Pollution Attacks and Defenses in Wireless Inter-flow Network Coding Systems

Jing Dong, Reza Curtmola, *Member, IEEE*, Cristina Nita-Rotaru, *Senior Member, IEEE*, and David K. Y. Yau, *Member, IEEE*

Abstract—We study data pollution attacks in wireless *inter-flow network coding* systems. Although several defenses for these attacks are known for intra-flow network coding systems, none of them are applicable to inter-flow coding systems. We formulate a model for inter-flow network coding that encompasses all the existing systems, and use it to analyze the impact of pollution attacks. Our analysis shows that the effects of pollution attacks depend not only on the network topology, but also on the location and strategy of the attacker nodes. We propose CodeGuard, a reactive attestation-based defense mechanism that uses efficient bit-level traceback and a novel cross-examination technique to unequivocally identify attacker nodes. We analyze the security of CodeGuard and prove that it is always able to identify and isolate at least one attacker node on every occurrence of a pollution attack. We analyze the overhead of CodeGuard and show that the storage, computation, and communication overhead are practical. We experimentally demonstrate that CodeGuard is able to identify attacker nodes quickly (within 500 ms) and restore system throughput to a high level, even in the presence of many attackers, thus preserving the performance of the underlying network coding system.

Index Terms—Pollution attacks, wireless networks, inter-flow network coding.

1 Introduction

Network coding has been shown to be effective in improving throughput in wireless networks. The core principle of network coding is that intermediate nodes actively mix (or *code*) input packets to produce output packets. The active mixing of packets increases packet diversity in the network, resulting in fewer redundant transmissions and better use of network resources.

Unfortunately, the very nature of packet mixing makes network coding systems vulnerable to a severe security threat known as *pollution attacks*, in which attackers inject corrupted packets into the network. Although packet injection is not a new attack, its impact on network coding is devastating. Specifically, as long as there is one corrupted packet that an intermediate node uses during the coding process, then all the packets that are coded and forwarded by the node will be corrupted. The result is an epidemic propagation of corrupted packets, as further nodes code and forward more corrupted packets.

Based on how packets are selected for coding, existing network coding systems can be classified into *intra-flow coding systems* [1]–[6], where nodes mix packets within the same flow only (*i.e.*, packets coming from one source), and *inter-flow coding systems* [7]–[14], where nodes mix packets across different flows (*i.e.*, packets coming from multiple sources), but not within the same flow. In intra-flow coding systems, coding occurs at all the nodes. In contrast, in inter-flow coding systems, coding occurs only at nodes at the intersection of flows. Nodes that do not perform coding simply forward packets unchanged.

Network coding introduces new challenges in the detection of corrupted (or polluted) packets, because it allows intermediate (possibly malicious) nodes to actively mix packets. Protection mechanisms proposed in the context of traditional routing (*e.g.*, monitoring [15] or authentication [16]) are not effective or feasible against pollution attacks under network coding. Monitoring-based corrupted packet detection requires nodes to be able to compare that the forwarded packets by their neighbors are coded from incoming packets. However, in network coding there are many scenarios in which upstream nodes cannot decode the packets coded and forwarded by downstream nodes and therefore cannot detect if coding was performed correctly. In intra-flow network coding, traditional authentication solutions based on digital signatures would require intermediate nodes to verify that each received coded packet is a valid combination of plain packets from the source. A brute force approach in which the source generates and disseminates signatures of all possible combinations of the plain packets has prohibitive computation/communication costs.

To address this challenging problem two approaches have been taken by the security and information theory research communities: providing an end-to-end defense mechanism and providing protection at intermediate nodes. The first approach provides recovery at the receivers by encoding redundant information at coding time [17]. The second approach primarily detects and filters packets at intermediate nodes preventing the propagation of corrupted packets in the network by using specialized homomorphic digital signatures, MACs, or hash functions [18]–[26].

Inter-flow network coding introduces additional vulnerabili-

Manuscript received ???; revised ???; accepted ???; published online ???

- *J. Dong, C. Nita-Rotaru, and D. K. Y. Yau are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907. E-mail: jingdong0@gmail.com, crisn@cs.purdue.edu, yau@cs.purdue.edu. D. Yau is also with the Advanced Digital Sciences Center, Singapore.*
- *R. Curtmola is with the Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102. E-mail: crix@njit.edu.*

ties because coding across flows results in complex inter-flow data dependencies and allows pollution attacks in one flow to contaminate other flows. The impact of such an attack is thus more devastating than in intra-flow coding systems. For example, by injecting a corrupted packet into a flow f, an attacker can corrupt all the flows coded with f. Similarly, an attacker node at the intersection of several flows can corrupt all the intersecting flows using a single incorrectly coded packet. The use of inter-flow coding also makes pollution attacks more challenging to defend against. Unlike intra-flow network coding, where coded packets are combinations of packets from the same source, inter-flow coding combines packets from different sources. Given these fundamental differences, cryptographic solutions proposed for intra-flow coding which assume a single point of trust (the source of the flow) are not applicable for inter-flow coding systems because there is no single source that is trusted for all the packets. Ideally, one needs a signature scheme that is homomorphic for XOR operations and that could be used to verify packets signed by different (and independent) sources. Unfortunately, we are not aware of such a cryptographic primitive (we emphasize that aggregate signatures [27] cannot be used, as they require the presence of the original packets used to obtain a coded packet).

In this paper, we focus on pollution attacks and defenses in inter-flow network coding systems. While most of the prior work in intra-flow network coding systems focused on detecting and filtering polluted packets, in inter-flow coding systems, an additional critical goal is to swiftly identify attacker nodes. This is because unlike intra-flow coding, where there are multiple paths from source to destination, inter-flow coding systems are usually based on single-path routing; hence it is critical to identify and avoid attacker nodes. To our knowledge, the only work that focuses on tracing pollution attackers is SpaceMac [28], which aims at identifying attackers in intra-flow network coding by relying on a trusted identity and pre-distributed keys. Instead, we are interested in mitigating pollution attacks in inter-flow coding systems by using a minimal set of assumptions. We propose CodeGuard, a defense mechanism that combines proactive node attestation and reactive traceback to identify the attacker nodes unequivocally.

Our contributions are:

• We formulate a general model for inter-flow network coding, which encompasses all the existing practical systems including [7]–[14]. We classify pollution attacks in inter-flow coding systems based on the type of packets injected by the attacker and the attacks' impact on flows in the network. In particular, we identify a new attack, *cross-flow pollution*, which exploits the coding dependencies among flows to propagate pollution across flows.

• We propose CodeGuard, the first defense mechanism against packet pollution attacks in inter-flow coding systems. The main novelty of CodeGuard consists of: (i) a mechanism which allows nodes to attest the correctness of their prior coding operations and which incurs a constant bandwidth overhead regardless of the number of coding nodes involved, and (ii) a *bit-level traceback* and *cross-examination* technique that efficiently tracks the history of a corrupted packet and identifies attacker nodes unequivocally. The proposed technique exploits two unique properties of inter-flow coding: the bit independence in the coding and decoding operations, and the sufficiency of packet consistency for the correctness of packet decoding.

• Using a representative inter-flow coding system, we quantify the impact of pollution attacks, and evaluate the efficiency and effectiveness of CodeGuard as a defense strategy. Our results show that due to cross-flow pollution effects, a pollution attack can cause nearly zero throughput for a significant number of affected flows, even if the attacker is not directly on the path of such a victim flow. Under a random network setting, the overall throughput of an unprotected network experiences a steady and severe decrease as the number of attackers increases. On the other hand, CodeGuard can successfully restore the system throughput to an extremely high level even if there are many attacker nodes present. This is due to CodeGuard's ability to identify and isolate attacker nodes quickly (within 500 ms of attack). Furthermore, the overhead for both the proactive and reactive components of CodeGuard is small.

Roadmap: Section 2 overviews related work. Section 3 presents a general model for inter-flow coding systems. Section 4 presents a detailed analysis of pollution attacks in inter-flow coding systems. Section 5 presents our pollution defense, CodeGuard. Section 6 presents simulation experimental results. Finally, Section 7 concludes the paper.

2 Related Work

To the best of our knowledge, there is no previous work that addresses pollution attacks in inter-flow network coding systems. We review related work on secure routing, defenses against pollution attacks in intra-flow network coding systems, tracing attackers, and network error correction coding.

Secure packet forwarding. There has been significant work on designing secure routing protocols in wireless networks, including *secure route establishment* and *secure packet forwarding*. Secure route establishment [16], [29], [30] protects the route selection process from being tampered by attacker nodes. Since pollution attacks are directed against packet forwarding, secure route establishment is complementary to our work. Closely related to our work is secure packet forwarding, which addresses packet injection and dropping attacks [15], [31]–[33]. However, they are designed for traditional routing protocols without the use of coding. With network coding, nodes cannot easily verify the correctness of their received packets. Hence, monitoring-based defenses (*e.g.*, watchdog [15]) are no longer effective, as an upstream node usually cannot decode packets coded by a downstream node.

Pollution attacks under intra-flow network coding. Previous research on defending against pollution attacks has focused exclusively on intra-flow coding systems. In such systems [1]–[3], [5], [6], only packets belonging to the same flow are coded together using random linear combinations. Solutions to packet pollution attacks for intra-flow coding systems can be categorized into cryptographic approaches and information theoretic approaches. Cryptographic approaches rely on

specialized homomorphic hash functions [18], [19], message authentication codes (MACs) [20], [25], [34], or homomorphic digital signatures [21]–[24], [26] to allow intermediate nodes to filter out polluted packets. Other approaches to thwart pollution attacks include checking if coded packets belong to the subspace spanned by the original packets ("null keys" [35]) and using time asymmetry and lightweight linear checksums (DART [36], [37]). Information theoretic approaches do not try to filter out polluted packets at the intermediate nodes. Rather, they either encode enough redundant information into packets so that receivers can detect pollution [38], or use a distributed protocol to allow the receivers to tolerate the pollution [39].

A key assumption in all these approaches is that the source node is trusted, which is natural for intra-flow coding since the input packets are all from the same source node. However, in inter-flow coding systems, packets from different source nodes are coded together, and any source node can be potentially malicious against other flows. Consequently, the solutions for intra-flow coding systems do not apply in inter-flow coding systems. The only exception is the work of Agrawal *et al.* [40], who introduce a security model and propose a generic construction for the problem of preventing pollution attacks in multi-source network coding, in which a source may act adversarially against other sources. They consider a many-to-many communication model, in which multiple sources disseminate files to multiple destinations, intermediate nodes combine file fragments from different sources, and every destination is interested in receiving the files from all the sources. This model is different than the one-to-one communication model (*i.e.*, multiple unicasts, with each destination interested in the files of a single source) considered by the inter-flow network coding literature [7]–[14] and also by our paper. As acknowledged by the authors, the practicality of security in this model is limited, as the network coding overhead grows linearly with the number of files of all the sources in the network; especially in a wireless network, this may reduce the benefits provided by network coding.

Tracing attackers. A related line of work is that which, in addition to preventing pollution attacks, takes one step further and tries to locate the malicious nodes that introduce corrupted packets. Le and Markopoulou [28] propose SpaceMac, a scheme for locating polluting attackers in intra-flow network coding systems, under the assumption of a central entity, the controller, which knows the complete network topology. The scheme is based on a novel homomorphic MAC scheme and on a previously proposed non-repudiation protocol [41]. Each node in the network is required to send several MAC tags with each transmitted packet. Any inconsistency between the packet and the associated tags allows downstream nodes to identify pollution attacks and alert the controller. The controller then asks each node in the network to report information about packets received from its upstream nodes and, based on these reports, the controller is able to identify all the polluting nodes. The scheme requires an initialization phase in which each node shares a secret key with the controller; for the non-repudiation protocol, the controller serves as a trusted intermediary to distribute a set of secret keys between each node and its downstream neighbors. Instead, we are interested in mitigating pollution attacks in inter-flow coding systems and in using a minimal set of assumptions (*i.e.*, no need for key pre-distribution in the initialization phase and no need for a trusted controller entity).

Another line of work that bears some similarities with our traceback procedure is that of IP traceback based on probabilistic packet marking (PPM) [42], [43]. In PPM, packets are marked probabilistically with information about the IP addresses of the routers they traverse. This allows a victim of a DDoS attack to trace the attack back to its source. Sattari *et al.* [44] combine a PPM scheme with network coding by marking packets with linear combinations of the router IDs, instead of individual IDs as in traditional PPM, which allows to reduce the number of packets required to reconstruct the attack paths.

Network error correction coding. Recent work [45]–[48] has developed a network error correction coding theory for detecting and correcting corrupted packets in network coding systems. In principle, the network error correction coding theory is parallel to classic coding theory for traditional communication networks, and also exhibits a fundamental trade-off between coding rate (bandwidth overhead of coding) and the error correction ability. Such schemes have limited error correcting ability and are inherently oriented toward network environments where errors only occur infrequently. In an adversarial wireless environment, the attackers are capable of injecting a large number of polluted packets that can easily overwhelm the error correction scheme and result in incorrect decoding.

3 System Model

In this section, we present a general model for inter-flow coding systems that encompasses all previously proposed systems [7]–[14], and discuss in details the packet forwarding and coding component, which is the target of pollution attacks. The attacks in Sec. 4 and the proposed defense in Sec. 5 are based on this general model.

3.1 Inter-flow Coding System Overview

The main idea of inter-flow coding is to leverage the abundant packet overhearing opportunities in wireless networks in order to reduce redundant transmissions, and to improve performance by having a node combine multiple unicasts into a single broadcast. For example, in Fig. 1(a), node A needs to deliver packet m_i to each neighboring node R_i. If each R_i has *overheard* all the other packets except m_i, node A can code (by XOR-ing) all the packets together and broadcast the resulting coded packet, from which every R_i can recover its desired m_i.

Like in all proposed inter-flow network coding systems for wireless networks [7]–[14], we consider a model in which a flow is defined between a source and a destination node (*i.e.*, a destination node is interested in receiving packets from a single source). However, nodes can leverage network coding and mix packets from different individual flows.

In general, an inter-flow coding system consists of three components: path selection, coding opportunity discovery,

and packet forwarding. The path selection component is responsible for selecting data delivery paths for flows. Coding opportunity discovery is responsible for deciding, for each node on the path of one or more flows, whether the node will perform coding, decoding, or simply packet forwarding for the flow(s). The packet forwarding component is responsible for the actual coding/forwarding of packets. Path selection and coding opportunity discovery may be performed independently [7], [8], [12], [13], or in an integrated manner [9]–[11], [14] in order to increase the overall coding opportunities.

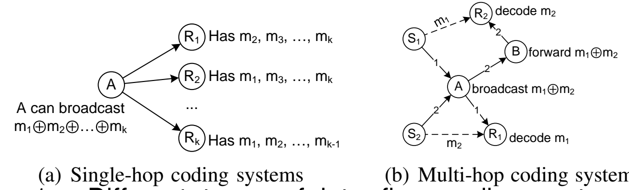
Packets transmitted in the network can be either *plain packets* or *coded packets*. A plain packet is an (uncoded) packet as sent by the source node. A *coded packet* is a bit-wise XOR of a set of plain packets from distinct flows, denoted as $e = m_1 \oplus m_2 \oplus \cdots \oplus m_k$, where each m_i is a plain packet.[1] Since bit-wise XOR is equivalent to addition modulo 2, we also write e as $e = \sum_{1 \le i \le k} m_i$, where the summation is performed modulo 2. We note that all the existing inter-flow coding systems use XOR as the coding operation.

Nodes on the path of one or more flows can be classified as *forwarding*, *coding*, or *decoding* nodes. A *forwarding* node simply forwards received coded/plain packets unmodified to its downstream node. For example, in Fig. 1(b) node B acts as a forwarder for the flow from source S_2 to receiver R_2. A *coding* node is a node that lies at the intersection of two or more flows. It codes received packets from the different flows into a single coded packet, which it broadcasts to the downstream nodes. For example, in Fig. 1(b) node A is on the flows from S_1 to R_1 and from S_2 to R_2. After receiving m_1 and m_2, node A produces a coded packet $e = m_1 \oplus m_2$ and broadcasts it to R_1 and B for both flows. A *decoding* node decodes a received coded packet by XOR-ing it with previously overheard plain/coded packets. For example, in Fig. 1(b) node R_1 decodes $e = m_1 \oplus m_2$ by using the overheard packet m_2 to compute $e \oplus m_2$ and recover m_1. A decoding node may also perform *partial decoding* which results in another coded packet. For example, if a node receives a coded message $e_1 = m_1 \oplus m_2 \oplus m_3$ and overhears m_1, the node can perform a partial decoding of e_1 by computing $e_1 \oplus m_1$ to obtain another (more simple) coded packet $e_2 = m_2 \oplus m_3$.

Based on the location of packet decoding, inter-flow coding systems can be classified as *single-hop* coding systems and *multi-hop* coding systems. In single-hop coding systems [8], [10], packet decoding occurs in the immediate next hop of the coding node, while in multi-hop coding systems [9], [12], [13]. decoding occurs at nodes several hops away from a coding node. Previous studies have shown that single-hop coding can improve system throughput by up to 70% [8] when compared with traditional routing. Multi-hop coding is able to further improve the throughput obtained by single-hop network coding systems by up to 20% [9].

3.2 General Coding Condition

A key component of an inter-flow coding system is the *coding condition* being used, which decides the set of packets that



(a) Single-hop coding systems (b) Multi-hop coding systems

Fig. 1. Different types of inter-flow coding systems. The number on an edge represents the ID of the flow that traverses the edge. Dashed lines represent packet overhearing.

can be coded (XOR-ed) together at a node for transmission. Intuitively, in an inter-flow coding system, a node can code a set of packets for different flows to produce a single coded packet for broadcast transmission *if and only if* the downstream nodes will have the necessary packets to recover, from the coded packet, the respective plain packets for their flow. In this section, we formalize this condition and present a general coding condition considered in this work.

We model the network as a directed graph $G = (V, E)$, where V is the set of nodes in the network and E is the set of directed edges such that $(v_1, v_2) \in E$ whenever v_2 is within the communication range of v_1. A flow f is a simple path in the network $s \to v_1 \to \cdots \to v_k \to r$, where s is the source node and r is the receiver node. We also denote the set of all downstream nodes of a node v on flow f as $D(v, f)$. We use $H(v)$ to denote the set of packets (both plain and coded) that a node v has received or overheard. Let S be a set of nodes. With a slight abuse of notation, we will use $H(S)$ to denote the union of packets that have been received by each node in the set, *i.e.*, $H(S) = \bigcup_{v_i \in S} H(v_i)$. In Appendix A, we prove:

Theorem 1. ***General Coding Condition.*** *A node v can broadcast a coded packet $e = m_1 \oplus m_2 \oplus \cdots \oplus m_k$ such that it is decodable by the downstream nodes at flows $f_1, f_2, \cdots, f_u$* if and only if *for each flow $f_i, 1 \le i \le u$, there exists a subset of packets P of $H(D(v, f_i))$ such that $\sum_{p \in P} p = \sum_{1 \le j \le k, j \ne i} m_j$.*

Theorem 1 specifies a basic condition that must be satisfied for coding to be useful in any inter-flow coding system. As such, it provides a general framework that encompasses all existing inter-flow coding systems [7]–[14]. By Theorem 1, to decide whether a coded packet e satisfies the coding condition, we need to decide the existence of a subset in a set of packets that can be summed to some given value. In Appendix B, we present a polynomial time algorithm for checking the existence of this subset and for finding this subset if it exists.

3.3 Specific Coding Systems

To illustrate the generality of our system model, we present two representative instantiations of the model.

Single-hop coding systems. Single-hop coding systems represent a basic form of inter-flow coding in which the coding condition at a node is only checked among its one-hop neighbors. More formally, in a single-hop coding system, to transmit k plain packets $m_1, m_2, \cdots, m_k$ to k next-hop nodes, a node can transmit a coded packet $e = m_1 \oplus m_2 \oplus \cdots \oplus m_k$ only if each intended next hop has all the $k-1$ packets m_j for $j \ne i$ (see Fig. 1(a)). Although single-hop coding systems use

1. We assume that all the plain packets are of equal length, which is a standard assumption in inter-flow coding systems.

only a subset of the possible coding opportunities allowed by the general coding condition, they have the benefit of a simple system design. Examples of single-hop coding systems include [7], [8], [10], [11], [14].

Multi-hop coding systems. Unlike single-hop coding systems, multi-hop coding systems use coding opportunities across multiple hops, providing increased coding performance gains. More formally, in a multi-hop coding system, to send k plain packets, $m_1, m_2, \cdots, m_k$ for k flows, a node can transmit a coded packet $e = m_1 \oplus m_2 \oplus \cdots \oplus m_k$ only if for each flow f_i, the union of all the plain packets recoverable at the downstream nodes contains all the $k-1$ packets m_j for $j \neq i$. For example, in Fig. 1(b), node A can code packets m_1 and m_2 together, since R_1 and R_2 (two hops away) have overheard necessary packets to decode the coded packet. Examples of multi-hop coding systems include [9], [12].

4 Pollution Attacks on Inter-Flow Network Coding Systems

In this section, we first present an adversarial model for pollution attacks in inter-flow coding systems. We then analyze the impact of pollution attacks under different network scenarios.

4.1 Adversarial Model

We consider a network in which malicious nodes may conduct packet pollution attacks. An attacker node can be any node on any flow in the network. Attackers may collude.

We define packet pollution attacks as the injection of corrupted packets into the network by malicious nodes. A corrupted packet can be either a plain packet or a coded packet. A *corrupted plain packet* is a packet that is labeled to be a plain packet p from a source but differs from the original p. A *corrupted coded packet* is a packet e' that is labeled as coded from plain packets $m_1, m_2, \cdots, m_k$, but $e' \neq m_1 \oplus m_2 \oplus \cdots \oplus m_k$.

We assume that a pollution attacker can inject corrupted packets to cause the corruption of packets in any flow. An attacker can always inject packets to corrupt its own flow, which is similar to attacks in intra-flow coding systems. In inter-flow network coding, an attacker node can also exploit the coding dependencies among flows to cause packet corruption of other flows, even if the attacker node is not on the data path of a victim flow. For example, an attacker can create coding opportunities to attack a victim flow by first establishing a fictitious flow that is to be coded with the victim flow and then injecting corrupted packets into the fictitious flow. Because any node can become the source of a flow, we assume that source nodes are not trusted to behave correctly towards other flows.

As discussed in [49], [50], besides pollution attacks, inter-flow coding systems are vulnerable to a wide range of other attacks. For example, attackers can subvert the path selection and coding opportunity discovery process by advertising fake routing or topology information. The packet forwarding process is vulnerable to other types of attack as well, such as packet dropping, refusal to decode packets, and coding too many packets together. In addition, wireless networks are vulnerable to attacks on the physical and MAC layers.

In this paper, we focus *exclusively* on pollution attacks, which pose as a generic and potent threat to any network coding system. Defending against pollution attacks is a critical step in any comprehensive solution to secure these systems.

4.2 Pollution Attacks

Pollution attacks can have different levels of severity depending on the strategy of the attacker, the network topology, and the specific network coding system under consideration. In this section, we present two classifications of pollution attacks, based on the type of packets used in the attack and based on the attack's effect.

Attack classification based on the type of packets used in the attack. In a pollution attack, an attacker node can inject either corrupted plain packets or corrupted coded packets. We refer to these two types of attack as *plain packet pollution* and *coded packet pollution*, respectively.

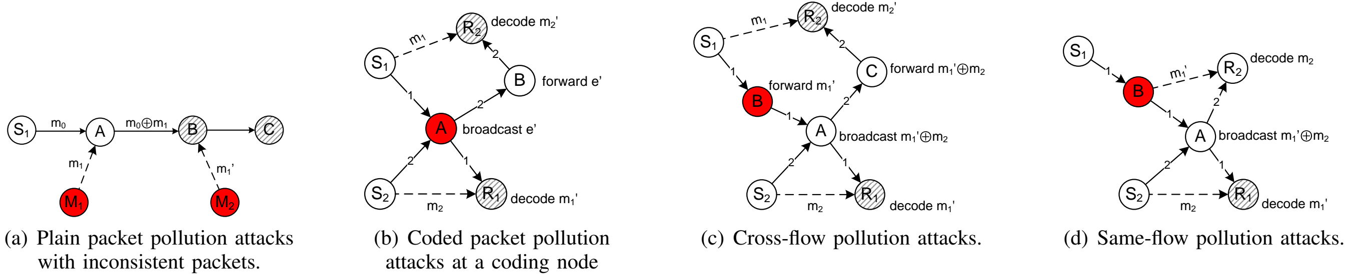
Plain packet pollution. An attacker node conducts plain packet pollution attacks by directly injecting corrupted plain packets, modifying plain packets, or injecting inconsistent plain packets into the network. Direct packet injection and modification attacks are similar to the respective attacks in traditional routing protocols, where an attacker node injects corrupted packets or maliciously modifies en-route packets to cause packet corruptions of a target flow.

The possibility to inject *inconsistent plain packets* is unique to attacks on inter-flow coding systems. In such an attack, the attacker injects different plain packets but labels them as the same packet, in order to cause inconsistency in the coding and decoding operations. One example is shown in Fig. 2(a), where two colluding attacker nodes M_1 and M_2 send a packet m_1 to node A and a different packet m_1' to node B, respectively, but both packets are labeled as m_1. If A codes the packet m_0 with the injected packet m_1 and sends the coded packet $e = m_0 \oplus m_1$ to B, then when B tries to decode e with packet m_1', it will recover a corrupted packet for m_0.

Note that in attacks using inconsistent plain packets, the injected packets originate from colluding attacker nodes who may share a single identity. Since attacker nodes are free to be the source of flows, the injected packets will pass message authentication, and circumvent authentication-based defenses.

Coded packet pollution. In coded packet pollution, attackers inject corrupted coded packets into the network, either at a malicious coding node or at a malicious forwarding node. In both cases, the attack can cause all the downstream nodes to decode incorrect packets. However, executing the attack at a coding node allows the attacker to corrupt multiple flows by injecting a single corrupted packet. For example, in Fig. 2(b) the attacker node A is a coding node, and a corrupted packet injected by A can cause both receiver nodes R_1 and R_2 to decode incorrect packets. On the other hand, if the attacker were the forwarding node B, then the attack would cause only receiver node R_2 to decode incorrect packets.

A key feature of coded packet pollution attacks is that they are difficult to address using cryptographic techniques. In inter-flow coding systems, each coded packet is an XOR of packets from different source nodes. Furthermore, the



(a) Plain packet pollution attacks with inconsistent packets.

(b) Coded packet pollution attacks at a coding node

(c) Cross-flow pollution attacks.

(d) Same-flow pollution attacks.

Fig. 2. Pollution attacks on inter-flow coding systems. The node in red is the attacker node and the shaded nodes are victim receiver nodes that receive or decode corrupted packets. The number on an edge represents the ID of the flow that traverses the edge. Dashed lines represent packet overhearing.

source nodes do not trust each other. Taking a cryptographic approach would require a cryptographic scheme that can combine authentication information from multiple mutually distrusting entities and is homomorphic with respect to the XOR operation. We are not aware of the existence of any such cryptographic scheme.

Attack classification based on the effects on flows in the network. Based on the scope of the flows affected, we classify pollution attacks into *cross-flow pollution* and *same-flow pollution*. The cross-flow pollution phenomenon we identify here is unique to inter-flow network coding.

Cross-flow pollution attacks exploit the coding dependencies among flows to cause an epidemic propagation of packet pollution across the flows. To execute such an attack, the attacker arranges for corrupted packets that it generates to be coded with (valid) packets of other flows. As an example, in Fig. 2(c), attacker node B injects corrupted packets into flow 1 for receiver R_1. These packets are to be coded by node A with the flow 2 packets for receiver R_2. As a result, both R_1 and R_2 will receive corrupted packets. A severe consequence of the cross-flow pollution is that the attack can bypass existing security mechanisms that try to ensure the correctness of nodes on a selected path [31], [51]. This is because even if the attacker is not selected for the path, it can still establish its own flow to corrupt the target flow.

In contrast to cross-flow pollution, packet corruption in a *same-flow pollution attack* is confined only to the flows directly modified by the attacker. Such an attack occurs either when the corrupted packet generated by the attacker is not coded with packets of other flows, or if the corrupted packet is used in such a coding so that its effects happen to be cancelled out by the operations of some other flow(s). For example in Fig. 2(d), although the corrupted packet from node B is coded with a packet for flow 2 at node A, the polluted packet is also overheard by node R_2 and used for decoding. The consistent use of the corrupted packet in both the coding and decoding cancels out the pollution effect for flow 2.

The above examples for cross-flow and same-flow pollution show that the effect of a pollution attack is sensitive to the location of the attacker node and to the overhearing relationships among nodes. They also show that in general, it is difficult to predict the impact of these attacks.

5 Pollution Defense: CodeGuard

In this section, we present CodeGuard, our approach to defend against pollution attacks in inter-flow network coding systems.

General definitions for inter-flow network coding systems

`Types of packets`
- **plain packet**: packet sent by the source
- **coded packet**: bitwise XOR of a set of plain packets

`Types of nodes`
- **forwarding node**: forwards plain or coded packets unmodified
- **coding node**: codes packets from different flows into a single coded packet, which is then forwarded
- **decoding node**: decodes a coded packet either into a plain packet (full decoding) or into another coded packet (partial decoding)

`Types of corrupted packets`
- **corrupted plain packet**: a packet that is labeled as a plain packet from a source, but is different than the original packet sent by that source
- **corrupted coded packet**: a packet e that is labeled to be coded from packets $m_1, m_2, \ldots, m_k$, but $e \neq m_1 \oplus m_2 \oplus \ldots \oplus m_k$

Definitions for CodeGuard

`Types of signed packets`
- **signed plain packet**: a packet $\widehat{p} = ((p, id_S, sig_S), 0, 0)$, where p is the plain packet, id_S is the ID of the source node, and sig_S is the signature of S over p
- **signed coded packet**: a packet $\widehat{e} = (e, id_C, sig_C)$, where e is the (component-wise) XOR of two or more signed plain packets, id_C is the ID of the coding node C, and sig_C is the signature of C over e

Fig. 3. Reference sheet for various definitions

We first state several assumptions. We then describe the defense scheme and analyze its security and its overhead. To facilitate the exposition, we include a reference sheet with various definitions in Fig. 3.

5.1 Assumptions

We assume that each node can authenticate its messages by using digital signatures. We further assume the existence of a reliable end-to-end communication path between every pair of nodes. Reliable end-to-end communication in wireless networks has been a subject of extensive study with numerous proposals [16], [29]–[31], any of which may be used with our protocol. Also, since reliable communication is required in our defense only after an attacker is positively identified, a straightforward implementation using flooding can be used.

We assume that mechanisms are deployed to prevent a node from conducting a Sybil attack, in which a single attacker node owns multiple (bogus) identities and their associated credential information. However, attackers may collude. Each receiver of a flow trusts only the source of the flow; it does not trust intermediate nodes or sources of other flows.

5.2 CodeGuard Overview

In CodeGuard, nodes use digital signatures as a means to unequivocally identify pollution attackers. Specifically, the

source signs every plain packet to ensure the authenticity and integrity of each plain packet; a coding/decoding node signs each coded packet it generates to attest to the correctness of the coding/decoding operation. Forwarding nodes verify the signature of the received packets and if the signature is valid, forward the packets unchanged. When any node receives a packet with an invalid signature on a link from one of its neighbors, it sends a bad link notification to the source so that the link is avoided in the future. We emphasize that in CodeGuard forwarding nodes do not have to sign the packets they forward; only nodes that participate in the coding and decoding process must sign the coded packets they generate.

Note that under our adversarial model, packets that pass the digital signature verification are not necessarily valid. For example, a malicious coding node can sign and inject polluted coded packets with its own signature. Alternatively, colluding attacker nodes can conduct the inconsistent plain packet attack by injecting two (or more) inconsistent plain packets signed using their shared identity as discussed in Sec. 4.2. However, in these cases, the attack will eventually cause a downstream node d to recover a corrupted plain packet which d can detect by checking the signature on the plain packet from its source. When that happens, node d will send a pollution notification to the source of the flow, including the corrupted packet in the notification. The source will then initiate a traceback procedure to identify the responsible attacker. Traceback is initiated by the source because coding and forwarding nodes are not trusted.

Thus, the source of a flow may receive two types of notifications. A *bad link notification* is sent by a node A that received a (plain or coded) packet with an invalid signature from node B, and contains the link between A and B[2]; the source will avoid that link in the future. A *pollution notification* occurs when a decoding node receives a coded packet with a valid signature, and after decoding it, the result is a plain packet which has an invalid signature. The decoding node alerts the source with a pollution notification, which includes the corrupted packet; as a result, the source initiates the traceback procedure. Note that the source node stores the plain packets it sends for a certain amount of time, in order to successfully trace attackers if pollution attacks occur. Also, coding/decoding nodes store the input packets in order to be able to exonerate themselves during a traceback.

A straight-forward, but expensive approach, is to traceback the entire packet. Instead, CodeGuard relies on the observation that each bit in a packet is coded/decoded independently. As a result, it is sufficient to trace the history of a single corrupted bit to identify the attacker. During the *bit-level traceback* procedure, the source queries iteratively each involved coding node about the bit in question. Upon being queried, a coding node reveals the bit values of the corresponding input packets used to create the coded packet as a proof of its correct coding. CodeGuard uses a novel *cross-examination* technique to ensure the consistency of the bit value answered by a node with the value reported by the node that sent the packet, and to

2. Note that node A cannot report node B as an attacker to the source, because A cannot prove that B is the originator of the corrupted packet.

Algorithm 1 Coding/decoding ($\widehat{\oplus}$) executed at node C

Input: packets $\widehat{e_1}, \ldots, \widehat{e}_k$; (where $\widehat{e}_i$ is either a signed plain packet $(e_i, 0, 0) = ((p, id, sig), 0, 0)$ or a signed coded packet (e_i, id, sig))
Output: a signed plain or coded packet $\widehat{e_1} \widehat{\oplus} \ldots \widehat{\oplus}\ \widehat{e}_k$

1: Check if the signatures on all input packets are valid; for each input packet that has an invalid signature, send a bad link notification to the source containing the link on which the packet was received
2: Compute $e = e_1 \oplus \ldots \oplus e_k$; (apply $\oplus$ component-wise)
3: **if** e is a signed plain packet **then**
4:     **if** the signature on e is valid **then**
5:         Output $(e, 0, 0)$
6:     **else**
7:         Send pollution notification to the source
8: **else**
9:     Compute signature sig_C over e
10:     Output (e, id_C, sig_C)

---

ensure that nodes do not conduct inconsistent packet attacks (i.e they sent different packets to different nodes labeled with the same packet identifier). The cross-examination relies on the observation that in inter-flow coding, consistency in using packets is sufficient for the correct recovery of plain packets. We prove that during the traceback, either a coding node will be unable to produce a correctness proof or a node injecting inconsistent plain packets will be found. In either case, the source node can always identify one attacker node and exclude it from the network because of the digital signature carried by each packet.

CodeGuard is independent of the route selection process and only assumes that the routing and coding can avoid the identified malicious links and nodes. Such avoidance can be easily achieved, *e.g.*, by including the list of identified malicious links and nodes in the route request messages.

Finally, we note that most of the defenses proposed for intra-flow network coding attempt to achieve hop-by-hop containment of polluted packets. However, in inter-flow network coding the nature of the coding/decoding operations and the existence of malicious sources makes efficient hop-by-hop containment more challenging. Unlike the approach of Agrawal *et al.* [40], which achieves hop-by-hop containment at the cost of network overhead linear with the number of sources in the network, CodeGuard has the advantage of constant-size network overhead due to coding; the trade-off is that CodeGuard may not always detect immediately polluted packets (although when decoded, corrupted plain packets are always detected).

5.3 Detection of Polluted Packets

In this section, we present in details the detection of polluted packets in CodeGuard. This detection relies on digital signatures. Each plain packet is signed by its source node and each coded packet is signed by the node that creates it (*i.e.*, codes/decodes it). A *signed plain packet* created by a source S is denoted by $\widehat{p} = (p, id_S, sig_S)$, where p is the plain packet, id_S is the ID of S, and sig_S is the signature of S over p. A *signed coded packet* created by a node C is denoted by $\widehat{e} = (e, id_C, sig_C)$, where e is the XOR of two or more signed plain packets, id_C is the ID of C, and sig_C is the signature of C over e. For consistency, we write a signed plain packet in the same format as a signed coded packet, as $\widehat{p} = ((p, id_S, sig_S), 0, 0)$. Using this notation, we define an

XOR-like operation $\widehat{\oplus}$ for coding and decoding signed packets as shown in Algorithm 1.

For example, coding two signed plain packets $\widehat{p}_1 = ((p_1, id_{S_1}, sig_{S_1}), 0, 0)$ and $\widehat{p}_2 = ((p_2, id_{S_2}, sig_{S_2}), 0, 0)$ at node C produces a signed coded packet $\widehat{e} = (e, id_C, sig_C)$, with $e = (p_1 \oplus p_2, id_{S_1} \oplus id_{S_2}, sig_{S_1} \oplus sig_{S_2})$. To decode $\widehat{e}$ using $\widehat{p}_2$, we compute $e \oplus (p_2, id_{S_2}, sig_{S_2}) = (p_1, id_{S_1}, sig_{S_1})$, which is a plain packet, and so the output packet of $\widehat{e}$ $\widehat{\oplus}$ $\widehat{p}_2$ is $((p_1, id_{S_1}, sig_{S_1}), 0, 0) = \widehat{p}_1$ as expected. Since a signed coded packet always has only one attached signature (from the coding node), we stress that *the size of a coded packet remains the same* regardless of the number of coding operations performed to obtain it.

A signed plain packet $\widehat{p}$ and a signed coded packet $\widehat{e}$ can be verified by checking the validity of sig_S over p, and the validity of sig_C over e, respectively. Note that the verification of a signed coded packet does not involve the verification of the plain packets used to obtain it; indeed, these plain packets may not be available. Each node performs the following actions:

• **Source node**: For every plain packet p, the source computes the signed version $\widehat{p} = ((p, id, sig), 0, 0)$ and forwards it to the next hop node. The source also stores the packet p for use in the traceback for attacker nodes in case of a packet pollution notification.

• **Forwarding node**: When a forwarding node receives a (plain or coded) packet, it verifies the packet's attached signature. If the verification passes, the node forwards the packet unchanged to its next hop node via an authenticated channel. Otherwise, the packet is dropped and the node sends a bad link notification to the source in order to report the link on which the packet is received as a malicious link to be avoided.

• **Coding/decoding node**: Like a forwarding node, a coding/decoding node first verifies the signatures of its received plain and coded packets. If a packet with an invalid signature is found, the node sends a bad link notification to the source. Only input packets with valid signatures are kept for further coding/decoding operations. If the result of an operation is another coded packet (*e.g.*, after coding or partial decoding), the node computes the signed version of the coded packet and forwards the signed packet downstream. The node also stores the input packets in case it needs to provide a correctness proof during a possible traceback in case some pollution attack will be detected. If the result of an operation is a plain packet, the node verifies the source signature on the packet. If the verification fails, then the packet is corrupted and the node sends to the appropriate source node a pollution notification, which includes the corrupted packet. The actions of a coding/decoding node are specified in Algorithm 1.

5.4 Tracing Pollution Attackers

There are cases in which coded packets passing the signature verification are not necessarily valid, but their pollution cannot be detected at that hop, but only downstream when they eventually result in the recovery of a plain packet with an invalid signature. When such an event occurs, the node that detects the invalid signed plain packet triggers a traceback procedure by sending to the source of the corresponding flow a pollution notification which contains the corrupted plain packet.

Upon receiving a pollution notification, the source starts a bit-level traceback procedure to identify the attacker that injected the polluted packet and caused the corrupted plain packet recovery. A key observation in this step is that each bit in a packet is coded/decoded independently (*i.e.*, the i-th bit of a coded packet is obtained by XOR-ing the i-th bits of the input packets). It is hence sufficient to perform a bit-level traceback instead of a more costly packet-level traceback. In bit-level traceback, each node only responds with a single bit in a packet, and so packet signatures cannot be used to ensure the correctness of the response. Instead, CodeGuard leverages another key observation in inter-flow coding, namely that consistency in using packets is sufficient for the correct recovery of plain packets. In other words, as long as a packet is used consistently in the coding and decoding process, even if the packet is corrupted, its effect will be cancelled out (for a concrete example, see the same-flow pollution attack in Fig. 2(d)). This observation motivates a *cross-examination* technique to ensure the consistency of the bit in question.

The bit-level traceback works as follows. On receiving a pollution notification from a node R for a corrupted plain packet $\widehat{p}'$, the source first finds an incorrect bit, say bit u, in $\widehat{p}'$ by comparing it with the signed plain packet $\widehat{p}$ it originally sent. Then the source starts tracing back the history of bit u by contacting all the involved coding nodes iteratively and asking them to prove their innocence by providing the input bits they used to create the coded bit that is traced. The detailed procedure is shown in Algorithm 2. The procedure terminates when an attacker node is identified.[3]

The source maintains two sets, coded_set and plain_set, containing information on coded and plain packets, respectively. At each step of the traceback, the source selects a node P to be a *prover node* and queries it for a proof of innocence. If the node proves its innocence, it is exonerated, otherwise it is identified as an attacker node. For each prover node P, the source also has a target bit value b_u^P for bit u of the packet output by that node. Initially, P is set to be the node R which sent the pollution notification, and the target bit value b_u^P is set to be bit u of $\widehat{p}'$ (lines 4–5). The source queries P for the u-th bit of the input packets coded to produce $\widehat{p}'$ (line 9). In response, P sends for each input packet $\widehat{p}_i$ a tuple $T_i = (b_i, pid_i, ptype_i, n_i)$, where b_i is the value of bit u in $\widehat{p}_i$, pid_i is the packet ID of $\widehat{p}_i$, $ptype_i$ is a bit indicating whether $\widehat{p}_i$ is a coded or plain packet, and n_i is the ID of the node that originated $\widehat{p}_i$ (line 10).

On receiving the set of tuples, the source verifies that P coded correctly by checking if $\sum b_i = b_u^P$, where the summation is performed over modulo 2. If the verification fails, P is identified as an attacker (lines 11–12), as P performs the coding incorrectly.

Otherwise, the source cross-examines the bit value b_i re-

3. In case of n (possibly colluding) attackers, the source node can identify all of them with at most n invocations of the traceback procedure.

Algorithm 2 The traceback procedure

```
Input: A corrupted plain packet p̂' and the pollution reporting node R
Output: The identified attacker node
Note: All query and response messages are signed by their originator nodes.
      When a query times out, the queried node is identified as the attacker.
 1: Let p̂ the correct packet stored by the source
 2: if p̂' and p̂ are identical then // Fake pollution notification
 3:    return node R
 4: Set P = R
 5: Let b_u^P be any bit in p̂' that differs from the corresponding bit in the
    correct packet p̂
 6: Let u be the differing bit position
 7: Set coded_set = ∅, plain_set = ∅
 8: loop
 9:    Query P for tuples for bit u of its input packets
10:    Let S = {(b_i, pid_i, ptype_i, n_i)} be the set of tuples returned by P
11:    if Σ b_i ≠ b_u^P then // Check if P codes correctly
12:       return node P
13:    for each tuple T_i = (b_i, pid_i, ptype_i, n_i) in S do
14:       Query n_i with tuple T_i for cross examination
15:       if n_i returns "NO" then // Either P or n_i is lying
16:          Query P for the signed packet p̂_i containing the bit
17:          if p̂_i is from n_i and bit u is b_i then
18:             return node n_i
19:          else
20:             return node P
21:    // Node P is now exonerated
22:    Add to coded_set all T_i's returned by P whose ptype_i is "coded"
23:    Add to plain_set all T_i's returned by P whose ptype_i is "plain"
24:    if plain_set contains two consistent tuples for the same packet then
25:       return the originator node of the tuples
26:    Remove a tuple (b_i, pid_i, ptype_i, n_i) from coded_set
27:    Set P = n_i and b_u^P = b_i to repeat the traceback procedure
```

ported by P with its claimed originator node n_i (lines 13–20). To do this, the source forwards each tuple T_i to n_i, which responds with an indication of whether it has indeed generated a packet with ID pid_i and whose bit u is indeed b_i (line 14). A positive response indicates that node n_i accepts the responsibility for generating the bit. If all the responses are positive, then node P is exonerated, since the coding operations are consistent up to node P.

If any response from n_i is negative, that is, if n_i disagrees with the claim made by P regarding the bit value b_i, then either P or n_i is lying and can be identified as the attacker. To find out which one, the source queries P for the signed input packet from n_i, which P should have stored in its buffer. If P is able to return a packet correctly signed by n_i whose u-th bit is b_i, then node n_i is identified as the attacker. Otherwise, P is identified as the attacker (lines 17–20).

The source adds all the received tuples T_i to coded_set or plain_set according to their $ptype_i$ field (lines 22–23). It then checks the consistency of the tuples in plain_set to ensure that no node has claimed conflicting bits for the same plain packet, that is, there do not exist two tuples T_i and T_j in plain_set with $i \neq j, n_i = n_j, pid_i = pid_j$, but $b_i \neq b_j$. If such a conflict is found, the responsible node is identified as the attacker (lines 24–25). This consistency check handles the inconsistent plain packet attacks (described in Sec. 4.2). Otherwise, the source removes a tuple from coded_set, sets P and b_u^P to be n_i and b_i in the removed tuple, respectively, and then repeats the traceback procedure on the coding node n_i (lines 26-27).

5.5 Security Analysis

In a pollution attack, an attacker can inject corrupted packets carrying an invalid signature but also a valid signature (*e.g.*, inconsistent plain packets or incorrectly coded packets, signed by itself). In the following, we show that CodeGuard can effectively address both types of attack. In addition, we show that repeatedly invoking the traceback procedure in an attempt to cause a potential DoS attack can only have limited impact.

Lemma 1. *Corrupted packets with invalid signatures do not propagate in the network and links involving the pollution attacker are identified.*

Proof: Since all honest nodes perform packet signature verification before processing a received packet, packets with invalid signatures are immediately identified and dropped at the attacker's first hop neighbor node. Thus, the polluted packet does not propagate. The link between the receiving node and the attacker node is also reported to the source as a malicious link to be avoided. There is no danger of false accusation, as the reporting node is required to be one end of the reported malicious link. Thus, an attacker node can only cause links adjacent to itself to be avoided (this is in fact desirable).

Lemma 2. *For every occurrence of a pollution attack which uses packets with correct signatures and causes incorrect packet decoding at an honest node, a traceback procedure is always invoked.*

Proof: Plain packets are attached with the signatures from their corresponding source nodes. When a pollution attack causes a corruption during packet decoding at some downstream honest node, the invalid signature on the decoded plain packet will cause the node to send a notification to the source node to invoke the traceback procedure. In the worst case, the attack will be detected at the receiver. The delivery of the notification message is ensured based on the reliable end-to-end communication assumption discussed in Section 5.1.

Lemma 3. *A traceback procedure always results in the identification of one attacker node, even in the presence of colluding attackers, and there are no false positives.*

Proof: In inter-flow coding systems, we can visualize the history of a decoded plain packet using a *coding tree* as in Fig. 4. If the decoded plain packet is incorrect, observe that one of the following two cases must have occurred: an incorrect XOR was performed at some interior node, or an inconsistent plain packet was used at a leaf node. The traceback performs a breadth-first search of the coding tree, verifying the correctness of an XOR operation and the consistency of plain packets at each step. Thus, eventually one attacker node will be identified.

We now explain the concept of a *coding tree* and illustrate it with an example in Fig. 4. First, we observe that in inter-flow coding systems, coding and decoding operations are essentially the same operation, *i.e.*, an XOR over multiple input packets. Thus, in the following, we use the term coding to mean both coding and decoding. Given a plain packet which is obtained from a decoding operation, we can trace back the history of the packet by constructing a coding tree recursively from top

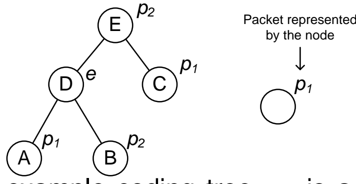


Fig. 4. An example coding tree. p_2 is a decoded plain packet which is incorrect. The leaf nodes represent plain packets and the interior nodes represent the XOR of their child nodes. In order to produce an incorrect plain packet at the root, it must be the case that either an XOR is performed incorrectly at some interior node (*e.g.*, $e \neq p_1 \oplus p_2$), or there are inconsistent plain packets at the leaves (*e.g.*, packets represented by node A and C are not the same even though they are both labeled as p_1).

to bottom as follows. Each tree node represents a packet, with interior nodes being coded packets and leaf nodes being plain packets. The root of the tree is the target plain packet. The children of each node in the tree are the input packets that are coded together (using XOR) to obtain the node. A plain packet node, except for the root node, does not have children. Such a coding tree can always be constructed for any plain packet.

Each tree node has a signature attached to it: the signature for a plain packet node is created by its source node, and the signature for a coded packet node is created by the coding node that coded the packet. Since CodeGuard requires each node to verify the validity of the attached signatures prior to the coding operation, if there is a node with an invalid signature, then its parent node is identified as the attacker node. Next, we assume all nodes in the tree have valid signatures. Since the root plain packet is a corrupted packet, it must be the case that at least one of the following occurs: i) some interior coding node does not label correctly the packet it codes using its children nodes, or ii) some plain packet nodes (at leaves) are inconsistent. In the first case, the incorrect interior node is the culprit attacker node, while in the second case, the source that signs inconsistent packets is the attacker node. During the traceback procedure of CodeGuard, the source performs a breadth first search from the root node of the coding tree, and is able to always find one attacker node.

Attacks on the traceback procedure. Below we analyze the effects of possible attacks on the traceback process: *attacker collusion*, *packet dropping*, *reporting pollution on old packets*, and *DoS attacks*.

In the presence of multiple (possibly colluding) attacker nodes, one attacker node may be able to prove the correctness of its coding by blaming another attacker node for providing incorrect input packets. However, regardless of how the attackers pass off responsibilities among themselves, the traceback procedure will always identify one attacker node.

Two attackers may collude to deceive the source about the bit being traced in the cross-examination procedure. However, in inter-flow network coding, as long as the bit is used consistently in both the coding and decoding processes, its impact cancels out and does not result in packet corruption. Such consistency of bits is examined at each step of the traceback; hence, collusion attacks on the cross-examination are ineffective.

An attacker may incriminate an honest node by dropping either a traceback query request or response message. However, such an attack is ruled out by the reliable end-to-end communication assumption discussed in Sec. 5.1.

An attacker may report a packet pollution for a packet long in the past such that the input packets stored at an honest node for generating the coding correctness proof have been purged, due to limited storage space. As shown in the storage overhead analysis (Sec. 5.6), a node is able to store packets for 1000 seconds even with a moderate storage space. Thus, the source can ignore pollution reports for old plain packets (*e.g.*, older than 1000 seconds) to avoid such attacks.

Finally, an attacker may try to invoke frequent and lengthy traceback procedures by injecting corrupted packets in order to exhaust the network bandwidth. However, the number of nodes contacted by the source during traceback is upper-bounded by the number of coding nodes involved, which is typically a small number. Furthermore, even if such an attack is executed, its effect cannot be sustained since at least one attacker node will be removed from the network after each traceback invocation. Note that attacks from malicious source nodes that repeatedly initiate tracebacks are a form of DoS attack and can be addressed by limiting the rate at which a source node is allowed to initiate tracebacks.

5.6 Overhead Analysis

We now analyze the storage, computation, and bandwidth overhead of the proposed defense.

Storage. CodeGuard requires each coding and decoding node to store its received packets because they may be needed for verification during traceback. Since the packets are needed only for traceback, they can be stored in secondary storage (which is increasingly abundant and cheaper), instead of in the more premium memory. For example, for a storage capacity of 1 GB, a packet size of 1 KB, and a packet rate of 1000 packets per second (equivalent to around 8 Mbps goodput), a packet can be stored for over 1000 seconds before it becomes unavailable. In Appendix C, we rigorously analyse the lower bound on the time duration that a node needs to store the input packets and the level of tolerance to packet delaying attacks.

Computation. The computation overhead of CodeGuard consists of signature generation at the source and coding nodes, and signature verification at the intermediate nodes. With increasingly more powerful CPUs and more prevalent use of cryptographic hardware accelerators, computing per-packet digital signatures is becoming practical. For instance, using the 160-bit Elliptic Curve DSA signature (ECDSA)[4], current consumer-grade processors can perform one signature generation/verification operation in less than 1 ms[5]. With hardware acceleration, rates exceeding 10,000 operations per second have been reported [52], [53], which would allow our protocol to accommodate rates that far exceed current wireless mesh network bandwidths. The computation overhead consists of two components: Proactive overhead (incurred regardless of the presence of attack, it is primarily due to the use of

4. Equivalent to the security level of 1024-bit DSA.
5. Based on OpenSSL version 0.9.8g on 2.26 GHz Intel Core 2 Duo CPU.

digital signatures on packets) and reactive overhead(incurred by efforts to identify and isolate attacker nodes). The reactive overhead is small, as it is bounded by the number of attackers in the network (each invocation of the traceback procedure results in one attacker being identified and excluded); moreover, typical wireless mesh networks have relatively small path lengths. We evaluate the computation overhead in Section VI.C

Communication. The communication overhead of CodeGuard under normal operation is due to signatures attached to packets. The size of a 160-bit ECDSA signature is 320 bits. Hence, for a data packet size of 1500 bytes, the signature incurs a bandwidth overhead of 2.7% for a signed plain packet and 5.4% for a signed coded packet.[6] The communication overhead of the traceback process is also small, because both query and response messages are only a few bytes long.

Overhead observation. Finally, we would like to emphasize that in CodeGuard only nodes that participate in the coding and decoding process incur the storage overhead and the computation overhead of signing packets. In flows that do not have coding opportunities, CodeGuard only imposes the overhead of the source signing its packets and intermediate nodes verifying their received packets. Such overhead is necessary even for addressing the conventional packet modification attacks.

6 Experimental Evaluations

In this section, we present simulation results to evaluate the effectiveness and overhead of CodeGuard in identifying attacker nodes.

6.1 Experimental Methodology

Simulator setup. We use the Glomosim simulator [54] configured with 802.11 as the MAC layer protocol and a raw link bandwidth of 2 Mbps. We augment the physical layer of Glomosim to use the measurement-based model from [55], which empirically maps the link distance to the transmission success probability, considering physical layer effects such as path-loss, shadowing, and multi-path fading.

Coding system. We implement an inter-flow coding protocol based on the general coding condition in Sec. 3 and refer to it as ICODE. It encompasses all existing protocols, including [7]–[14]. As we are only interested in the impact of pollution attacks, ICODE is implemented in an off-line manner using global knowledge of the network. Specifically, we first select a set of source and destination pairs and establish a path between each pair using the ETX metric [56]. Then each node on a path is examined for coding opportunities. At runtime, nodes forward or code packets according to the off-line analysis results.

Experiment scenarios. We first examine the impact of pollution attacks using the network configurations in Figures 2(b) and 2(c). These configurations demonstrate coded packet pollution attacks and plain packet pollution attacks with cross-flow pollution. For each experiment, we vary the *offered load* in the network by varying the packet sending rate at the source nodes.

6. Because a signed coded packet contains two signatures. See Sec. 5.3.

We then examine the impact of pollution attacks and the effectiveness and overhead of CodeGuard under a general network configuration. The network consists of 100 honest nodes and up to 30 attacker nodes distributed randomly in a 1500m by 1500m area. Attacker nodes selected on a data delivery path always send corrupted packets to their downstream nodes. A variable number of flows are established between randomly selected source and destination pairs. The total offered load of all the flows is kept at 1 Mbps, which is sufficient for full utilization of the network bandwidth without causing over-congestion. We present results for 25 flows (results for different number of flows are similar). Each simulation is run for 300 seconds and is repeated 20 times with different random seeds.

Metrics. We use the following metrics for evaluating the impact of pollution attacks and the effectiveness and overhead of CodeGuard.

– *Network throughput.* This is the aggregate data receiving rate (in kbps) of all receivers in the network.

– *Bandwidth overhead.* This is the amount of data transmitted due to CodeGuard, including both signatures attached to packets and the data sent during traceback.

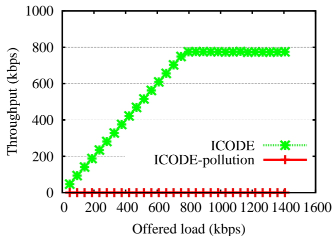
– *Computation overhead.* We measure computation overhead in terms of the number of digital signatures performed at a source or a coding node, because digital signatures are the most computation-intensive operations in CodeGuard.

– *Delay in attacker identification.* This is the average time over all attackers from when an attacker starts the pollution attack to when the attacker is identified by a source node.

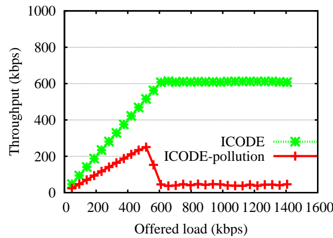
6.2 Results for Illustrative Scenarios

In Fig. 5, we show the impact of coded packet pollution and cross-flow pollution using the illustrative network scenario in Fig. 2(b) and 2(c) as presented in Sec. 4.2, respectively. The figure shows that inter-flow coding improves performance significantly. In particular, when the offered load is high, network congestion causes the throughput to decrease for traditional routing, while with network coding, the throughput can be sustained at a much higher level. Also, the pollution attack has a significant impact on the network throughput: It degrades the throughput to nearly zero for both scenarios, especially when the offered load is high.

The network throughput under pollution attacks in Fig. 5(b) exhibits an interesting trend: it first increases and then decreases as the offered load increases. This demonstrates the cross-flow impact of packet pollution attacks. In the network scenario in Fig. 2(c), flow 2 has no attacker on its path. When the network is under a lighter load, packets do not accumulate in the output queue of node A. Thus, as soon as node A receives a packet, it forwards the packet immediately without inter-flow coding, and flow 2 is not affected by the pollution attack on flow 1. As the offered load increases, node A starts to have packets queuing up, which enables inter-flow coding of the packets. As a result, the polluted packets in flow 1 start to contaminate the packets in flow 2, bringing the throughput of both flows to nearly zero.



(a) Scenario in Fig. 2(b)



(b) Scenario in Fig. 2(c)

Fig. 5. Throughput for different offered loads with and without using network coding, and the impact of pollution attacks on network coding for illustrative scenarios in Fig. 2(b) and 2(c).

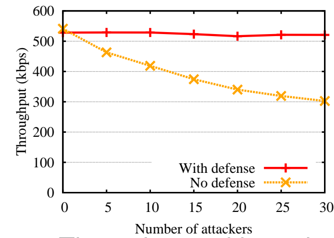


Fig. 6. Throughput with and without CodeGuard defense in a random network.

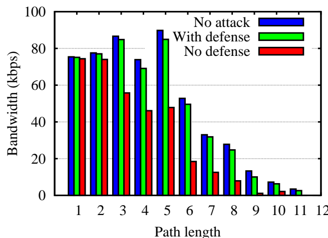


Fig. 7. Attack impact on aggregate throughput of flows with 20 attackers.

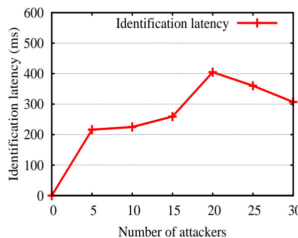


Fig. 8. Average delay of attacker identification.

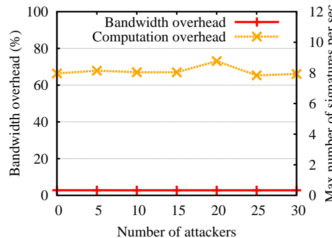


Fig. 9. Proactive bandwidth (left-Y) and computation (right-Y) overheads.

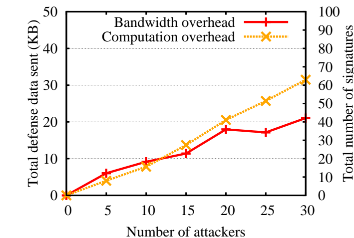


Fig. 10. Reactive bandwidth (left-Y) and computation (right-Y) overheads.

6.3 Results for Random Network Scenarios

In this section, we present the impact of pollution attacks and the effectiveness and overhead of CodeGuard under random network topologies.

Impact of pollution. Fig. 6 shows the total throughput achievable in the network under the presence of different numbers of pollution attackers, with and without the use of CodeGuard in a general network setting. We observe that without the defense, pollution attacks cause a steady degradation of network throughput as the number of attackers increases. However, the impact is not as drastic as in the illustrative examples where the throughput drops to nearly zero. The reason is that since the sources and destinations of flows are randomly selected and attackers are randomly placed, shorter flows have a smaller chance of having an attacker on their path. They are thus less likely to be affected by an attack, but they usually contribute more towards the total network throughput. Fig. 7 demonstrates the phenomenon by showing the effects of pollution attacks on flows of different path lengths, when there are 20 attacker nodes. We see that flows of only one or two hops are only marginally affected by the attack, while longer flows may experience a severe degradation in throughput.

Effectiveness of defense. From Fig. 6, observe that with the use of CodeGuard, the throughput is maintained at a high level similar to the case of no attack. CodeGuard is highly effective in restoring the throughput because of its ability to identify and isolate attacker nodes quickly after a single polluted packet is injected. To demonstrate this, Fig. 8 shows the average delay of attacker detection for different numbers of attackers. This latency is independent of the number of attackers; it is a function of the number of coding nodes on the path and the time required for a round trip between any two nodes in the network, both of which are small in typical wireless mesh networks. From the figure we see that the time to identify an attacker node is relatively stable at less than 500 ms. Thus, the overall impact that an attacker can inflict on a flow is minimum.

Overhead evaluation. We evaluate both the proactive and reactive overhead of CodeGuard. Fig. 9 shows the proactive bandwidth and computation overheads of CodeGuard in terms of the percentage of data transmitted due to the defense and the *maximum* number of signatures performed at a node, respectively. We see that CodeGuard has a proactive bandwidth overhead of only around 3%, while the maximum number of signatures performed at a node is around 8 per second. We note that the proactive computation overhead will increase proportionally as the data rate increases. However, as discussed in Sec. 5.6, we do not expect it to become a performance bottleneck given the power of current processors and the use of cryptographic hardware.

Fig. 10 shows the reactive bandwidth and computation overheads in terms of the total number of bytes delivered and the total number of signatures performed for defense packets, respectively. The results are shown for different numbers of attacker nodes in the network. Both the reactive bandwidth and computation overheads exhibit a trend of linear increase as the number of attackers increases, but both remain at a low level. For example, with as many as 30 attacker nodes, CodeGuard incurs a bandwidth overhead of less than 25 KB and it uses less than 70 total signature operations for the entire network.

Acknowledgments

A preliminary version of this paper appeared in [58]. This research was sponsored in part by US National Science Foundation grants NETS 0905266-CNS and CNS-0963715.

7 Conclusion

We have studied packet pollution as a potentially devastating attack against wireless inter-flow coding systems. We first presented a general framework that captures the essential properties of these systems. We then studied the impact of

pollution attacks within the framework for different network configurations and attacker strategies. Finally, we proposed a pollution defense, CodeGuard, that uses node attestation and bit-level trace-back to efficiently identify attackers. Through analysis and simulation experiments, we showed that CodeGuard provides highly effective protection while incurring small computation and communication overheads.

REFERENCES

[1] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *SIGCOMM*, 2007.

[2] X. Zhang and B. Li, "Optimized multipath network coding in lossy wireless networks," in *Proc. of ICDCS '08*, 2008.

[3] X. Zhang and B. Li, "DICE: a game theoretic framework for wireless multipath network coding," in *Proc. of Mobihoc*, 2008.

[4] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard, "Symbol-level network coding for wireless mesh networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, 2008.

[5] J.-S. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Medard, "Codecast: a network-coding-based ad hoc multicast protocol," *Wirel. Commu.*, 06.

[6] C. Gkantsidis *et al.*, "Multipath code casting for wireless mesh networks," in *CoNEXT '07*.

[7] S. Katti, D. Katabi, W. Hu, H. Rahul, and M. Médard, "The importance of being opportunistic: Practical network coding for wireless environments," in *In Proc. of Allerton Conference*, 2005.

[8] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: practical wireless network coding," in *SIGCOMM*, '06.

[9] J. Le, J. C. S. Lui, and D. M. Chiu, "DCAR: Distributed coding-aware routing in wireless networks," in *Proc. of ICDCS '08*, 2008.

[10] S. Das, Y. Wu, R. Chandra, and Y. C. Hu, "Context-based routing: Technique, applications, and experience," in *Proc. of NSDI '08*, 2008.

[11] Q. Dong, J. Wu, W. Hu, and J. Crowcroft, "Practical network coding in wireless networks," in *Proc. of MobiCom*, 2007.

[12] S. Omiwade, R. Zheng, and C. Hua, "Practical localized network coding in wireless mesh networks," *In Proc. of SECON 2008*.

[13] S. Omiwade, R. Zheng, and C. Hua, "Butteries in the mesh: Lightweight localized wireless network coding," in *Proc. of NetCod 2008*, Jan. 2008, pp. 1–6.

[14] B. Ni, N. Santhapuri, Z. Zhong, and S. Nelakuditi, "Routing with opportunistically coded exchanges in wireless mesh networks," *Wireless Mesh Networks, 2006. WiMesh 2006. 2nd IEEE Workshop on*.

[15] S. Marti, T. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Proc. of MOBICOM*, August 2000.

[16] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: a secure on-demand routing protocol for ad hoc networks," *Wirel. Netw.*, 2005.

[17] T. Ho, B. Leong, R. Koetter, M. Mdard, M. Effros, and D. R. Karger, "Byzantine modification detection in multicast networks with random network coding," *IEEE Trans. on Inform. Theory*, vol. 54, no. 6, 2008.

[18] M. Krohn, M. Freedman, and D. Mazieres, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *Proc. of Symposium on Security and Privacy*, 2004.

[19] C. Gkantsidis and P. Rodriguez Rodriguez, "Cooperative security for network coding file distribution," *Proc. of INFOCOM 2006*.

[20] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-based integrity for network coding," in *Proc. of ACNS '09*.

[21] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *Proc. of CISS*, 2006.

[22] Q. Li, D.-M. Chiu, and J. C. S. Lui, "On the practical and security issues of batch content distribution via network coding," in *Proc. of ICNP '06*.

[23] F. Zhao, T. Kalker, M. Medard, and K. Han, "Signatures for content distribution with network coding," in *Proc. of ISIT '07*, 2007.

[24] Z. Yu *et al.*, "An efficient signature-based scheme for securing network coding against pollution attacks," in *Proc. of INFOCOM '08*, 2008.

[25] Z. Yu, Y. Wei, and Y. Guan, "An efficient scheme for securing XOR network coding against pollution attacks," in *Proc. of INFOCOM*, 2009.

[26] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a linear subspace: Signature schemes for network coding," in *Proc. of PKC '09*, 2009.

[27] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps." in *Proc. of EUROCRYPT '03*.

[28] A. Le and A. Markopoulou, "Locating Byzantine attackers in intra-session network coding using SpaceMac," in *Proc. of IEEE International Symposium on Network Coding (NetCod)*, 2010.

[29] Y.-C. Hu, D. B. Johnson, and A. Perrig, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," in *WMCSA*, 2002.

[30] M. Guerrero Zapata and N. Asokan, "Securing Ad hoc Routing Protocols," in *WiSe*, 2002.

[31] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens, "ODSBR: An on-demand secure byzantine resilient routing protocol for wireless ad hoc networks," *ACM Trans. Inf. Syst. Secur.*, 2008.

[32] R. Curtmola and C. Nita-Rotaru, "BSMR: Byzantine-resilient secure multicast routing in multi-hop wireless networks," in *SECON*, 2007.

[33] J. Dong, R. Curtmola, and C. Nita-Rotaru, "On the pitfalls of using high-throughput multicast metrics in adversarial wireless mesh networks," in *Proc. of SECON '08*, June 2008.

[34] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "RIPPLE authentication for network coding," in *Proc. of INFOCOM'10*, 2010.

[35] E. Kehdi and B. Li, "Null keys: Limiting malicious attacks via null space properties of network coding," in *Proc. of INFOCOM '09*, 2009.

[36] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in wireless network coding," *ACM Trans. Inf. Syst. Secur.*, vol. 14, pp. 7:1–7:31, June 2011.

[37] 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 WiSec '09*, 2009.

[38] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, "Byzantine modification detection in multicast networks using randomized network coding," in *Proc. of ISIT '04*, 2004.

[39] S. Jaggi *et al.*, "Resilient network coding in the presence of byzantine adversaries," in *INFOCOM '07*.

[40] S. Agrawal, D. Boneh, X. Boyen, and D. Freeman, "Preventing pollution attacks in multi-source network coding," in *Proc. of PKC 2010*, 2010.

[41] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "MIS: malicious nodes identification scheme in network-coding-based peer-to-peer streaming," in *Proc. of IEEE INFOCOM '10*, 2010.

[42] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network support for IP traceback," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, 2001.

[43] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for IP traceback," in *Proc. of IEEE INFOCOM '01*, 2001.

[44] P. Sattari, M. Gjoka, and A. Markopoulou, "A network coding approach to IP traceback," in *Proc. of IEEE International Symposium on Network Coding (NetCod)*, 2010.

[45] D. Silva, F. Kschischang, and R. Koetter, "A rank-metric approach to error control in random network coding," *IEEE Inf. Theory for Wireless Ntwks*, 2007.

[46] R. Koetter and F. R. Kschischang, "Coding for errors and erasures in random network coding," *IEEE Trans. on Information Theory*, 2008.

[47] R. W. Yeung and N. Cai, "Network error correction, part i: Basic concepts and upper bounds," *Commun. Inf. Syst.*, vol. 6, no. 1, pp. 19–36, 2006.

[48] N. Cai and R. W. Yeung, "Network error correction, part ii: Lower bounds," *Commun. Inf. Syst.*, vol. 6, no. 1, pp. 37–54, 2006.

[49] J. Dong, R. Curtmola, R. Sethi, and C. Nita-Rotaru, "Toward secure network coding in wireless networks: Threats and challenges," in *Proc. of NPSec*, 2008.

[50] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Secure network coding for wireless mesh networks: Threats, challenges, and directions," *Elsevier Computer Communications*, 2009.

[51] R. Curtmola and C. Nita-Rotaru, "BSMR: Byzantine-resilient secure multicast routing in multi-hop wireless networks," *IEEE Transactions on Mobile Computing (TMC)*, vol. 8, no. 4, 2009.

[52] H. Eberle, N. Gura, S. C. Shantz, V. Gupta, L. Rarick, and S. Sundaram, "A public-key cryptographic processor for RSA and ECC," in *ASAP '04*.

[53] H. Eberle, S. Shantz, V. Gupta, N. Gura, L. Rarick, and L. Spracklen, "Accelerating next-generation public-key cryptosystems on general-purpose CPUs," *IEEE Micro*, vol. 25, no. 2, pp. 52–59, 2005.

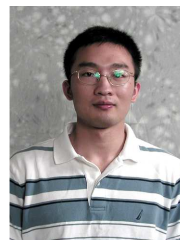
[54] "Global mobile information systems simulation library - glomosim," http://pcl.cs.ucla.edu/projects/glomosim/.

[55] J. Camp, J. Robinson, C. Steger, and E. Knightly, "Measurement driven deployment of a two-tier urban mesh access network," in *MobiSys '06*.

[56] D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *MobiCom '03*.

[57] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[58] J. Dong, R. Curtmola, C. Nita-Rotaru, and D. Yau, "Pollution attacks and defenses in wireless inter-flow network coding systems," in *Proc. of IEEE Int'l Workshop on Wireless Network Coding (WiNC)*, 2010.



Jing Dong received his PhD degree in Computer Science from Purdue University in 2009. During his PhD study, he was a member of the Dependable Distributed Systems Laboratory. He received his BS and MS degree in Computer Science in 2003 and 2004, both from University of Massachusetts, Boston. His research interests are in wireless networks with a focus on resilience and security of such networks.

Reza Curtmola is an Assistant Professor in the Department of Computer Science at NJIT. He received the B.Sc. degree in Computer Science from the "Politehnica" University of Bucharest, Romania, in 2001, the M.S. degree in Security Informatics in 2003, and the PhD degree in Computer Science in 2007, both from The Johns Hopkins University. He spent one year as a post-doctoral research associate at Purdue University. He is the recipient of the NSF CAREER award. His research focuses on storage security, applied cryptography, and security aspects of wireless networks. He is a member of the ACM and the IEEE Computer Society.

Cristina Nita-Rotaru is an Associate Professor in the department of Computer Science at Purdue University. She leads the Dependable and Secure Distributed Systems Laboratory. She received BS and MS degrees from Politechnica University of Bucharest, Romania, in 1995 and 1996, and a Ph.D. degree in Computer Science from The Johns Hopkins University in 2003. She served on the technical program committee of over 40 conference in networking, distributed systems, and security. She received the NSF CAREER award. Her research interests include security and fault-tolerance for distributed systems, and networks. She is a member of the ACM and IEEE Computer Society.

David K. Y. Yau (M'10) received the B.Sc. (first class honors) from the Chinese University of Hong Kong, and the M.S. and Ph.D from the University of Texas at Austin, all in computer science. He is currently Distinguished Scientist at the Advanced Digital Sciences Center, Singapore, and Associate Professor of Computer Science at Purdue University, West Lafayette, IN, USA. Dr. Yau was the recipient of an NSF CAREER award for research in quality of service. His other areas of research interest are protocol design and implementation, wireless and sensor networks, network security and incentives, and Smart Grid IT. Dr. Yau served as Associate Editor of *IEEE/ACM Trans. Networking* (2004–09); Vice General Chair (2006), TPC co-Chair (2007), and TPC Area Chair (2011) of IEEE Int'l Conf. Network Protocols; TPC co-Chair (2006) and Steering Committee member (2007–09) of IEEE Int'l Workshop Quality of Service; and TPC Track co-Chair of IEEE Int'l Conf. Distributed Computing Systems 2012.

Appendix A
Proof of Theorem 1

We first prove the sufficiency of the condition, that is, if for each flow $f_i, 1 \leq i \leq u$, there exists a subset P of $H(D(v, f_i))$ such that $\sum_{p \in P} p = \sum_{1 \leq i \leq k, i \neq i} m_i$, then the coded packet $e = m_1 \oplus m_2 \oplus \cdots \oplus m_k$ can be decoded by some downstream node of the coding node for the all flows $f_i, 1 \leq i \leq u$. Without loss of generality, consider a specific flow f_i, for some i between 1 and u. Let the subset P of the packets in $H(D(v, f_i))$ that satisfies the condition. We can divide P into l disjoint sets $P_i, 1 \leq i \leq l$, such that $P = \bigcup_{1 \leq i \leq l} P_i$ and each P_i is received at a node $v_i \in D(v, f_i)$. To decode the packet e, each v_i computes $e_1 = e \oplus \sum_{p \in P_i} p$ and forwards the resulting message to its downstream nodes. Then the resultant message e_l at node v_l is

$$e_l = e \oplus \sum_{1 \leq j \leq l} \sum_{p \in P_i} p = e \oplus \sum_{p \in P} p = e \oplus \sum_{1 \leq i \leq k, i \neq i} m_i = m_i$$

Hence, the packet e is decoded at node v_l.

We now prove the necessity of the condition. We show that if the coded packet e is decodable at flow $f_i, 1 \leq i \leq u$, then there exists a subset P of $H(D(v, f_i))$ that satisfies the equation $\sum_{p \in P} p = \sum_{1 \leq j \leq k, j \neq i} m_j$, for each f_i. Without loss of generality, we consider f_i for some $1 \leq i \leq u$.

Suppose the packet e is decodable to recover the plain packet m_i at some downstream node $v_l \in D(v, f_i)$. Let $v_1, v_2, \cdots, v_{l-1}$ be the sequence of nodes in $D(v, f_i)$ proceeding v_l. Since the only possible operation that a node v_i can perform on the coded packet is to perform xor operation of the coded packet with some of its received packet, let e_i be the resultant coded packet from node v_i, then

$$e_i = e_{i-1} \oplus \sum_{p \in P_i} p, \tag{1}$$

where $e_0 = e$ and P_i is some set of packets received at node v_i. Solve the recursive relation defined in Eqn. 1, we have

$$e_l = e \oplus \sum_{1 \leq j \leq l} \sum_{p \in P_i} p = e \oplus \sum_{p \in P} p,$$

where P is a set of packets consists of all the packets whose total number of appearance in each P_i is an odd number. Clearly, $P \subseteq \bigcup_{1 \leq j \leq l} P_i \subseteq H(D(v, f_i))$.

Since the packet is decoded at node v_l, *i.e.*, $e_l = m_i$, we have $\sum_{p \in P} p = \sum_{1 \leq i \leq k, i \neq i} m_i$. Hence, there exists a subset P of $H(D(v, f_i))i$ that satisfies the equation $\sum_{p \in P} p = \sum_{1 \leq j \leq k, j \neq i} m_j$.

Appendix B
A polynomial time algorithm for finding a subset P that satisfies the coding condition

We formally state the problem as follows. Given a set of packets $S = H(D(v, f_i))$ containing both plain and coded packets, a coded packet $e = m_1 \oplus m_2 \oplus \cdots \oplus m_k$, and a packet $m_i, 1 \leq i \leq k$, find a subset of packets $P \subseteq S$ if it exists, such that

$$e \oplus \sum_{p \in P} p = m_i. \tag{2}$$

Naively, one can try all possible subsets of S and see if Eqn. 2 holds, but this would require an exponential amount of time. Note that the problem is a specialized form of the general subset sum problem [57], which is NP-hard. The special property of XOR operation allows us to solve this problem in polynomial time by transforming the problem into solving a system of linear equations as follows.

Without loss of generality, we consider $m_i = m_1$ in Eqn. 2. Let $S = \{p_1, p_2, \cdots, p_s\}$, where each $p_i = \sum_{j=1}^{u_i} m_j^i$, with each m_j^i being a plain packet. Due to the property of XOR, we can assume each m_j^i is unique, since otherwise they would cancel out. Thus, we can also regard p_i as a set of plain packets $\{m_1^i, m_2^i, \cdots, m_{u_i}^i\}$. Note that if $u_i = 1$, p_i is a plain packet.

Assume the subset $P \subseteq S$ exists such that $e \oplus \sum_{p \in P} p = m_i$, or equivalently, $\sum_{p \in P} p = \sum_{i=2}^{k} m_i$. We assign a binary variable b_i to each packet p_i in S, such that $b_i = 0$ if $p_i \in P$, or $b_i = 1$ if $p_i \notin P$.

Denote the set $\{m_2, m_3, \cdots, m_k\}$ as T. Regarding p_i as a set of plain packets as discussed above, denote $\mathcal{P} = \bigcup_{i=1}^{s} p_s$. Observe that for each $m \in T$, it must be the case that the total number of occurrences of m in all $p \in P$ is odd, and for all $m \in \mathcal{P}$ and $m \notin T$, it must be the case that the total number of occurrences of m in all $p \in P$ is even. Thus, we can establish a system of linear equations that b_i must satisfy as follows. For each $m'_j \in \mathcal{P}$, we add the following equation to the system, $\sum_{1 \leq i \leq s, m'_j \in p_i} b_i = c_j$, where $c_j = 1$ if $m'_j \in T$, or $c_j = 0$ if $m'_j \notin T$.

If the system of linear equations admits no solution, then the subset P does not exist. Otherwise, we can determine P by including only those p_i for which their corresponding $b_i = 1$.

Let $l = \max_{1 \leq i \leq s}(|p_s|)$, then the total number of equations in the system is at most ls and the total number of variables per equation is at most s, thus the size of the system of linear equations is $O(ls^2)$, and can be solved in polynomial time using the standard method of Gaussian elimination.

Appendix C
Analysis of Packet Storing Time

We analyze the time duration a node needs to store the input packets for the coded packets it generates in order to provide proof of coding correctness to the source node when queried.

Let Δ be the maximum path latency between any two nodes in the network, t_s, t_r, t_q be the time when the source sends out the packet, the node receives the packet, and the source sends out a query to the node for the proof of coding correctness for the packet, respectively. Denote T to be the time duration that a node stores the input packets for a coded packet it generates. Then in order for the node to be able to supply the input packets as its proof of coding correctness, we must have $t_q + \Delta \leq t_r + T$, or equivalently, $t_q \leq t_r + T - \Delta$. Since $t_r \geq t_s$, it is sufficient to have $t_q \leq t_s + T - \Delta$. That is, the source node should send out query for proof of coding correctness no later than $T - \Delta$ after the time it sends out the packet.

Let the node be the kth node being queried for the proof of coding correctness. Without considering packet delaying attacks, the time that the source queries the node for the proof of its coding correctness is at most $t_s + 2k\Delta$. Thus we need to have $T - \Delta \geq 2k\Delta$, or $T \geq (2k+1)\Delta$. Since the number of multi-coding involved for a packet typically is small, we can

assume $k \leq n_c$, where n_c is a system parameter that upper bounds the maximum number of multi-coding performed for a packet. For example, in single-hop coding and multi-hop coding systems, we have $n_c = 1$. By taking k at its maximum value, we have $T \geq (2n_c + 1)\Delta$. In single-hop or multi-hop coding systems, where $n_c = 1$, we have $T \geq 3\Delta$. In the general case that includes multi-coding, since it is very rare for a packet to have more than two or three coding nodes involved, it is sufficient to set n_c to be a small number, *e.g.*, 5.

Since the only overhead for a large T is more storage space, which is getting increasing more abundant and less expensive, we can set T to be much larger than the lower bound we derived. For example, assuming a packet size of 1KB, a storage space of 1GB, and a packet rate of 1000 input packets for coding per second, we can store a packet for over 1000 seconds before the storage is full. The overly large value for T also allows tolerance to packet delaying attacks that delay packets up to $T - T_{min}$, where $T_{min} = (2n_c + 1)\Delta$ is the lower bound of T. For example, in a typical network with $\Delta = 100ms, n_c = 5$, setting $T = 1000$ seconds, allows tolerance of packet delay up to nearly 999 seconds. Such a long packet delay can be treated as a packet drop, and can be addressed with existing techniques such as watchdog [15].