A Tag Encoding Scheme against Pollution Attack to Linear Network Coding

Xiaohu Wu, Yinlong Xu, Chau Yuen, and Liping Xiang

Abstract—Network coding allows intermediate nodes to encode data packets to improve network throughput and robustness. However, it increases the propagation speed of polluted data packets if a malicious node injects fake data packets into the network, which degrades the bandwidth efficiency greatly and leads to incorrect decoding at sinks. In this paper, insights on new mathematical relations in linear network coding are presented and a key predistribution-based tag encoding scheme *KEPTE* is proposed, which enables all intermediate nodes and sinks to detect the correctness of the received data packets. Furthermore, the security of KEPTE with regard to pollution attack and tag pollution attack is quantitatively analyzed. The performance of KEPTE is competitive in terms of: 1) low computational complexity; 2) the ability that all intermediate nodes and sinks detect pollution attack; 3) the ability that all intermediate nodes and sinks detect tag pollution attack; and 4) high fault-tolerance ability. To the best of our knowledge, the existing key predistribution-based schemes aiming at pollution detection can only achieve at most three points as described above. Finally, discussions on the application of KEPTE to practical network coding are also presented.

Index Terms—Network coding, pollution attack, Byzantine attack, compromised nodes, security

1 Introduction

Network coding, where an intermediate node encodes incoming packets before forwarding, has been theoretically proven to maximize network throughput and enhance network robustness [1], [2], [3]. It has received extensive attentions and been applied to various computer network systems, such as multicast networks [28], wireless networks [26], and P2P systems [27].

However, when there is a malicious node in a network and the malicious node injects *fake data packets* into its downstream nodes, the fake data packets will be encoded together with correct data packets by the downstream nodes and the outputs of the downstream nodes will be polluted and fake. The pollution propagates in the network quickly with the transmission of *polluted data packets*, which not only leads to incorrect decoding at sinks, but also wastes network resources. So it is crucial to prevent pollution attacks in practical applications of network coding.

The existing approaches against pollution attack to network coding can be categorized into three classes, error correction, malicious node localization, and pollution detection. Error correction-based approaches [4], [19], [23], [24] only allow a small portion of packets to be polluted, and focus on correcting error only at sinks. The pollution will inevitably propagate in networks with the error correction approaches.

• *X. Wu, Y. Xu, and L. Xiang are with the Key Laboratory on High Performance Computing, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230026, P.R. China. E-mail: {wuxh11, xlping}@mail.ustc.edu.cn, ylxu@ustc.edu.cn.*
• *C. Yuen is with the Singapore University of Technology and Design, 20 Dover Drive, Singapore 138682. E-mail: yuenchau@sutd.edu.sg.*

Manuscript received 15 July 2012; revised 10 Jan. 2013; accepted 20 Jan. 2013; published online 14 Feb. 2013.
Recommended for acceptance by W. Jia.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2012-07-0638.
Digital Object Identifier no. 10.1109/TPDS.2013.24.

Some schemes [17], [18], [22] are proposed to locate malicious nodes and make those nodes unable to further inject polluted data packets. Those schemes either assume that there exists a powerful controller that knows the entire topology of a network [17], [18] or assume a clock synchronization of all nodes in a network [22]. Those schemes are of limited practicality when multiple malicious nodes exist.

The existing schemes of pollution detection, which are mainly based on key delay distribution, public key cryptography (PKC), and key predistribution, focus on detecting and filtering fake or polluted data packets at intermediate nodes or sinks directly and can prevent pollution propagation efficiently. The schemes in [12], [16] based on key delay distribution require a clock synchronization of all the nodes in a network. So it is difficult to implement them in an adversarial distributed environment. The implementation of the schemes based on PKC or key predistribution are relatively simple. However, the PKC-based schemes in [5], [6], [7], [8], [9], [10], [11], [29] require a large field, for example, the field in [8] is $F_{3^{97}}$, which implies that the computational complexities of PKC-based schemes are very high.

The schemes in [13], [14], [15], [25] are based on key predistribution. The computational complexities of the schemes in [13], [14] are low, but they experience tag pollution attack that leads to numerous correct data packets being discarded. With the scheme in [14], the correctness of a data packet will be verified after several hops (usually at least 3 hops). The MacSig scheme [15] requires a field of size 128 bits with very high computational complexity. In addition, even if the schemes in [13], [15], [25] tolerate less nodes to be compromised, very large redundancy is needed to append to each data packet. For example, if the most recent MacSig scheme [15] allows 15 nodes to be compromised with a probability 99.5 percent, the size of redundancy appended to an IP data packet of 1,400 bytes will be 5,144 bytes.[1]

This paper will present some insights on mathematical relations in linear network coding and propose a key predistribution-based tag encoding (KEPTE) scheme to detect polluted data packets. The basic idea of KEPTE is as follows: A source s uses N keys to generate N tags for each data packet. Each node g except the source holds a unique pair of keys (Z_g, V_g), where Z_g, V_g, and the N keys held by s satisfy a certain relation. When a node g receives a data packet W with its N tags $t_{W,1}, \dots, t_{W,N}$, it uses Z_g to encode the N tags $t_{W,1}, \dots, t_{W,N}$ and generates a new tag t that can be viewed as a tag generated according to W and the key Z_g. Then, the node g can verify the correctness of W with V_g and t. An illustration of the packet correctness verification process of KEPTE will also be seen in Fig. 3.

Compared with the existing key predistribution-based schemes in [13], [14], [15], [25], KEPTE shows competitive performance as follows:

1. It is computationally efficient because it does not require a large field.
2. All intermediate nodes and sinks are able to detect pollution attack.
3. All intermediate nodes and sinks are able to detect tag pollution attack.
4. Suppose that random linear network coding is executed over finite field F_p. When $N\lceil \log_2 p \rceil$ bits of redundancy is appended to a data packet, it allows no more than $N - \theta (N \geq \theta + 1)$ nodes to be compromised, where θ mainly depends on F_p and will be explained in Section 4.5. In particular, if $F_p = F_{2^8}$ and $F_p = F_{2^{16}}$, θ can be set to be 8 and 5, respectively.

The rest of this paper is organized as follows: In Section 2, we introduce the models and assumptions of KEPTE. Then, we present KEPTE in Section 3. Subsequently, a quantitative security analysis is taken in Section 4. In Section 5, we give performance analysis and performance comparison with other typical schemes, and discuss the application of KEPTE to a practical network coding case. At last, a conclusion is made in Section 6.

Part of this work appeared at NetCod′ 2011 [30]. Compared with [30], this paper proposes a lightweight key distribution, presents detailed security analysis of KEPTE, and detailed comparison of KEPTE with other schemes.

2 Models and Assumptions

2.1 Network Coding Model

Suppose that a network consists of a source s, some intermediate nodes, and a set $\mathcal{R}$ of sinks, and random linear network coding is exploited in the network without the knowledge of its global topology.

The source s is to multicast a file of n *data blocks* $B_1, B_2, \dots, B_n$ to a set $\mathcal{R}$ of sinks, where each $B_i (1 \leq i \leq n)$ is a vector of m dimensions over field F_p (*Note: Table 1 summarizes the main notations in this paper*). To mark the coefficients of encoded data blocks, before sending $B_1, B_2, \dots, B_n$, a unit vector E_i of length n over field F_p is appended to $B_i (1 \leq i \leq n)$, where the ith coordinate of E_i is 1. Set $P_i = (E_i, B_i) \in F_p^{m+n}$ as *a data packet* that consists of a data block B_i and the encoded coefficient vector E_i. With random linear network coding, an encoded data packet is a linear combination of data packets $P_1, P_2, \dots, P_n$. Let W be an encoded data packet, then W will be an $m+n$ dimensional vector with the first n coordinates being encoding coefficients. If the first n coordinates of W are $w_1, \dots, w_n$, W can be represented as

$$W = (w_1, \dots, w_n) \cdot \begin{pmatrix} P_1 \\ \dots \\ P_n \end{pmatrix}, \qquad (1)$$

where $(w_1, w_2, \dots, w_n)$ is usually called the global encoding vector of W. A data packet W' is called *fake or polluted* if and only if $W' \notin Span\{P_1, P_2, \dots, P_n\}$.[2]

TABLE 1
Notation Interpretation

Notations	The Corresponding Meanings
$B_1, \cdots, B_n \in F_p^m$	a file of n data blocks, each with length m over Field F_p
$P_i = (E_i, B_i) \in F_p^{m+n}$	i-th data packet, with length $m+n$ over F_p, $1 \leq i \leq n$
N	a parameter that decides the fault-tolerant ability of KEPTE
$X_1, \cdots, X_N \in F_p^{m+n}$	N secret vectors that the source holds
$t_{P,i}$	the i-th tag of data packet P, $1 \leq i \leq N$
$Z_g \in F_p^N$, $V_g \in F_p^{m+n}$	two secret vectors that a node g except the source holds
$g_1, \cdots, g_r$	r nodes compromised by an adversary
$\kappa_1, \kappa_2, \kappa_3$	parameters evaluating security level
$G_1 : K_{G_1} \to F_p^{m+n}$	pseudo random function (PRF)
$G_i : K_{G_i} \to F_p^N$	PRF , $i = 2, 3$
$k_1, \cdots, k_N \in K_{G_1}$	N public keys, each k_i as the input of G_1 to generate $Y_i \in F_p^{m+n}$, $1 \leq i \leq N$
$b_1, \cdots, b_N \in K_{G_2}$	N private keys held by the source, each b_i as the input of G_2 to generate $U_i \in F_p^N$, $1 \leq i \leq N$
$c_g \in K_{G_3}, Z_g \in F_p^N$	two private keys held by node g, c_g is the input of G_3 to generate $C_g \in F_p^N$

With random linear network coding, a node randomly selects encoding coefficients from a field F_p. Ho et al. [3] show that if F_p is sufficiently large, any n encoded data packets received by a node will be linearly independent and further the n original data packets $P_1, P_2, \dots, P_n$ can be decoded from them with high probability. F_{2^8} and $F_{2^{16}}$ are two typical fields in practical network coding [20].

From the view of practical network coding, if a file is too large, it is usually divided into some subfiles, which are called groups or generations. Each generation is further divided into some data blocks. Random linear network coding will be executed among the data blocks in the same generation, which is usually identified by a generation *id*. So we can perform KEPTE in each generation as a separate file. In the following, we only consider that the source s multicasts n data packets to sinks.

1. We set $\delta = 0.1$ and $\varepsilon = 0.005$ in MacSig, and we will explain the settings in the supplementary, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.24.

2. This is the linear space based on $P_1, \dots, P_n$, it consists of all linear combinations of $P_1, \dots, P_n$.

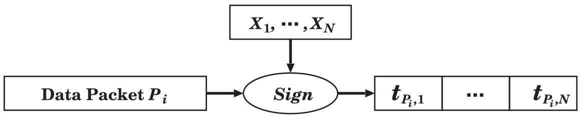


Fig. 1. Tag generation at source s.

2.2 Adversary Model

We assume that the source s is trustworthy. An adversary may compromise some intermediate nodes or sinks, and knows the secret information held by those compromised nodes. The adversary is able to wiretap all the data packets that are transmitted in a network.

A compromised node can inject fake data packets into its output links, which is called *pollution attack*. The objective of pollution attack is to make intermediate nodes or sinks unable to detect error data packets, which not only leads to incorrect decoding at sinks but also makes polluted data packets be transmitted in a network, leading to bandwidth waste.

In addition, a compromised node can modify the verification tags of a correct data packet, and injects the correct data packet with modified tags to its output links, which is called *tag pollution attack*. The objective of tag pollution attack is to get correct data packets be judged as wrong and be discarded by intermediate nodes or sinks, which wastes bandwidth greatly.

2.3 Cryptographical Assumptions

We assume that there is a key distribution center (KDC) in our model. KDC is a common tool for key distribution. If KDC is not available, key distribution based on PKC can replace the function of KDC. By PKC, the source distributes secret information, which is also secret keys, to each node g except the source in a network. The previous cryptographical schemes against pollution attack to network coding have the similar assumptions on KDC or PKC. For the details to distribute keys securely, please refer to [31].

3 KEPTE

3.1 Three Basic Algorithms

Before presenting KEPTE, we give an overview of three basic algorithms, namely $Sign$, $Combine$, and $Verify$. The algorithm $Sign$ computes N tags for each of the n data packets $P_1, P_2, \ldots, P_n$, where N is a security parameter and we will analyze the security performance that N tags can provide in the next section. Given multiple data packets, each with N tags, $Combine$ produces N tags for a linear combination of those multiple data packets. $Verify$ checks the correctness of a data packet with its N tags. In KEPTE, the source, intermediate nodes or sinks will perform one or two of those three algorithms:

- $Sign(X_1, \ldots, X_N, P_i)$:
  Input: N secret vectors $X_1, \ldots, X_N \in F_p^{m+n}$, the ith data packet $P_i \in F_p^{m+n}$.
  Output: N tags $t_{P_i,1}, \ldots, t_{P_i,N} \in F_p$ for P_i, where $t_{P_i,j} = P_i \cdot X_j^T \in F_p (1 \le j \le N)$.

- $Combine((W_i, t_{W_i,1}, \ldots, t_{W_i,N})_{i=1}^h, (\alpha_i)_{i=1}^h)$:
  Input: h vectors $W_i \in F_p^{m+n} (1 \le i \le h)$ and each with N tags $t_{W_i,1}, \ldots, t_{W_i,N} \in F_p (1 \le i \le h)$, h constants $\alpha_1, \ldots, \alpha_h \in F_p$.
  Output: $(W, t_{W,1}, \ldots, t_{W,N}) = \sum_{i=1}^h \alpha_i (W_i, t_{W_i,1}, \ldots, t_{W_i,N}) \in F_p^{m+n+N}$.

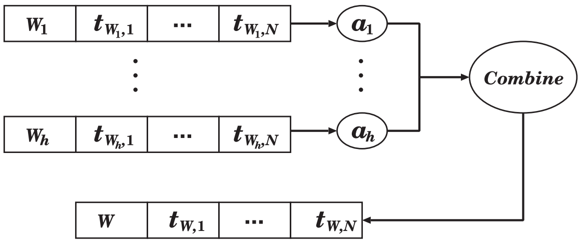


Fig. 2. Encoding at a node g.

- $Verify(Z_g, V_g, (W, t_{W,1}, \ldots, t_{W,N}))$:
  Input: two secret vectors $Z_g \in F_p^N$ and $V_g \in F_p^{m+n}$, a vector $W \in F_p^{m+n}$ with its N tags $t_{W,1}, \ldots, t_{W,N}$.
  Output: If $Z_g \cdot (t_{W,1}, \ldots, t_{W,N})^T = W \cdot V_g^T$, output 1; otherwise, output 0.

3.2 The Outline of KEPTE

In this section, we describe the process of generating tags for $P_1, \ldots, P_n \in F_p^{m+n}$ at the source s, the encoding of multiple data packets at intermediate nodes, and the correctness verification of data packets at each node g except s in a network. In the following of this paper, we will simply refer to *each node g except the source s in a network* with *each node g*:

1. *Setup*. KDC distributes N *secret vectors* $X_1, \ldots, X_N \in F_p^{m+n}$ to the source s, and distributes *two secret vectors* $Z_g \in F_p^N$ and $V_g \in F_p^{m+n}$ to each node g, where Z_g, V_g, and $X_1, \ldots, X_N$ satisfy the following:
$$V_g = Z_g \cdot \begin{pmatrix} X_1 \\ \cdots \\ X_N \end{pmatrix}. \tag{2}$$
We will discuss the detailed process of KDC distributing these secret vectors to s and g in Section 3.3.
2. *Tag Generation*. For each $P_i \in F_p^{m+n} (1 \le i \le n)$, the source uses the algorithm $Sign(X_1, \ldots, X_N, P_i)$ to generate N tags $t_{P_i,1}, \ldots, t_{P_i,N}$. Fig. 1 depicts this process.
3. *Encoding*. Assume that an intermediate node g receives h correct data packets $W_i \in F_p^{m+n} (1 \le i \le h)$, each with N tags $t_{W_i,1}, \ldots, t_{W_i,N} \in F_p (1 \le i \le h)$. For an output link, g randomly selects h constants $\alpha_1, \ldots, \alpha_h \in F_p$ and performs the algorithm $Combine((W_i, t_{W_i,1}, \ldots, t_{W_i,N})_{i=1}^h, (\alpha_i)_{i=1}^h)$ to generate a new encoded data packet W with N tags $t_{W,1}, \ldots, t_{W,N}$ as the output of this link. Fig. 2 depicts this process.
4. *Verification*. Upon receiving a data packet W with its N tags $t_{W,1}, \ldots, t_{W,N}$, a node g checks the correctness of W with algorithm $Verify$ and its secret vectors Z_g, V_g. If its output is 1, g judges W being correct; otherwise, g judges W being fake or polluted, and discards W. Fig. 3 depicts this process. The correctness of the verification algorithm will be analyzed in Section 3.4.

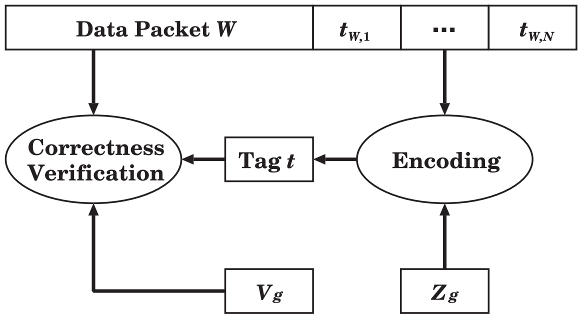


Fig. 3. Verification at a node g.

3.3 Key Distribution in Practical Network Coding

With KEPTE, the source s needs N secret vectors $X_1, \ldots, X_N$ to produce N tags for each packet, and each node g needs two secret vectors Z_g, V_g to check the correctness of its received packets. Each of the secret vectors $X_1, \ldots, X_N$ and V_g is of the same size as a data packet and N may be several tens or even more. So distributing $X_1, \ldots, X_N$ to s and distributing V_g to node g may bring heavy load to a network and KDC. To reduce the cost of distributing the secret vectors, we design a scheme for key distribution at KDC in the following.

Before introducing the proposed key distributing scheme, we first present three cryptographically secure pseudorandom functions ($PRFs$), $G_1 : \mathcal{K}_{G_1} \to F_p^{m+n}$, $G_2 : \mathcal{K}_{G_2} \to F_p^N$, and $G_3 : \mathcal{K}_{G_3} \to F_p^N$. G_1, G_2, and G_3 are publicly known to KDC and all the nodes in a network. An element in $\mathcal{K}_{G_i} (1 \leq i \leq 3)$ is called a seed, which is used as an input of G_i. Usually, the size of a seed is 128 bits [31].

The following key distributing process is to replace the "*Setup*" step for distributing $X_1, \ldots, X_N$ to s and distributing V_g to node g. It is shown in Fig. 4. We will compare the overheads of the following distributing process with which in the "*Setup*" step:

1. N seeds $k_1, k_2, \ldots, k_N \in \mathcal{K}_{G_1}$ are the public keys, which are known to all the nodes in a network. From the N seeds, N public vectors $Y_1, Y_2, \ldots, Y_N \in F_p^{m+n}$ are generated at the source s and at each node g, where $Y_i = G_1(k_i) \in F_p^{m+n} (1 \leq i \leq N)$.
2. For the source s, KDC selects N seeds $b_1, b_2, \ldots, b_N \in \mathcal{K}_{G_2}$ such that $((U_1)^T, (U_2)^T, \ldots, (U_N)^T)^T$ is full ranked, where $U_i = G_2(b_i) \in F_p^N (1 \leq i \leq N)$. Then, KDC distributes the N private keys $b_1, b_2, \ldots, b_N$ to the source s. So with the following, the source s gets N secret vectors $X_1, \ldots, X_N \in F_p^{m+n}$:
$$\begin{pmatrix} U_1 \\ \cdots \\ U_N \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ \cdots \\ X_N \end{pmatrix} = \begin{pmatrix} Y_1 \\ \cdots \\ Y_N \end{pmatrix}. \tag{3}$$
3. For each node g, KDC selects a seed $c_g \in \mathcal{K}_{G_3}$, sets $C_g = G_3(c_g) = (c_{g,1}, c_{g,2}, \ldots, c_{g,N}) \in F_p^N$ and computes a key $Z_g \in F_p^N$ with the following:
$$Z_g = C_g \cdot \begin{pmatrix} U_1 \\ \cdots \\ U_N \end{pmatrix}. \tag{4}$$
Then, KDC distributes the two private keys c_g, Z_g to the node g and the node g gets a secret vector $V_g \in F_p^{m+n}$ with the following:
$$V_g = C_g \cdot \begin{pmatrix} Y_1 \\ \cdots \\ Y_N \end{pmatrix} \in F_p^{m+n}. \tag{5}$$

From Steps 1 and 2, the source s gets N secret keys $X_1, \ldots, X_N$; and from Steps 1 and 3, a node g gets two secret keys Z_g, V_g. From (3), (4), and (5), $X_1, \ldots, X_N$ and Z_g, V_g satisfy (2). So the above key distribution process can successfully replace the "*Setup*" step.

After the above key distribution for a file, the source s obtains the secret vectors $X_1, \ldots, X_N$ from N short keys $b_1, \ldots, b_N$, and each node g obtains the secret vectors Z_g, V_g from Z_g and a short key c_g, where the size of a short key is usually 128 bits. But each of $X_1, \ldots, X_N \in F_p^{m+n}$ and V_g may be over 10,000 bits. So the above key distributing process greatly reduces the overhead of key distribution.

For each round of key distribution, $b_1, b_2, \ldots, b_N$, c_g, and Z_g can be used until their key lifetime[3] ends. Usually the key lifetime is long enough for the source s to send a file to the sinks securely.

3.4 The Correctness of KEPTE

Given an encoding data packet $W = (w_1, w_2, \ldots, w_{m+n})$ with its N tags $t_{W,1}, \ldots, t_{W,N}$. From (1) and the algorithm *Combine*, the following is satisfied:

$$(W, t_{W,1}, \ldots, t_{W,N}) = \sum_{i=1}^{n} w_i (P_i, t_{P_i,1}, \ldots, t_{P_i,N}). \tag{6}$$

Then, according to the algorithm *Sign*, we have

$$\begin{aligned} t_{W,j} &= (w_1, \ldots, w_n) \cdot (t_{P_1,j}, \ldots, t_{P_n,j})^T \\ &= (w_1, \ldots, w_n) \cdot \left(P_1^T, \ldots, P_n^T\right)^T \cdot X_j^T \\ &= W \cdot X_j^T. \end{aligned} \tag{7}$$

Upon receiving an encoding data packet $W = (w_1, w_2, \ldots, w_{m+n})$ with its N tags $t_{W,1}, \ldots, t_{W,N}$, a node g will use the algorithm *Verify* and two secret vectors Z_g, V_g to check the correctness of W.

From (7) and (2),

$$\begin{aligned} W \cdot V_g^T &= W \cdot ((X_1)^T, \ldots, (X_N)^T) \cdot Z_g^T \\ &= (t_{W,1}, \ldots, t_{W,N}) \cdot Z_g^T. \end{aligned} \tag{8}$$

From (8), the node g can use the algorithm *Verify* to check the correctness of a packet with N tags $t_{W,1}, \ldots, t_{W,N}$ and two secret vectors Z_g, V_g. So with KEPTE, a node is able to check the correctness of a received encoded data packet.

3.5 Main Features of KEPTE

KEPTE is enlightened by the tag generation method in [13] and is similar to its Message Authentication Codes (MAC)

3. Key lifetime depends on many factors related to the specific system [32]: the operating environment and the volume and the importance of the transmitted data, and so on. For the similar symmetric authentication key in communication application, the maximal key lifetime can be set to be 2 years.

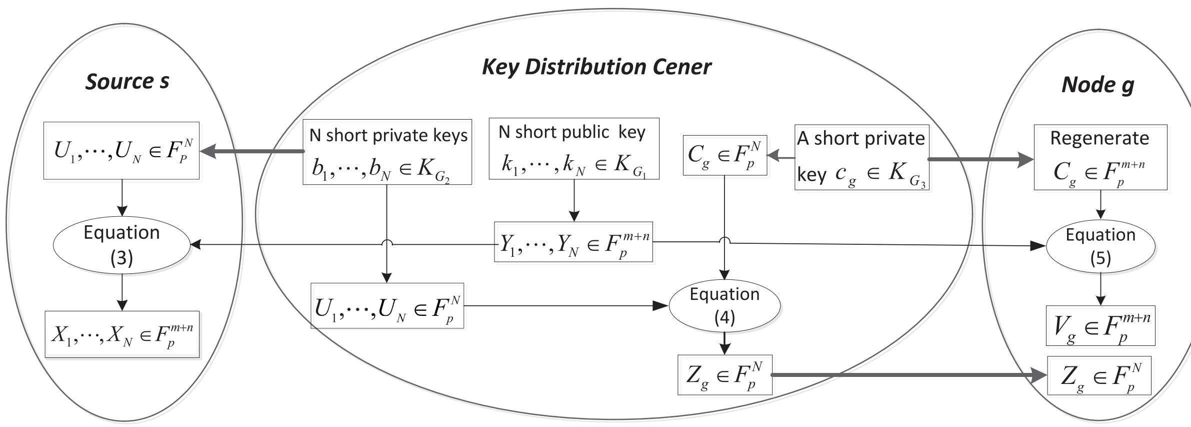

Source s
$U_1,\cdots,U_N \in F_p^N$
Equation (3)
$X_1,\cdots,X_N \in F_p^{m+n}$
Key Distribution Cener
N short private keys $b_1,\cdots,b_N \in K_{G_2}$
N short public key $k_1,\cdots,k_N \in K_{G_1}$
$C_g \in F_p^N$
A short private key $c_g \in K_{G_3}$
$Y_1,\cdots,Y_N \in F_p^{m+n}$
$U_1,\cdots,U_N \in F_p^N$
Equation (4)
$Z_g \in F_p^N$
Node g
Regenerate $C_g \in F_p^{m+n}$
Equation (5)
$V_g \in F_p^{m+n}$
$Z_g \in F_p^N$

Fig. 4. Key distribution. The bold arrow indicates KDC distributes the key(s) to the corresponding entity (s or g), and the entity may use the key(s) as the input of a PRG to compute the secret vectors at the end of the arrow. Other arrows represent a certain computation process, or the input or output of a certain computation.

scheme. We modify the tag generation method in [13] with some insights to random linear network coding such as (2)-(5)and (8).

However, unlike the previous key predistribution-based or PKC-based schemes, KEPTE falls into neither digital signature nor MAC because digital signature uses PKC to generate and verify the signature or tags of a data packet, while in MAC both the tag generation at the source and the correctness verification at other nodes in a network uses the same key.

4 Security Analysis

In a network with network coding, the source s sends a group of n data packets P_i, each with N tags $t_{P_i,1},\dots,t_{P_i,N}$, $1 \le i \le n$, to sinks. N secret vectors $X_1,\dots,X_N$ are held by the source and two secret vectors Z_g, V_g are held by each node g. In this section, we will analyze the possibilities of an adversary launching a pollution attack or a tag pollution attack successfully. The following Definition 1 shows the essence of pollution attack.

Definition 1. *$W' \in F_p^{m+n}$ with N tags $t_{W',1},\dots,t_{W',N}$ is called* a deceptive data packet *over $P_1,\dots,P_n$, if the following two conditions are satisfied:*

1. *W' is a fake data packet, that is, $W' \notin Span\{P_1,\dots,P_n\}$.*
2. *For each node g with (Z_g, V_g), $W' \cdot V_g^T = Z_g \cdot (t_{W',1},\dots,t_{W',N})^T$.*

The condition 1 in Definition 1 means that W' is not a linear combination of $P_1,\dots,P_n$, i.e., is a fake or polluted data packet. The condition 2 in Definition 1 means that W' with N tags $t_{W',1},\dots,t_{W',N}$ will pass the correctness check at all nodes. So for an adversary, the essence of pollution attack is to find a deceptive data packet W' with N tags $t_{W',1},\dots,t_{W',N}$ over $P_1,\dots,P_n$.

In this section, we first analyze the possible information an adversary can get for launching its attack. Then, based on these information, we analyze the possible pollution attacks and give the respective quantitative security requirements to resist on the pollution attacks. Next, we analyze the reason why tag pollution attack fails in KEPTE. At last, we make a security requirement summary of KEPTE.

4.1 The Possible Information for Launching Attacks

KEPTE assumes that the source s is trustworthy and the process of key distribution is secure. So during the process of key distribution, the adversary cannot get any information about the keys $b_1,\dots,b_N \in \mathcal{K}_{G_2}$ distributed to the source s and the keys c_g, Z_g distributed to node g. Further, considering that the used three $PRFs$ G_1, G_2, and G_3 are all cryptographically secure, the adversary also cannot get any information about $X_1,\dots,X_N \in F_p^{m+n}$ held by s and (C_g, Z_g, V_g) held by g during the process of key distribution.

In KEPTE, an adversary can wiretap all the data communication in a network and may compromise some intermediate nodes or sinks. Hence, the useful information that the adversary can obtain includes: 1) data packets transmitted in a network; 2) key information of the nodes compromised by the adversary. Here, we assume that the adversary knows all n original data packets P_i together with their tags $t_{P_i,1},\dots,t_{P_i,N}, 1 \le i \le n$ because it can decode $P_1,\dots,P_n$ from n linearly independent packets it received. In addition, *we assume that the adversary compromises r nodes $g_1, g_2,\dots,g_r$, each node $g_i (1 \le i \le r)$ with secret keys C_{g_i}, Z_{g_i}, V_{g_i}.* So the adversary also knows $C_{g_i}, Z_{g_i}, V_{g_i}$ for $1 \le i \le r$.

According to the algorithm $Sign$, the adversary can get the following as an estimation of $X_1,\dots,X_N$:

$$\begin{pmatrix} P_1 \\ \cdots \\ P_n \end{pmatrix} \cdot X_j^T = \begin{pmatrix} t_{P_1,j} \\ \cdots \\ t_{P_n,j} \end{pmatrix} (1 \le j \le N). \tag{9}$$

According to the key distribution, the adversary can get the following as an estimation of $X_1,\dots,X_N$ or $U_1,\dots,U_N$:

$$\begin{pmatrix} Z_{g_1} \\ \cdots \\ Z_{g_r} \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ \cdots \\ X_N \end{pmatrix} = \begin{pmatrix} V_{g_1} \\ \cdots \\ V_{g_r} \end{pmatrix} \tag{10}$$

and

$$\begin{pmatrix} C_{g_1} \\ \cdots \\ C_{g_r} \end{pmatrix} \cdot \begin{pmatrix} U_1 \\ \cdots \\ U_N \end{pmatrix} = \begin{pmatrix} Z_{g_1} \\ \cdots \\ Z_{g_r} \end{pmatrix}, \tag{11}$$

where $U_i = (u_{i,1},\dots,u_{i,N}) (1 \le i \le N)$ and $Z_{g_i} = (z_{g_i,1},\dots,z_{g_i,N}) (1 \le i \le r)$.

According to the algorithm $Verify$, the following is useful information for the adversary to find out a deceptive data packet W' with N tags $t_{W',1}, \dots, t_{W',N}$:

$$\begin{pmatrix} Z_{g_1} \\ \cdots \\ Z_{g_r} \end{pmatrix} \cdot \begin{pmatrix} t_{W',1} \\ \cdots \\ t_{W',N} \end{pmatrix} = \begin{pmatrix} V_{g_1} \\ \cdots \\ V_{g_r} \end{pmatrix} \cdot (W')^T. \tag{12}$$

As a summary, the information which the adversary can use to launch pollution attack includes

1. data packets $P_1, \dots, P_n$, each P_i with N tags $t_{P_i,1}, \dots, t_{P_i,N}, 1 \le i \le n$;
2. $C_{g_i}, Z_{g_i}, V_{g_i}$ at a compromised node g_i for $1 \le i \le r$;
3. the schemes to generate N secret vectors $X_1, \dots, X_N$ and N tags for each data packet at the source s and to generate secret vectors C_g, Z_g, V_g at a node g. Equations (9)-(12) show the relationship among the n data packets, their tags, and the secret vectors.

In the case that the adversary compromises $r \ge N$ nodes, it is very possible that the rank of the matrix $((Z_{g_1})^T, (Z_{g_2})^T, \dots, (Z_{g_r})^T)^T$ is N. And if so, the adversary can derive $X_1, \dots, X_N$ directly from (10). So in the following, we assume that the adversary is not able to compromise more than $N-1$ nodes.

4.2 The First Pollution Attack Behavior: Searching for $X_1, \dots, X_N$

Because $Y_1, \dots, Y_N$ are publicly known to all nodes, $U_1, \dots, U_N$ are privately known by s and C_g, Z_g, V_g are privately known by node g, there are two ways for an adversary to get $X_1, \dots, X_N$:

1. *The First Way*. First getting $U_1, \dots, U_N$ with (11) and C_g, Z_g, V_g held by comprised node g, and then getting $X_1, \dots, X_N$ with $U_1, \dots, U_N$ and (3).
2. *The Second Way*. Using (9) and (10) with C_g, Z_g, V_g held by comprised node g.

In the following, we will analyze the probability of the adversary getting $X_1, \dots, X_N$ using the above two ways separately.

4.2.1 The First Way

$U_1, \dots, U_N$ are the key information for the adversary to get $X_1, \dots, X_N$. We first analyze the probability that the adversary derives $U_1, U_2, \dots, U_N$ in the case of $r < N$.

Given $C_{g_1}, \dots, C_{g_r}$ and $Z_{g_1}, \dots, Z_{g_r}$, let $\mathcal{U}$ be the set of all the solutions of $U_1, \dots, U_N$ in (11). Then, we have the following Theorem 1:

Theorem 1. *In the case of $r < N$, if the adversary randomly selects an element from $\mathcal{U}$, the probability that the element is exactly the N secret vectors $U_1, U_2, \dots, U_N$ held by s is no greater than $1/p^{N(N-r)}$, where p is the size of field F_p.*

Proof. Because the adversary comprises $g_1, g_2, \dots, g_r$, it knows the secret vectors C_{g_i}, Z_{g_i} held by node g_i for $1 \le i \le n$. The only way that the adversary tries to get $U_1, U_2, \dots, U_N$ is solving the following linear equation:

$$\begin{pmatrix} C_{g_1} \\ \cdots \\ C_{g_r} \end{pmatrix} \cdot \begin{pmatrix} u_{1,i} \\ \cdots \\ u_{N,i} \end{pmatrix} = \begin{pmatrix} z_{g_1,i} \\ \cdots \\ z_{g_r,i} \end{pmatrix} \tag{13}$$

for $1 \le i \le N$, where $U_i = (u_{i,1}, \dots, u_{i,N})$. Because the rank of the matrix $((C_{g_1})^T, \dots, (C_{g_r})^T)^T$ is no greater than r, there are no less than p^{N-r} solutions in (13), further $|\mathcal{U}| \ge p^{N(N-r)}$. Hence, if the adversary randomly selects an element from $\mathcal{U}$, the probability that the element is exactly the N secret vectors $U_1, U_2, \dots, U_N$ held by s is no greater than $1/p^{N(N-r)}$. □

Let $\kappa_1 (0 < \kappa_1 < 1)$ be a small real number such that $1/p^{N(N-r)} \le \kappa_1$, which means that the probability of the adversary getting $U_1, \dots, U_N$ is small enough. From (3), getting $U_1, \dots, U_N$ is equivalent to getting $X_1, \dots, X_N$. So KEPTE is secure against the adversary getting $X_1, \dots, X_N$ with the first way on the condition that $1/p^{N(N-r)} \le \kappa_1$.

4.2.2 The Second Way

Provided that the adversary has no knowledge of $U_1, \dots, U_N$ held by s, the adversary cannot get any useful information about $X_1, \dots, X_N$ from (3). In this section, we will analyze the probability of the adversary getting $X_1, \dots, X_N$ with the second way. Given $P_1, \dots, P_n$, each P_i with N tags $t_{P_i,1}, \dots, t_{P_i,N}$, and secret vectors Z_{g_i}, V_{g_i} at node g_i for $1 \le i \le r$, let $\mathcal{X}$ be the set of all the solutions of $X_1, \dots, X_N$ that satisfy both (9) and (10). Then, we have the following Theorem 2.

Theorem 2. *In the case of $r < N$, if the adversary randomly selects an element from $\mathcal{X}$, the probability that the element is exactly the N secret vectors $X_1, \dots, X_N$ held by s is no greater than $1/p^{m(N-r)}$.*

Proof. A node g can receive enough encoded packets to decode the packets $P_1, \dots, P_n$, and the N tags $t_{P_i,1}, \dots, t_{P_i,N}$ of P_i for $1 \le i \le n$. Because X_j^T is a vector of $m+n$ dimensions, provided that m coordinates of $X_j (1 \le j \le N)$ are known, the vector $X_j (1 \le j \le N)$ is determined by (9).

So to find $X_1, \dots, X_N$ satisfying both (9) and (10), we need m coordinates of each $X_j (1 \le j \le N)$. And the rank of $((Z_{g_1})^T, (Z_{g_2})^T, \dots, (Z_{g_r})^T)^T$ is no greater than r. Similar to the proof of Theorem 1, $|\mathcal{X}| \ge p^{m(N-r)}$ and Theorem 2 holds. □

Let $\kappa_2 (0 < \kappa_2 < 1)$ be a small real number such that $1/p^{m(N-r)} \le \kappa_2$, which means that the probability of the adversary getting $X_1, \dots, X_N$ is small enough. KEPTE is secure against the adversary getting $X_1, \dots, X_N$ with the second way on the condition that $1/p^{m(N-r)} \le \kappa_2$.

4.3 The Second Pollution Attack Behavior: Finding a Deceptive Data Packet

Without the knowledge of $X_1, \dots, X_N$, (12) is the only information that the adversary can use to find a deceptive data packet W' with N tags $t_{W',1}, \dots, t_{W',N}$ that can pass the verifications at all nodes. There are the following two ways that the adversary uses:

The First Way. The adversary randomly selects $t_{W',1}, \dots, t_{W',N} \in F_p$, and then tries to inject a fake data packet W' with N tags $t_{W',1}, \dots, t_{W',N}$. So the adversary will try to find W' which satisfies (12) and $W' \notin Span\{P_1, P_2, \dots, P_n\}$, and W' with its N tags can pass the correctness check at all nodes. Let $\mathcal{W}_2$ be the set of all the deceptive data packets with $t_{W',1}, \dots, t_{W',N}$ as their tags. Let $\mathcal{W}_1$ be the set of all $W^{'}$s which not only are solutions of (12) but also satisfy $W^{'} \notin Span\{P_1, P_2, \ldots, P_n\}$, but $W^{'}$ may not pass the correctness check at some node(s). From the definition of a deceptive data packet, $\mathcal{W}_2 \subseteq \mathcal{W}_1$ holds. The adversary randomly selects an element from $\mathcal{W}_1$ and hopes that the element belongs to $\mathcal{W}_2$.

The Second Way. The adversary fixes $W^{'}$ first, and try to find $(t_{W',1}, \ldots, t_{W',N}) \in F_p^N$ to make $W^{'}$ with $t_{W',1}, \ldots, t_{W',N}$ be a deceptive data packet. Similarly to the first way, let $\mathcal{W}_3$ be the set of all the solutions $(t_{W',1}, \ldots, t_{W',N})$ in (11) with the fixed $W^{'}$. Then, the adversary randomly selects an element $(t_{W',1}, \ldots, t_{W',N})$ for $W^{'}$ from $\mathcal{W}_3$ and hopes $W^{'}$ with the tags $(t_{W',1}, \ldots, t_{W',N})$ is a deceptive data packet, i.e., $t_{W',i} = W^{'} \cdot X_i^T (1 \leq i \leq N)$.

In the following, we analyze the probability of the adversary finding a deceptive data packet with the above two ways.

4.3.1 The First Way

Now, we analyze the probability of the adversary finding a deceptive data packet with the first way.

Theorem 3. *Suppose that the adversary randomly selects an element $W^{'}$ from $\mathcal{W}_1$. In the case of $r < N$, the probability of $W^{'}$ belonging to $\mathcal{W}_2$ is no greater than $\frac{1}{p^{N-r}}$.*

Proof. A deceptive data packet $W^{'}$ with given N tags $(t_{W',1}, \ldots, t_{W',N})$ satisfies

$$\begin{pmatrix} t_{W',1} \\ \cdots \\ t_{W',N} \end{pmatrix} = \begin{pmatrix} X_1 \\ \cdots \\ X_N \end{pmatrix} \cdot (W^{'})^T. \tag{14}$$

Any solution $W^{'}$ of (14) will pass the correctness verification at each node g with (Z_g, V_g). Because $W^{'}$ is an $m+n$ dimensional vector over F_p and there are p^n elements in $Span\{P_1, \ldots, P_n\}$, the order of $\mathcal{W}_2$ is $p^{m+n-N} - p^n$.

Because the rank of the matrix $((Z_{g_1})^T, \ldots, (Z_{g_r})^T)^T$ is no greater than r and there are p^n elements in $Span\{P_1, \ldots, P_n\}$, the order of $\mathcal{W}_1$ is no less than $p^{m+n-r} - p^n$.

So if the adversary randomly selects an element $W^{'}$ from $\mathcal{W}_1$, the probability of $W^{'}$ belonging to $\mathcal{W}_2$ is $\frac{p^{m+n-N}-p^n}{p^{m+n-r}-p^n} = \frac{p^{m-N}-1}{p^{m-r}-1} \leq \frac{1}{p^{N-r}}$. □

According to Theorem 3, when the adversary randomly selected p^{N-r} elements from $\mathcal{W}_1$, it expects to find no more than one element (i.e., one data packet) that can pass the verification at all nodes, i.e., it belongs to $\mathcal{W}_2$. Let $\kappa_3 (0 < \kappa_3 < 1)$ be a small real number such that $1/p^{N-r} \leq \kappa_3$, which means the probability of the adversary finding an element in $\mathcal{W}_2$ is negligible. So, KEPTE can prevent the adversary from getting a deceptive data packet on the condition that $1/p^{N-r} \leq \kappa_3$.

4.3.2 The Second Way

If the adversary seeks for a deceptive data packet with the second way, we have the following Theorem 4.

Theorem 4. *Given $W^{'} \in F^{m+n}$ as a data packet, suppose that the adversary randomly selects an element $(t_{W',1}, \ldots, t_{W',N})$ from $\mathcal{W}_3$ as N tags of $W^{'}$. In the case of the number of compromised nodes $r < N$, the probability of $t_{W',i} = W^{'} \cdot X_i^T (1 \leq i \leq N)$ is no greater than $\frac{1}{p^{N-r}}$.*

Proof. Omitted, as it can be proved in the similar way as the proof of Theorem 3. □

Similar to the analysis of the first way, on the same condition as the first way that $1/p^{N-r} \leq \kappa_3$, the probability is small enough to keep the adversary from finding a deceptive data packet with the second way.

4.4 Tag Pollution Attack

In a type of schemes based on key predistribution [13], [14], [15], N keys are used to generate N tags for a data packet at the source s and each node g just possesses part of the N keys. The adversary only modifies one or several tags of a data packet and makes no modification to the data packet. If a node receives the data packet with the modified tags and does not hold the keys corresponding to the modified tags, the data packet will pass its verification because the node can only verify the unmodified tags corresponding to the keys it holds. If a node receives the encoded data packets with the incorrect tags and exactly holds the keys corresponding to the incorrect tags, those data packets will be discarded.

Tag pollution attack may waste bandwidth greatly. The schemes in [13], [14] are susceptible to tag pollution attack, while the MacSig scheme in [15] adopts an extra method based on PKC to resist on tag pollution attack and requires a large field of size 128 bits. So the computational complexity of the MacSig scheme is much higher than the schemes in [13], [14].

KEPTE uses the algorithm *Verify* to check the correctness of a packet with N tags. The following Theorem 5 presents the probability of a correct data with modified tag(s) passing the correctness check at a node.

Theorem 5. *If the adversary modifies $d (1 \leq d \leq N)$ tags of a data packet, the probability that the d modified tags do not be checked out by a node during the verification process is $1/p^d$.*

Proof. Suppose that a node g with (Z_g, V_g) verifies the correctness of a data packet W with tags $t_{W,1}, \ldots, t_{W,N}$. If the ith coordinate of the vector $Z_g \in F_p^N$ is zero, the ith tag of W will not be checked; otherwise, it will be checked. Because the ith coordinate of Z_g is randomly selected from F_p, the probability of it being zero is $1/p$. So the probability of the ith tag of W being not checked out is $1/p$ and the probability that all d modified tags are not checked out by g in the verification of W is $1/p^d$. □

$\frac{1}{p^d}$ is maximized when $d = 1$. Hence, in KEPTE, it is most advisable for the adversary to modify only one of the N tags. If $F_p = F_{2^8}$ and $d = 1$, $\frac{1}{p^d} \approx 0.4\%$, and if $F_p = F_{2^{16}}$ and $d = 1$, $\frac{1}{p^d} \approx 0.00153\%$. The probability of a packet with modified tag(s) passing the checks at two nodes is very small. So KEPTE is efficient to prevent tag pollution attack under common used setting of network coding.

4.5 Security Summary

The essence of pollution attack is to find a deceptive data packet defined in Definition 1. From Sections 4.1, 4.2, and 4.3, KEPTE can prevent the adversary to get a deceptive data packet from the two attack behaviors if the following three conditions are satisfied: 1) $1/p^{N(N-r)} \le \kappa_1$; 2) $1/p^{m(N-r)} \le \kappa_2$; 3) $\frac{1}{p^{N-r}} \le \kappa_3$, where κ_1, κ_2, and κ_3 are security parameters. In particular, setting $\kappa_1 = \kappa_2 = \frac{1}{10^{100}}$ and $\kappa_3 = \frac{1}{10^{18}}$, the following two common used settings in practical network coding both satisfy the above three conditions: 1) $F_p = F_{2^8}$, $r \le N-8$, and $m \ge 128$, and 2) $F_p = F_{2^{16}}$, $r \le N-5$, and $m \ge 64$.

If the adversary cannot obtain a deceptive data packet, it may launch tag pollution, i.e., it modifies the tag(s) of a correct data packet. A node verifies the correctness of a data packet using the algorithm $Verify$, and the probability that a packet with modified tag(s) passes the verification at a node is no greater than $1/p$. If $F_p = F_{2^8}$, the probability is no greater than $1/256 \approx 0.4\%$. In the case that $F_p = F_{2^8}$, the probability that the packet passes the verifications at two nodes is no greater than $\frac{1}{256\times256} \approx 0.00153\%$. If the system needs a higher security level, it can select a larger field, for example, $F_p = F_{2^{16}}$. If $F_p = F_{2^{16}}$, the probability that a packet with modified tag(s) passes the verification at a node is no greater than 0.00153 percent.

As a summary of this section, provided that the three security conditions 1, 2, and 3 stated in the first paragraph of this section are satisfied, all intermediate nodes and sinks are able to resist on pollution attack and tag pollution attack with the protection of KEPTE.

5 Performance and Application

In this section, we will analyze the performance of KEPTE in terms of computational complexity, communication overhead, and storage overhead. We also discuss the application of KEPTE to a practical network coding case. The comparison of KEPTE with some typical schemes is presented as supplementary, available online.

5.1 Performance Analysis

5.1.1 Computational Complexity

The computational complexity of KEPTE consists mainly of two parts: 1) the key and tag generation at the source; 2) the key generation and the correctness verification of packets at each node g:

1. *At the source*. The source uses G_1 to generate N vectors $Y_1, \ldots, Y_N$ and uses G_2 to generate N vectors $U_1, \ldots, U_N$. To get the N vectors $X_1, \ldots, X_N$ from (3), the source needs to solve a group of linear equations, whose computational complexity is $O(N^3 + N^2(m+n))$. According to the algorithm $Sign$, $N(m+n)$ multiplications and $N(m+n-1)$ additions over field F_p are needed for the tag generation of each data packet at a node, whose computational complexity is $O(N(m+n))$.

2. *At each node g*. The node g uses G_1 to generate N vectors $Y_1, \ldots, Y_N$ and uses G_3 to generate the vector C_g. Then, with (5), $(N-1)(m+n)$ additions and $N(m+n)$ multiplications over F_p are needed to generate a vector V_g. For the verification of a data packet at a node, according to the algorithm $Verify$, $N+m+n$ multiplications and $m+n+N-2$ additions over F_p are performed. The computational complexity for the correctness verification of a data packet at a node is $O(m+n+N)$.

We can see from Table 1 in the supplementary, available online, that the computational complexity of KEPTE is always less than the scheme in [13], while the time for the verification of a packet with the scheme in [13] is at microsecond level under their experimental setup. So KEPTE is computational efficient for practical application.

5.1.2 Communication and Storage Overhead

The communication overhead includes the tags appended to each data packet and keys distributed to the source s and each node g.

The size of tags appended to each data packet is $N\lceil\log_2 p\rceil$ bits. The ratio of the size of N tags to the size of a data packet is $\frac{N}{m+n}$. Suppose a data block is no less than 128 bytes = 1,024 bits and the system can at most tolerate t nodes to be compromised. From the security analysis, if $F_p = F_{2^8}$, $t = N-8$ and the size of redundancy appended to a data packet is $N\lceil\log_2 p\rceil = 8N = 8(t+8)$ bits. If $F_p = F_{2^{16}}$, $t = N-5$ and the size of redundancy appended to a data packet is $N\lceil\log_2 p\rceil = 16N = 16(t+5)$ bits.

The lengths of the keys distributed to the source s and each node g are $N \times Len_{key}$ bits and $N\lceil\log_2 p\rceil + Len_{key}$ bits, respectively, where Len_{key} is the number of bits of a key.

In the "Setup" phase of KEPTE, the source needs to store N secret vectors $X_1, \ldots, X_N \in F_p^{m+n}$ to generate N tags for each of n data packets. The total size of the N secret vectors is $N(m+n)\lceil\log_2 p\rceil$ bits and equals to N data packets. After the source generating N tags for each of the n data packets, the N secret vectors can be destroyed and not stored anymore. For each node g, it needs to store two secret vectors Z_g, V_g, each of size $(N+m+n)\lceil\log_2 p\rceil$ bits.

5.2 A Practical Case in the Wired Network

In this section, we take a practical network coding case from [20] to show the practicality of KEPTE. Suppose that a wired network consists of 89 nodes, $F_p = F_{2^8}$ and the size of an IP packet is set to 1,400 Bytes, i.e., $m+n = 1{,}400$, and the system tolerates 32 nodes to be compromised and $N = 40$.

During key distribution, KDC distributes to the source $N = 40$ keys $b_1, \ldots, b_{40}$, the total size of $N \times Len_{key} = 5{,}120$ bits, and KDC distributes to each node g the keys Z_g, c_g, total size of $N + Len_{key} = 448$ bits.

Before sending the n data packets, the source needs to solve a group of $N = 40$ linear equations over F_{2^8} to get 40 secret vectors $X_1, X_2, \ldots, X_{40}$ according to (3). The computational complexity of solving (3) is $O(N^3 + N^2(m+n))$ over F_{2^8}. In addition, during the tag generation of the n data packets, the source needs to store those N vectors $X_1, X_2, \ldots, X_{40}$, total size of 54.7 KB approximately and which can be removed from the memory once the tags of these n data packets being generated. The memory of a modern PC usually has several GBs. So the storage cost of $X_1, X_2, \ldots, X_{40}$ can be neglected.

To verify the correctness of a received data packet, a node g with c_g, Z_g in the network needs to get V_g using equation $V_g = C_g \cdot ((Y_1)^T, \ldots, (Y_N)^T)^T$, where $C_g = G_3(c_g) \in F_p^N$ and the size of V_g equals to the size of a data packet. The size of the tags appended to each data packet is 320 bits, and the ratio of the tag size to the length of a data packet is $\frac{40}{1{,}400} \approx 2.9\%$.

From the analysis above, KEPTE can applied to this case in [20] with a reasonable overhead.

In P2P systems, any node could upload and download data packets as source and sink simultaneously, but its basic function is to distribute a file from a node (say u) to other nodes. KEPTE can be applied to P2P systems with a little modification, i.e., regarding u as the source node and all other nodes as intermediate nodes or destination nodes.

6 Conclusion

This paper presented some insights to linear network coding and proposed a key predistribution-based tag encoding scheme, KEPTE, which is used to secure network coding against pollution attack and tag pollution attack. This paper also quantitatively gave the security analysis of KEPTE and compared its performance and overhead with other schemes. The main advantages of KEPTE can be summarized as: 1) It is computationally efficient; 2) All intermediate nodes and sinks are able to detect pollution attack and tag pollution attack; 3) When the size of redundancy appended to each data packet is $N\lceil\log_2 p\rceil$ bits, it allows no more than $N-\theta(N\geq\theta+1)$ nodes to be compromised, where θ is a security parameter. Roughly, θ mainly depends on the size p of the selected field F_p. In practical network coding, F_{2^8} and $F_{2^{16}}$ are two common used fields. If $F_p=F_{2^8}$, θ can be set to 8, and if $F_p=F_{2^{16}}$, θ can be set to be 5. Finally, discussions on a practical network coding case showed the practicality of KEPTE.

The fault-tolerant ability of KEPTE is related to the size of the redundancy appended to each data packet. How to release this constraint may be our future work.

Acknowledgments

This paper was partially supported by the Fundamental Research Funds for the Central Universities (No. WK0110000007) and National Nature Science Foundation of China (No. 61073038). Chau Yuen's work was partly supported by the International Design Center (No. IDG31100102 and IDD11100101).

References

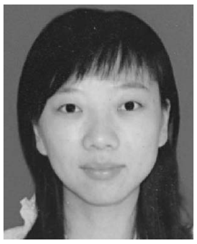
[1] R. Ahlswede, N. Cai, S.-Y.R. Li, and R.W. Yeung, "Network Information Flow," *IEEE Trans. Information Theory,* vol. 46, no. 4, pp. 1204-1216, July 2000.
[2] S.R. Li, R. Yeung, and N. Cai, "Linear Network Coding," *IEEE Trans. Information Theory,* vol. 49, no. 2, pp. 371-381, Feb. 2003.
[3] T. Ho, M. Medard, R. Koetter, D.R. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Trans. Information Theory,* vol. 52, no. 10, pp. 4413-4430, Oct. 2006.
[4] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Medard, and M. Effros, "Resilient Network Coding in the Presence of Byzantine Adversaries," *IEEE Trans. Information Theory,* vol. 54, no. 6, pp. 2596-2603, June 2008.
[5] A. Yun, J. Cheon, and Y. Kim, "On Homomorphic Signatures for Network Coding," *IEEE Trans. Computers,* vol. 59, no. 9, pp. 1295-1296, Mar. 2010.
[6] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An Efficient Signature-Based Scheme for Securing Network Coding against Pollution Attacks," *Proc. IEEE INFOCOM,* Apr. 2008.
[7] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin, "Secure Network Coding over the Integers," *Proc. 13th Int'l Conf. Practice and Theory in Public Key Cryptography (PKC '10),* May 2010.
[8] Y. Jiang, H. Zhu, M. Shi, X. Shen, and C. Lin, "An Efficient Dynamic-Identity Based Signature Scheme for Secure Network Coding," *Computer Networks: The Int'l J. Computer and Telecomm. Networking,* vol. 54, no. 1, pp. 28-40, Jan. 2010.
[9] M. Krohn, M. Freedman, and D. Mazieres, "On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution," *Proc. IEEE Symp. Security and Privacy,* May 2004.
[10] F. Zhao, T. Kalker, M. Medard, and K.J. Han, "Signatures for Content Distribution with Network Coding," *Proc. IEEE Int'l Symp. Information Theory,* June 2007.
[11] G. Gkantsidisand and P. Rodriguez, "Cooperative Security for Network Coding File Distribution," *Proc. IEEE INFOCOM,* Apr. 2006.
[12] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical Defenses Against Pollution Attacks in Wireless Network Coding," *ACM Trans. Information and System Security,* vol. 14, no. 1, article 7, May 2011.
[13] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-Based Integrity for Network Coding," *Proc. Int'l Conf. Applied Cryptography and Network Security,* June 2009.
[14] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An Efficient Scheme for Securing XOR Network Coding against Pollution Attacks," *Proc. IEEE INFOCOM,* Apr. 2009.
[15] P. Zhang, Y. Jiang, C. Lin, H. Yao, A. Wasef, and X.S. Shen, "Padding for Orthogonality : Efficient Subspace Authentication for Network Coding," *Proc. IEEE INFOCOM,* Apr. 2011.
[16] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "RIPPLE Authentication for Network Coding," *Proc. IEEE INFOCOM,* Mar. 2010.
[17] A. Le and A. Markopoulou, "Cooperative Defense against Pollution Attacks in Network Coding Using SpaceMac," *IEEE J. Selected Areas in Comm. on Cooperative Networking Challenges and Applications,* vol. 30, no. 2, pp. 442-449, Feb. 2012.
[18] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "Identifying Malicious Nodes in Network-Coding-Based Peer-to-Peer Streaming Networks," *Proc. IEEE INFOCOM,* Mar. 2010.
[19] R. Koetter and F.R. Kschischang, "Coding for Errors and Erasures in Random Network Coding," *IEEE Trans. Information Theory,* vol. 54, no. 8, pp. 3579-3591, Aug. 2008.
[20] P. Chou, Y. Wu, and K. Jain, "Practical Network Coding," *Proc. Allerton Conf. Comm., Control, and Computing,* Oct. 2003.
[21] H. Wang, "Cover-Free Families and Their Applications in Network Security," technical report, Division of Math. Sciences School of Physical and Math. Sciences, Nanyang Technological Univ., 2009.
[22] A. Le and A. Markopoulou, "TESLA-Based Defense against Pollution Attacks in P2P Systems with Network Coding," *Proc. IEEE Int'l Symp. Network Coding (NetCod),* July 2011.
[23] N. Cai and R.W. Yeung, "Network Error Correction, Part 2: Lower Bounds," *Comm. in Information and Systems,* vol. 6, no. 1, pp. 37-54, 2006.
[24] Z. Zhang, "Network Error Correction Coding in Packetized Networks," *Proc. IEEE Information Theory Workshop,* Oct. 2006.
[25] F. Oggier and H. Fathi, "An Authentication Code against Pollution Attacks in Network Coding," *IEEE/ACM Trans. Networking,* vol. 19, no. 6, pp. 1587-1596, Mar. 2011.
[26] D. Petrovic, K. Ramchandran, and J. Rabaey, "Overcoming Untuned Radios in Wireless Networks with Network Coding," *IEEE Trans. Information Theory,* vol. 52, no. 6, pp. 2649-2657, June 2006.
[27] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale File Distribution," *Proc. IEEE INFOCOM,* Mar. 2005.
[28] Y. Zhu, B. Li, and J. Guo, "Multicast with Network Coding in Application Layer Overlay Networks," *IEEE J. Selected Areas in Comm. on Recent Advances in Service Overlay Networks,* vol. 22, no. 1, pp. 107-120, Jan. 2004.
[29] Q. Li, J.C.S. Lui, and D.-M. Chiu, "On the Security and Efficiency of Content Distribution via Network Coding," *IEEE Trans. Dependable and Secure Computing,* vol. 9, no. 2, pp. 211-221, Jan. 2011.
[30] X. Wu, Y. Xu, L. Xiang, and W. Xu, "A Hybrid Scheme against Pollution Attack in Network Coding," *Proc. IEEE Int'l Symp. Network Coding (NetCod),* July 2011.
[31] W. Stallings, *Cryptography and Network Security,* fifth ed. Prentice Hall, 2011.

[32] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for Key Management - Part 1: General," NIST Special Publication 800-57, Mar. 2007.

Xiaohu Wu received the master's degree from the University of Science and Technology of China (USTC) in June 2012. He is currently working toward the PhD degree at EURECOM in Sophia-Antipolis, France. His research interests include security in network coding and pricing of sharing resources, especially for cloud computing resources.

Yinlong Xu received the BS degree in mathematics from Peking University in 1983, and the MS and PhD degrees in computer science from the University of Science and Technology of China (USTC), in 1989 and 2004, respectively. He is currently a professor with the School of Computer Science and Technology at USTC. Before that he served as an assistant professor, a lecturer, and an associate professor at the Department of Computer Science and Technology at USTC . He is currently leading a group of research students in doing some networking and high-performance computing research. His research interests include network coding, wireless network, combinatorial optimization, design and analysis of parallel algorithm, parallel programming tools, and so on. He received the Excellent PhD Advisor Award of Chinese Academy of Sciences in 2006.

Chau Yuen received the BEng and PhD degrees from Nanyang Technological University, Singapore, in 2000 and 2004, respectively. He was a postdoc fellow in Lucent Technologies Bell Labs, Murray Hill during 2005. He was also a visiting assistant professor of Hong Kong Polytechnic University in 2008. During the period of 2006-2010, he was at the Institute for Infocomm Research (Singapore) as a senior research engineer. He joined Singapore University of Technology and Design as an assistant professor from June 2010. He also serves as an associate editor for *IEEE Transactions on Vehicular Technology*. In 2012, he received the IEEE Asia-Pacific Outstanding Young Researcher Award.



Liping Xiang received the BS degree from the Department of Information and Computational Science, Anhui University, China, in 2007. She is currently working toward the PhD degree at the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. Her research interests include distributed storage system, data recovery, and network coding.