RIPPLE Authentication for Network Coding

Yaping Li[1], Hongyi Yao[2], Minghua Chen[1], Sidharth Jaggi[1], and Alon Rosen[3]
[1] The Chinese University of Hong Kong [2] Tsinghua University [3] Herzliya Interdisciplinary Center, Israel

***Abstract*—By allowing routers to randomly mix the information content in packets before forwarding them, network coding can maximize network throughput in a distributed manner with low complexity. However, such mixing also renders the transmission vulnerable to *pollution attacks*, where a malicious node injects corrupted packets into the information flow. In a worst case scenario, a single corrupted packet can end up corrupting *all* the information reaching a destination. In this paper, we propose RIPPLE, a symmetric key based in-network scheme for network coding authentication. RIPPLE allows a node to efficiently detect corrupted packets and encode only the authenticated ones. Despite using symmetric key based homomorphic Message Authentication Code (MAC) algorithms, RIPPLE achieves asymmetry by delayed disclosure of the MAC keys. Our work is the first symmetric key based solution to allow arbitrary collusion among adversaries. It is also the first to consider *tag pollution attacks*, where a single corrupted MAC tag can cause numerous packets to fail authentication farther down the stream, effectively emulating a successful pollution attack.**

I. Introduction

Network coding allows the routers to mix the information content in packets before forwarding them. This mixing has been theoretically proven to maximize network throughput [1]–[4]. It can be done in a distributed manner with low complexity, and is robust to packet losses and network failures [5], [6]. Furthermore, recent implementations of network coding for wired and wireless environments demonstrate its practical benefits [7], [8].

But what if the network contains malicious nodes? A malicious node may mount a *pollution attack* by pretending to forward packets from source to destination, while in reality it injects corrupted packets into the information flow. Since network coding makes the routers mix packets' content, a single corrupted packet can end up corrupting *all* the information reaching a destination. Unless this problem is solved, in the presence of adversaries network coding schemes may perform much worse than pure forwarding schemes.

Source authentication prevents the pollution attack by allowing a node to ensure that the received data originates from the source and was not modified en-route. Previous research on providing source authentication for network coding falls into two broad categories: end-to-end and in-network schemes. An end-to-end approach makes minimal changes to existing network coding algorithms; only the source and sink are involved in performing computations to enable detection and/or correction of errors introduced by malicious nodes [9]. However, these schemes are geared towards a worst-case view of adversarial action; namely, that the adversaries locate themselves at the weakest part of the network (the bottlenecks). Hence the achievable rate guarantees of such schemes can be unduly pessimistic when the adversary does not have a free hand at choosing which parts of the network to attack.

The in-network solutions apply cryptographic primitives to enable routers to detect and drop corrupted packets. The rates they achieve scale gracefully with the adversaries' actual attacks, rather than their worst-case attacks. The in-network approach can be further divided into public key based and symmetric key based solutions. Researchers have proposed various public key signature schemes [10]–[15]. These schemes are elegant, but too slow for online traffic.

Two recent research endeavors use symmetric key cryptography to reduce computational complexity in network coding source authentication. Yu et al. [16] exploit symmetric key encryption and probabilistic key pre-distribution to defend against pollution attacks. Agrawal and Boneh [17] design a homomorphic Message Authentication Code (MAC) [1] system that allows in-network verification of the authenticity of network coded data. Both schemes are limited in being only c-collusion resistant for some pre-determined c; they become vulnerable when more than c malicious nodes collude. Moreover, both systems are susceptible to a subtle *tag pollution attack*, where an attacker tampers with MAC tags. In [16] and [17], a message carries multiple tags, and a node only verifies a subset of them. An attacker can modify a tag that will only be verified farther down the stream. This single corrupted tag can cause numerous packets to fail authentication, effectively emulating a successful pollution attack.

In this paper, we propose RIPPLE, a symmetric key based in-network solution for network coding authentication. RIPPLE differs from the above two schemes in two main respects: (a) tolerating arbitrary collusion among adversaries, and (b) being tag pollution resistant. We make two main contributions. First, we construct a symmetric key based homomorphic MAC that enables routers to both verify the authenticity of a packet, and compute new MACs for encoded packets. Second, we propose a transmission protocol to allow source authentication in network coding. To ensure that data comes from a claimed sender, asymmetry is essential. Inspired by TESLA [18], RIPPLE achieves asymmetry by delayed disclosure of the MAC keys, and thus provides source authentication of the received data. We name our network coding authentication scheme RIPPLE for packets moving in the network from level to level, in a wavelike fashion. A wave of packets reaches the nodes at a level, pauses for key disclosure, verification, and coding, and then flows to the next level.

[1] A MAC algorithm, sometimes called a keyed (cryptographic) hash function, accepts as input a secret key and an arbitrary-length message to be authenticated, and outputs a MAC, sometimes known as a tag.

A. Related Work

Work on network coding started with a pioneering paper by Ahlswede et al. [1], which establishes the value of coding in the routers and provides theoretical bounds on the capacity of such networks. The combination of [2], [3], and [4] shows that, for multicast traffic, linear codes achieve the maximum capacity bounds, and both design and implementation can be done in polynomial time. Additionally, Ho et al. show that the above is true even when the routers perform random linear operations [5]. Researchers have extended the above results to a variety of areas including wireless networks [7], [19], secrecy [20], content distribution [8], and distributed storage [21]. See [22] for a nice survey on network coding.

A Byzantine attacker is a malicious adversary hidden in a network, capable of eavesdropping and jamming communications. Prior research has examined such attacks in the presence of network coding and without it. In the *absence* of network coding, Dolev et al. [23] consider the problem of communicating over a known graph containing Byzantine adversaries. They show that for k adversarial nodes, reliable communication is possible only if the graph has more than $2k+1$ vertex connectivity. Subramaniam extends this result to unknown graphs [24]. Pelc et al. address the same problem in wireless networks by modeling malicious nodes as locally bounded Byzantine faults, i.e., nodes can overhear and jam packets only in their neighborhood [25].

Existing work in defending against such pollution attacks on network coding falls into two broad categories: end-to-end, and in-network approaches.

End-to-end schemes: The end-to-end schemes in the literature allow internal nodes to mix *all* incoming packets, without verifying the veracity of their contents, to generate outgoing packets. The motivation is two-fold. For one, a paradigm where internal nodes perform only very simple operations meshes well with the distributed, low-complexity nature of random linear network coding algorithms in the literature [5] – all the complexity is pushed to the source's encoder and the receiver's decoder. For another, the worst-case throughput performance of such schemes (when the adversary can choose "the weakest links" in the network to attack) can be shown to be equivalent to that achievable by more sophisticated schemes in which interior nodes can perform *arbitrary* operations.

The work in [26], detects the existence of an adversary but does not provide an error-correction scheme. This work demonstrates that as long as there is even one pollution-free path from the sender to the receiver, a pollution attack can be detected. Such performance is attained by requiring that each source packet satisfy a (non-linear) hash. With high probability any pollution injected by the adversary results in the receiver decoding packets that do not satisfy such a hash, and therefore the attack can be detected.

The work of Cai and Yeung [20], [27], [28] generalizes standard bounds on error-correcting codes to networks, and demonstrates the existence of codes that achieves these bounds. However, no tractable algorithms to design and implement that achieve these bounds are presented in [20], [27], [28].

The work of [29] and [9] presents the first efficient schemes to design and implement error-correction against Byzantine adversaries in the distributed network coding setting. They concentrate on communicating in the presence of a wiretapping and pollution-injecting adversary, and present distributed schemes with an information-theoretically optimal rate. In a nutshell, [29] reduce the model of network coding to a certain point-to-point channel. They then construct generalizations of Reed-Solomon codes for this channel, which enables the authors to construct deterministic network error-correcting codes as mentioned above. The work of [9] is in a similar vein, but looks at the performance of random linear network codes against Byzantine adversaries. It also considers the interplay between eavesdropping and error-injection – it turns out that if the adversary is limited in its eavesdropping power, significantly higher throughputs can be achieved even against pollution attacks.

In-network schemes: However, in realistic scenarios the adversary often cannot choose which parts of the network to attack. In these scenarios the rates achievable via end-to-end schemes are pessimistic. If one requires honest internal nodes to verify incoming packets via cryptographic primitives, one can achieve higher rates in such situations than would be achievable via end-to-end schemes.

For such in-network cryptographic schemes, researchers propose various signature schemes [10]–[14] in the context of integrity verifications of network coding. These public key signature schemes are elegant, but too slow for online traffic.

Krohn et al. [30] first propose using collision-resistant homomorphic hashing in the context of verification of rateless erasure codes. Gkantsidis et al. [10] apply such homomorphic hash functions for integrity verification of network coded data. Since these homomorphic hash functions are computationally expensive, batch verification schemes are used to improve efficiency. To further reduce the cryptographic computation, Gkantsidis et al. propose a security scheme where honest nodes probabilistically check received packets and cooperatively detect and alert each other of malicious activities. However, this scheme allows some bogus packets to propagate.

Zhao et al. [12] suggest an authentication scheme that breaks a file into a number of blocks viewed as vectors spanning a subspace V. The sender computes an authenticator for V based on a vector orthogonal to V and signs it with a common public key signature scheme. A node can verify the integrity of an encoded message (viewed as a vector v) by checking the membership of v in V based on the authenticator. A drawback of the schemes in [10], [12], [30] is their lacking of data streaming support since the sender needs to know the entire file before generating the authentication information.

Boneh et al. [14] propose two signature schemes for network coding authentication. The first homomorphic signature scheme differs from that of Zhao et al. [12] in associating signatures with individual vectors instead of the entire subspace. This scheme supports data steaming and has a constant public-

key size and per-packet overhead. The second scheme modifies that of Krohn et al. [30] to prevent vectors from different files from being combined.

Yu et al. [16] exploit probabilistic key pre-distribution and symmetric key encryption to defend against pollution attacks. A probabilistic key pre-distribution protocol assigns each node a random subset of secret keys from a global key pool $\mathcal{K}$ such that any two nodes have a certain probability to share a key. For each message, the sender generates t authenticators each with a different secret key. An authenticator only verifies some part of a message and a node can check its validity if it knows the encryption key. Multiple downstream nodes can thus collaboratively verify different parts of an encoded message and potentially detect a bogus message within a few hops with high probability. The scheme does not detect a bogus packet at every first honest node thus allows it to propagate to pollute more packets. Another drawback is that an encoded message has a communication overhead x times higher than that of a source message if it is computed from x source messages [16].

Agrawal and Boneh [17] design a homomorphic MAC system which allows checking the integrity of network coded data. The construction evolves in three stages. The first stage constructs a homomorphic MAC which allows end-to-end detection of bogus packets. The second stage converts the previous homomorphic MAC into a broadcast homomorphic MAC which allows in-network integrity verification. This construction is limited in being only c-collusion resistant for some pre-determined c. The system becomes vulnerable when more than c malicious nodes collude. The third stage constructs an integrity system for multiple senders and receivers and we are concerned only with the single sender setting in this paper.

Most related to our proposal is the work of [15], in which Dong et al. propose DART. DART is a time-based authentication in combination with random linear transformations to defend against pollution attacks. Inspired by TESLA [31], both DART and our work leverage time in source authentication. However, RIPPLE differs from DART in two main respects. First, RIPPLE uses symmetric key cryptography (except for the two public key operations per node per multicast session). In contrast, DART is essentially public key cryptography. DART requires that every node performs one public key verification per generation. Frequent public key verification is expensive and exposes DART to denial of service attacks, where an attacker floods a node with signature packets for verification. Second, RIPPLE supports multicast and DART is implicitly unicast only. A source starts sending packets for the next generation only after it receives an acknowledgement from the receiver which has received all packets in a generation. It is not clear how to extend DART to efficiently support multicast with the acknowledgement mechanism.

II. Problem Statement

A. System Setting

Consider a network modeled as a directed acyclic graph $G = (V, E)$. A source $\mathcal{S}$ multicasts a stream of messages to a set of receivers. All nodes in $V - \{\mathcal{S}\}$, i.e., all receivers and forwarders, perform random linear network coding.

In this paper, we consider a well adopted random linear network coding scheme based on generations. $\mathcal{S}$ partitions a stream of messages into generations, each of m messages. For clarity, we focus on the coding and transmission of a single generation. Only messages from the same generation are encoded. A message $\mathbf{x} \in \mathbb{F}_q^n$ is a vector of n symbols, each an element of the finite field $\mathbb{F}_q$. Following the treatment in [3], all arithmetic operations henceforth are done over $\mathbb{F}_q$. $\mathcal{S}$ starts a multicast session by transmitting messages to its neighbors. As the messages propagate through the network, a node generates coded messages as random linear combinations of the received messages and transmits the coded messages to its neighbors. Specifically, consider a node v with w incoming links. For incoming link i and outgoing link j, V chooses a coding coefficient $\alpha_{ij} \in \mathbb{F}_q$ uniformly at random. Let $\mathbf{x}_i$ denote a message received from link i. v generates a coded message $\mathbf{y}_j$ leaving for link j as

$$\mathbf{y}_j = \sum_{i=1}^{w} \alpha_{ij} \mathbf{x}_i. \tag{1}$$

A receiver can recover the original messages from any m random linear combinations that form a full rank matrix.

In order for a receiver to decode, a message $\mathbf{x}$ must carry the *global coding coefficients* that result in $\mathbf{x}$ as the random linear combination of the original messages. $\mathcal{S}$ thus expands each original message $\bar{M}_i \in \mathbb{F}_q^n$ $(1 \le i \le m)$ by m symbols as

$$M_i = (\bar{M}_i, \underbrace{0, \ldots, 0, 1}_{i}, \overbrace{0, \ldots, 0}^{m}) \in \mathbb{F}_q^{n+m} \tag{2}$$

with a single 1 in the i^{th} position. The augmented M_i's are coded as they traverse the network. As a result, if a message $\mathbf{x}$ is a linear combination of the original messages, i.e.,

$$\mathbf{x} = \sum_{i=1}^{m} c_i M_i \left(c_i \in \mathbb{F}_q\right), \tag{3}$$

the last m symbols of $\mathbf{x}$ are the global coding coefficients c_i.

The network coding scheme described above implicitly assumes that messages of the same generation, traveling along different paths from the source to a node, arrive at the same time. However, in typical scenarios where multiple paths from the source to a node introduce heterogeneous delays, messages of the same generation arrive asynchronously. In such situations, a node buffers for a long enough time (e.g., more than the longest network propagation delay) for all outstanding messages of the same generation to arrive before performing network coding. Due to this buffering, the end-to-end decoding delay may increase, and nodes may need large buffers to perform network coding. Nevertheless, the scheme still achieves the optimal multicast throughput asymptotically in the number of generations [32]. The work of [33] demonstrates the stability of the rates achievable and buffer-lengths even in lossy networks operated asynchronously.

For example, consider a skewed Butterfly network in Fig. 1(b), where every link has unit capacity and unit propagation delay. The source $\mathcal{S}$ multicasts messages $\{u_i, v_i\}\,(i \geq 1)$ to receivers f and g. The first generation of packets u_1 and v_1 reaches node d at time 2 and 3, respectively. Node d buffers u_1 that arrives at time 2, and skips the opportunity to transmit on link $d \rightarrow e$ at time 2. It sends out $u_1 + v_1$ when v_1 arrives at time 3. u_1 (resp. v_1) reaches f (resp. g) at time 2 through route $\mathcal{S} \rightarrow a \rightarrow f$ (resp. $\mathcal{S} \rightarrow b \rightarrow g$). When $u_1 + v_1$ reaches f (resp. g) at time 5, f (resp. g) can decode the first generation of messages. Although node d wastes one transmission opportunity at time 2, the arrivals of the streaming generations of packets allow it to utilize every transmission opportunity afterwards. Consequently, f and g can decode the second generation of messages at time 6, the third generation of messages at time 7, and so on. The throughput f and g obtain is still the optimal value 2. Thus, amortized over the number of packets across generations, the loss in transmission efficiency due to pipelining initialization can be made negligible.

B. Threat Model

We consider adversaries that control an arbitrary subset of the nodes in the network. The adversaries may attempt to inject corrupted packets into the information flow, aiming to corrupt the information on its way to the destination. They may also try to modify the packets going through the nodes it controls, and to fiddle with the authentication tags that are appended to these packets.[2] An adversary is considered to have successfully modified a packet if it has managed to produce an authentic (vector, tag) pair for which the vector component does not lie in the linear subspace corresponding to the delivered messages (see Section IV for details).

We assume that the adversary does not have access to the randomness used by the source in order to produce the various cryptographic keys, and that its running time is polynomial in the security parameter (which may in some cases be polylogarithmic in the size of the underlying finite field $\mathbb{F}_q$ – see Section IV-A for discussion).

Our work is the first symmetric key based scheme to allow collusion among an arbitrary number of corrupted nodes. It is also the first to consider adversaries that modify tags on their way to verification. Indeed, one of our main objectives is to prevent a "tag pollution" attack, in which an adversary injects errors in tags that are verified far down the information flow. Tag pollution attacks are mostly relevant in symmetric key based authentication schemes. In public key based solutions, a message is authenticated with a single authenticator. If an attacker modifies either the message or the authenticator, the authentication will fail immediately and the packet is discarded. However, this is not the case in the two previously proposed symmetric key based schemes [16] and [17]. In these two schemes, a message carries multiple tags, and each node only verifies a subset of them. As a result, it is possible for an attacker to tamper with a tag that will only be verified farther down the information flow. The consequences of such attacks may be devastating, not only because they evade early detection (since the polluted tags are verified only at a much later phase), but also because errors in as little as a single tag may snowball into errors in many tags (thus resulting in numerous packets that fail authentication and are subsequently dropped, effectively emulating a successful pollution attack).

[2]Allowing adversaries to fiddle with tags is a new consideration that has not been previously addressed. It is most relevant in the symmetric-key cryptography realm.

III. Design Goals and Approach

Under our settings, the goals of in-network authentication of network coding are as follows:

- *In-network source authentication.* Any node in the network can verify that the received data originates from the source and was not modified en-route.
- *Arbitrary collusion resistant.* An honest node can verify the authenticity of the received data even in the presence of an arbitrary number of colluding adversaries.
- *Light-weight operation.* Forwarders need little extra power to process message authentication.

With these goals in mind, we present our RIPPLE scheme for network coding authentication in the next two sections. The main set of ideas behind our scheme are: 1) designing homomorphic MACs to allow in-network tag regeneration for coded messages, 2) utilizing symmetric key cryptography for light-weight in-network message authentication, and 3) using time to create asymmetry in message authentication; ideas of this kind were first proposed in [18].

RIPPLE has two main components. The first is a homomorphic MAC for broadcast authentication. The second is the RIPPLE transmission protocol for message distribution and authentication. Our homomorphic MAC is information theoretically secure, whereas the security of the RIPPLE transmission protocol relies on computational hardness.

IV. Homomorphic MAC

A homomorphic MAC [17]) allows to linearly combine any sequence of given (vector, tag) pairs. That is, given a sequence of pairs $\{(M_i, t_i)\}_{i=1}^{m}$, where $M_1, M_2, \ldots, M_m \in \mathbb{F}_q^{n+m}$, anyone can create a valid authentication tag t for the vector $\mathbf{y} = \sum_{i=1}^{m} \alpha_i M_i$ for any $\alpha_1, \alpha_2, \ldots, \alpha_m \in \mathbb{F}_q$. We assume that a message $M_i \in \mathbb{F}_q^{n+m}$ is generated as in Equation 2. To define security, we consider the strongest type of attacks, an *adaptive chosen message attack*, where an adversary first learns messages and the corresponding tags of its choice. Loosely speaking, the security requirement is that, even under an adaptive chosen message attack, creating a valid tag t for a vector outside the linear span of the original messages is only possible with negligible probability.

Syntax. We define a (q, n, m) homomorphic MAC via four probabilistic, polynomial-time (PPT) algorithms, (**Generate**, **MAC**, **Verify**, **Combine**):

- **Generate** is a PPT algorithm that randomly samples a key K from a key space $\mathcal{K}$.

- **MAC** is a PPT algorithm that takes as input a secret key K, a vector M and outputs a tag t for M.
  The **MAC** algorithm authenticates a vector space spanned by $M_1, M_2, \ldots, M_m \in \mathbb{F}_q^{n+m}$ by running $\mathbf{MAC}(K, M_i)$ for $i = 1, \ldots, m$. This produces a tag for each of the basis vectors M_i. To authenticate a message (whose packets form the subspace spanned by $M_1, M_2, \ldots, M_m$) the sender transmits the pairs (M_i, t_i).
- **Verify** is a PPT algorithm that takes as input a triplet (K, M, t), where K is a secret key, $M \in \mathbb{F}_q^{n+m}$, and t is a tag, and outputs either 1 (accept) or 0 (reject).
- **Combine** is a PPT algorithm that takes a sequence of triplets $(M_1, t_1, \alpha_1), (M_2, t_2, \alpha_2), \ldots, (M_w, t_w, \alpha_w)$, where $w \leq m$ and $M_i \in \mathbb{F}_q^{n+m}$ and $\alpha_i \in \mathbb{F}_q$, and outputs a tag t for the vector $\mathbf{y} = \sum_{i=1}^{w} \alpha_i M_i \in \mathbb{F}_q^{n+m}$.

We require that for any $M_1, M_2, \ldots, M_w \in \mathbb{F}_q^{n+m}$ and any $\alpha_1, \ldots, \alpha_d \in \mathbb{F}_q$, it holds that

$$\mathbf{Verify}\left(K, \sum_{i=1}^{w} \alpha_i M_i, \mathbf{Combine}((M_i, t_i, \alpha_i)_{i=1}^{w})\right) = 1$$

where $t_i = \mathbf{MAC}(K, M_i)$.

Security. To define security of the homomorphic MAC, we consider an adaptive chosen message attack. The adversary should be unable to produce a pair (M', t) that passes verification, and so that M' does not lie within the subspace spanned by the messages $M_1, \ldots, M_m$.

A bit more formally, security is defined through the following game between a challenger C and an adversary A. First, C generates a random key K. Then A adaptively submits MAC queries of the form $M_1, \ldots, M_m$. To respond to a query, C computes $t_i = \mathbf{MAC}(K, M_i)$ and sends $(t_1, \ldots, t_m)$ to A. Finally, A outputs a vector $M' \in \mathbb{F}_q^{n+m}$ and a tag t. The adversary A is said to have forged if: (1) $\mathbf{Verify}(K, M', t) = 1$, and (2) $M' \notin \mathrm{Span}(M_1, \ldots, M_m)$.

Definition 1: A (q, n, m) homomorphic MAC scheme is said to be secure if for all PPT adversary A the probability that A forges is upper bounded by $1/q$.

Basic construction. We start with a basic construction of a homomorphic MAC. Denote by

$$\langle M, K \rangle \in \mathbb{F}_q$$

the inner product operation between two vectors M and K over $\mathbb{F}_q$ of equal length. Our basic (q, n, m) homomorphic MAC scheme, which we call the *inner-product* MAC, is constructed as follows:

- **Generate:** Sample $K \xleftarrow{R} \mathbb{F}_q^{n+m}$.
- **MAC:** Given $M, K \in \mathbb{F}_q^{n+m}$, output $t = \langle M, K \rangle \in \mathbb{F}_q$.
- **Verify:** Given (K, M, t), check that $\langle M, K \rangle = t$.
- **Combine:** Given $(M_i, t_i, \alpha_i)_{i=1}^{w}$ with $w \leq m$, output $\sum_{i=1}^{w} \alpha_i t_i$.

Theorem 2: The inner-product scheme is a secure homomorphic MAC.

Proof: **Homomorphism:** Define $t = \sum_{i=1}^{w} \alpha_i t_i$. We then have:

$$t = \sum_{i=1}^{w} \alpha_i \langle M_i, K \rangle = \langle \sum_{i=1}^{w} \alpha_i M_i, K \rangle.$$

Security. Assume the adversary knows $(M_i, t_i)_{i=1}^{m}$ by adaptively choosing $(M_i)_{i=1}^{m}$. Let (M', t) be the adversary's output, which presumably satisfies $M' \notin \mathrm{Span}(M_1, M_2, \ldots, M_m)$.

It suffices to prove that for any $t \in \mathbb{F}_q$ there exists a set of equal size, such that any K in the set satisfies $\langle M', K \rangle = t$.

For any $t \in \mathbb{F}_q$, a possible K is a solution of :

$$(M_1; M_2; \ldots; M_m; M')K = (t_1; t_2; \ldots; t_m; t).$$

By applying basic linear algebra result we get that if the first m equations have a solution, and M' is not in the linear subspace spanned by $\{M_i\}_{i=1}^{m}$, then for any t, the solutions set has cardinality $q^{(n-r_k)}$ where r_k is the rank of $(M_1; M_2; \ldots; M_m; M')$. ■

An appealing feature of the inner-product MAC is that its tags are elements in $\mathbb{F}_q$. Thus, concatenating a vector M with its authentication tag t actually results in a longer vector (M, t), which can be itself authenticated by taking its inner product with a longer key. This process can be continued inductively, effectively producing a "chain" of nested authentications, each intended to a different level in the network. Conveniently, this "nesting" operation preserves both the security and the homomorphic properties of the original authentication scheme.

Full construction. We now present the homomorphic MAC scheme that is used in RIPPLE. This scheme takes advantage of the ability to nest authentications in order to prevent tag pollution attacks.

- **Generate:** Sample a sequence $K = (K^1, K^2, \ldots, K^L)$, where $K^j \xleftarrow{R} \mathbb{F}_q^{n+L-j}$.
- **MAC:** Given a message $M \in \mathbb{F}_q^{n+m}$ and keys $K^1, K^2, \ldots, K^L$ as above, output the following L tags:

$$\begin{aligned} t^L &= \langle M, K^L \rangle \\ t^{L-1} &= \langle (M, t^L), K^{L-1} \rangle \\ &\vdots \\ t^1 &= \langle (M, t^L, t^{L-1}, \ldots, t^2), K^1 \rangle \end{aligned}$$

- **Verify:** Given $P = (M, t^L, t^{L-1}, \ldots, t^1) \in \mathbb{F}_q^{n+m+L}$ and K^j, check that $t^j = \langle (M, t^L, t^{L-1}, \ldots, t^{j+1}), K^j \rangle$ for every $j = 1, \ldots, L$.
- **Combine**: Given $(M_i, t_i^L, t_i^{L-1}, \ldots, t_i^1, \alpha_i)_{i=1}^{w}$ with $w \leq m$, output a tag $t = t^L, t^{L-1}, \ldots, t^1$ where $t^j = \sum_{i=1}^{w} \alpha_i t_i^j$.

We next argue that the scheme described above is secure.

Theorem 3: . The nested inner-product scheme is a secure homomorphic MAC.

Proof: **Homomorphism:** Given $(M_i, t_i^1, \ldots, t_i^L, \alpha_i)_{i=1}^{w}$, consider a tag $t = t^1, \ldots, t^L$ where $t^j = \sum_{i=1}^{w} \alpha_i t_i^j$.

Firstly, note that $t^L = \sum_{i=1}^{w} \alpha_i \langle M_i, K^L \rangle = \langle \sum_{i=1}^{w} \alpha_i M_i, K^L \rangle$ passes the verification. For any $j = L-1, L-2, \ldots, 1$, we then have:

$$\begin{aligned} t^j &= \sum_{i=1}^{w} \alpha_i t_i^j \\ &= \sum_{i=1}^{w} \alpha_i \langle (M_i, t_i^L, t_i^{L-1}, \ldots, t_i^{j+1}), K^j \rangle \\ &= \langle \sum_{i=1}^{w} \alpha_i (M_i, t_i^L, t_i^{L-1}, \ldots, t_i^{j+1}), K^j \rangle \\ &= \langle (M, t^L, t^{L-1}, \ldots, t^{j+1}), K^j \rangle \end{aligned}$$

And so t^j passes the verification.

Security. Suppose that an adversary outputs $M' \notin \text{Span}(M_1, M_2, \ldots, M_m)$. By Theorem 2, this means that the adversary can forge t^j for any $j = 1, 2, \ldots, L$ with probability at most $1/q$.

For any $j = L, L-1, \ldots, 2$, if $(M, t^L, t^{L-1}, \ldots, t^j)$ is not in the space spanned by $\{(M_i, t_i^L, t_i^{L-1}, \ldots, t_i^j)\}_{i=1}^m$, then by Theorem 2, an adversary can forge t^k for any $k = 1, 2, \ldots, j-1$ with probability at most $1/q$. ∎

Arbitrary Collusion Resistant. Our RIPPLE system is secure against arbitrary collusion among the adversaries. Assume that the adversary knows all the *legal packets*, i.e., the linear subspace spanned by $\{P_i = (M_i, t_i^1, \ldots, t_i^L)\}_{i=1}^m$ are observed. Any adversarial behaviors, i.e., nonlinear operations, on M or t^j can pass the verification of t^k with probability at most $1/q$ for any $j > k$.

A. Security beyond $1/q$

As proved in Theorem 3, an adversary can forge with probability at most $1/q$. For small q (e.g. 2), this may not yield satisfactory security. To ameliorate this state of affairs, we consider two possible approaches to strengthen the security of the MAC system:

1) **Using a larger field**. Let q be the size of the original field and $q' = q^c$ be the size of a larger field used to achieve a reliable security parameter $1/q' = 1/q^c$. Such a larger field would result in tag communication overhead c times of the original one, while the computational complexity of field multiplication over $\mathbb{F}_{q'}$ is $c \log c$ times of that over $\mathbb{F}_q$.
2) **Using Multiple Tags**. For each level $j \in [1, L]$ we use c tags $\{t_1^j, t_2^j, \ldots, t_c^j\}$ generated by c independent keys[3] $\{K_1^j, K_2^j, \ldots, K_c^j\}$. The probability that an adversary can forge c tags is $1/q^c$. Thus the tag communication overhead increases by a factor of c, while the number of multiplication increases by only c times, as opposed to $c \log c$ times in the first approach.

We suggest using the second approach to improve security for its lower computational complexity. However, for ease of understanding, we will use a single tag per level in the description of the RIPPLE transmission protocol next.

[3]Note that t_a^j is not used to authenticate t_b^j for any $a, b \in [1, c]$. Thus the length of K_a^j is $n + (L-j)c$ for any $j \in [1, L]$ and $a \in [1, c]$.

V. RIPPLE Transmission Protocol

In our problem setting, a sender $\mathcal{S}$ schedules packet transmission based on two coordinates, space and time. The network space is hierarchically organized. A node is called a level-j node if it is at most j hops away from $\mathcal{S}$. We define the children of $\mathcal{S}$ to be level-1 nodes and assume a maximum of L levels in a network. Time is divided into uniform intervals. $\mathcal{S}$ sends zero or multiple packets during an interval. We refer to packets sent during interval i as interval-i packets.

We name our network coding authentication scheme RIPPLE for packets moving in the network from level to level, in a wavelike fashion. A batch of packets reaches the nodes at a level, pauses for key disclosure, verification, coding, and finally flows to the next level. For ease of understanding, we first describe the RIPPLE scheme at a high level, and then break it into four stages and elaborate on each of them.

$\mathcal{S}$ broadcasts a batch of packets in each time interval and a batch moves from level to level driven by delayed key disclosures. Specifically, assume that $\mathcal{S}$ transmits a batch of packets to its children during interval i. A packet carries L tags, each for verification by nodes at a different level. A tag is produced with a key generated specifically for a particular interval and level. $\mathcal{S}$ reveals the key for interval i and level 1 after a fixed amount of delay to ensure that all level-1 nodes have received the interval-i packets. Upon receiving the key, a level-1 node verifies all the buffered interval-i packets by checking their level-1 tags and encodes only the authenticated packets. Due to the homomorphic property of our MAC, the level-1 nodes are able to produce new tags for the encoded packets for the remaining levels. Driven by one key disclosure, this batch of packets moves to level-2 nodes. Based on a predetermined key disclosure schedule, $\mathcal{S}$ distributes the keys for levels 2 through L in an increasing order of levels. Every key disclosure drives the batch one level forward. Eventually, this batch of packets reaches all the receivers.

A. Sender Setup

We now describe our RIPPLE transmission protocol in four stages. Recall that time is divided into intervals of a uniform length. We use the following notations: T_0 denotes the starting time of a multicast transmission, N the maximum number of intervals in a multicast session, W the maximum number of packets sent in an interval, T_i the starting time of interval i, and δ the interval length. For the last three variables, we have the following equation

$$T_i = T_0 + i \cdot \delta, \quad \forall 1 \le i \le N. \tag{4}$$

In the setup stage, $\mathcal{S}$ determines the network level L and network delay D.

1) Determining Network Level L: For each node $v \in V - \{\mathcal{S}\}$, its level $\mathrm{Lv}(v)$ on a coding graph $G = (V, E)$ is defined as the length (i.e., hop count) of the longest path from $\mathcal{S}$ to v. Network level L is thus defined as

$$L \triangleq \max_{v \in V - \{\mathcal{S}\}} \mathrm{Lv}(v). \tag{5}$$

For example, the level of node d on the butterfly coding graph in Fig. 1(b) is 3, and the network level L is 5.

Given a coding graph[4], finding the length of the longest path from $\mathcal{S}$ to a node $v \in V$ can be transformed to the problem of finding the shortest path between $\mathcal{S}$ and v by changing the signs of the weights on the edges. Hence, by using existing shortest path algorithms such as Dijkstra's algorithm, the length of the longest path can be computed in $O(|V| \log |V| + |E|)$ time when no adversary exists. In the presence of adversaries, the length of the longest path can be computed by using recently proposed passive network tomography schemes [34] for network coding based multicast.

2) Bounding Network Delay D: We define network delay D as the sum of these two terms:

- $\mathrm{RTT}(\mathcal{S}, v)$, the maximum round trip time between $\mathcal{S}$ and a node $v \in V - \{\mathcal{S}\}$, and
- T_p, the bound on a node's processing time to authenticate up to W messages of an interval, encode the authenticated messages, calculate new tags for the encoded messages, and forward the newly generated packets to the next level.

More precisely,

$$D \triangleq \max_{v \in V - \{\mathcal{S}\}} \mathrm{RTT}(\mathcal{S}, v) + T_p. \tag{6}$$

D is thus an upper bound on the maximum time for messages of an interval to move one level forward. We emphasize that RIPPLE only requires $\mathcal{S}$ to know the upper bounds of L and D to operate. However, knowledge of accurate values of L and D reduces both the communication and computational overhead.

3) One-way Key Chain: For each level of nodes, we generate a different sequence of N random values and use them in reverse order to derive the MAC keys each for a different time interval. We use a pseudo-random function [35] F to generate the sequences. For each sequence, we choose a different random number r_0 as the base value and generate the sequence recursively as $r_i = F(r_{i-1})$ where $1 \le i \le N$ (recall that N is the maximal number of intervals in a broadcast session). We refer to the sequence as the *one-way key chain*.

To authenticate the values in a one-way key chain, $\mathcal{S}$ signs the last value r_N with a common public key signature scheme such as RSA [36] or DSA [37]. We refer to the signed r_N a *commitment* to the key chain. Anyone can verify the authenticity of r_N with $\mathcal{S}$'s public key. Given an authenticated r_N and an index i, anyone can check if a value r is the i^{th} value in the one-way key chain by checking if $r_N = F^{N-i}(r)$, where $F^n(x)$ denotes n consecutive applications of F. By convention, $F^0(x) = x$.

[4]Recall that we focus on directed acyclic coding graphs in this paper.

We use the one-way key chain in reverse order to derive MAC keys. A value in the one-way key chain is used as the "key" to derive the next value, and the convention is not to use the same key in different cryptographic operations. We thus apply a second pseudo-random function F' to derive the MAC keys. Specifically, let r^j_{N-i} denote the $(N-i)^{th}$ value in the one-way key chain for level-j. we use key $K^j_i = F'\left(r^j_{N-i}\right)$ to generate tags for interval-i messages, for verification by level-j nodes.

4) Key Disclosure Delay d: $\mathcal{S}$ sets the key disclosure delay for the one-way key chain to be

$$d = \lceil D/\delta \rceil + 1 \tag{7}$$

in units of intervals. For packets sent in interval i, $\mathcal{S}$ delays the key disclosure for level-j until interval $i + dj$, by when level-j nodes should have already received the interval-i packets.

B. Initializing Nodes

Before data transmission, $\mathcal{S}$ and the nodes in the network loosely synchronize their clocks. By using a synchronization protocol with low complexity [18], each node v in $V - \{\mathcal{S}\}$ compares its local time with that of $\mathcal{S}$'s, and records the difference Δ_v. We assume that the clock drifts between $\mathcal{S}$ and the receivers are negligible during a multicast session.

In addition to loose time synchronization, $\mathcal{S}$ sends each node a bootstrapping packet. These packets are digitally signed with a public signature scheme. A bootstrapping packet includes: the starting time of the transmission T_0, the interval length δ, the key disclosure delay d, the level of the recipient node, and the commitment to the key chain associated with the level of the recipient node.

C. Sending Authenticated Packets

To prevent tag pollution attacks, $\mathcal{S}$ generates L tags each for a different level in a nested fashion. Let M denote a message to be sent in interval i and $K = K^1_i, K^2_i, \ldots, K^L_i$ be the L level keys for interval i. $\mathcal{S}$ prepends M with i; without loss of generality, let the resulting vector $(i, M) \in \mathbb{F}_q^{n+m}$. We will discuss the purpose of prepending i in Section V-D. $\mathcal{S}$ computes tags $\mathbf{MAC}((i, M), K) = (t^L, t^{L-1}, \ldots, t^1)$ and sends packet

$$P \triangleq (i, M, t^L, t^{L-1}, \ldots, t^1) \in \mathbb{F}_q^{n+m+L}. \tag{8}$$

Key disclosure schedule. RIPPLE leverages the time difference between packet dispatch and key disclosure to authenticate packets. For packets sent in interval i, $\mathcal{S}$ delays the release of the key K^j_i for level j and interval i until after level-j nodes have received those packets. Specifically, it releases a key packet (j, K^j_i) for interval i and level j in interval $i + dj$ where d is the key disclosure delay.

D. Packet Authentication and Coding

Packet buffering. After a forwarder $\mathcal{F}$ receives a packet $P = \left(i, M, t^L, t^{L-1}, \ldots, t^l\right)$, it buffers it only if $\mathcal{S}$ has not disclosed the key K^l_i for $\mathcal{F}$'s level l and interval i. To this end, $\mathcal{F}$ first derives the latest possible time y that $\mathcal{S}$ could be at based on the loose time synchronization. Assume that the

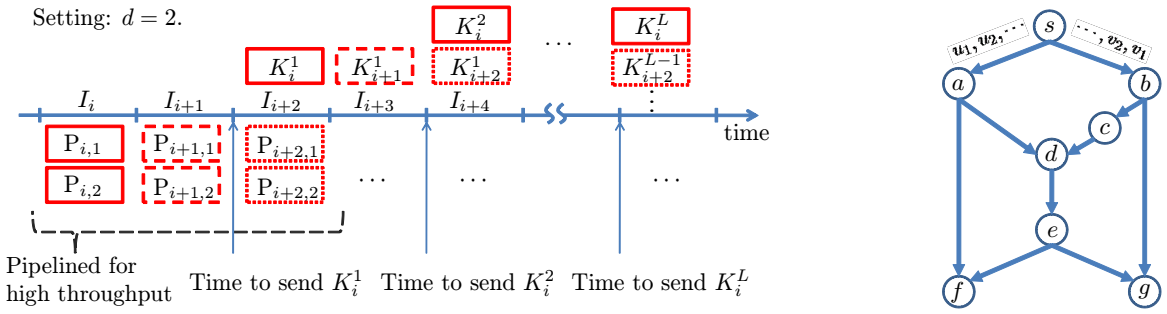

Fig. 1. a) This figure illustrates the packet and key disclosure schedule of source $\mathcal{S}$. In this example, network delay D is 0.9δ and the key disclosure delay d is 2. $\mathcal{S}$ sends two packets per interval. For the two packets $P_{i,1}$ and $P_{i,2}$ sent in interval I_i, $\mathcal{S}$ releases the corresponding level j keys K_i^j in interval I_{i+dj}, by when $P_{i,1}$ and $P_{i,2}$ must have reached all level j nodes. Specifically, $\mathcal{S}$ releases K_i^1, K_i^2, ..., K_i^L for levels 1,2, ..., L in intervals I_{i+2}, I_{i+4}, ..., I_{i+2L} respectively. For ease of understanding, we use the same box frame for packets sent in an interval and the corresponding level keys for that interval. b) $\mathcal{S}$ sends multiple batches of messages $\{u_i, v_i\}_i$ to nodes f and g over a skewed Butterfly network, where each link has unit capacity and unit delay. Node d buffers u_i and waits for v_i before transmitting $u_i + v_i$ to node e. Due to pipelining, the achieved multicast throughput is still 2.

synchronization error between $\mathcal{S}$ and $\mathcal{F}$ is $\Delta_{\mathcal{F}}$ and the current time at $\mathcal{F}$ is T_k. Then $y = (T_k + \Delta_{\mathcal{F}})/\delta$. For packets sent in interval i, $\mathcal{S}$ releases the key for level-l nodes in interval $i+dl$ based on the key disclosure schedule. As a result, $\mathcal{F}$ checks if $i \le y \le i+dl$. If not, $\mathcal{F}$ discards P. Otherwise, $\mathcal{F}$ buffers P.

Key verification. $\mathcal{F}$ verifies the key's legitimacy after it receives a key packet $\left(j, K_i^j\right)$. If $j < l$, $\mathcal{F}$ discards it. If $j > l$, $\mathcal{F}$ forwards it to its children. Otherwise, $\mathcal{F}$ keeps it. If $\mathcal{F}$ already knows K_i^j or a later key, then it does nothing. Otherwise, K_i^j is the latest key received so far. $\mathcal{F}$ checks K_i^j's legitimacy and derives its birth interval i via the one-way key chain. Specifically, for a previously received key K_k^j ($k < i$), if $\mathcal{F}$ can find a positive integer x such that $K_k^j = F^x(K_i^j)$, then K_i^j is legitimate with birth interval $i = x + k$. Having an upper bound on x limits the number of applications of F to compute. To this end, $\mathcal{F}$ first derives the latest possible time y that $\mathcal{S}$ could be at as described above. Then the latest key that $\mathcal{S}$ has disclosed for level j is for interval $\lfloor y - dj \rfloor$. So x is bounded by $\lfloor y - dj \rfloor - k$ if $\mathcal{F}$ uses K_k^j in key verification. To reduce future computation, $\mathcal{F}$ buffers the latest key in a pair (i, K_i^j) and removes the old pair.

Packet authentication. With a verified key K_i^j, $\mathcal{F}$ authenticates the buffered interval-i packets. For packet $P = (i, M, t^L, t^{L-1}, \ldots, t^j)$ and key K_i^j, $\mathcal{F}$ checks if $t^j = \langle (i, M, t^L, t^{L-1}, \ldots, t^{j+1}), K_i^j \rangle$. If so, $\mathcal{F}$ removes t_i^j from P. Otherwise, it discards P. For w authenticated packets $(i, M_k, t_k^T, t_k^{T-1}, \ldots, t_k^{j+1})_{k=1}^w$, $\mathcal{F}$ generates w coding coefficients α_i as described in Section II-A. $\mathcal{F}$ generates the network coded message $\tilde{M} = \sum_{k=1}^w \alpha_k M_k$. It then calls **Combine** which takes $(i, M_k, t_k^T, t_k^{T-1}, \ldots, t_k^{j+1}, \alpha_k)_{k=1}^w$ with $w \le m$, and outputs a tag $t = \tilde{t}^T, \tilde{t}^{T-1}, \ldots, \tilde{t}^{j+1}$ where $\tilde{t}^j = \sum_{k=1}^w \alpha_k t_k^j$. $\mathcal{F}$ sends $(i, \tilde{M}, t)$ to its children. See Fig. 1(a) for an example.

E. Reducing Key Disclosure Traffic

The baseline RIPPLE scheme broadcasts L keys to the entire network per interval, one key for nodes at each level. For large network with large L and large number of nodes, this could lead to heavy key disclosure traffic, reducing the overall transmission efficiency.

We describe a mechanism to reduce the key disclosure traffic to one key per interval. Our mechanism is inspired by the work of [38] and works as follows. Instead of using one independent key chain per level, we use the same key chain for all levels, but different functions of the same key for authentication at different levels. All nodes at different levels share the same key chain. Each key K_i in the key chain is associated with the corresponding time interval i, and will be disclosed in interval i. When nodes at level j receive K_i, they authenticate it using the commitment of the key chain, and generate their authentication key as $K_i^j = G(K_i, j)$, where G is a pseudo-random function. This way, nodes at different levels verify the same "seed" key using the shared key chain, but derive different authentication keys.

VI. Performance Analysis

In this section, we study the performance characteristics of the RIPPLE transmission protocol.

A. Computational Overhead

The computational overhead of the RIPPLE transmission protocol is dominated by the homomorphic MAC scheme used in the last two stages of RIPPLE. The overhead incurred by the first two stages, the sender setup and node initialization stages, is one time cost for a multicast session and asymptotically negligible. Recall that our MAC scheme consists of four functions, **Generate**, **MAC**, **Verify**, and **Combine** (Section IV, full construction). Function **Generate** produces L random numbers as level keys for a multicast session; its cost is negligible.

The cost of **MAC** is dominated by the number of finite field multiplications. **MAC** takes a message M and L keys and outputs L tags, t^L, $t^{L-1}, \ldots, t^1$. Recall that a tag t^j for level j is computed as the inner product of two vectors in $\mathbb{F}_q^{(n+m)+(L-j)}$, i.e., $t^j = \langle (M, t^L, t^{L-1}, \cdots, t^{j+1}), K^j \rangle$. Generating t^j thus takes $n + m + L - j$ multiplications in $\mathbb{F}_q$. The number of

Number of Tags Per Level	**MAC** (ns)	**Verify** and **Combine** (ns)	Tag Size (bytes)	Security
1	61.7	4.0	16	$1/2^8$
4	246.8	24.0	64	$1/2^{32}$

TABLE I
COMPUTATION AND COMMUNICATION OVERHEAD UNDER DIFFERENT SECURITY SETTINGS

multiplications to generate all L tags is

$$S = \sum_{j=1}^{L} (n + m + L - j) = L\left(n + m + \frac{L-1}{2}\right). \quad (9)$$

The cost of **Verify** is similar to that of **MAC**. Given a packet $P = (M, t^L, t^{L-1}, \ldots, t^j) \in \mathbb{F}_q^{n+m+L-j+1}$ and a key K^j for level j, a level-j node checks if $t^j = \langle (M, t^L, t^{L-1}, \ldots, t^{j+1}), K^j \rangle$. This takes $n + m + L - j$ multiplications in $\mathbb{F}_q$. For a packet to travel from level 1 to L, a total number of $L\left(n + m + \frac{L-1}{2}\right)$ multiplications is required. On average, each node computes $\left(n + m + \frac{L-1}{2}\right)$ multiplications.

The cost of **Combine** depends only on L and the number w of incoming edges of a node. Recall that in the RIPPLE transmission protocol, a level-j node receives packets in the form of $(M, t^T, t^{T-1} \ldots, t^j,)$; tag $t^{j-1}, t^{j-2}, \ldots, t^1$ have been removed by upper level nodes. **Combine** takes $(M_i, t_i^T, t_i^{T-1}, \ldots, t_i^{j+1}, \alpha_i)_{i=1}^{w}$ with $w \leq m$, and outputs a tag $t = t^T, t^{T-1}, \ldots, t^{j+1}$ where $t^j = \sum_{i=1}^{w} \alpha_i t_i^j$. To compute the tags for each outgoing packet, a level-j node performs $(L-j)w$ multiplications. If we assume that every node in the network has the same number of parents, then on average, a node computes $(L-1)w/2$ multiplications.

B. Experiments

Table I summarizes RIPPLE's performance. We assume that the network has 10K nodes and estimate the maximum number of levels $L = 16$ (note that $\log 10000 \approx 13$). We assume $q = 2^8$, a packet size $n = 1024$ bytes, a generation size $m = 32$, and the number of parents per node $w = 6$. We use the C/C++ library [39] which implements fast Galois field multiplications with table lookups. We conduct the experiments on a GNU/Linux system with 2.33GHz Intel Core 2 Duo processors. Recall that in Section IV-A, we discuss using multiple tags to increase security. For a fixed q, the computational and per packet communication overhead grows linearly with the number of tags per level, while the security grows exponentially. We show this relationship in Table I.

VII. CONCLUSIONS

In this work we present RIPPLE, an efficient in-network authentication scheme that is well-matched to distributed random linear network codes. Operating in tandem, they provide a practical and low-complexity scheme for achieving rate-optimal throughput even in the presence of a disruptive adversary in the network. The low complexity of RIPPLE's authentication scheme arises from using symmetric-key authentication together with time-asymmetry, *i.e.*, keys are transmitted after their corresponding messages. Hence we achieve security akin to that of public-key verification schemes, without their undesirable properties (prohibitive computational complexity for moderate key sizes). We show that RIPPLE is robust to arbitrary collusion among adversaries. It is also resilient against the subtle tag pollution attack discussed in this work, wherein an adversarially injected packet with a single corrupted tags may cause nodes in prior schemes to drop *numerous* packets.

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, 2000.

[2] S. R. Li, R. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[3] R. Kotter and M. Medard, "Beyond routing: An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 782–796, 2003.

[4] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Transactions on Information Theory*, vol. 51, no. 6, pp. 1973–1982, 2003.

[5] T. Ho, R. Koetter, M. Mdard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *Proc. of IEEE International Symposium on Information Theory (ISIT)*, 2003.

[6] D. S. Lun, M. Médard, and M. Effros, "On coding for reliable communication over packet networks," in *Proc. of the 42nd Allerton Conference*, 2004.

[7] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "Xors in the air: Practical wireless network coding," in *Proc. of ACM SIGCOMM 2006*, 2006.

[8] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," in *Microsoft Research Technical Report*, 2004.

[9] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Médard, and M. Effros, "Resilient network coding in the presence of byzantine adversaries," *IEEE Transactions on Information Theory, Special Issue on Information Theoretic Security*, vol. 54, no. 6, pp. 2596–2603, June 2008.

[10] C. Gkantsidis and P. Rodriguez Rodriguez, "Cooperative security for network coding file distribution," in *Proc. of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, 2006.

[11] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *Proc. of the Fortieth Annual Conference on Information Sciences and Systems*, 2006.

[12] F. Zhao, T. Kalker, M. Medard, and K. J. Han, "Signatures for content distribution with network coding," in *Proc. of IEEE International Symposium on Information Theory (ISIT)*, 2007.

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

[14] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a linear subspace: Signature schemes for network coding," in *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009, vol. 5443/2009, pp. 68–87.

[15] 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 the second ACM conference on Wireless network security*, 2009.

[16] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient scheme for securing XOR network coding against pollution attacks," in *Proc. of the 28th IEEE International Conference on Computer Communications (INFOCOM)*, 2009.

[17] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-Based Integrity for Network Coding," in *2009 Applied Cryptography and Network Security*, 2009, pp. 292–305.

[18] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proc. of the IEEE Symposium on Security and Privacy (OAKLAND)*, 2000.

[19] S. Jaggi, M. Effros, T. Ho, and M. Médard, "On linear network coding," in *Proc. of the 42nd Allerton Conference*, 2004.

[20] N. Cai and R. W. Yeung, "Network coding and error correction," in *Proc. of IEEE International Symposium on Information Theory (ISIT)*, 2002.

[21] A. Jiang, "Network coding for joing storage and transmission with minimum cost," in *Proc. of IEEE International Symposium on Information Theory (ISIT)*, 2006.
[22] R. W. Yeung, S. Y. R. Li, N. Cai, and Z. Zhang, "Network coding theory," *Foundation and Trends in Communications and Information Theory*, vol. 2, pp. 241–381, 2005.
[23] D. Dolev, C. Dwork, O. Waarts, and M. Yung, "Perfectly secure message transmission," *Journal of the Association for Computing Machinery*, vol. 40, no. 1, pp. 17–47, January 1993.
[24] L. Subramanian, "Decentralized security mechanisms for routing protocols," Ph.D. dissertation, University of California at Berkeley, Computer Science Division, Berkeley, CA 94720, 2005.
[25] A. Pelc and D. Peleg, "Broadcasting with locally bounded byzantine faults," *Information Processing Letters*, vol. 93, no. 3, pp. 109–115, February 2005.
[26] T. C. Ho, B. Leong, R. Koetter, M. Médard, M. Effros, and D. R. Karger, "Byzantine modification detection in multicast networks using randomized network coding," in *International Symposium on Information Theory*, 2004.
[27] R. W. Yeung and N. Cai, "Network error correction, Part I: Basic concepts and upper bounds," *Communications in Information and Systems*, vol. 6, no. 1, pp. 19–36, 2006.
[28] N. Cai and R. W. Yeung, "Network error correction, Part II: Lower bounds," vol. 6, no. 1, pp. 37–54, 2006, communications in Information and Systems.
[29] D. Silva, F. R. Kschischang, and R. Koetter, "A rank-metric approach to error control in random network coding," in *IEEE Information Theory Workshop*, 2007.
[30] M. Krohn, M. Freedman, and D. Mazieres, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *Proc. of IEEE Symp. on Security and Privacy*, 2004.
[31] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The TESLA broadcast authentication protocol," *RSA CryptoBytes*, Summer 2002.
[32] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. of Allerton*, 2003.
[33] D. Lun, "Efficient Operation of Coded Packet Networks," Ph.D. dissertation, Massachusetts Institute of Technology, 2006.
[34] H. Yao, S. Jaggi, and M. Chen, "Network coding tomography for network failures," in *Proc. of the 29th IEEE International Conference on Computer Communications (INFOCOM)*, 2009.
[35] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM (JACM)*, vol. 33, no. 4, pp. 792 – 807, 1986.
[36] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.
[37] U. S. National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS), Federal Register 56. FIPS PUB 186," Aug. 1991.
[38] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2001, pp. 35–46.
[39] J. S. Plank, "Fast galois field arithmetic library in C/C++," University of Tennessee, Tech. Rep. UT-CS-07-593, March 2007.