

TESLA-Based Defense Against Pollution Attacks in P2P Systems with Network Coding

Anh Le, Athina Markopoulou
University of California, Irvine
{anh.le, athina}@uci.edu

Abstract—Pollution attacks are well-known to have detrimental effect on intra-session network coding in general, and in peer-to-peer (P2P) systems with network coding in particular. Previously proposed defense mechanisms against pollution attacks in intra-session network coding face various challenges that make them ill-suited for P2P systems: they are either computationally expensive, not collusion resistant, or work only on fixed, known topologies. In this work, we propose a novel, complete defense system for network coding-based P2P systems that can (i) quickly detect corrupted blocks, (ii) precisely identify the attackers, thereby eliminating them from the network, (iii) resist arbitrary collusion, and (iv) work with unknown, dynamic topologies, as it is the case in P2P systems. Our scheme uses and builds on two key ingredients: homomorphic message authentication codes and time asymmetry (as in TESLA [1]) to provide *source authentication* for the detection scheme and *non-repudiation* for the identification scheme. Our mechanisms introduce significantly less communication and computation overhead than other comparable state-of-the-art schemes for P2P systems. Using implementation in both C/C++ and Java, on both a PC and an Android device, we show that the computational delay per block at each peer is as low as 600 microseconds and the bandwidth overhead is as low as 1.3%.

Keywords—Peer-to-peer; network coding; pollution; detection; identification; TESLA; homomorphic MAC.

I. INTRODUCTION

Peer-to-peer (P2P) systems for content distribution [2] and streaming [3], [4] have been shown to greatly benefit from using random network coding [5]. The benefits stem from the distributed operation of the peers and include robustness to packet loss, delay, as well as dynamic network topologies.

Network coding, in general, and P2P systems with random network coding in particular, are known to be vulnerable to pollution (a.k.a. Byzantine modification) attacks. During such attacks, malicious peers can upload corrupted blocks to their neighbors. These neighbors may (unknowingly) combine the corrupted blocks with other blocks and spread them further. This eventually results in a large number of corrupted blocks, which waste network resources and prevent the peers from decoding the original blocks, thus degrading the performance of the whole system.

Due to the severity of pollution attacks, a large number of defense mechanisms has been proposed by the research community. Existing defense mechanisms using homomorphic hashes [6], [7] or homomorphic signatures [8]–[10] are computationally expensive, incurring a large delay per block at each peer. The ones that use homomorphic message authentication codes (MACs) or subspace properties are either only c -collusion

resistant for a predetermined small c [11], [12], not collusion resistant [17] (thus vulnerable to a large number of colluding peers), or simply only works on fixed directed acyclic graph networks [13], [14] (thus are not applicable to P2P systems). Furthermore, the majority of the existing approaches focuses solely on the detection of corrupted blocks but do not address the attack at its root, which is the identification and elimination of malicious peers.

Motivated by these observations, in this work, we propose a novel defense mechanism for network coding-based P2P systems. Our defense scheme is complete because it includes both a detection scheme, which allows peers to detect attacks quickly, and an identification scheme, which can precisely pinpoint the malicious peers and eliminate them from the network. Our schemes are inspired by TESLA [1] and RIPPLE [14] and require loose time synchronization among peers. In particular, we leverage homomorphic MACs, pre-distribution of MAC tags, and delayed key disclosure to provide both source authentication and non-repudiation properties. The source authentication property allows every peer to verify the integrity of any block on-the-fly, right after downloading it. The non-repudiation property holds each peer accountable for the blocks it uploads, which enables the identification of malicious peers.

To the best of our knowledge, our mechanism is the first complete defense mechanism for P2P systems that simultaneously provides the following features: (i) **In-network Detection**: Each peer can verify the integrity of each block on-the-fly; (ii) **Precise Identification**: Malicious peers are precisely identified and eliminated from the network; (iii) **Arbitrary Collusion Resistance**: Our defense mechanism can deal with a large number of colluding malicious peers; moreover, it also resists *tag pollution* (first studied by Li *et al.* [14]); and (iv) **Low Overhead**: Our defense mechanism incurs low communication and computation overhead. Furthermore, we do not require a known or fixed topology, which makes our approach well-suited for P2P topologies.

We evaluate our detection and identification schemes in a practical setting, and we show that they introduce lower overhead compared to state-of-the-art schemes for P2P systems [6], [15]. We find that the total communication overhead of the combination of both schemes is only about 1.3%. Furthermore, our implementation of the homomorphic MAC scheme in both C/C++ and Java show that both the detection and identification schemes incur a very small amount of computational delay: $\sim 600 \mu\text{s}$ in C/C++ on a PC and just $\sim 45 \text{ ms}$ in Java on an

Android resource-constrained device (Samsung Captivate).

The rest of this paper is organized as follows. Section II discusses related work. Section III formulates the problem. Section IV describes our detection mechanism. Section V describes how we identify the attackers. Section VI presents the evaluation results. Finally, we conclude in Section VII.

II. RELATED WORK

Defense mechanisms against pollution attacks could be classified into three main categories: error correction, attack detection, and attacker identification. Due to lack of space, we only discuss closely related work in the latter two categories, and we refer the reader to [13] for a discussion of error correction approaches.

Attack Detection. In [7], Krohn *et al.* proposed a homomorphic hash scheme for verification of rateless erasure codes. Gkantsidis and Rodriguez [6] later propose probabilistic checking and cooperation mechanisms among nodes to reduce the computation overhead when using the scheme in [7] in P2P file distribution systems. This approach suffers from a major drawback: it requires a large number of modular exponentiations performed by each peer, which is computationally expensive and results in high latency.

Agrawal and Boneh [12] proposed a defense mechanism based on a homomorphic MAC scheme. This approach relies on cover-free set systems for pre-distributing keys to provide in-network detection, and thus, only c -collusion resistant. It is also susceptible to tag-pollution attacks, where malicious nodes tamper with subsets of tags of the packets they receive, as presented in [14]. Li *et al.* [14] present RIPPLE, a defense mechanism also built on a homomorphic MAC scheme but utilizes time asymmetry (inspired by TESLA [1]) to achieve collusion and tag pollution resistance. This approach, however, only works on fixed directed acyclic graph networks: it needs to know the longest path from each node to the source in order to precompute the MAC tags for the nodes. The detection scheme in our previous work [13] also suffers from the same drawback. Our detection scheme presented in this work is inspired by RIPPLE. In particular, similarly to RIPPLE, we combine the use of homomorphic MAC and time asymmetry; however, different from RIPPLE, we leverage the pre-distribution of MAC tags to support general P2P network topologies.

Another approach that also leverages the time asymmetry to provide detection is the DART scheme proposed by Dong *et al.* [16]. In this work, checksums computed from random linear transformations are periodically released by the source to enable in-network detection. This scheme is essentially a public-key scheme, as opposed to a symmetric-key scheme (like RIPPLE and ours). It requires one public key signature verification per checksum at each node, which incurs high delay and make the peers susceptible to denial of service attacks, where attackers flood peers with signature packets for verification.

In [17], Kehdi and Li proposed an in-network detection scheme which exploits subspace properties of network coding.

In their scheme, intermediate nodes verify the integrity of a vector (packet) by checking if it belongs to the subspace spanned by the source vectors. Null keys, which are vectors orthogonal to the space spanned by the source vectors, are used for the verification. The verification in this scheme can be done very efficiently; however, this scheme is not collusion resistant: multiple nodes can collude to infer the null keys and make benign nodes accept corrupted vectors.

Attacker Identification. Jafarisavoshani *et al.* [18] leveraged the subspace properties of random network coding to approximately identify the location of attackers. Our prior work [19] and its extended version [13] build on [18]; they combine the subspace properties with an appropriately designed MAC, called SpaceMac, to provide exact identification of attackers. SpaceMac is homomorphic and supports expanding subspaces. In this paper, we also use SpaceMac as a building block but SpaceMac is not necessary here as it was in [13], [19]: any other homomorphic MAC scheme could replace it in the system proposed in this paper. Another difference between our system presented in this work and the previous approaches [13], [18], [19] is that they assume a fixed directed acyclic graph network, which is not the case in P2P systems, while we do not.

Wang *et al.* [15] introduced a light-weight non-repudiation protocol and use it to support their identification scheme for P2P systems. This scheme requires a distribution of multiple checksums (of all the blocks sent by the source) to all the peers when an attack is detected, which incurs significant communication overhead. By leveraging TESLA to provide the non-repudiation property, our identification scheme is able to achieve significantly higher security per byte overhead than [15] while avoiding the need of checksum dissemination.

III. PROBLEM FORMULATION

Network Operations. We consider an overlay P2P network where every peer can download from any other peer and the network employs random linear network coding. Let there be a source S who has a file \mathcal{F} which is of interest to every peer. Let \mathcal{V} denote the set of peers. To distribute file \mathcal{F} , the source first divides the file into multiple generations, each of which consists of m blocks. For clarity, we focus on a single generation. We denote the blocks as $\hat{\mathbf{b}}_1, \dots, \hat{\mathbf{b}}_m$. Each block is a vector in a n -dimensional linear space \mathbb{F}_q^n , where q is the field size. The source then augments every block $\hat{\mathbf{b}}_i$ with its m global coding coefficients. The resulting blocks, \mathbf{b}_i , called *source blocks*, have the following form:

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

space spanned by the source blocks *source space*.

At the beginning, the source S uploads linear combinations of source blocks to a subset of peers. Each peer then upload blocks to other peers. A peer upload to another peer using either pull-based [2] or push-based mechanism [4]. Peer A , when uploading to a peer B , linearly combines its downloaded blocks and uploads encoded blocks to B . For instance, if A currently have ℓ blocks, \mathbf{e}_i , $i \in [1, \ell]$, then it uploads $\mathbf{e} =$

$\sum_{i=1}^{\ell} \alpha_i \mathbf{e}_i$ to B , where the α 's are local coding coefficients randomly chosen from \mathbb{F}_q .

On one hand, when a peer is benign, the encoded blocks that it uploads are linear combinations of the source blocks, thus belong to the source space. On the other hand, when a peer is malicious, it uploads blocks that are not in the source space to pollute the network. Peers accumulate blocks over time. A peer can subsequently decode the m source blocks by performing Gaussian elimination as soon as it downloaded m linear independent blocks from other peers.

Threat Model. We assume the source S is trusted and the adversaries may compromise an arbitrary subset of peers and inject corrupted blocks from any of these peers, at any time. The adversaries may also collude and tamper with the MAC tags. A successful modification of tags constitutes an attack called tag pollution, which could be as devastating as a pollution attack [14]. Similar to other approaches that use cryptography, we assume that the adversaries are aware of our cryptographic constructions and applications; however, their running time is polynomial in the security parameter.

Assumptions. We assume that each generation is downloaded by all the peers in a small window of time. This is native to a P2P live streaming system, however, is a restriction on a P2P content distribution system. We further assume each peer P shares with the source S a pair of keys (k_1^P, k_2^P) . The shared keys can be established with the help of a Public Key Infrastructure (PKI) Other symmetric-key approaches also assume the existence of a PKI [11], [14] or the existence of shared secret keys [12]. In general, the problem of establishing shared secret keys is a challenging problem of its own and is orthogonal to this work. Finally, similar to [14], [16], we assume loose time synchronization among peers, which can be achieved using a low complexity protocol [1].

IV. DETECTION SCHEME

A. Homomorphic MACs

Our detection scheme requires the use of a MAC scheme which has homomorphic property. In particular, we require that any peer able to construct a valid MAC tag for any legitimate block it wants to upload, given that it has valid MAC tags of the source blocks. To this end, any of the proposed homomorphic MAC schemes, such as, the schemes proposed in [12]–[14], suffice. We will use the SpaceMac scheme proposed in our prior work [13].

SpaceMac consists of a triplet of algorithms: Mac, Combine, and Verify. The Mac algorithm generates a tag (in \mathbb{F}_q) for any block (in \mathbb{F}_q^{n+m}) under a given key. The Combine algorithm generates a tag for any linear combination of blocks by linearly combining their tags. The Verify algorithm verifies if a tag is valid for a block under a given key. We refer the reader to [13] for the detailed construction and formal proof.

SpaceMac satisfies our requirement as follows: Let t_i , $i \in [1, m]$, be the MAC tags of the source blocks, \mathbf{b}_i , generated using the Mac algorithm with key K . Assume that A wants to upload a legitimate block, \mathbf{e} , to its peers. Since

\mathbf{e} is a legitimate block, it belongs to the source space. Let $\mathbf{e} = \sum_{i=1}^m \alpha_i \mathbf{b}_i$, where $\alpha_i \in \mathbb{F}_q$. Then the MAC tag t of \mathbf{e} under key K can be computed using Combine: $t = \text{Mac}_K(\mathbf{e}) = \text{Combine}(\{\mathbf{b}_i, t_i, \alpha_i\}_{i=1}^m) = \sum_{i=1}^m \alpha_i t_i$.

We note that both the Mac and Verify algorithms need the identifier of the source space as an input. Since we focus on a single generation, we omit the identifier in our discussion for clarity. Finally, a MAC tag is an element of \mathbb{F}_q , and the probability of forging a valid MAC tag without knowing the key is $\frac{1}{q}$. To improve the security, one can either increase the field size or use multiple tags.

B. Overview of the Detection Scheme

The source S and other peers in the network upload blocks following a time schedule. Time is divided into uniform intervals. Each interval, j , is associated with a different key, K_j . At the start of each interval, j , the source discloses the key of that interval, K_j , to every peer. Let N be the upper bound of the number of intervals required for distributing a generation to all peers¹.

Before uploading the encoded blocks, S sends to every peer the MAC tags of the source blocks generated under the keys associated with the N intervals. In particular, for each key, K_j , $j \in [1, N]$, S creates m MAC tags of the m source blocks using the Mac algorithm with key K_j and uploads them. We called these tags *source tags*.

At interval j_0 , to upload a legitimate block, \mathbf{e} , to a peer B , a peer A has to construct a valid MAC tag of the block and upload the tag along with the block. To this end, A uses the source tags of interval j_1 ($j_1 > j_0$) to construct a MAC tag of \mathbf{e} . It is possible for A to construct the tag because (i) \mathbf{e} is a combination of the source blocks, (ii) A received the source tags of interval j_1 previously from S , and (iii) the MAC scheme used is homomorphic.

Upon downloading \mathbf{e} , B accepts \mathbf{e} if the source has not yet released the key for the interval j_1 , K_{j_1} . After accepting the block, the block is still unverified, *i.e.*, it could be corrupted. B then verifies the integrity of \mathbf{e} when it receives the key K_{j_1} at a later time. If the block fails the check, then the block is corrupted, and a pollution attack is detected; otherwise, the block is legitimate, and B can use it in subsequent operations.

From the attacker point of view, in order to pollute the system, he must have his peers accept the corrupted block. For this, he has to generate a valid MAC tag of the corrupted block. Note that a corrupted block is a block not in the source space, and that the attacker cannot use a key that is already disclosed by the source to generate the MAC tag (otherwise its peers will simply reject the block). Without knowing the key, it is computationally difficult for the attacker to generate a valid tag for a block that does not belong to the source space (by the security guarantee of the MAC scheme). As a result, since the attacker cannot generate a valid MAC tag of the corrupted block, the attack will be detected as soon as the receiving peer checks the block.

¹ N is the maximum number of intervals before a new set of keys need to be generated. For support of unbounded duration, we refer the reader to [20].

C. Full Detection Scheme

Setup. We follow the technique in TESLA [1] to setup the time intervals and key chains. Let T_0 denote the starting time of the transmission of the generation, T_j denote the starting time of interval j , and d denote the interval length. By definition, we have $T_j = T_0 + j \cdot d$, $j \in [1, N]$. To create the MAC keys, a one-way function, F , is constructed using a pseudorandom function (PRF) f : $F(K) = f_K(0)$. This one-way function is used to create a *one-way key chain*. The key chain is then used in reverse order of generation to derive the MAC keys. In particular, the source first picks a random value for K_N . Then, the other keys of the key chain are computed recursively using F : $K_j = F(K_{j+1})$, $j \in [0, N - 1]$.

To avoid using the same key multiple times in different cryptographic operations, *i.e.*, using key K_j to derives key K_{j-1} as well as to compute a MAC tag, we use another one-way function, F' , to derive the MAC keys. In particular, we construct F' using another PRF, f' : $F'(K) = f'_K(1)$; then, F' is used to derive the MAC keys: $K'_j = F'(K_j)$, $j \in [0, N]$. For each key K'_j , a set of source tags is created using the Mac algorithm. In particular, $t_{j,i} = \text{Mac}_{K'_j}(\mathbf{b}_i)$, $i \in [1, m]$. Let \mathcal{T}_j denote the set of source tags created using K'_j : $\mathcal{T}_j = \{t_{j,i}, \forall 1 \leq i \leq m\}$.

At the beginning of each interval j , the source discloses K_j to all the peers by uploading $\{j \parallel K_j\}$, where ‘ \parallel ’ denote the concatenation. Note that given an authenticated K_0 and an index j , anyone can check if a value K is the j -th value in the one-way key chain by checking if $K_0 = F^j(K)$, where $F^n(x)$ denotes n consecutive applications of F .

Bootstrapping. The source and peers in the network loosely synchronize their clocks before the transmission using the protocol proposed in [1]. The requirement is that each peer knows the upper bound on S clock. S then uploads a bootstrapping packet to each peer. This bootstrapping packet includes all the source tags, \mathcal{T}_j , $j \in [0, N]$; the time S starts uploading content blocks, T_0 ; the interval length, d ; and the commitment, K_0 . Each packet is uploaded to each peer, P , through an authenticated channel achieved with k_1^P and a traditional MAC scheme, *e.g.*, HMAC.

Uploading Blocks. To upload an encoded block \mathbf{e} to a peer B , a peer A needs to compute a MAC tag for \mathbf{e} . Let δ_{AB} be the upper bound of the time it takes to send \mathbf{e} from A to B . A needs to pick a value j_1 such that when B receives \mathbf{e} , the source S has not released the key K_{j_1} . Let T_S denote the upper bound of the time of S when A sends \mathbf{e} , then the upper bound of time of S when B receives \mathbf{e} is $T_S + \delta_{AB}$. A computes j_1 as follows: $j_1 = \lfloor \frac{T_S + \delta_{AB} - T_0}{d} \rfloor + 1$. An accurate estimation of δ_{AB} will help to reduce the time B waits to verify \mathbf{e} as B will need K'_{j_1} (derived from K_{j_1}) for the verification.

Let $\alpha_1, \dots, \alpha_m$ be the global coding coefficients of \mathbf{e} , A compute the MAC tag of \mathbf{e} under key K'_{j_1} using the Combine algorithm: $t_{\mathbf{e}}^1 = \text{Mac}_{K'_{j_1}}(\mathbf{e}) = \text{Mac}_{K'_{j_1}}(\sum_{i=1}^m \alpha_i \mathbf{b}_i) = \text{Combine}(\{\mathbf{b}_i, t_i, \alpha_i\}_{i=1}^m) = \sum_{i=1}^m \alpha_i t_i$. A finally uploads to B the packet p : $p = \{j_1 \parallel t_{\mathbf{e}}^1 \parallel \mathbf{e}\}$.

Downloading Blocks. We call a key which is not yet disclosed by the owner of the key a *safe key*. Upon receiving the packet p from A , B first needs to determine if K_{j_1} is safe based on the key release schedule of S . To this end, let T'_S denote the upper bound of the time of S when B receives p . B then checks if $j_1 \leq \lfloor \frac{T'_S - T_0}{d} \rfloor$. If it is, it means that S has already released K_{j_1} ; B then discards the packet. This is because K_{j_1} is no longer a safe key, *i.e.*, there may be other peers that know K_{j_1} , thus are able to forge the MAC tag. B also drops \mathbf{e} if $j_1 > N$ (out of range). Otherwise, B buffers p and waits for the verification.

When B receives $\{j_1 \parallel K_{j_1}\}$ from S , it first verifies if the key is a valid key by checking if $F^{j_1}(K_{j_1}) = K_0$. If not, then K_{j_1} is not a valid key, B will drop this key. Otherwise, B derives the MAC key, $K'_{j_1} = F'(K_{j_1})$, and performs the Verify algorithm to check the integrity of \mathbf{e} , *i.e.*, checking if $\text{Verify}_{K'_{j_1}}(\mathbf{e}, t_{\mathbf{e}}^1) = 1$. If \mathbf{e} fails the verification, \mathbf{e} is a corrupted block; otherwise, \mathbf{e} is classified as legitimate.

Remark. Since each block is buffered at a receiving peer for a certain amount of time for the verification, the throughput of the system may be reduced. Nevertheless, as discussed in [16], one may incorporate *pipelining*, where multiple generations are transmitted concurrently to saturate the upload and download bandwidth of the peers.

D. Security Analysis

Assume that A is the attacker who tries to make B accept a corrupted block \mathbf{e} . A may pick a small j_1 such that K_{j_1} is already disclosed by the source and known to A , thus it can forge a valid tag of \mathbf{e} . This strategy fails because B will not accept \mathbf{e} if K_{j_1} is no longer a safe key. A may also pick a large value of j_1 , but this only delays the check by B . Consequently, the only way A can make B accept \mathbf{e} is to forge a valid MAC tag of \mathbf{e} without knowing the key or using Combine (as \mathbf{e} does not belong to the source space). By the security guarantee of SpaceMac, the probability that A successfully make B accept a corrupted block is negligible: $\frac{1}{q^{t_1}}$, when using ℓ_1 MAC tags. Hence, we have the following Lemma:

Lemma 1. *Let ℓ_1 be the number of MAC tags used in the detection scheme. The probability that an attacker successfully injects a corrupted block into the network is $\frac{1}{q^{\ell_1}}$.*

The above analysis holds even when there is a large number of colluding peers. This is because no matter how many or which peers are colluding, they still do not learn a safe key; thus, they cannot make a benign peer accept a corrupted block. Furthermore, since each MAC tag is immediately verified by the receiving peer, our scheme is inherently resistant to tag pollution attacks.

V. IDENTIFICATION SCHEME

A. Overview of the Identification Scheme

Central Authority. In order to identify and eliminate malicious peers, we need a central authority who manages the peers in the network. In this work, we use S for this task. When a peer

B detects a corrupted block uploaded by a peer A , it reports this block to S . S will determine if the report is correct and inform every peer to put A into its blacklist.

Scenario. For clarity, we discuss the case of single adversary first. Let A be the malicious peer who tries to pollute the network by uploading multiple corrupted blocks to its peer B . The probability that B accepts a corrupted block from A is $\frac{h}{q^{c_1}}$, where h is the number of corrupted blocks A uploaded to B . We assume that A uploads more than one corrupted blocks, i.e., $h > 1$, to B to increase its success probability.

Evidence Tag. We require A , when uploading a block e to B , to also upload a MAC tag of the block (in addition to the MAC tags of the detection scheme). The tag must be generated under a secret key, K^A , shared by A and the source: $t = \text{Mac}_{K^A}(e)$. We call this tag *evidence tag* as it serves as an evidence indicating that A did upload e . In particular, when B detects that e is corrupted, B can report to S both e and its evidence tag, t . Since S knows the source space, S can check if e is corrupted; moreover, since A and S are the only parties that know K^A , the tag can only be generated by either A or S . Based on this, S can hold A accountable for uploading the corrupted block e , thereby identifying A as the attacker.

Valid Tag Enforcement. Note that to avoid being identified as the attacker, A can upload a bogus tag, t' , of e : $t' \neq \text{Mac}_{K^A}(e)$. In this case, t' can no longer be used as the evidence because when receiving the report from B , S checks that t' is not a valid tag of e under key K^A ; thus, S cannot conclude that A sent e . As a result, we need an additional mechanism to enforce A to upload only valid tags of its outgoing blocks.

To this end, we apply another instance of TESLA. We first introduce two states for a peer: *suspicious state* and *normal state*. The states are how a downloading peer perceives an uploading peer. A peer considers another peer suspicious as soon as it detects a corrupted block uploaded by the peer. After an amount of time, Δ , called *suspicious window*, the downloading peer puts the suspicious peer back to the normal state if it does not detect any additional corrupted block uploaded by the peer during this time. Δ is a configurable parameter. We require a peer to disclose its shared secret key (used to compute the evidence tag) to a downloading peer after a period of time, and we require a downloading peer to verify all evidence tags of blocks of a suspicious peer before using the blocks.

For example, assume that A wants to avoid the identification scheme by uploading to B corrupted blocks with bogus evidence tags. B detects the first corrupted block and put A into suspicious state. B then buffers all subsequent corrupted blocks sent by A for the verification of their evidence tags. Thus, even if one of the subsequent corrupted blocks bypasses the detection scheme, B will still drop the block due to its bogus evidence tag. As a result, if A wants to pollute the network, it is in its best interest to send valid evidence tags.

Summary. To identify the attackers, we apply another instance of delayed key disclosure, but this time, to each pair of

uploading and downloading peers instead of pairs consisting of the source and a peer. We require the uploading peer to attach to each of its outgoing blocks a MAC tag that can only be generated by itself. The tags serve as evidence to hold it accountable for what it uploaded.

B. Full Description of the Identification Scheme

Setup. The peers still upload blocks based on the defined time intervals T_j , $j \in [0, N]$. For each peer P , $P \in \mathcal{V} \setminus \{S\}$, the source creates a one-way key chain, $\{K_j^P\}_{j=0}^N$. Each key, K_j^P , in the key chain is then used to create a MAC key, $K_j^{\prime P}$. Both the chains and the MAC keys are constructed in a way similar to the way we create the chain $\{K_j\}$ and derive the MAC keys K_j' in the previous section. The keys $K_j^{\prime P}$ are used to create evidence tags by the peer P . The keys in the chain $\{K_j^P\}$ are initially known only to the peer P and S . Each key, K_j^P , in the key chain is eventually disclosed by P to its downloading peers at the beginning of the interval T_j .

Bootstrapping. Each pair of peers in the network and the source loosely synchronize their time. The requirement here is that each peer and the source know the upper bound of the time of all other peers. In addition to the bootstrapping packet used in the detection scheme, for each peer P , the source also uploads another bootstrapping packet. This packet includes the seed K_N^P to generate the key chain $\{K_j^P\}$ and the commitments $\{K_0^{P'}, P' \in \mathcal{V} \setminus \{P\}\}$. This packet is sent to P over a secure and authenticated channel achieved with k_1^P (for authentication) and k_2^P (for encryption) using a standard encrypt-then-authenticate scheme².

Uploading Blocks. When a peer A uploads an encoded block, e , to a peer B , beside the tag(s) of the detection scheme, A also needs to upload an evidence tag. Let δ_{AB} and δ_{BS} be the upper bounds of the time it takes to send e from A to B and from B to S , respectively. A needs to pick j_2 such that by the time e reaches S (in case B reports), $K_{j_2}^A$ has not been disclosed by A . Let T_A denote the time at A when A uploads e . A computes j_2 as follows: $j_2 = \lfloor \frac{T_A + \delta_{AB} + \delta_{BS} - T_0}{d} \rfloor + 1$.

A then creates the evidence tag of e under key $K_{j_2}^{\prime A}$. This tag can be created using a traditional MAC scheme, such as, HMAC; nevertheless, the SpaceMac scheme can also be used. For the convenience of notation, we use the SpaceMac scheme here as well. The evidence tag is then computed as follows: $t_e^2 = \text{Mac}_{K_{j_2}^{\prime A}}(e)$. Finally, A uploads to B the packet p' (replacing p in the previous section): $p' = \{j_2 \parallel t_e^2 \parallel j_1 \parallel t_e^1 \parallel e\}$, where j_1 and t_e^1 are computed as described in the last section.

Downloading Blocks. As usual, when downloading p' , B drops p' if K_{j_1} is no longer a safe key. In addition, B checks if j_2 is chosen such that it has time to report e to S . In particular, let T'_A denote the upper bound of the time of A when B receives p' . The upper bound of the time of A when the report reaches S is $T'_A + \delta_{BS}$. Thus, B checks if $j_2 \leq \lfloor \frac{T'_A + \delta_{BS} - T_0}{d} \rfloor$. If it is,

²We refer the reader to Chapter 4.9 in [21] for more details on how to achieve secure communication using encrypt-then-authenticate schemes.

by the time the report reaches S , key $K_{j_2}^A$ has been released by A . This means that B does not have enough time to report e . In this case, B drops e . B also drops e if $j_2 > N$ (out of range). Otherwise, B buffers e and waits for K_{j_1} to check the integrity of e .

Upon receiving K_{j_1} , B verifies the integrity of K_{j_1} as usual. Then, how B handles e depends on whether e is corrupted and whether A is suspicious:

- If A is in the normal state and e is not corrupted, B classifies e as legitimate.
- If A is in the normal state and e is corrupted, B drops e and also puts A into the suspicious state.
- If A is in the suspicious state and e is not corrupted, B buffers p' and waits for $K_{j_2}^A$ to verify the evidence tag. Upon receiving $K_{j_2}^A$, B uses the commitment, K_0^A , to verify the integrity of the key, *i.e.*, checking if $F^{j_2}(K_{j_2}^A) = K_0^A$. If not, B drops the key; otherwise, B uses this key to derive the MAC key, $K_{j_2}^A = F'(K_{j_2}^A)$. B then performs the Verify algorithm to check the integrity of the evidence tag, *i.e.*, checking if $\text{Verify}_{K_{j_2}^A}(e, t_e^2) = 1$. If the evidence tag is invalid, B drops e ; otherwise, B classifies e as legitimate.
- If A is in the suspicious state and e is corrupted, B sends to S a report r : $r = \{A \parallel j_2 \parallel t_e^2 \parallel e\}$.

Identifying the Attacker. Assume that A is the attacker and B detects the attack and reports r to S . Upon receiving r , S first checks if A has not yet released $K_{j_2}^A$. In particular, let T_A'' denote the upper bound of the time of A when S receives r , then S checks if $j_2 \leq \lfloor \frac{T_A'' - T_0}{d} \rfloor$. If it is, then $K_{j_2}^A$ is no longer a safe key, thus S ignores the report. Otherwise, S proceeds by checking if e is corrupted. In particular, let $\alpha_1, \dots, \alpha_m$ denote the global coding coefficients of e ; S then checks whether $e = \sum_{i=1}^m \alpha_i \mathbf{b}_i$. If so, e is not corrupted, and S ignores the report. Otherwise, S proceeds by checking if the evidence tag is valid, *i.e.*, by checking whether $\text{Verify}_{K_{j_2}^A}(e, t_e^2) = 1$. If the evidence tag fails the verification, S ignores the report; otherwise, S concludes that A is the attacker. S then broadcasts this result by sending to each peer P the packet $\{A\}$ through an authenticated channel achieved with k_1^P . Each peer P then puts A into its blacklist and stops any transmission with A .

Remark. In the absence of adversaries, blocks are used right after the integrity check. Thus, the identification scheme does not affect the network throughput in a benign condition.

C. Security Analysis

Assume that A is an attacker who uploads corrupted blocks to B , and that B detects the attack and sends a report to S . A may strategically pick a small value of j_2 such that by the time the report reaches S , S deems that the report invalid because the shared secret key between A and S used to generate the evidence tag of the corrupted block in the report is no longer safe. Since B drops any block that have such value of j_2 , this means all corrupted blocks sent by A with such values of j_2 will be dropped even if they bypass the detection scheme.

The only other option for A to void the report of B is to send along a bogus tag of the corrupted block. This, however, also results in B dropping corrupted blocks from A even if they bypass the detection scheme. Consequently, it is of A best interest to pick valid values for j_2 as well as upload valid evidence tags. If so then A has no way to avoid the identification by S when its attack is detected. The probability of successful identification depends on the probability of successful detection of two corrupted blocks within the suspicious window, which is described in the following Lemma.

Lemma 2. Assume that an attacker picks valid j_2 values and sends valid evidence tags. Let $h > 1$ be the number of corrupted blocks the attacker uploaded to a peer within a window of time Δ and ℓ_1 be the number of MAC tags used in the detection scheme. The probability of successful identification of the attacker is $\sum_{i=2}^h \binom{h}{i} (1 - \frac{1}{q^{\ell_1}})^i (\frac{1}{q^{\ell_1}})^{h-i}$.

A malicious peer, B , may also frame a benign peer, A , by reporting to S that A uploaded a corrupted block. In order to frame A , B has to generate a valid evidence tag for a corrupted block under a safe key of A to convince S . By the security guarantee of SpaceMac, this probability is negligible, depending on the number of evidence tags used:

Lemma 3. Let ℓ_2 be the number of MAC tags used in the identification scheme. The probability that an attacker successfully convinces S that a benign peer injects a corrupted block is $\frac{1}{q^{\ell_2}}$.

Colluding malicious peers cannot frame a benign peer because the collusion does not help the malicious peers to learn a safe key of the benign peer. In addition, colluding peers may delay the identification of a malicious peers, *e.g.*, a malicious peer receives a corrupted block of another malicious peer but does not report. Nonetheless, as soon as a malicious peer attempts to pollute a benign peer, the benign peer will detect the attack with high probability and report the malicious peer; thus, the malicious peer will be identified. In this manner, eventually, all malicious peers will be eliminated from the network. Last but not least, evidence tags used in the identification scheme do not propagate in the network, *i.e.*, its valid scope is limited to one peer (unless it is reported to S), and all the evidence tags are immediately verified by the receiving peer (in the suspicious case). As a result, our identification scheme is also not susceptible to tag pollution attacks

VI. PERFORMANCE EVALUATION

A. Communication Overhead

Detection Scheme. The overhead of our detection scheme is dominated by the predistribution of source tags and the online overhead of the tags of each block. The predistribution sends to each peer $\ell_1 N m$ tags per generation. Since each MAC tag is a symbol in \mathbb{F}_q , the overhead of this predistribution is $\ell_1 N m \lceil \log_2 q \rceil$ bits. When uploading a block, a peer sends along ℓ_1 tags, thus the online overhead per block is $\ell_1 \lceil \log_2 q \rceil$ bits.

To be concrete, we use a parameter set from R^2 , a live streaming P2P system [4]: the video quality is 64 KB per second, $q = 2^8$, $n = 2048$, $m = 128$, and a generation lasts

	Detection (μs)	Identification (μs)	Combined (μs)
C/C++ (PC)	201	402	603
Java (PC)	363	726	1,089
Java (Android)	15,084	30,168	45,252

TABLE I
COMPUTATION OVERHEAD OF OUR SCHEMES

4 seconds. We set $\ell_1 = 3$ (security is 2^{-24}) and $d = 1$ second following the setting in [16], where the source discloses a checksum every 32 blocks. We conservatively set $N = 8$, which covers 8 seconds, double the time of a generation. Using this setting, the predistribution overhead is 3 KB per generation ($\sim 1\%$), and the online overhead per block is 3 B ($\sim 0.1\%$). The predistribution overhead thus dominates the online one.

For comparison, our overhead per generation is significantly smaller than that of [6]. In [6], the predistribution of hash values of blocks of a generation requires $m \lceil \log_2 q \rceil$ bits. Since the homomorphic hash in [6] requires a large field size, 1024 bits, the overhead of the predistribution for a generation of size 128 is at least 16 KB. (The relative percentage overhead depends on the generation size in byte.)

Identification Scheme. The overhead of our identification scheme is dominated by the online overhead of the evidence tags of each block. When uploading a block, a peer is required to send along ℓ_2 evidence tags, thus leading to an additional online overhead of $\ell_2 \lceil \log_2 q \rceil$ bits. For example, if we set $q = 2^8$, $\ell_1 = \ell_2 = 3$ and assume the attacker sends 2 corrupted blocks ($h = 2$) within the suspicious window, the probability that the attacker avoids the identification is less than 2^{-23} , and the probability that the attacker successfully frames a benign peer is 2^{-24} . The total online overhead is 6 bytes per block ($\sim 0.3\%$). Compared to the scheme in [15], our scheme achieves much higher security per byte. For instance, in order to achieve similar security as above, the scheme in [15] needs at least 22 bytes per block.

B. Computation Overhead

The major computation overhead of both of our schemes comes from the MAC algorithms performed by each peer on each block. We note that the computational cost of the Mac and Verify algorithms dominate the Combine algorithm. The overhead per block of the detection scheme is mainly from ℓ_1 runs of Verify. Meanwhile, the overhead per block of the identification is mainly from ℓ_2 runs of Mac and also ℓ_2 runs of Verify. Note that the identification algorithm only runs Verify when a corrupted block is detected and not in a benign operating condition.

We implemented the algorithms of SpaceMac in both C/C++ and Java as a library and make it available online [22]. More details about our implementation can be found in [13]. We evaluate the overhead of both of the schemes on both a PC and an Android device, using both the C/C++ and Java implementation. The PC has a 2.8 Ghz quad-core CPU and 32 GB of RAM. The Android device is a Samsung Captivate with 1 Ghz CPU and 512 MB of RAM. We set $q = 2^8$, $n = 2048$, $m = 128$, and $\ell_1 = \ell_2 = 3$ as above for our evaluation. Table VI-B shows the computation overhead, each of which taken as

an average of 10^5 runs, of both of our schemes. The results show that our schemes have very low overhead, in the order of sub-millisecond on a PC and tens of milliseconds even on a resource-constrained Android device.

For comparison, our computation overhead per block is more than two orders of magnitude less than those of the schemes that use homomorphic hashes or signatures. These schemes perform expensive modular exponentiations, which incur hundred of milliseconds delay per block even when run on a modern PC, e.g., 3.4 Ghz Intel Pentium IV [16] or 2 Ghz Intel Core 2 [11].

VII. CONCLUSION

In this paper, we introduce a complete defense mechanism for network coding-based P2P systems. Our defense mechanism simultaneously provides both in-network detection and precise attacker identification for P2P systems. Our detection and identification schemes are collusion resistant as well as tag-pollution resistant. We also demonstrate that our schemes have very low communication and computation overhead, significantly less than other state-of-the-art schemes.

VIII. ACKNOWLEDGMENT

This work was supported by NSF CAREER grant 0747110 and by AFOSR MURI award FA9550-09-1-0643.

REFERENCES

- [1] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The TESLA Broadcast Authentication Protocol," *RSA CryptoBytes*, vol. 5, 2002.
- [2] C. Gkantsidis and P. R. Rodriguez, "Network Coding for Large Scale Content Distribution," in *INFOCOM'05*.
- [3] Z. Liu, C. Wu, B. Li, and S. Zhao, "UUSee : Large-Scale Operational On-Demand Streaming with Random Network Coding," in *INFOCOM'10*.
- [4] M. Wang and B. Li, "R²: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *JSAC*, vol. 25, no. 9, Dec. 2007.
- [5] B. Li and D. Niu, "Random Network Coding in Peer-to-Peer Networks : From Theory to Practice," *IEEE*, vol. 99, no. 3, 2011.
- [6] C. Gkantsidis and P. R. Rodriguez, "Cooperative Security for Network Coding File Distribution," in *INFOCOM'06*.
- [7] M. N. Krohn, M. J. Freedman, and D. Mazieres, "On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution," in *SP'04*.
- [8] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a Linear Subspace : Signature Schemes for Network Coding," in *PKC'09*.
- [9] F. Zhao, T. Kalkert, M. Medard, and K. J. Han, "Content Distribution with Network Coding," in *ISIT'07*.
- [10] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *Info Sciences and Systems*, vol. 1, no. 1, 2006.
- [11] P. Zhang, Y. Jiang, C. Lin, H. Yao, A. Wasef, and X. S. Shen, "Padding for Orthogonality : Efficient Subspace Authentication for Network Coding," in *INFOCOM'11*.
- [12] S. Agrawal and D. Boneh, "Homomorphic MACs : MAC-Based Integrity for Network Coding," in *ACNS'09*.
- [13] 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>
- [14] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "RIPPLE Authentication for Network Coding," in *INFOCOM'10*.
- [15] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "Identifying Malicious Nodes in Network-Coding- Based Peer-to-Peer Streaming Networks," in *Mini INFOCOM'10*.
- [16] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical Defenses Against Pollution Attacks in Intra-Flow Network Coding for Wireless Mesh Networks," in *ACM WiSec'09*.
- [17] E. Kehdi and B. Li, "Null Keys : Limiting Malicious Attacks Via Null Space Properties of Network Coding," in *INFOCOM'09*.
- [18] M. Jafarisiavoshani, C. Fragouli, and S. Diggavi, "On Locating Byzantine Attackers," in *NetCod'08*.
- [19] A. Le and A. Markopoulou, "Locating Byzantine Attackers in Intra-Session Network Coding using SpaceMac," in *NetCod'10*.
- [20] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and Secure Source Authentication for Multicast," in *NDSS'01*.
- [21] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC Press, 2007.
- [22] A. Le, "SpaceMac Library," 2011. [Online]. Available: <http://www.ics.uci.edu/~anhml/software.html#SpaceMac>