

# Cooperative Defense against Pollution Attacks in Network Coding Using SpaceMac

Anh Le, *IEEE Student Member*, and Athina Markopoulou, *IEEE Member*

**Abstract**—Intra-session network coding is inherently vulnerable to pollution attacks. In this paper, first, we introduce a novel homomorphic MAC scheme called SpaceMac, which allows an intermediate node to verify whether received packets belong to a specific subspace, even if the subspace is expanding over time. Then, we use SpaceMac as a building block to design a cooperative scheme that provides complete defense against pollution attacks: (i) it can detect polluted packets early at intermediate nodes, and (ii) it can identify the exact location of all, even colluding, attackers, thus making it possible to eliminate them. Our scheme is cooperative: parents and children of any node cooperate to detect any corrupted packets sent by the node, and nodes in the network cooperate with a central controller to identify the exact location of all attackers. We implement SpaceMac in both C/C++ and Java as a library, which we make publicly available. Our evaluation on both a PC and an Android device shows that the SpaceMac algorithms can be computed quickly and efficiently and that our cooperative defense scheme has low computation overhead and significantly lower communication overhead than those of state-of-the-art schemes.

**Index Terms**—Communication networks, network coding, cooperative systems, communication system security, cryptographic protocols, message authentication, pollution attacks.

## I. INTRODUCTION

THE network coding paradigm advocates that intermediate nodes in a network should mix incoming packets instead of simply forwarding them, and receivers should decode to obtain the original packets. This idea, originally introduced by Ahlswede *et al.* [1], has been shown to bring benefits in terms of throughput and distributed operation of networks, and has received much attention. In this work, we consider networks that employ intra-session linear network coding.

An inherent weakness of network coding is that it is particularly vulnerable to pollution (a.k.a. Byzantine) attacks. Malicious nodes can inject corrupted packets into a network, which get combined and forwarded by downstream nodes, thus causing a large number of corrupted packets propagating in the network. This wastes resources and eventually prevents the decoding of the original packets at the receivers. The detrimental effect of pollution attacks has been shown through both theoretical analysis [2] as well as experimentation [3], [4].

Proposed defense mechanisms against pollution attacks can be classified into three categories: error correction [5]–[7], attack detection [4], [8]–[13], and locating attackers [14], [15].

A. Le (anh.le@uci.edu) is with the Computer Science Department and A. Markopoulou (athina@uci.edu) is with the Electrical Engineering and Computer Science Department, University of California, Irvine, CA, USA.

This work was supported by the NSF CAREER award (0747110) and by an AFOSR MURI (FA9550-09-1-0643).

Manuscript received on Feb. 1st, 2011; last revised on Sept. 20th, 2011.

In this paper, we are interested in the latter two approaches. In particular, we set out to design a complete defense system that can not only detect polluted packets in a timely manner but can also accurately locate and eliminate all, even colluding, attackers. This allows for dealing with an attack early and at its root. To the best of our knowledge, none of the existing defense mechanisms can provide this level of protection.

To this end, we first propose a novel homomorphic message authentication code (MAC) scheme for expanding spaces, called SpaceMac. SpaceMac allows a node to verify if its received packets belong to a specific subspace, even if the subspace is expanding over time. Then, we design a novel cooperative defense system which includes both a detection scheme and a locating scheme, using SpaceMac as their building block. Our detection scheme relies on SpaceMac to force intermediate nodes to send only linear combinations of packets that they actually receive from their parents. Parents and children of any intermediate node cooperate to detect corrupted packets sent by the intermediate node. Our locating scheme uses SpaceMac to force nodes in the network to truthfully cooperate with a central controller so that the controller can exactly locate the pollution attackers. Finally, by leveraging multiple generations, our scheme is able to deal with an arbitrary number of colluding attackers.

The main contributions of this paper are the following:

- *The design and implementation of SpaceMac*: We describe the construction of SpaceMac and provide a formal security proof for the construction. We implement SpaceMac in both C/C++ and Java as a ready-to-use library, which we make available online [16]. Our Java implementation is compatible with the current Android OS (Android 2.2 Froyo).
- *The design of a novel cooperative defense system based on SpaceMac*: To the best of our knowledge, our defense system is the first that meets all of the following requirements: (i) it can provide timely in-network detection, (ii) it can exactly locate all pollution attackers, (iii) it can deal with an arbitrary number of colluding attackers, and (iv) it has low communication and computation overhead.

We have extensively evaluated the computation overhead of SpaceMac's algorithms and the computation and communication overhead of our defense system, through implementation in both C/C++ and Java, and on both a PC and an Android device (Samsung Captivate). Our evaluation results show that for a practical parameter setting (detailed in Section VIII), all three algorithms of SpaceMac (Mac, Combine, and Verify) can be computed efficiently (requiring 64 KB of memory in

C/C++ or 128 KB in Java) and also quickly on a PC ( $< 28 \mu\text{s}$  in C/C++) and even on a smartphone ( $< 2.3 \text{ ms}$ ). Evaluation results also demonstrate that for the same practical parameter setting, when implementing our defense system, nodes in the network introduce very small computational delay per packet (in the order of sub-millisecond on the PC and millisecond on the smart phone). Moreover, our defense system introduces very low communication overhead per packet (2%), significantly less than those of state-of-the-art schemes.

The rest of the paper is organized as follows. Section II discusses existing approaches to protect network coding against pollution attacks. Section III formulates the problem, describes the threat model, and discusses our design goals. Section IV presents our key observations and the overview of our approach. Section V presents the construction of SpaceMac and the formal security proof. Section VI describes our detection scheme. Section VII describes our locating scheme. Section VIII presents our implementation and the evaluation results. Finally, Section IX concludes the paper.

## II. RELATED WORK

For a comprehensive discussion of related work, please see [17]. Here, we review only the most closely related work, and we put our work in perspective.

### A. Attack Detection

Two homomorphic MAC schemes have been proposed by two groups of researchers: Agrawal and Boneh [9], and Li *et al.* [10]. The scheme in [9], HomMac, relies on cover free set systems for pre-distributing keys to provide in-network detection, and thus is only  $c$ -collusion resistant. Furthermore, it is susceptible to tag-pollution attacks, where malicious nodes tamper with some subset of tags of a packet. The scheme in [10] is collusion resistant as well as resistant against tag-pollution attacks; however, it requires time synchronization among nodes in the network. Both schemes have low computation overhead since they only require simple addition and multiplication operations at intermediate nodes for both combining MAC tags and tag verification.

Our SpaceMac scheme is inspired by and generalizes HomMac in [9], with the difference that SpaceMac allows intermediate nodes to sign subspaces that *expand* over time, while HomMac allows to sign only fixed subspaces. A detailed description and comparison are provided in Section V. In our preliminary work [18], we proposed the first construction of SpaceMac and we used it to locate attackers. In this paper, (i) we provide a much more efficient construction of SpaceMac, and (ii) we show that the ability to authenticate expanding subspaces can also be utilized to provide in-network detection.

### B. Locating Attackers

Early work by Jafarisiavoshani *et al.* [14] leveraged the subspace properties of randomized network coding to locate pollution attackers. The main observation was that packets sent by a node have to belong to the space spanned by source packets and also the space spanned by the packets the node

receives from its parents. Using this observation, in a general network topology having a single attacker, the authors can locate the attacker with an uncertainty of at most two nodes; when there are multiple attackers, the uncertainty is within a set of nodes including the attackers and their parents and children. Our scheme builds on and significantly improves this work: we make it possible to pinpoint the exact location of the attackers, even in the case where there are multiple colluding attackers, thereby allowing for the removal of all attackers.

Wang *et al.* [15] introduced a light-weight non-repudiation protocol ensuring that (i) a malicious node that injected a polluted packet cannot deny its behavior and (ii) a malicious node cannot disparage any innocent node. They built a defense scheme based on the protocol to identify malicious nodes. The scheme in [15] performs flooding of multiple checksums of all the packets sent by the source to all the nodes, which incurs significant communication overhead. Moreover, because the success of the locating scheme relies on the successful reception of the checksums at every node, this scheme is vulnerable to colluding attackers. The scheme is also unable to locate all the attackers. We use the non-repudiation protocol of [15] as a building block in our locating scheme. However, unlike the scheme in [15], our scheme is able to locate all, even colluding attackers, without the need of checksums.

## III. PROBLEM FORMULATION

### A. Network Model and Operation

We follow the notation used in [9] and [14]. Consider a fixed, directed acyclic graph (DAG), denoted by  $G = (\mathcal{V}, \mathcal{E})$ . There is a single source node  $S$  that multicasts packets to a set of receivers, denoted by  $\mathcal{R}$ . Denote the set of intermediate nodes as  $\mathcal{I}$ , *i.e.*,  $\mathcal{I} = \mathcal{V} \setminus \{\mathcal{R} \cup \{S\}\}$ . Nodes in  $\mathcal{I}$  perform generation-based linear network coding. A generation consists of  $m$  packets,  $\hat{\mathbf{v}}_1, \dots, \hat{\mathbf{v}}_m$ , in an  $n$ -dimensional linear space  $\mathbb{F}_q^n$ , where  $m, n$  and  $q$  are fixed ahead of time and  $q \gg 1$ . The source augments every packet  $\hat{\mathbf{v}}_i$  with  $m$  additional symbols, which are the coefficients of  $\hat{\mathbf{v}}_i$ . The resulting packets,  $\mathbf{v}_i$ 's, called *source packets*, have the following form:

$$\mathbf{v}_i = (-\hat{\mathbf{v}}_i, \underbrace{0, \dots, 0, 1, 0, \dots, 0}_i) \in \mathbb{F}_q^{n+m}.$$

If a packet  $\mathbf{y}$  is a linear combination of the packets  $\mathbf{v}_i$ 's, then the last  $m$  symbols of packet  $\mathbf{y}$  contain the global linear combination coefficients. The source  $S$  sends  $\mathbf{v}_i$ 's to the network. Denote the subspace spanned by  $\mathbf{v}_i$ 's by  $\Pi^S \subseteq \mathbb{F}_q^{n+m}$ . We refer to  $\Pi^S$  as the *source space*.

We use  $\mathcal{P}_N$  and  $\mathcal{C}_N$  to denote the sets of parents and children of a node  $N$ , respectively. Each intermediate node  $N$  receives from  $\mathcal{P}_N$  some packets, which are linear combinations of the source packets. It then creates linear combinations of the received packets and sends them to its adjacent downstream nodes. We use  $\Pi_N^P(t) \subseteq \mathbb{F}_q^{n+m}$  to denote the space spanned by the packets received by node  $N$  from a parent node  $P$ ,  $P \in \mathcal{P}_N$ , up to time  $t$ . We further use  $\Pi_N(t)$  to denote the space spanned by all the packets received by node  $N$  from all its parents up to time  $t$ . We also denote the space spanned by

all the packets sent by a node  $N$  up to time  $t$  as  $\Pi^N(t)$ . When there is no ambiguity, we omit the time index  $t$ . A receiver  $R \in \mathcal{R}$  can successfully decode the original source packets using Gaussian elimination if its received space  $\Pi_R$  equals the source space  $\Pi^S$ .

If all the nodes in the network are benign, the space spanned by packets sent by  $N$ ,  $\Pi^N$ , must be a subspace of the space spanned by the packets that  $N$  receives,  $\Pi_N$ . This is a property of networks that implement random linear network coding. This observation was also made in [14]. Formally,

**Lemma 1.** *If every node in the network is benign then for every node  $N$ ,  $\Pi^N(t) \subseteq \Pi_N(t)$ .*

Also, observe that for any parent  $P$  of  $N$ , both  $\Pi_N^P$  and  $\Pi_N$  expand over time. Formally,  $\Pi_N^P(t_0) \subseteq \Pi_N^P(t_1)$  and  $\Pi_N(t_0) \subseteq \Pi_N(t_1)$ , for all  $t_0 \leq t_1$

Furthermore, when all the intermediate nodes  $N \in \mathcal{I}$  are benign, the incoming spaces of all the intermediate nodes and the receivers are subspaces of the source space. Assume that there is a pollution attacker in the network. The attacker combines a subspace  $\Pi^* \not\subseteq \Pi^S$  with its incoming space and sends the resulting packets to its children. As a result, the incoming spaces of these children are not subspaces of  $\Pi^S$ . Formally, for every node  $N, N \in \mathcal{V} \setminus \{S\}$ , its incoming space  $\Pi_N^P$  from its parent  $P$  can be decomposed as follows:  $\Pi_N^P = \Pi_N^{S_P} \oplus \hat{\Pi}_N^P$ , where  $\oplus$  denotes the direct sum of spaces,  $\Pi_N^{S_P} \stackrel{\text{def}}{=} \Pi^S \cap \Pi_N^P$ , and  $\hat{\Pi}_N^P$  contains packets not in  $\Pi^S$ .

Following the framework of [14], we define a *polluted directed edge* as follows:

**Definition 1.** *A directed edge is polluted if it transmits any packet which is not a linear combination of the source packets.*

The following lemma, adapted from [14], is a direct consequence of the definition:

**Lemma 2.** *A directed edge  $e(P, N)$  is polluted iff  $\Pi_N^P \not\subseteq \Pi^S$ .*

## B. Threat Model

We assume that both the source and the receivers are trustworthy but the intermediate nodes may be malicious. We assume that there may be multiple pollution attackers, located at an arbitrary set of intermediate nodes in the network. Each attacker may inject corrupted packets into a single or multiple downstream edges to pollute the network. They may also modify other data associated with the packets, such as, authentication tags. We consider both cases where the pollution attackers launch their attacks independently or collude and coordinate their attacks. We further assume that the attackers are aware of our defense scheme, *i.e.*, the construction and application of SpaceMac. However, similar to other cryptographic approaches, we assume that the attackers' running time is polynomial in the security parameter.

## C. Design Goals

With this threat model in mind, we set out to design a defense system with the following capabilities and properties:

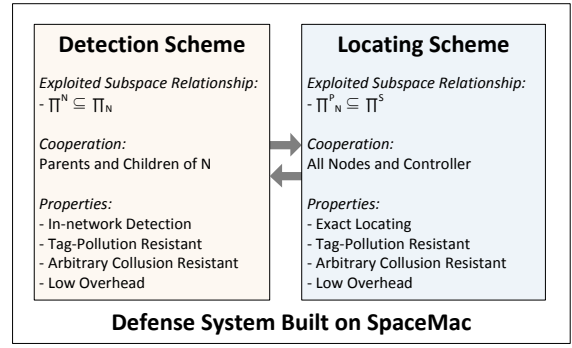


Fig. 1. Overview of the main components of our cooperative defense system and their properties. The detection scheme is active all the time, and the locating scheme is only executed when an attack is detected.

- (1) *In-network Detection.* Any intermediate node in the network should be able to detect the attack as soon as its malicious parent sends it a corrupted packet. This prevents corrupted packets from polluting the downstream edges.
- (2) *Exact Locating.* The location of all pollution attackers should be precisely identified. This allows for the removal of the attackers from the network.
- (3) *Arbitrary Collusion Resistance.* The defense system should be able to detect attacks and remove the attackers from the network even when they collude.
- (4) *Low Overhead.* The system should require a small amount of computing from intermediate nodes and should not introduce a large amount of traffic, *e.g.*, bandwidth of the MAC tags, to the network.

To achieve the above goals, we design a defense system which consists of two main components: the detection scheme and the locating scheme. The detection scheme provides in-network detection while the locating scheme provides exact locating. Both the detection scheme and locating scheme impose low computation as well as low communication overhead. The defense system as a whole is arbitrarily collusion resistant. Fig. 1 illustrates the overall structure of our defense system.

## IV. KEY OBSERVATIONS AND APPROACH OVERVIEW

### A. In-Network Detection

Previous work that use homomorphic MACs to detect corrupted packets, such as, the work by Agrawal and Boneh [9] and Li *et al.* [10], leverage the observation that if a packet does not belong to the source space, then it is a corrupted packet. The detection works by first establishing shared secret MAC keys between the source and the intermediate nodes. Then, using these secret keys, the source node can sign the fixed source space and the intermediate nodes can verify if their received packets belong to the source space.

Our detection scheme leverages a different observation: A packet sent by an intermediate node must belong to the space spanned by all packets that it received from its parents (Lemma 1). For example, consider a subset of nodes in a network in Fig. 2. A packet sent by  $C$  must belong to the space spanned by the packets it received from its parents:  $A$  and  $B$ ; otherwise,  $C$  must be polluting the network. Formally, at any moment  $t$  in the multicast session, if an intermediate node  $N$  sends out a vector  $\mathbf{y}$  then  $\mathbf{y} \in \Pi_N(t)$ ; otherwise,  $\mathbf{y}$  is corrupted.

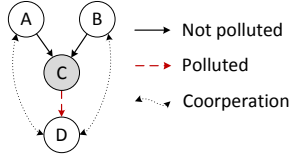


Fig. 2. An example illustrates how SpaceMac helps to detect pollution attacks. Using SpaceMac,  $A$  and  $B$  are able to sign the expanding space  $\Pi_C$  (the received space of  $C$ ) and  $D$  is able to verify any packet sent by  $C$  to see if it belongs to  $\Pi_C$ . If there is a packet sent by  $C$  that is not in  $\Pi_C$ , the attack is detected by  $D$ . The cooperation among  $A$ ,  $B$ , and  $D$  helps to detect the attack from  $C$ .

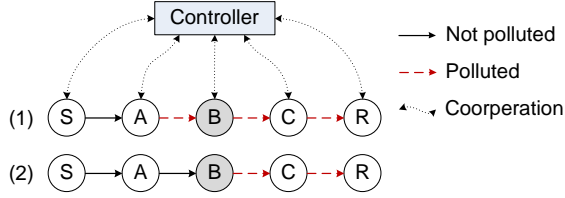


Fig. 3. An example of inferring an attacker's location using information about polluted edges from [14]: The attacker is at node  $B$ . Scenarios (1) and (2) correspond to the sets of polluted edges when the attacker lies and is honest about its incoming subspace, respectively. The controller can narrow down the attacker to two candidate nodes:  $A$  and  $B$ , which have outgoing polluted edges. The cooperation among nodes in the network and the controller helps the locating process.

We use SpaceMac to enable the parents of  $N$  to sign the expanding space  $\Pi_N$  and the children of  $N$  to verify any packet that  $N$  sends to see if it belongs to  $\Pi_N$ . The cooperation of the parents of  $N$  and the children of  $N$  enables the children to detect any corrupted packet sent by  $N$  immediately. For example, in Fig. 2, at any time  $t$ ,  $D$  is able to check if any packet it receives from  $C$  belongs to  $\Pi_C(t)$ ; hence,  $D$  is able to detect any pollution attempt by  $C$  as soon as  $D$  receives a corrupted packet from  $C$ .

### B. Exact Locating

Leveraging the cooperation between nodes in the network and a central controller, Jafarisiavoshani *et al.* [14] have shown that when there is a single pollution attacker, its location can be narrowed down to a set of at most two nodes. This is done by analyzing the polluted edges identified based on the incoming subspaces reported by all nodes to the central controller. An example is shown in Fig. 3. When there are multiple attackers in a general network topology, the number of suspected nodes increases to include the attackers and their parents and children.

Our key observation here is that the uncertainty about the location of the attackers originates from the fact that the attackers can lie about their received spaces. Therefore, by ensuring that all nodes in the network cannot lie about their received spaces, we can exactly locate the attackers. For instance, if the attacker cannot lie in the example of Fig. 3, then the only possible scenario is scenario (2); hence, one can infer that the attacker is located at node  $B$ . We explicitly design and use SpaceMac to achieve this goal, *i.e.*, prevent nodes from lying. To prevent attacker  $B$  from lying,  $A$  cooperates with the controller by signing the space spanned by the packets it sends to  $B$  using SpaceMac. This is so that when  $B$  reports

a fake space, it will not have the proper signature of the fake space to convince the controller.

## V. THE CONSTRUCTION OF SPACEMAC

In this section, we describe the construction of SpaceMac. This construction is new and significantly more efficient than our previous construction presented in [18]. This construction is inspired by and generalizes the HomMac scheme by Agrawal and Boneh [9]. The key difference between SpaceMac and HomMac is the security property that SpaceMac brings: SpaceMac *allows for signing spaces that expand over time, while HomMac only allows for signing fixed spaces*. This is directly reflected in the difference between the security games of SpaceMac and HomMac, and consequently in the difference between the constructions of the two schemes.

### A. Definitions

A  $(q, n, m)$  homomorphic MAC scheme is defined by three probabilistic, polynomial-time algorithms: Mac, Combine, and Verify. The Mac algorithm generates a tag for a given vector; the Combine algorithm computes a tag for a linear combination of some given vectors; and the Verify algorithm verifies whether a tag is a valid tag of a given vector.

- $\text{Mac}(k, \text{id}, \mathbf{y})$ :
  - Input: a secret key  $k$ , the identifier  $\text{id}$  of the source space  $\Pi^S$ , and a vector  $\mathbf{y} \in \mathbb{F}_q^{n+m}$ .
  - Output: tag  $t$  for  $\mathbf{y}$ .
- $\text{Combine}((\mathbf{y}_1, t_1, \alpha_1), \dots, (\mathbf{y}_p, t_p, \alpha_p))$ :
  - Input:  $p$  vectors  $\mathbf{y}_1, \dots, \mathbf{y}_p$ , their tags  $t_1, \dots, t_p$  under key  $k$ , and their coefficients  $\alpha_1, \dots, \alpha_p \in \mathbb{F}_q$ .
  - Output: tag  $t$  for vector  $\mathbf{y} \stackrel{\text{def}}{=} \sum_{i=1}^p \alpha_i \mathbf{y}_i$ .
- $\text{Verify}(k, \text{id}, \mathbf{y}, t)$ :
  - Input: a secret key  $k$ , the identifier  $\text{id}$  of the source space  $\Pi^S$ , a vector  $\mathbf{y} \in \mathbb{F}_q^{n+m}$ , and its tag  $t$
  - Output: 0 (reject) or 1 (accept)

Also, the scheme must satisfy the following correctness requirement: Let  $t = \text{Combine}((\mathbf{y}_1, t_1, \alpha_1), \dots, (\mathbf{y}_p, t_p, \alpha_p))$ , then  $\text{Verify}(k, \text{id}, \sum_{i=1}^p \alpha_i \mathbf{y}_i, t) = 1$ .

### B. Attack Game

We consider the following attack game for a homomorphic MAC  $\mathcal{T} = (\text{Mac}, \text{Combine}, \text{Verify})$ , a challenger  $\mathcal{C}$ , and an adversary  $\mathcal{A}$ :

- *Setup*.  $\mathcal{C}$  generates a random key  $k \stackrel{\mathcal{R}}{\leftarrow} \mathcal{K}$
- *Queries*.  $\mathcal{A}$  adaptively queries  $\mathcal{C}$ , where each query is of the form  $(\text{id}, \mathbf{y})$ . For each query,  $\mathcal{C}$  replies to  $\mathcal{A}$  with the corresponding tag  $t \leftarrow \text{Mac}(k, \text{id}, \mathbf{y})$ .
- *Output*.  $\mathcal{A}$  eventually outputs a tuple  $(\text{id}_*, \mathbf{y}_*, t_*)$ .

Up to the time  $\mathcal{A}$  outputs, it has queried  $\mathcal{C}$  multiple times. Let  $l$  denote the number of times  $\mathcal{A}$  queried  $\mathcal{C}$  using  $\text{id}_*$  and get tags for  $l$  vectors,  $\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l}$ , of these queries. Let  $\mathbf{y}_* = (y_*^{(1)}, \dots, y_*^{(n+m)})$ . We consider that the adversary wins the security game if

- (i)  $(y_*^{(n+1)}, \dots, y_*^{(n+m)}) \neq \mathbf{0}$  (trivial forge otherwise),

- (ii)  $\text{Verify}(k, \text{id}_*, \mathbf{y}_*, t_*) = 1$ , and
- (iii)  $\mathbf{y}_* \notin \text{span}(\mathbf{y}_{*1}, \dots, \mathbf{y}_{*l})$ .

Let  $\text{Adv}[\mathcal{A}, \mathcal{T}]$  denote the probability that  $\mathcal{A}$  wins the above attack game. We define a secure homomorphic MAC scheme as follows:

**Definition 2.** A  $(q, n, m)$  homomorphic MAC scheme  $\mathcal{T}$  is secure if and only if for all probabilistic polynomial-time adversaries  $\mathcal{A}$ ,  $\text{Adv}[\mathcal{A}, \mathcal{T}]$  is negligible.

### C. Construction

Let  $\mathcal{K}$  and  $\mathcal{D}$  denote the domains of the keys and the id's of the spaces sent by the source, respectively. Let  $[n]$  denote  $\{1, \dots, n\}$ . We use a pseudorandom generator (PRG)  $G: \mathcal{K}_G \rightarrow \mathbb{F}_q^{n+m}$  and a pseudorandom function (PRF)  $F: \mathcal{K}_F \times \mathcal{D} \times [m] \rightarrow \mathbb{F}_q$ . A key  $k$  for this construction consists of a pair  $(k_1, k_2)$ , where  $k_1 \in \mathcal{K}_G$  and  $k_2 \in \mathcal{K}_F$ .

- $\text{Mac}(k, \text{id}, \mathbf{y})$ : A tag for a vector  $\mathbf{y} = (y^{(1)}, \dots, y^{(n+m)})$  using key  $k = (k_1, k_2)$  is generated as follows:
  - $\mathbf{r} \leftarrow G(k_1) \in \mathbb{F}_q^{n+m}$
  - $b \leftarrow \sum_{j=1}^m [y^{(n+j)} \cdot F(k_2, \text{id}, j)] \in \mathbb{F}_q$
  - $t \leftarrow (\mathbf{r} \cdot \mathbf{y}) + b \in \mathbb{F}_q$
- $\text{Combine}((\mathbf{y}_1, t_1, \alpha_1), \dots, (\mathbf{y}_p, t_p, \alpha_p))$ : The tag  $t$  of  $\mathbf{y}$  is computed as follows:
  - $t \leftarrow \sum_{i=1}^p \alpha_i t_i \in \mathbb{F}_q$
- $\text{Verify}(k, \text{id}, \mathbf{y}, t)$ : To verify if  $t$  is the valid tag of  $\mathbf{y}$  using key  $k = (k_1, k_2)$ , we proceed as follows:
  - $\mathbf{r} \leftarrow G(k_1) \in \mathbb{F}_q^{n+m}$
  - $b \leftarrow \sum_{j=1}^m [y^{(n+j)} \cdot F(k_2, \text{id}, j)] \in \mathbb{F}_q$
  - $a \leftarrow \mathbf{r} \cdot \mathbf{y} \in \mathbb{F}_q$
  - If  $a + b = t$  output 1; otherwise, output 0

The correctness of the scheme is proved as follows. Suppose  $\mathbf{y} = (y^{(1)}, \dots, y^{(n+m)}) = \sum_{i=1}^p \alpha_i \mathbf{y}_i$ . Then,

$$\begin{aligned} a + b &= \mathbf{r} \cdot \sum_{i=1}^p \alpha_i \mathbf{y}_i + \sum_{j=1}^m \left[ \left( \sum_{i=1}^p \alpha_i y_i^{(n+j)} \right) \cdot F(k_2, \text{id}, j) \right] \\ &= \sum_{i=1}^p \alpha_i \left[ (\mathbf{r} \cdot \mathbf{y}_i) + \sum_{j=1}^m \left( y_i^{(n+j)} \cdot F(k_2, \text{id}, j) \right) \right] = \sum_{i=1}^p \alpha_i t_i. \end{aligned}$$

Compared to HomMac [9], our construction replaces the Sign algorithm, which generates tags for basis vectors of a fixed space, with the Mac algorithm, which generates a tag for any vector in  $\mathbb{F}_q^{n+m}$ . SpaceMac generalizes HomMac because the Mac algorithm can substitute the Sign algorithm, *i.e.*, it can generate valid, combinable tags for all basis vectors.

### D. Security

The security of SpaceMac can be proved by assuming  $F$  is a secure PRF and  $G$  is a secure PRG. Let  $\mathcal{B}_1$  and  $\mathcal{B}_2$  denote a PRF adversary and a PRG adversary, respectively. Let  $\text{PRF-Adv}[\mathcal{B}_1, F]$  and  $\text{PRG-Adv}[\mathcal{B}_2, G]$  denote the advantages in winning the PRF and PRG security games [19], respectively.

**Theorem 3.** For any fixed  $q, n, m$ , SpaceMac is a secure  $(q, n, m)$  homomorphic MAC assuming  $F$  is a secure PRF

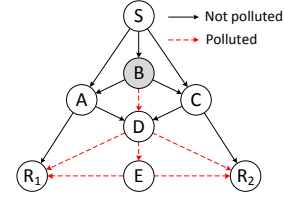


Fig. 4. A network consisting of 8 nodes (resembling a network given in [14]).  $B$  is the attacker. After every node (except for  $S$ ) reports to the controller its true incoming spaces,  $B$  is identified as the attacker since it has no incoming polluted edge but has one outgoing polluted edge.

and  $G$  is a secure PRG. In particular, for every homomorphic MAC adversary  $\mathcal{A}$ , there is a PRF adversary  $\mathcal{B}_1$  and a PRG adversary  $\mathcal{B}_2$ , who have similar running time to  $\mathcal{A}$ , such that

$$\text{Adv}[\mathcal{A}, \text{SpaceMac}] \leq \text{PRF-Adv}[\mathcal{B}_1, F] + \text{PRG-Adv}[\mathcal{B}_2, G] + \frac{1}{q}.$$

Due to limited space, we refer the reader to [17] for the proof of the theorem. The theorem states that an adversary  $\mathcal{A}$  can break the scheme with probability  $\frac{1}{q}$ . For a small field size, *e.g.*,  $q = 2^8$ , the security of the MAC scheme may be unsatisfactory. To improve the security, one could either increase the field size or use multiple tags as suggested in [9] and [10]. The security of our scheme using  $l$  tags is  $(\frac{1}{q})^l$ .

## VI. DETECTION SCHEME

For ease of presentation, we describe the detection scheme within the scope of a single generation, *i.e.*, considering a single source space id.

**1) Assumptions.** We assume that there is a controller who knows the complete topology of the graph. The controller could be the source itself. We further assume that each node  $N$  shares with the controller a pair of secret keys  $(k_N^1, k_N^2)$ . These keys can be established with a public key infrastructure.

**2) Bootstrapping.** First, for every intermediate node  $N$ , the controller determines the key  $k_N$ , which is secret to  $N$  itself and is used by the parents and children of  $N$  when using SpaceMac. Each node can serve as either a parent or a child. Hence, each node, depending on its position in the network, must know a different set of keys to participate in the detection scheme. For example, consider the network in Fig. 4, node  $D$  needs to know  $k_A, k_B$ , and  $k_C$  to detect corrupted packets sent by  $A, B$ , and  $C$ , respectively. It also needs to know  $k_E$  to help  $R_1$  and  $R_2$  to detect corrupted packets sent by  $E$ .

Second, the source and all the receivers need to share an end-to-end key,  $k^*$ . This key is used to ensure detection in the presence of colluding adversaries, in which case a node  $N$  colludes with its parent to obtain  $k_N$  and thus can bypass the verification of its children.

The controller then sends to each node  $N$  a bootstrapping packet consisting of the set of keys that are necessary for it to participate in the detection scheme. In particular, the controller constructs the bootstrapping packet  $b_N: b_N = \{k_{\bar{X}}, X \in \{\mathcal{P}_N \cup \mathcal{C}_N\}, k^*\}$ . Note that  $b_N$  contains  $k^*$  if and only if  $N$  is either the source or a receiver. The controller then sends  $b_N$  to  $N$  through a secure and authenticated channel achieved with  $k_N^1$  (for encryption) and  $k_N^2$  (for authentication) using a standard encrypt-then-authenticate algorithm. For example,

node  $D$  in Fig. 4 receives  $\{k_{\bar{A}}, k_{\bar{B}}, k_{\bar{C}}, k_{\bar{E}}\}$  while node  $R_1$  receives  $\{k_{\bar{A}}, k_{\bar{D}}, k_{\bar{E}}, k^*\}$ .

**3) Sending and Coding.** Before sending out each source packet,  $\mathbf{v}_i$ , the source  $S$  calculates an *end-to-end tag*,  $t_{\mathbf{v}_i}^{k^*}$ , using the Mac algorithm of SpaceMac with key  $k^*$ :  $t_{\mathbf{v}_i}^{k^*} = \text{Mac}(\mathbf{v}_i, k^*)$ .  $S$  then attaches this tag to every source packet and sends  $\mathbf{w}_i \stackrel{\text{def}}{=} \{t_{\mathbf{v}_i}^{k^*} \parallel \mathbf{v}_i\}$  instead of  $\mathbf{v}_i$ , where ‘ $\parallel$ ’ denotes concatenation. The packets traversing the network are linear combinations of  $\mathbf{w}_i$ ’s instead of  $\mathbf{v}_i$ ’s. For ease of presentation with regards to the input length of the SpaceMac algorithms, assume that the size of  $\mathbf{w}_i$  is still  $n$ .

Consider a parent  $P$  who wants to send a packet  $\mathbf{y}$  to its child  $N$ .  $P$  needs to calculate a *helper tag* which helps the children of  $N$  to detect corrupted packets sent by  $N$ . In particular, before sending  $\mathbf{y}$  to  $N$ ,  $P$  needs to calculate a MAC tag,  $t_{\mathbf{y}}^{k_{\bar{N}}}$ , using Mac under key  $k_{\bar{N}}$ :  $t_{\mathbf{y}}^{k_{\bar{N}}} = \text{Mac}(\mathbf{y}, k_{\bar{N}})$ . Besides the helper tag,  $P$  must also pass along a *verification tag* of  $\mathbf{y}$ , which is used by  $N$  to verify the integrity of  $\mathbf{y}$ . Assume  $P$  received  $\{\mathbf{y}_1, \dots, \mathbf{y}_l\}$  and their helper tags  $\{t_{\mathbf{y}_1}^{k_{\bar{P}}}, \dots, t_{\mathbf{y}_l}^{k_{\bar{P}}}\}$  from its parents, and  $P$  computes  $\mathbf{y}$  as  $\mathbf{y} = \sum_{i=1}^l \alpha_i \mathbf{y}_i$ . Then, the verification tag,  $t_{\mathbf{y}}^{k_{\bar{P}}}$ , of  $\mathbf{y}$  can be computed using Combine:  $t_{\mathbf{y}}^{k_{\bar{P}}} = \text{Combine}((\mathbf{y}_1, t_{\mathbf{y}_1}^{k_{\bar{P}}}, \alpha_1), \dots, (\mathbf{y}_l, t_{\mathbf{y}_l}^{k_{\bar{P}}}, \alpha_l))$ . The final packet that  $P$  sends to  $N$  includes  $\mathbf{y}$  and its helper and verification tags:  $\{t_{\mathbf{y}}^{k_{\bar{N}}} \parallel t_{\mathbf{y}}^{k_{\bar{P}}} \parallel \mathbf{y}\}$ .

**4) Receiving and Verification.** When a node  $N$  receives from its parent  $P$  a packet  $\{t_{\mathbf{y}}^{k_{\bar{N}}} \parallel t_{\mathbf{y}}^{k_{\bar{P}}} \parallel \mathbf{y}\}$ , it uses  $k_{\bar{P}}$  and the Verify algorithm to verify the integrity of the packet. The packet is deemed non-corrupted if  $\text{Verify}(k_{\bar{P}}, \mathbf{y}, t_{\mathbf{y}}^{k_{\bar{P}}}) = 1$ . Since  $P$  does not know  $k_{\bar{P}}$ , the probability that  $P$  can forge a valid tag,  $t_{\mathbf{y}}^{k_{\bar{P}}}$ , when  $\mathbf{y}$  is outside of its received space,  $\Pi_P$ , is  $\frac{1}{q}$ . As soon as  $N$  receives a corrupted packet from  $P$ , with high probability,  $N$  is able to detect the attack immediately.

If  $N$  is a receiver, it further verifies the end-to-end tag using key  $k^*$ . It first parses  $\mathbf{y}$  as  $\{t_{\mathbf{w}}^{k^*} \parallel \mathbf{w}\}$  and accepts  $\mathbf{w}$  if  $\text{Verify}(k^*, \mathbf{w}, t_{\mathbf{w}}^{k^*}) = 1$ . Since none of the malicious intermediate node knows  $k^*$ , if  $\mathbf{w}$  is outside of the source space, the adversary can only forge a valid tag of  $\mathbf{w}$ ,  $t_{\mathbf{w}}^{k^*}$ , with probability  $\frac{1}{q}$ . This second level of verification provides a detection mechanism in the presence of colluding adversaries.

## VII. LOCATING SCHEME

**1) Reporting.** The following lemma, originally presented in [14] and [20], implies that for each received subspace,  $\Pi_N^P$ , from a parent  $P$ , node  $N$  may report a randomly chosen packet,  $\mathbf{y}_r$ , of the space instead of the space itself; and by checking if  $\mathbf{y}_r \in \Pi^S$ , the controller can determine if  $\Pi_N^P \subseteq \Pi^S$  to identify the polluted edges.

**Lemma 4** (Jafarisiavoshani *et al.* [14], [20]). *Let  $\Pi_1$  and  $\Pi_2$  be two subspaces of  $\mathbb{F}_q^{n+m}$  and assume that  $\mathbf{y}_r$  is a randomly selected packet from  $\Pi_1$ . Let  $d_{12}$  and  $d_1$  be the dimensions of  $\Pi_1 \cap \Pi_2$  and  $\Pi_1$ , respectively. With probability  $1 - q^{d_{12}-d_1}$ ,  $\mathbf{y}_r \in \Pi_2$  if and only if  $\Pi_1 \subseteq \Pi_2$ .*

**2) Using SpaceMac.** We use SpaceMac to prevent nodes from lying about their received spaces as follows. Whenever  $P$  sends a vector  $\mathbf{y}_i$  to  $N$ , it generates a tag,  $t_{\mathbf{y}_i}$ , of  $\mathbf{y}_i$  using the Mac algorithm with a secret key shared by  $P$  and

the controller. Then, when  $N$  reports  $\mathbf{y}_r$ , if  $\mathbf{y}_r$  is a linear combination of vectors that it received from  $P$ ,  $\mathbf{y}_i$ ’s, then  $N$  can generate a valid tag of  $\mathbf{y}_r$  by using the Combine algorithm on the tags of  $\mathbf{y}_i$ ’s that it received. If  $\mathbf{y}_r$  is not a linear combination of  $\mathbf{y}_i$ ’s then  $N$  can forge a valid tag for  $\mathbf{y}_r$  with only a negligible probability of  $\frac{1}{q}$ .

**3) Non-Repudiation Transmission Protocol.** SpaceMac forces nodes to report only true received subspaces since it is computationally difficult to forge valid tags otherwise. However, it does not prevent a malicious node from sending invalid tags to its children to prevent the children from reporting polluted spaces.

To address this, we utilize an efficient non-repudiation transmission protocol proposed by Wang *et al.* [15]. For a parent  $P$  and a child  $N$ , the controller generates a set of secret keys, denoted by  $\mathcal{X}$ , based on the private key of the parent and the ID  $N$  of the child. After that, the controller randomly selects a set of keys  $\mathcal{Y}$  from  $\mathcal{X}$  based on the private key of the child and the ID  $P$  of the parent; then, it sends  $\mathcal{Y}$  to the child. We denote  $\mathcal{X} \setminus \mathcal{Y}$  as  $\bar{\mathcal{Y}}$ ; also, let  $\lambda \stackrel{\text{def}}{=} |\mathcal{X}|$  and  $\delta \stackrel{\text{def}}{=} |\mathcal{Y}|$ .

When sending a packet,  $P$  generates  $\lambda$  tags (instead of one) using the Mac algorithm and all keys in  $\mathcal{X}$ . When receiving a packet,  $N$  uses its set of keys  $\mathcal{Y}$  and the Verify algorithm to verify  $\delta$  out of  $\lambda$  tags. Finally, when receiving a randomly chosen packet  $\mathbf{y}_r$  chosen from  $\Pi_N^P$ , and its  $\lambda$  tags from the  $N$ , the controller uses all keys in  $\bar{\mathcal{Y}}$  and the Verify algorithm to verify all  $\lambda - \delta$  tags. The controller, in this case, keeps track of a counter  $\theta$ ,  $\theta \leq \lambda - \delta$ . If at least  $\theta$  tags pass the verification then the controller accepts the report. The security when using this non-repudiation protocol in our context is described in detailed in [17]. Examples of values for these parameters and the corresponding probabilities are provided in Table I.

**4) Locating the Attackers.** After the controller collects the true subspaces from every node, we proceed similar to the approach by Jafarisiavoshani *et al.* [14] to locate the attackers. We proceed by partitioning the edges into two set: the set of polluted edges,  $\mathcal{E}_p$ , and non-polluted edges,  $\mathcal{E}_s$ , then analyzing the nodes with respect to the identified  $\mathcal{E}_p$  and  $\mathcal{E}_s$ . Preventing nodes from lying, our scheme produces an unambiguous partitioning of  $\mathcal{E}_p$  and  $\mathcal{E}_s$ , which helps to precisely locate the attacker. In particular, the adversary is always the node that has no incoming edge belonging to  $\mathcal{E}_p$  but has at least one outgoing edge belonging to  $\mathcal{E}_p$ . Fig. 4 shows the case where  $B$  is an attacker whom get identified because it has no incoming polluted edge but one outgoing polluted edge.

Due to limited space, we refer the reader to [17] for (i) a detailed description of the locating scheme and (ii) a detailed analysis of the security of the defense system, where we discuss (ii-a) multiple colluding adversaries, (ii-b) tag pollution attacks, (ii-c) denial of service attacks, as well as (ii-d) malicious receiver scenarios.

## VIII. PERFORMANCE EVALUATION

### A. Communication Overhead

**Detection scheme.** For the online overhead per packet, *i.e.*, the additional bandwidth required per packet sent in the

$q$	$\lambda$	$\delta$	$\theta$	$\Pr[P]$	$\Pr[N]$	Space Overhead
$2^8$	19	9	3	$2^{-10}$	$2^{-17}$	20 bytes
$2^8$	24	12	3	$2^{-14}$	$2^{-16}$	25 bytes
$2^8$	29	14	4	$2^{-16}$	$2^{-21}$	30 bytes

TABLE I

$q$ ,  $\lambda$ ,  $\delta$ , and  $\theta$  are the parameters of the locating scheme.  $\Pr[P]$  is the probability a malicious parent succeeds in preventing its child to report.  $\Pr[N]$  is the probability a malicious child succeeds in disparaging its parent. The space overhead refers to the size of the MAC tags used.

	# Tags	Mac	Combine	Verify	Security
C/C++	1	28	0.02	28	$2^{-8}$
	4	112	0.08	112	$2^{-32}$
Java	1	61	0.09	61	$2^{-8}$
	4	244	0.36	244	$2^{-32}$
Android	1	2,273	0.66	2,273	$2^{-8}$
	4	9,092	2.64	9,092	$2^{-32}$

TABLE II

Computational time in  $\mu\text{s}$  of SpaceMac algorithms in  $\mathbb{F}_{2^8}$

network, our detection scheme requires each packet to carry three SpaceMac tags: an end-to-end tag, a helper tag, and a verification tag. Each tag is a symbol in the field  $\mathbb{F}_q$ ; hence, the total overhead is  $3|q| = 3\lceil\log_2 q\rceil$  bits. Our communication overhead is fixed, regardless of the network topology.

In contrast, the online overhead of RIPPLE [10] varies depending on the network level. In particular, in [10], the authors define a level of a node  $N$  as the length of the longest path from  $S$  to  $N$ . The network level  $\ell$  is defined as the maximum among the levels of the nodes. In their scheme, each packet carries  $\ell$  MAC tags initially, then one or more tags are peeled off by every node the packet goes through. The average overhead is therefore approximately  $\frac{\ell}{2}\lceil\log_2 q\rceil$  bits.

In [9], to achieve security  $\frac{1}{q^a}$  and  $c$ -collusion resistance, *i.e.*, secure against any  $c$  colluding attackers, each packet carries  $|\mathbb{X}|$  MAC tags and each node verifies  $|\mathbb{B}|$  tags, where  $(\mathbb{X}, \mathbb{B})$  is a  $(c, d)$ -cover free family. For instance, to provide security  $\frac{1}{q}$  and 2-collusion resistance, each packet needs to carry 49 tags and each node verifies 7 out of these 49 tags [9]. The overhead is  $49\lceil\log_2 q\rceil$  in this case, or  $|\mathbb{B}|\lceil\log_2 q\rceil$  in general.

**Locating scheme.** The communication overhead of our locating scheme includes the  $\lambda$  tags carried by each packet, the reporting vectors sent by the nodes, and the announcement sent by the controller. The online overhead is the first one; the latter two overheads exist only when there is a pollution attack detected, thus are asymptotically negligible in the number of packets. The overhead per packet is  $\lambda\lceil\log_2 q\rceil$  bits. Table I shows that with an overhead of about 20 bytes per packet, the probability that a malicious parent succeeds in preventing its child from reporting,  $\Pr[P]$ , and the probability that a malicious child succeeds in disparaging its parent,  $\Pr[N]$ , are both very small.

**Combined Scheme.** The total online communication overhead of our defense scheme is  $(3 + \lambda)\lceil\log_2 q\rceil$  bits. Note that this overhead depends neither on the packet size nor on the generation size; hence, it becomes less expensive when the packet size is large. For instance, for  $\lambda = 19$ ,  $n = 1024$ ,  $m = 32$ , and  $q = 2^8$ , the per-packet communication overhead is only 2% while the security of the detection is  $2^{-8}$  and of the security of the locating is  $2^{-10}$ . In contrast to the other two schemes, [9] and [10], our overhead is constant in terms of the network size and the number of attackers.

## B. Computation Overhead

The major online computation overhead, *i.e.*, the computation done by each node for each packet, of both of our schemes come from the algorithms Mac, Combine, and Verify performed at each node. Both the Mac and the Verify algorithms incur one PRG call,  $m$  PRF calls, and  $(n + 2m)$  finite field multiplications. If we let  $w$  be the average number of packets combined by each node, then the Combine algorithm incurs  $w$  multiplications on average.

**Detection Scheme.** The operations performed by each node in the detection scheme for each packet  $\mathbf{y}$  include (i) verifying the integrity of  $\mathbf{y}$  using the Verify algorithm, (ii) combining the received helper tags to generate a verification tag for an outgoing packet  $\mathbf{z}$  using the Combine algorithm, (iii) computing a helper tag for  $\mathbf{z}$  using the Mac algorithm. If the node is the source, it needs to compute the end-to-end tag using the Mac algorithm; however, in this case it does not need to verify the integrity of packets. If it is a receiver, it needs to verify the end-to-end tag by performing another Verify algorithm. The worst case computation overhead, *i.e.*, when the node is a receiver, in terms of the number of finite field multiplications, is  $3(n + 2m) + w$ .

**Locating Scheme.** For each packet received, each node verifies  $\delta$  tags using the Verify algorithm. For each packet it sends out, each node needs to compute  $\lambda$  tags using the Mac algorithm. The total overhead is therefore  $(\delta + \lambda)(n + 2m)$  number of multiplications.

**Combined Scheme.** The overall computation overhead per packet per node of our combined scheme is  $(3 + \delta + \lambda)(n + 2m) + w$ . Table III summarizes the computation overhead to achieve the security  $2^{-8}$  for our scheme, RIPPLE [10], and HomMac [9]. We omit the detailed description of the overhead of the other two schemes and refer the reader to [17]. We implement multiplication in  $\mathbb{F}_{2^8}$  by creating an offline multiplication table, storing all  $2^{16}$  products of pairs of elements in this field. The table only occupies about 64 KB in C/C++ and 128 KB in Java. The numbers reported are averaged over  $10^6$  multiplications. Our PC has a quad-core 2.8 Ghz processor and 32 GB of RAM, and our Android device is a Samsung Captivate with a single 1 Ghz processor and 512 MB RAM.

As shown in Table III, the computational latency of our detection scheme is in the same order of magnitude as the other two detection schemes. We note, however, that our detection scheme achieves all of the desired properties: in-network detection, arbitrary collusion resistance, and tag pollution resistance without the need of time synchronization as in [10]. Table III also shows that the latency of our combined detection-locating scheme is about 10 times higher than that of our detection scheme. This is the trade-off when one wants to locate and eliminate all attackers.

## C. Library

We implement SpaceMac in both C/C++ and Java and provide them as a library. As mentioned before, we implement field multiplication using a look-up table. We implement addition as a simple XOR operation. Finally, we implement

	<b>RIPPLE</b> [10]	<b>Broadcast MAC</b> [9]	<b>Our Detection</b>	<b>Our Full System</b>
<b>Features</b>	In-network detection Arbitrary collusion res. Tag-pollution res. <i>No</i> locating	In-network detection <i>c</i> -collusion res. <i>No</i> tag pollution res. <i>No</i> locating	In-network detection Arbitrary collusion res. Tag pollution res. <i>No</i> locating	In-network detection Arbitrary collusion res. Tag pollution res. Exact locating
<b>Network Params.</b>	$q = 2^8, n = 1024, m = 32, w = 4, \ell = 9$			
<b>Scheme Params.</b>	$ \mathbb{X}  = 49,  \mathbb{B}  = 7, c = 2$		$\lambda = 19, \delta = 9, \theta = 3$	
<b>Security</b>	$2^{-8}$			
<b># Multiplications</b>	1,096	7,812	3,268	33,732
<b>C/C++ (<math>\mu</math>s)</b>	5.5	39.1	16.3	168.7
<b>Java (<math>\mu</math>s)</b>	6.6	46.9	19.6	202.4
<b>Android (<math>\mu</math>s)</b>	116.2	828.1	346.4	3,575.6

TABLE III

Online computation overhead per packet per node in terms of the number of finite field multiplications, computing latency in C/C++, Java, and on an Android platform (Samsung Captivate).

PRF and PRG using AES with CBC mode of operation. We make our SpaceMac library's source code available online [16]. Table II provides the benchmark of all three algorithms of SpaceMac. For the benchmark, we set  $n = 1024$ ,  $m = 32$ , and  $w = 4$ . The reported values correspond to the averages taken over  $10^5$  runs of each algorithm. Table II shows that in order to achieve high security,  $2^{-32}$ , the computational latency of the C/C++ implementation is only in the order of hundreds of microseconds.

## IX. CONCLUSION

In this work, we introduce a novel homomorphic MAC scheme for expanding spaces, called SpaceMac, and we propose a cooperative defense system against pollution attacks built on SpaceMac. To the best of our knowledge, our system is the first that can provide both in-network detection and exact locating of the attackers. In addition, our system is collusion resistant and tag-pollution resistant. Our evaluation results using real implementation in C/C++ and Java on multiple devices demonstrate that our defense system incurs both low communication and low computation overhead. We implemented SpaceMac as a ready-to-use library and make it available online [16].

## X. ACKNOWLEDGEMENT

We would like to thank Prof. Stanislaw Jarecki at UC Irvine for insightful discussions on the SpaceMac scheme.

## REFERENCES

- [1] R. Ahlswede, N. Cai, S.-y. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.
- [2] M. Kim, L. Lima, F. Zhao, J. Barros, M. Medard, R. Koetter, T. Kalkert, and K. J. Han, "On Counteracting Byzantine Attacks in Network Coded Peer-to-Peer Networks," *IEEE JSAC*, vol. 28, no. 5, pp. 692–702, Jun. 2010.
- [3] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical Defenses against Pollution Attacks in Intra-Flow Network Coding for Wireless Mesh Networks," in *Proc. of ACM WiSec 2009*, Zurich, Switzerland, pp. 111–122.
- [4] C. Gkantsidis and P. R. Rodriguez, "Cooperative Security for Network Coding File Distribution," in *Proc. of IEEE INFOCOM 2006*, Barcelona, Spain, pp. 1–13.
- [5] N. Cai and R. W. Yeung, "Secure Network Coding," in *Proc. of ISIT 2002*, Lausanne, Switzerland, p. 323.
- [6] S. Jaggi, M. Langberg, T. Ho, and M. Effros, "Correction of Adversarial Errors in Networks," in *Proc. of ISIT 2005*, Adelaide, Australia, pp. 1455–1459.
- [7] R. Koetter and F. R. Kschischang, "Coding for Errors and Erasures in Random Network Coding," in *Proc. of ISIT 2007*, Nice, France, pp. 791–795.

- [8] 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 SP 2004*, Oakland, CA, USA, pp. 226–240.
- [9] S. Agrawal and D. Boneh, "Homomorphic MACs : MAC-Based Integrity for Network Coding," in *Proc. of ACNS 2009*, Paris-Rocquencourt, France, pp. 292–305.
- [10] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "RIPPLE Authentication for Network Coding," in *Proc. of IEEE INFOCOM 2010*, San Diego, CA, USA, pp. 2258–2266.
- [11] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a Linear Subspace : Signature Schemes for Network Coding," in *Proc. of PKC 2009*, Irvine, CA, USA, pp. 68–87.
- [12] E. Kehdi and B. Li, "Null Keys : Limiting Malicious Attacks Via Null Space Properties of Network Coding," in *IEEE INFOCOM 2009*, Rio de Janeiro, Brazil, pp. 1224–1232.
- [13] P. Zhang, Y. Jiang, C. Lin, H. Yao, A. Wasef, and X. S. Shen, "Padding for Orthogonality : Efficient Subspace Authentication for Network Coding," in *IEEE INFOCOM 2011*, Shanghai, China, pp. 1026–1034.
- [14] M. Jafarisiavoshani, C. Fragouli, and S. Diggavi, "On Locating Byzantine Attackers," in *Proc. of NetCod 2008*, Hong Kong, China, pp. 1–6.
- [15] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "Identifying Malicious Nodes in Network-Coding-Based Peer-to-Peer Streaming Networks," in *Proc. of IEEE Mini INFOCOM 2010*, San Diego, CA, USA, pp. 1–5.
- [16] A. Le, "SpaceMac Library," 2011. [Online]. Available: <http://www.ics.uci.edu/~anhml/software.html#SpaceMac>
- [17] A. Le and A. Markopoulou, "Cooperative Defense Against Pollution Attacks in Network Coding Using SpaceMac," in *Technical Report*. [Online]. Available: <http://arxiv.org/abs/1102.3504>
- [18] A. Le and A. Markopoulou, "Locating Byzantine Attackers in Intra-Session Network Coding using SpaceMac," in *Proc. of NetCod 2010*, Toronto, ON, Canada, pp. 1–6.
- [19] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC Press, 2007.
- [20] M. Jafarisiavoshani, C. Fragouli, and S. Diggavi, "Subspace Properties of Randomized Network Coding," in *Proc. of ITW 2007*, Bergen, Norway, pp. 1–5.



**Anh Le** is a Ph.D. candidate in the Computer Science Program at UC Irvine. He received a B.S. degree in Computer Science - Mathematics from the University of Manitoba, Canada in 2006 and a M.Math. degree in Computer Science from the University of Waterloo, Canada in 2008. He was an intern with Google, Mountain View, CA in summer 2011. His research interests are in the areas of security and privacy, including network coding security, collaborative firewalls, phishing detection, network intrusion detection, smartphone and browser privacy.



**Athina Markopoulou** (SM'98, M'02) is an assistant professor in the EECS Dept. at the University of California, Irvine. She received a Diploma degree in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 1996, and M.S. and Ph.D. degrees, both in Electrical Engineering, from Stanford University in 1998 and 2003, respectively. She has been a postdoctoral fellow at Sprint Labs (2003) and at Stanford University (2004–2005), and a member of the technical staff at Arastra Inc. (2005). Her research interests include

network coding, online social networks, network measurement and security. She received the NSF CAREER award in 2008.