$ the set of parents of $v_i$ at level $l$ such that $P^l(i) = P(P^{l-1}(i))$. Let $\mathcal{E}_{ij}$ be the set of edges on the paths connecting nodes $v_i$ and $v_j$, and $\mathcal{E}_i$ be the set of edges between the source and $v_i$. Denote by $c_{ij} = \text{mincut}(v_i, v_j)$ the minimum cut between nodes $v_i$ and $v_j$, and $c_i$ the minimum cut between the source and node $v_i$.

### A. Malicious Model

In our model, the threat is imposed by internal malicious nodes that have access to the information exchanged in the network as other participating nodes. Intuitively, a malicious node $v_m \in V$ has an in-degree $\Gamma_I(v_m)$, an out-degree $\Gamma_O(v_m)$ and has access to $\delta_I(v_m)$ null keys. Malicious nodes do not send any null key that contributes in protecting their neighbors. On the contrary, the null keys sent to downstream nodes are assumed to be recognized by all malicious nodes. Those malicious nodes continuously attempt to pollute the network by injecting bogus blocks to neighbors. Jamming attacks succeed when the injected block is not a valid linear combination of the source blocks $\{x_1, ..., x_r\}$, but passes the verification process. In other words, the malicious attack is successful if the corrupted block does not belong to $\Pi_X$ but is orthogonal to the null keys of the receiver.

In Section V-B, we show that sending random blocks is not an efficient strategy to jam a network protected with *Null Keys*. However, a malicious node can take advantage of the information it has access to. Consider the case where a malicious node $v_m$ and node $v_i$ receive the null keys matrices $K_m$ and $K_i$ respectively. A corrupted block should map $K_i$ to zero in order to pass the verification process at node $v_i$. In case the row space of $K_i$ is included in the row space of $K_m$, the malicious node can pollute $v_i$'s cache by sending blocks that are orthogonal to the rows of $K_m$, but do not belong to $\Pi_X$. This defines the smart malicious behavior in our model. A malicious node sends to neighbor nodes, random combinations of vectors that span the null space of $K_m$ but do not belong to $\Pi_X$. All the neighbor nodes, that possess null keys spanning a space included in the row space of $K_m$, would be corrupted. Hence, a malicious node is stronger when the rows of $K_m$ span a larger space.

### B. Malicious Behavior Evaluation

We use the following lemmas in the proof of theorems that characterize the security level of *Null Keys*.

*Lemma 1:* Let $A$ be a $m \times n$ matrix consisting of $m$ independent blocks of dimension $n$, in the finite field $\mathbb{F}_q$. The probability that a random $n$-dimensional vector maps $A$ to zero is $\frac{1}{q^m}$.

*Proof:* Any $n$-dimensional vector $w$, that maps $A$ to zero, belongs to $\Pi_A^{\perp}$, the null space of $A$. Following Equation (1), $\dim(\Pi_A^{\perp})$ is equal to $n-m$. Hence the probability of choosing a random vector that maps $A$ to zero is

$$\Pr(Aw^T = 0) = \frac{q^{n-m}}{q^n} = \frac{1}{q^m}.$$

■

*Lemma 2:* Consider a network where the source $S$ has $d$ independent null keys. If node $v_i$ has a $\text{mincut}(S, v_i) = c_i \leq d$, then with high probability $\dim(\Pi_{K_i}) = c_i$.

*Proof:* The source $S$, sends one random linear combination of its null keys on each of its outgoing links. With a $\text{mincut}(S, v_i) = c_i$, node $v_i$ receives $c_i$ random linear combinations from the source $S$. This is equivalent to the case where $v_i$ constructs its subspace by selecting $c_i$ keys, uniformly at random, from the $d$ source null keys. By Lemma 1 of [14], with high probability, $\dim(\Pi_{K_i}) = c_i$. ■

If a malicious node attempts to jam the network by sending random blocks to neighbor nodes, with high probability the attack fails. In fact, by Lemma 2, node $v_i$ collects null keys matrix $K_i$ of dimension $c_i \times (r + d)$. Applying Lemma 1, the probability of polluting the content of $v_i$ is $\frac{1}{q^{c_i}}$.

Since the random attack is not efficient, a malicious node can benefit from the data blocks or null keys it receives. However, out of all possible random linear combinations of $X^{\perp}$, only $\delta_O(S)$ null keys are injected in the network. Thus, the data blocks are not useful for finding the injected null keys and the appropriate bogus blocks that map $K_i$ to zero. Even when a malicious node is able to decode all the data blocks, it cannot determine the null keys combinations at its neighbor nodes. On the other hand, as other participating nodes, a malicious node has access to null keys exchanged in the network. The most efficient behavior, for a malicious node $v_m$, is to send blocks belonging to the null space of $\Pi_{K_m}$. Following this procedure, any node $v_i$, with $\Pi_{K_i} \subseteq \Pi_{K_m}$, would be polluted. In such scenario we have:

*Theorem 1:* The source $S$ injects $\delta_O(S)$ null keys in the network. Suppose that each node $v_i$, including the malicious nodes, selects $c_i \leq \delta_O(S)$ random linear combinations of the injected keys. Then, the probability that a malicious node $v_m$ conducts a successful attack is $\frac{1}{q^{\delta_O(S)-c_m}}$.

*Proof:* A malicious node $v_m$ can corrupt the content of another node $v_i$ by sending a bogus block $w$ that maps $K_i$ to zero. The null keys $K_i$ and $K_m$ are random linear combinations of the keys injected by the source. By Lemma 1 of [14], with high probability, $\dim(\Pi_{K_i}) = c_i$, $\dim(\Pi_{K_m}) = c_m$ and $\dim(\Pi_{K_i} \cap \Pi_{K_m}) = c_i + c_m - \delta_O(S)$.

The matrix $K_i$ can be represented by the basis vectors $\beta_i$ that span $\Pi_{K_i} \cap \Pi_{K_m}$ and the remaining basis vectors $\alpha_i$.

$$K_i = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_{c_i+c_m-\delta_O(S)} \\ \alpha_1 \\ \vdots \\ \alpha_{\delta_O(S)-c_m} \end{bmatrix}.$$

Following the malicious behavior defined in Section V-A, $w$ belongs to $\Pi^{\perp}_{K_m}$. Since the vectors $\beta_i \in \Pi_{K_m}$, $w$ maps the sub matrix formed by the basis vectors $\beta_i$ to zero with probability equal to one. Applying Lemma 1, the probability that $v_m$ conducts a successful attack is

$$\Pr\left( \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{\delta_O(S)-c_m} \end{bmatrix} w^T = 0_{(\delta_O(S)-c_m)\times 1} \right) = \frac{1}{q^{\delta_O(S)-c_m}}.$$

■

Applying Theorem 1, if the minimum cut between the source and the malicious node is equal to the number of injected null keys, the malicious node can successfully attack any participant in the network. However, if the minimum cut is less than the number of injected null keys, the probability drops to the order $O(\frac{1}{q})$. This is true, if the selection of the keys does not take into account the paths shared by the nodes.

As shown in [14], the subspaces collected at the nodes is connected to the topology of the network. The shared parents, the connections and the minimum cut between the participants, define the topology of the network. Since the efficiency of malicious attacks depends on the intersection of subspaces collected by the nodes, those factors should be considered in evaluating the *Null Keys* algorithm.

### C. Topological Approach

The intersection of the subspace spanned by the null keys of a malicious node with other subspaces, depends on its connections with other nodes. As shown in Theorem 1, if the subspace collected by a participant is not included in the subspace collected by a malicious node, then with high probability the participant is protected from pollution attacks. Hence, the capabilities of a malicious node is limited to the topology and its location in the network.

Consider the case where a malicious node $v_m$ is trying to attack node $v_i$. The intersection between $\Pi_{K_i}$ and $\Pi_{K_m}$ depends on the parents shared by $v_i$ and $v_m$, and the paths $\mathcal{E}_{im}$ connecting them. In fact, if those factors are not considered, then, as shown in Lemma 1 of [14], the intersection between subspaces is the minimum possible. The conditions on the minimum cut between nodes $v_i$ and $v_m$, depend on the shared paths to the source. We have:

*Corollary 1:* Suppose that there exist $t$ paths between the source and node $v_i$, excluding $\mathcal{E}_{im}$, that do not intersect with any cut between the source and node $v_m$. If $c_{im} < t$, then $\Pi_{k_i} \nsubseteq \Pi_{k_m}$.

*Proof:* The information sent on the $t$ paths are hidden from $v_m$. If $\mathcal{E}_{im}$ are removed, $\Pi_{K_i} \cap \Pi_{K_m} \leq c_i - t$. Hence, $v_m$ has to collect at least $t$ independent blocks from $\Pi_{K_i}$ on $\mathcal{E}_{im}$ in order to successfully attack $v_i$. However, the number of independent blocks on $\mathcal{E}_{im}$ is limited by $c_{im}$. Thus, if $c_{im} \leq t$ then $\Pi_{k_i} \nsubseteq \Pi_{k_m}$. ■

In the case of an acyclic graph, $c_{im} = 0$. Therefore, if there exists one path from $S$ to $v_i$ that does not intersect with any cut from $S$ to $v_m$, then, by Corollary 1, $\Pi_{K_i} \nsubseteq \Pi_{K_m}$. As the network size grows, the probability of the existence of such path increases. Hence, $\Pr(\Pi_{K_i} \subseteq \Pi_{K_m})$ decreases and malicious attacks are better restricted. A more general graphical interpretation is provided in the following theorem that also considers the minimum cut between the parents of nodes $v_i$ and $v_m$.

*Theorem 2:* Let $P^c_{im} = \{v_p \in \bigcup_l P^l(i) \cap \bigcup_l P^l(m) | \mathcal{E}_{pm} \nsubseteq \mathcal{E}_{pi} \cup \mathcal{E}_{im} \text{ and } \mathcal{E}_{pi} \nsubseteq \mathcal{E}_{pm} \cup \mathcal{E}_{mi}\}$ be the set of common parents providing blocks to $v_i$ and $v_m$ on different paths, excluding the source $S$. Let $\tilde{v_p}$ be a super node belonging to $\mathcal{P}(P^c_{im})$, the power set of $P^c_{im}$. If

$$\text{mincut}(S, \{v_i\} \cup \tilde{v_p}) > \text{mincut}(\{v_i\} \cup \tilde{v_p}, v_m), \forall \tilde{v_p} \in \mathcal{P}(P^c_{im}),$$

then,

$$\Pi_{K_i} \nsubseteq \Pi_{K_m}.$$

*Proof:* If the minimum cut difference between $\text{mincut}(S, v_i)$ and $\text{mincut}(v_i, v_m)$ is equal to $t$, then $\Pi_{K_i} \nsubseteq \Pi_{K_m}$, unless there exist at least $t$ of $v_i$'s upstream nodes that possess null keys belonging to $\Pi_{K_m}$. Those nodes

are parents shared by $v_i$ and $v_m$, hence belong to $P^c_{im}$. When they are grouped with node $v_i$ forming a super node, this minimum cut difference decreases by at least $t$. However, if $\text{mincut}(S, \{v_i\} \cup \tilde{v_p}) > \text{mincut}(\{v_i\} \cup \tilde{v_p}, v_m)$, for all possible $\tilde{v}_p \in \mathcal{P}(P^c_{im})$, then $\Pi_{K_i} \nsubseteq \Pi_{K_m}$. ■

Figure 1 is an example that illustrates Theorem 2. In this example, $m = 7$ and $i = 6$. The set of common parents is $P^c_{67} = \{v_1, v_2, v_3\}$. Hence, $\mathcal{P}(P^c_{67}) = \{\emptyset, \{v_1\}, \{v_2\}, \{v_3\}, \{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_1, v_2, v_3\}\}$. All the sets in $\mathcal{P}(P^c_{67})$ satisfy the condition in Theorem 2 except $\tilde{v_p} = \{v_1, v_2\}$, as shown in the figure. In this case, $\text{mincut}(S, \{v_6\} \cup \tilde{v_p})$ is equal to $\text{mincut}(\{v_6\} \cup \tilde{v_p}, v_7) = 3$.

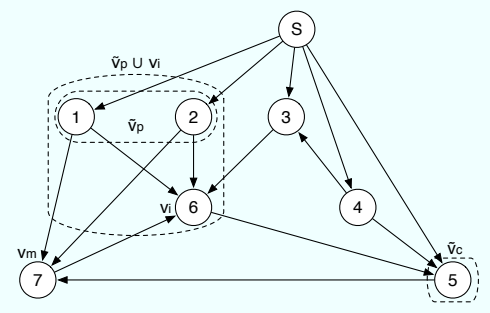



Fig. 1. A network consisting of 8 nodes. The malicious node is $v_7$ and the target node is $v_6$.

It is more probable to find a set $\tilde{v_p}$ that does not satisfy the condition in Theorem 2, if $|\mathcal{P}(P^c_{im})|$, the cardinality of $\mathcal{P}(P^c_{im})$, is larger. Therefore, for larger values of $\delta_I(v_m)$, the number of incoming edges to $v_m$, the malicious node is able to share a larger number of parents with other nodes and hence it can conduct successful attacks with higher probability.

However, when we consider Theorem 2 for nodes $v_i$ and $v_m$, having a sufficient minimum cut between the super node $\{v_i\} \cup \tilde{v_p}$ and node $v_m$ does not imply that $v_m$ is capable of corrupting the content of $v_i$. For instance, a null key sent on a path, crossing that minimum cut, does not increase the intersection between $\Pi_{K_i}$ and $\Pi_{K_m}$ once it is mixed with other random null keys. This random combination generates different values of null keys and guarantees an efficient protection against jamming attacks. The following theorem presents the conditions that apply on the paths crossing the minimum cut between $v_i$ and $v_m$.

*Theorem 3:* Consider a path $p$ from $v_i$ to $v_m$. Let $\tilde{v_c}$ be the set of nodes on the path $p$. If $\exists v_c \in \tilde{v_c}$ such that $\Pi_{K_c} \nsubseteq \Pi_{K_m}$, then the null key received on the path $p$ does not contribute in increasing the probability that $v_m$ successfully attacks $v_i$.

*Proof:* Denote by $\tilde{K_c}$ the null keys matrix of $\tilde{v_c}$. Since node $v_i$ sends a null key to $\tilde{v_c}$ on the path $p$, then $\dim(\Pi_{K_i} \cap \Pi_{\tilde{K_c}}) \geq 1$. Therefore, if $\Pi_{\tilde{K_c}} \subseteq \Pi_{K_m}$ then $\dim(\Pi_{K_m} \cap \Pi_{K_i}) \geq 1$. On the other hand, applying Lemma 2 of [14], if $\Pi_{\tilde{K_c}} \nsubseteq \Pi_{K_m}$, then with high probability, the null key sent by $\tilde{v_c}$ to $v_m$ on the path $p$ does not belong to $\Pi_{K_i}$. Hence, the path $p$ does not contribute in increasing the dimension of $\Pi_{K_i} \cap \Pi_{K_m}$, unless, $\Pi_{\tilde{K_c}} \subseteq \Pi_{K_m}$. This condition is satisfied if and only if $\forall v_c \in \tilde{v_c}, \Pi_{K_c} \subseteq \Pi_{K_m}$. ■

In the example shown in Figure 1, $\tilde{v}_c = \{v_5\}$. The null keys received by $v_5$ span a space $\Pi_{K_5} \nsubseteq \Pi_{K_7}$. Thus, the null key sent on the path $(v_5, v_6)$ does not increase the intersection space $\Pi_{K_6} \cap \Pi_{K_7}$. Although there exists a $\tilde{v}_p = \{v_1, v_2\}$, that does not satisfy the condition in Theorem 2, $v_7$ fails to attack $v_6$.

Let $p_c = \Pr(\Pi_{K_c} \subseteq \Pi_{K_m})$. Then, by Theorem 3, the probability that the null key sent on path $p$, between $v_i$ and $v_m$, increases $\Pi_{K_i} \cap \Pi_{K_m}$ is $\prod_{\forall v_c \in \tilde{v_c}} p_c$. This probability decreases as the number of nodes on the path $p$ increases. It also applies to the paths connecting the parents of $v_i$ and node $v_m$. In fact, under the conditions of Theorem 2, when the paths between $\tilde{v_p}$ and $v_m$ contain more nodes, it is less probable that $\Pi_{K_i} \subseteq \Pi_{K_m}$. The corruption probability drops when the target node or the shared parents are far from the malicious node. This explains how our security scheme is able to limit the pollution in the network and isolate malicious nodes.

Furthermore, as the network size grows, nodes can select parents from a larger set of participants in a random topology. Hence the probability that a malicious node shares parents with others, drops. Also, it is less probable to find a set of nodes $\tilde{v_p}$, as defined in Theorem 2, that has a sufficient minimum cut with the malicious node. Since the number of participants is larger, if this set exists, the paths that separate it from the malicious node contain a greater number of nodes. Hence, applying Theorem 3, the probability that these paths increase the intersection of subspaces collected by the nodes also drops.

Hence, our security scheme is more efficient when the network size is larger. This is obvious, since the null keys go through a greater number of combinations when there are more participants. Thus, it is less probable that a malicious node shares null keys with neighbor nodes.

The analysis shows that the pollution produced by a malicious node can be limited to a small area covering some of its neighbor nodes. If there exist multiple malicious nodes, our security scheme is capable of isolating each malicious node separately and limiting the pollution spread as we show in Section VI-A. However, as noted in Section V-A, an upstream malicious node does not provide a null key that helps in detecting corruption. Hence, in the condition of Theorem 2, the path between the source and the target node $v_i$ containing a malicious node can be ignored. As a result, when the upstream nodes of $v_i$ are malicious, node $v_i$ is more susceptible to jamming attacks.

Moreover, for large networks, the probability that a malicious node shares enough parents and paths with others is minimal. Therefore, the corruption of node $v_i$ depends on the number of upstream malicious nodes that can limit the effectiveness of $\Pi_{K_i}$. Hence, for large populations, as the number of malicious nodes increases linearly with the network size, the percentage of corrupted nodes stabilizes at a level that depends on the ratio of malicious nodes to the network size.

In the discussion, we assumed that the minimum cut between the source and the malicious node is less than the out-degree of the source. However, if we allow only the

source to transmit twice to its direct downstream nodes when distributing the null keys, we can remove this restriction. In fact, the number of null keys present in the network doubles, and the damage caused by malicious nodes decreases. This is equivalent to doubling the out-degree of the source node. Hence, the minimum cut between the source and the nodes increases. As a result, following the conditions of Theorem 2, $\Pr(\Pi_{K_i} \subseteq \Pi_{K_m})$ decreases. The only drawback in such scenario is that, the direct downstream nodes of the source have to validate the integrity of the null keys twice as many as in the previous scenario. On the other hand, if multiple malicious nodes $\{v_{m_1}, v_{m_2}, \cdots\}$ are allowed to exchange their null keys then they are equivalent to a single malicious node, with an in-degree $\bigcup \Gamma_I(v_{m_i})$, on which the same conditions apply.

### D. Time Delay and Overhead

The distribution of null keys imposes an initial delay on the system. However, the source needs not wait until every node receives its null keys. It can start sending its blocks directly after injecting the null keys. As indicated in Section IV, the operations at the intermediate nodes remain unchanged during the null keys distribution phase. As a node receives a new null key, it can clean its buffer if it contains corrupted blocks.

In *Null Keys*, no redundancy is added to the source blocks. Adding extra bits to the data blocks can consume a significant capacity. For instance, in Charles *et al.* [13], the signature is transmitted together with the data. For a large file, these extra bits add up to become a substantial overhead cost to the system. On the other hand, although other methods that use homomorphic hashing do not add any overhead to the source blocks, they require heavy computations at intermediate nodes. This is the main drawback of those schemes. However, the *Null Keys* algorithm only requires a simple multiplication, which allows the nodes to perform fast verifications. The cost of the security check at node $v_i$ is the computation cost of Equation (2), which necessitates $O(c_i(r+q))$ operations. There is a trade-off between computation complexity and security performance. For instance, homomorphic hashing guarantees a high level of security since the hash functions are collision-free. However, due to the computation cost, security schemes that use homomorphic hashing require that nodes check blocks probabilistically. In our security scheme, we use homomorphic hashing to protect only the null keys. Another drawback of hashing is the large field size that would be required. Limiting the use of homomorphic hashing operations preserves the network performance. Cooperative security is one approach that uses probabilistic checking. In Section VI, we compare the performance of *Null Keys* to cooperative security.

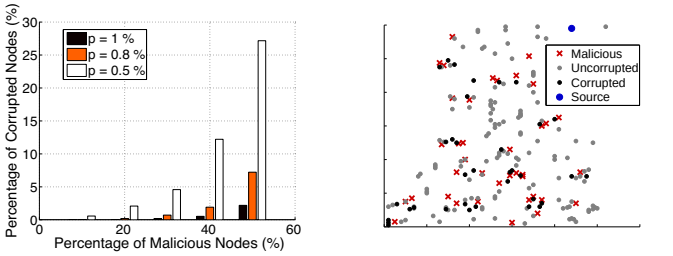
## VI. Simulations

In this section, we present simulation scenarios and results to evaluate the performance of *Null Keys*. We also compare with cooperative security introduced by Gkantsidis *et al.* [5]. In their approach, homomorphic hashing is used, where nodes cooperate to protect themselves by sending alert messages. Gkantsidis *et al.* argue that cooperative security reduces the cryptographic overhead of homomorphic hashes, and show by simulations that cooperation guarantees better efficiency than other probabilistic approaches that use homomorphic hashing.

We use the percentage of corrupted nodes as a metric to measure the efficiency of security schemes with network coding. In the simulations, the network consists of 1000 nodes and the block size is set to 128. The topology is a directed random graph with one source node. Each pair of nodes is connected with a probability $p$. The attack model of malicious nodes is defined in Section V-A. The simulator is round-based, where in each round a node can upload and download blocks. Each round, the attackers send one bogus block on each of their outgoing links. We randomly choose malicious nodes from the population. All the results are averaged over several runs, specified by each scenario.

### A. Pollution Spread

The important factors in studying *Null Keys* are the edge connection probability $p$ and the number of malicious nodes. As the probability $p$ increases, the nodes gain access to a larger set of null keys. On the other hand, when the number of malicious nodes grows, the nodes become more susceptible to jamming attacks. In fact, malicious nodes do not provide neighbor nodes with null keys that can help in protecting their content from corruption.



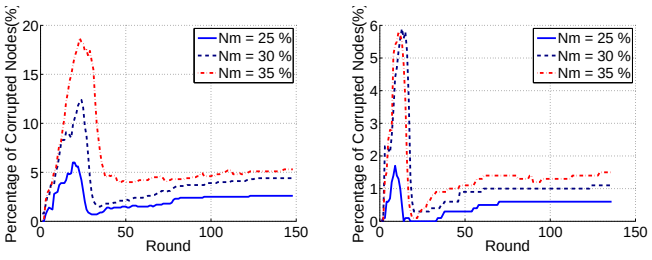

(a) Network consisting of 1000 nodes (b) Snapshot of a network of 250 nodes

Fig. 2. Pollution spread.

Figure 2(a) shows the percentage of corrupted nodes under the attack of different number of malicious nodes. The population is set to 1000 nodes. The result is averaged over 50 runs. Note, for each specific percentage of malicious nodes, the corruption expands as the edge connection probability $p$ decreases. The result validates the theoretical analysis in Section V. When $p$ increases, the connection between the nodes increases. As a result, the nodes gain access to a larger set of null keys that helps in the protection against bogus blocks. In fact, each node connects to a larger set of neighbors, and the number of parents increases. Therefore, the source injects additional null keys and the keys go through a greater number of linear combinations which limits the capabilities of a malicious node. On the other hand, it is clear that increasing the number of malicious nodes expands the pollution spread. The result is obvious since the connection with malicious nodes grows. Thus, the protection capabilities decrease, since

malicious nodes do not provide neighbors with null keys that help detect bogus blocks. Figure 2(b) provides a snapshot of a network consisting of 200 nodes, plotted such that the distance between more connected nodes is shorter. We can clearly observe that the effects of malicious nodes are isolated in the network, and the corruption spread is limited to small areas. This helps to identify the locations of malicious nodes.

In Section V-D, we mentioned that the source node can start distributing its blocks directly after injecting the null keys. In such a scenario, some nodes receive null keys before others. As a result, the number of corrupted nodes varies as the participants receive the keys. Once a node receives a new null key, it can clean its buffer from bogus blocks. Figure 3 shows how the corruption varies in a network of 1000 nodes with an edge connection probability $p$ equal to $0.5\%$ in (a) and $0.7\%$ in (b). It is evident that the corruption can peak at the initial rounds when there are few null keys available at the nodes. However, the corruption converges to a stable percentage. The curve descends when nodes are cleaning their buffers with null keys, recently received. When the corruption stabilizes at a low level, the malicious nodes are isolated and their locations can be approximated. Figure 3 shows that the *Null Keys* solution is able to drive the network to a stable state, where the number of corrupted nodes is fixed. This helps in isolating and locating the malicious nodes.




(a) $p = 0.5\%$ (b) $p = 0.7\%$

Fig. 3. Corruption variation in a network consisting of 1000 nodes. $N_m$ refers to the percentage of malicious nodes.

In Section V-C, we showed that the percentage of corrupted nodes decreases as the network grows. We also indicated that for large populations, the percentage of corrupted nodes stabilizes when the percentage of malicious nodes is fixed. Figure 4 shows how the pollution spread varies with the network size. The results are averaged over 30 runs.

As indicated in Section V, the corruption depends on two factors, the number of malicious nodes and the topology which defines the paths shared by the nodes. Theorem 2 can explain the fast decrease in corruption when the network size is less than 400 nodes. In fact, when the population grows, the probability that malicious nodes share parents with others decreases. However, for large networks, this probability becomes minimal and dominated by the first factor. Hence, the corruption depends on the number of malicious nodes that can limit the effectiveness of the null keys at neighbor nodes. In Figure 4, this number grows linearly with the network size. This explains the slow decrease followed by a stable level of corruption percentage, when the population exceeds 500.

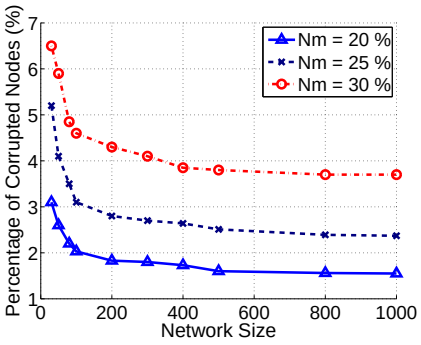



Fig. 4. Corruption as a function of network size. $N_m$ refers to the number of malicious nodes.

### B. Performance Comparison

As shown in Section V-D, for the *Null Keys* algorithm, the computation cost of the verification process at node $v_i$, is $O(c_i(r+q))$. On the other hand, homomorphic hashing is slower than the conventional hash function *SHA1*, as indicated in [7]. The *SHA1* algorithm performs around 1740 basic arithmetic operations on the message block elements, which imposes a large computation cost. In addition, Koetter *et al.* show in [7] that the expected per-block cost for a hash verification is $(m\lambda_q/2+d)\text{MultCost}(p)$, where $m$ is the block size, $\lambda_q$ is a large random prime security parameter and $\text{MultCost}(p)$ is the cost of multiplication in $\mathbb{Z}_p$. Our approach is much simpler, but homomorphic hashing guarantees a higher level of security since it is collision-free. However, because homomorphic hashes are computationally expensive, they require that nodes check blocks probabilistically, as in cooperative security [5].

Figure 5 shows the comparison between *Null Keys* and cooperative security proposed in [5]. In their model, Gkantsidis *et al.* use homomorphic hashing to check for malicious blocks. The nodes perform probabilistic verifications in order to decrease the computation cost imposed by homomorphic hashes. In order not to weaken the verification process, they proposed that nodes cooperate in checking for malicious blocks. Whenever a node detects the presence of bogus blocks, it sends alert messages to neighbor nodes. In order to compare both security approaches, we implemented the cooperative security scheme. We assume, as in [5], instantaneous propagation of alert messages. A node checks blocks with probability $p_c$. Once a malicious block is detected, all the infected nodes are informed. A node sends alert messages to the upstream nodes that sent unsecured blocks, and downstream nodes that received unsecured blocks. In our model, the graph is directed, however we assume that alert messages can propagate to upstream nodes.

Figure 5 shows that the *Null Keys* scheme succeeds in improving the protection against malicious attacks. On the other hand, as we increase the checking probability in cooperative security, the corruption decreases but the network performance

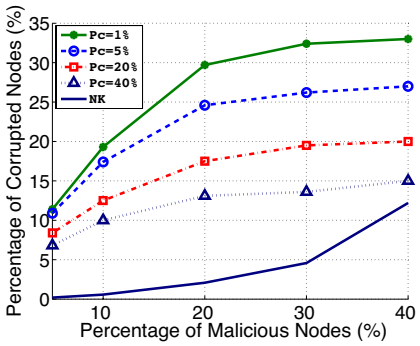



Fig. 5. Pollution in a Network of 1000 nodes with $p = 0.5\%$. NK refers to *Null Keys* and $p_c$ refers to the probabilistic checking of cooperative security.

degrades due to higher computational complexity. We note that a checking probability of $40\%$ imposes a significant computation overhead. When the percentage of malicious nodes is over $20\%$, the slow increase in the corruption can be explained by the overlap in the affected regions. In fact, when the polluted regions overlap, the same block can be corrupted multiple times. Hence, all overlapping attacks can be discovered when such a block is detected. Even with a checking probability $p_c = 40\%$, the *Null Keys* scheme performs better. On the other hand, we assumed that when a node receives an alert message, it does not count as a corrupted node. In fact, in their model [5], a node stops using unsecured blocks when an alert message is received. This is another drawback that decreases the network performance, since non-corrupted blocks can be part of these unsecured blocks. The cleaning process would be slow since homomorphic hashing is used.

In addition, the edge connection probability does not change the results in the case of cooperative security, as claimed in [5]. However, if this probability is increased, our security approach can perform better as shown in Figure 2(a). In the comparison, the edge connection probability is set to $0.5\%$, which is the worst case in Figure 2(a). Also, in the case of cooperative security, the corruption is dynamic. In contrast with our approach, the corrupted nodes always vary. Depending on which nodes check their blocks and how the alert messages propagate, the corruption of the nodes changes. Hence, the locations of malicious nodes cannot be approximated.

## VII. Conclusion

In this paper, we have presented the *Null Keys* algorithm, a new security scheme for network coding that does not require large computations, nor add any redundancy to the original blocks. We verify the integrity of a block by checking if it belongs to the original subspace formed at the source. The verification is possible when each node receives vectors from the null space of the original blocks that map legitimate blocks to zero. These vectors, referred to as null keys, go through a random combination when distributed. The malicious nodes have access to some of these keys, however path diversity and distributed randomness hide the null keys content at neighbors. As the connections between the nodes increase, the percentage of corrupted nodes decreases. The null keys can be signed using homomorphic hashing since the number of signed blocks and their verifications do not degrade the network performance. In fact, the number of null keys injected is equal to the out-degree of the source. Our simulations show that *Null Keys* effectively limits the pollution and isolates malicious nodes. The stable corruption spread, helps in identifying the locations of the malicious nodes. Moreover, the *Null Keys* algorithm guarantees better protection than cooperative security that uses homomorphic hashing in a probabilistic fashion.

## References

[1] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[2] R. Koetter and M. Medard, "An Algebraic Approach to Network Coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, 2003.

[3] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," in *Proc. of IEEE International Symposium on Information Theory (ISIT 2003)*, 2003.

[4] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. of IEEE INFOCOM 2005*, 2005.

[5] ——, "Cooperative Security for Network Coding File Distribution," in *Proc. of IEEE INFOCOM 2006*, April 2006, pp. 1–13.

[6] N. Cai and R. W. Yeung, "Network Coding and Error Correction," in *Proc. of IEEE Information Theory Workshop (ITW 2002)*, 2002, pp. 119–122.

[7] M. N. Krohn, M. J. 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*, 2004, pp. 226–240.

[8] Z. Zhang, "Network Error Correction Coding in Packetized Networks," in *Proc. of IEEE Information Theory Workshop (ITW 2006)*, October 2006, pp. 433–437.

[9] S. Jaggi, M. Langberg, T. Ho, and M. Effros, "Correction of Adversarial Errors in Networks," in *Proc. of International Symposium on Information Theory (ISIT 2005)*, 2005, pp. 1455–1459.

[10] L. Nutman and M. Lamberg, "Adversarial Models and Resilient Schemes for Network Coding," in *Proc. of IEEE International Symposium on Information Theory (ISIT 2008)*, July 2008, pp. 171–175.

[11] R. Koetter and F. R. Kschischang, "Coding for Errors and Erasures in Random Network Coding," in *Proc. of IEEE International Symposium on Information Theory (ISIT 2007)*, June 2007.

[12] Q. Li, D. Chiu, and J. C. S. Lui, "On the Practical and Security Issues of Batch Content Distribution Via Network Coding," in *Proc. of IEEE International Conference on Network Protocols (ICNP 2006)*, 2006, pp. 158–167.

[13] D. Charles, K. Jain, and K. Lauter, "Signatures for Network Coding," in *Proc. of 40th Annual Conference on the Information Sciences and Systems (CISS 2006)*, March 2006, pp. 857–863.

[14] M. Jafarisiavoshani, C. Fragouli, and D. Suhas, "Subspace Properties of Randomized Network Coding," in *Proc. of IEEE Information Theory Workshop on Information Theory for Wireless Networks (ITW 2007)*, July 2007, pp. 1–5.