2009

Practical security scheme design for resource-constrained wireless networks

Zhen Yu
Iowa State University

Practical security scheme design for resource-constrained wireless networks

by

Zhen Yu

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Iowa State University

Ames, Iowa

2009

DEDICATION

To my wife Min and my son Gavin —

Without your love and support, I would not have been able to complete this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this dissertation.

First and foremost, I would like to thank my advisor Dr. Yong Guan for his support, encouragement and an endless source of ideas. His breadth of knowledge, his enthusiasm for research, and his vision for the future amaze and inspire me. I thank him for the countless hours he has spent with me, discussing everything from research to career choices, reading my papers, and guiding my studies. His direction during my time at ISU has been invaluable — my life has been enriched professionally, intellectually, and personally by working with Dr. Guan.

I am greatly thankful to my labmates, Yawen Wei and Bhuvaneswari Ramkumar, for their collaboration during this research and contributions to this dissertation.

I would like to convey my appreciation sincerely to the committee members, Dr. Arun Somani, Dr. Manimaran Govindarasu, Dr. Daji Qiao, and Dr. Johnny Wong, for their time and input. I am also thankful to Dr. Lei Ying for his valuable discussions.

My deep gratitude goes to my wife (Dr. Min Xu), my son (Gavin Yu), and my parents (Guojian Yu and Yuying Liu) for their love, sacrifice and support during my life.

ABSTRACT

The implementation of ubiquitous computing (or pervasive computing) can leverage various types of resource-constrained wireless networks such as wireless sensor networks and wireless personal area networks. These resource-constrained wireless networks are vulnerable to many malicious attacks that often cause leakage, alteration and destruction of critical information due to the insecurity of wireless communication and the tampers of devices. Meanwhile, the constraints of resources, the lack of centralized management, and the demands of mobility of these networks often make traditional security mechanisms inefficient or infeasible. So, the resource-constrained wireless networks pose new challenges for information assurance and call for practical, efficient and effective solutions.

In this research, we focus on wireless sensor networks and aim at enhancing confidentiality, authenticity, availability and integrity, for wireless sensor networks. Particularly, we identify three important problems as our research targets: (1) key management for wireless sensor networks (for confidentiality), (2) filtering false data injection and DoS attacks in wireless sensor networks (for authenticity and availability), and (3) secure network coding (for integrity).

We investigate a diversity of malicious attacks against wireless sensor networks and design a number of practical schemes for establishing pairwise keys between sensor nodes, filtering false data injection and DoS attacks, and securing network coding against pollution attacks for wireless sensor networks. Our contributions from this research are fourfold: (1) We give a taxonomy of malicious attacks for wireless sensor networks. (2) We design a group-based key management scheme using deployment knowledge for wireless sensor networks to establish pair-wise keys between sensor nodes. (3) We propose an en-route scheme for filtering false data injection and DoS attacks in wireless sensor networks. (4) We present two efficient schemes for securing normal and XOR network coding against pollution attacks. Simulation and experimental results show that our solutions outperform existing ones and are suitable for resource-constrained wireless sensor networks in terms of computation overhead, communication cost, memory requirement, and so on.

CHAPTER 1 OVERVIEW

1.1 Introduction

Ubiquitous computing [74] (or pervasive computing) integrates computing technology into our day lives, embeds computing devices into our physical surroundings, and provides us the ability of computing (or information processing) everywhere. The implementation of ubiquitous computing can leverage the techniques such as wireless sensor networks [2, 3] and wireless personal area networks [80]. We term these networks *resource-constrained wireless networks*, which typically contain a lot of tiny wireless devices that are limited in power resource, computation capacity, memory size and communication range.

The resource-constrained wireless networks suffer various malicious attacks, because wireless communication through the air is insecurely open to the public and the wireless devices are not tamper-resistant. For instance, adversaries can easily eavesdrop on wireless communication to gain confidential information. They can interfere with wireless communication to change or disrupt the transmitted information. Through compromises devices, the adversaries can inject false information into networks, alter or drop the forwarded messages, or flood a large number of messages to cause networks unavailable. These attacks lead to leakage, alteration and destruction of critical information, and violate the basic requirement of information assurance such as confidentiality, authenticity, availability and integrity.

Besides the property of resource-constrained, these networks are often distributed and lack centralized management, while their wireless devices may even be mobile. Thus, many traditional security mechanisms supporting information assurance become inefficient or infeasible for these networks. For example, public-key cryptography may not be suitable for these networks due to the high computation overhead and the lack of the desired infrastructure within the networks, while the mobility of wireless devices greatly increase the complexity of group key manage. So, the resource-constrained wireless networks pose new challenges for information assurance and call for more practical, efficient and effective solutions.

Among different types of resource-constrained wireless networks, wireless sensor networks are the most popular one that has been widely used for various applications such as battlefield surveillance, target tracking and traffic control. In this research, we focus on wireless sensor networks and aim at enhancing confidentiality, authenticity, availability and integrity for wireless sensor networks. Particularly, we identify three important problems as our research targets:

1. *Key management for wireless sensor networks (for confidentiality).* Wireless communication between sensor nodes should be secured, which demands the efficient distribution of secret keys. Key management provides not only the fundamental cryptographic services, but also the basic component to construct other security mechanisms, hence, should be carefully studied.

2. *Filtering false data injection and DoS attacks in wireless sensor networks (for authenticity and availability).* For various applications to function correctly, it is important to make sure that valid information can be delivered to desired destinations. However, false data injection produces invalid information and DoS attacks disrupts information delivery. So, they must be efficiently filtered.

3. *Secure network coding (for integrity).* Network coding [1, 50] is promising to maximize network throughput and gains more and more applications in wireless networks. However, it poses a lot of new security problems that have not been well addressed.

We investigate a diversity of malicious attacks against wireless sensor networks and design a number of practical schemes for establishing pairwise keys between sensor nodes, filtering false data injection and DoS attacks, and securing network coding against pollution attacks. Our contributions from this research are as follows:

1. We classify the malicious attacks into different categories and provide a taxonomy of these attacks.

2. For enhancing confidentiality, we design a group-based key management scheme for wireless sensor networks to establish pair-wise keys between sensor nodes.

3. For enhancing authenticity and availability, we propose a dynamic en-route scheme for filtering false data injection and DoS attacks in wireless sensor networks.

4. For enhancing integrity, we present two schemes for securing network coding against pollution attacks, where one of our solutions is the first one addressing secure XOR coding problem.

1.2 Resource-constrained Wireless Networks

1.2.1 Ubiquitous Computing

Mark Weiser, the Chief Technologist of Xerox Palo Alto Research Center, coined the term *Ubiquitous Computing* [74] as "*Computers everywhere. Making many computers available throughout the physical environment, while making them effectively invisible to the user.*"

He also said that "*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*"

Ubiquitous computing provides us the ability to process information everywhere. For example, a person's location can be automatically tracked by sensor networks (Active Badge [77]) so that the telephone calls or emails could be forwarded to the correct location, while a user may use a handheld personal device (Xerox PARCTAB [82]) to gain access to other personal devices, LAN or Internet. We envision that a lot of small, inexpensive, wireless devices such as sensor nodes, mobile phones, PDAs and RFID tags, can be distributed at all scales throughout everyday life and turned to distinctly quotidian ends to support ubiquitous computing. That is, the implementation of ubiquitous computing can leverage the techniques such as wireless sensor networks [2, 3] and wireless personal area networks [80].

1.2.2 Wireless Sensor Networks

Akyildiz et al, [3] stated that "*Recent advances in micro-electro-mechanical systems (MEMS) technology, wireless communications, and digital electronics have enabled the development of low-cost, low-power, multifunctional sensor nodes that are small in size and communicate untethered in short distances. These tiny sensor nodes, which consist of sensing, data processing, and communicating components, leverage the idea of sensor networks based on collaborative effort of a large number of nodes.*"

The specifications of some types of sensor nodes can be found as follows:

Crossbow MICAz [55]:

- Microprocessor: Atmel ATmega 128L (8 MHz)
- Program Memory: 4K RAM
- External Memory: 128K Flash ROM
- RF Transceiver: 802.15.4/ZigBee compliant (2.4 GHz)
- Data Rate: 250 kbps
- Communication Range: 20-100 m
- Battery: 2× AA
- Weight: 0.7 oz

Crossbow TelosB [71]:

- Microprocessor: TI MSP430 (8 MHz)
- Program Memory: 10K RAM
- External Memory: 48K Flash ROM
- RF Transceiver: 802.15.4 compliant (2.4 GHz)
- Data Rate: 250 kbps
- Communication Range: 20-100 m
- Battery: 2× AA
- Weight: 0.8 oz

ETH Zurich BTnode [11]:

- Microprocessor: Atmel ATmega 128L (8 MHz)
- Program Memory: 64+128K RAM
- External Memory: 128K Flash ROM + 4K EEPROM
- RF Transceiver: Chipcon CC1000 (433-915 MHz) and Bluetooth (2.4 GHz)
- Battery: 2× AA
- Size: 58.15 x 33 mm attached to a 2× AA battery holder

1.2.3 Wireless Personal Area Networks

A wireless personal area network (WPAN) is a personal, short distance area wireless network for interconnecting devices centered around an individual person's workspace. The initial incarnation of ubiquitous computing was in the form of *Tab* [82], *Pad* [83], and *Board* built at Xerox PARC, 1988-1994, while the *Tab* and *Pad* can be assumed as devices of WPAN.

The Tab was a prototype handheld computer, which was 2"×3"×0.5", had a 2 week battery life on rechargeable batteries, and weighed 7 oz. It used a Phillips 8051 processor with 128k NVRAM. It featured an external I^2C external bus, a custom resistive touch screen, and a 128×64 mono display. A complete Tab system included an infrared base station in the ceiling for LAN connectivity.

The Pad was a prototype pen computer, which was 9"×11"×1", had a four 4 hour battery life, and weighed 5 oz. It used a Motorola 683×× processor with 4MB RAM running a unique real-time operating system. The Pad featured a PCMCIA slot, an electronic pen of our own design with a built-in microphone, a 640×480 4 level display, keyboard and serial ports. The Pad could communicate through infrared at 19.2kbs, through a unique near field radio at 240kbs, and through a 1Mbs tether which also supplied external power for operation and recharging. A complete Pad system also had a radio base station that served as a recharging station and Ethernet gateway for the Pads.

1.2.4 Observations

From the specifications of sensor nodes, Tab and Pad, we conclude that they are typically tiny wireless devices with limited power resource, computation capacity, memory space, data rate and communication range. Thus, we term the networks of these devices *resource-constrained wireless networks*, because these devices are often wireless with limited resources.

We also observe that the resource-constrained wireless networks are typically distributed, because a base station is unnecessary to be present during all the life time of the networks. Thus, such networks may lack a centralized management. In addition, the mobility is often demanded for the devices of these networks.

1.3 Taxonomy of Malicious Attacks against Wireless Networks

Wireless networks are vulnerable to various malicious attacks. We list some possible attacks as follows:

- *Eavesdropping*: The adversaries may obtain critical or sensitive information by eavesdropping on wireless communication.
- *Eavesdropping*: The adversaries may observe traffic flows to deduce information from the patterns of wireless communication, even when the flows are encrypted and cannot be decrypted.
- *Removing/Moving*: The adversaries may remove some wireless nodes or move them to other locations to make the network partitioned or the topology changed,
- *Jamming*: The adversaries can broadcast a high-power signal to disrupt or interfere with wireless communication.
- *Replaying*: The adversaries may replay the previously received messages to disturb the functionalities of wireless networks.
- *Wormhole*: The adversaries can record the information received at one location and replay them at another location via some *wormhole* tunnel, e.g., a fast wired link, which fools the wireless nodes far from each other to believe they are neighbors.
- *Dropping/Selective Forwarding*: The adversaries may compromise some nodes, and drop all or some of the messages that should be forwarded by those nodes.
- *Flooding*: The adversaries may inject a huge amount of useless information into wireless networks via some compromised nodes to disrupt wireless communication and/or deplete the energy of wireless nodes.
- *Modification/Pollution*: The adversaries can modify or corrupt the information transmitted in wireless networks.

- *False Data Injection*: The adversaries can inject false information into wireless networks to disturb the functionalities of wireless networks and/or deplete the energy of wireless nodes.

- *Impersonating/Sybil Attack*: A compromised node can impersonate another legal node or claim the identities of multiple legal nodes.

We categorize these attacks into different ways and provide the following taxonomy. Careful and thorough classification is important for correctly addressing these attacks.

- Attacks can be *passive* or *active*. Passive attacks do not generate new (malicious) information or modify old information. They are hard to observe and hence may not be well protected without careful design. For example, traffic analysis is a passive attack and can not be protected using encryption mechanisms solely. Active attacks either produce new (malicious) information and/or change old one. One example is flooding.

- The targets of malicious attacks may be different. Information, wireless node, or network services are all possible targets. Identifying the targets of attacks is crucial to address these attacks. For example, the target of eavesdropping is information that should be encrypted, while that of impersonating attacks is wireless nodes that should be authenticated.

- Attacks may occur at different *protocol layers* such as physical layer, MAC layer, network layer, transport layer and application layer. This classification help us select security mechanisms properly. For example, jamming is a physical layer attack that can be addressed using advanced modulation techniques such as DSSS or FHSS, while sybil attack works on application layer and can be countered using authentication approaches.

- Some attacks happen only when some special *condition* is satisfied. For example, dropping attacks require some nodes compromised, but replaying attacks do not. So, making nodes tamper-resistant can effectively deal with dropping attacks, but cannot prevent replaying.

- Attacks can be *intermittent* or *persistent* from the perspective of how long they last. Persistent attacks last long, but intermittent ones happen and disappear very quickly. So in most cases, intermittent attacks do not demand us to design specific countermeasures.

We note that some complicated attack may even consist of multiple attacks from different categories. For example, a Blackhole attack [42] against routing protocols consists of a wormhole and a dropping attack, where the adversaries first create a wormhole to attract data from victim nodes and then drop the data.

1.4 Information Assurance

National Information Assurance Glossary, Committee on National Security Systems (CNSS) Instruction No. 4009 [21], defines *Information Assurance* as:

Measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. These measures include providing for restoration of information systems by incorporating protection, detection, and reaction capabilities.

Generally, we consider the following security goals of information assurance.

- *Confidentiality*: Confidential information cannot be disclosed to unauthorized users, processes, or devices.

- *Integrity*: Information cannot be created, modified or deleted without proper authorization.

- *Availability*: Information, the computing systems used to process the information, and the security controls used to protect the information are all available and functioning correctly when the information is needed.

- *Authenticity*: The identity of users, processes, or devices, and the source of information can be correctly verified or is genuine.

- *Non-repudiation*: The sender of information cannot deny having sent the information and the receiver cannot deny having received it.
- *Access control*: The access to information is only granted to an authorized user.
- *Privacy*: A user is secure from unauthorized disclosure of information about oneself.

Among these goals, *confidentiality*, *integrity* and *availability* (known as CIA triad) are the most important and core principles.

Although these security goals are suitable for any type of networking systems such as Internet, WLAN and mobile ad-hoc networks, achieving these goals in resource-constrained wireless networks pose new challenges. For example, the traditional approaches for achieving authenticity based on public-key cryptography may be impractical for wireless sensor networks, because (1) public-key algorithms are too expensive for sensor nodes due their high computation requirement and energy consumption, and (2) the distributed wireless sensor networks might not be able to provide the desire public-key infrastructure. Another example is that achieving availability is difficult in wireless sensor networks, because sensor nodes are often failed or temporarily unavailable due to power depletion or unexpected interference. Our research is motivated by addressing these special challenges to enhance information assurance for resource-constrained wireless networks.

1.5 Objectives of Research

Resource-constrained wireless networks are subject to malicious attacks and demand efficient mechanisms for providing information assurance. In this research, we focus on wireless sensor networks and aim to enhance *confidentiality*, *integrity*, *availability* and *authenticity* for wireless sensor networks, because these are the core principles of information assurance. Particularly, we identify three problems as our research targets, which are: (1) key management for wireless sensor networks (for confidentiality), (2) filtering false data injection and DoS attacks in wireless sensor networks (for authenticity and availability), and (3) secure network coding (for integrity).

Wireless communication is prone to eavesdropping. In wireless sensor networks, distributing secret keys among sensor nodes allows them to secure their communication with encryption. Key management, including key generation, distribution, revocation and update, not only provides the basic cryptographic service, but also are critical to implement other security mechanisms. However, there are still a lot of unsolved problems in providing efficient key management for wireless sensor networks. For example, sensor nodes may not have suffice memory for key storage and cannot support heavy public-key algorithms. These problems motivate us to study key management for wireless sensor networks.

Wireless sensor networks support various applications such as battlefield surveillance, target tracking or traffic control, in which sensor nodes should report detected events or sensing readings to the base station. However, the data reports are vulnerable to forging, modifying and dropping and the adversaries can inject false data into networks. The forged reports about battlefield may cause false alarms in the base station and the dropped reports might be critical for traffic control. So, it is important to ensure that the information transmitted with wireless sensor networks is valid, correct and available. Some solutions for filtering false data injection are not efficient and robust for dynamic sensor networks, while few of them consider DoS attacks simultaneously. Thus, we are inspired to address both attacks for wireless sensor networks.

Network coding [1, 50] is a new forwarding technique that allows a forwarder to encode its input messages for creating a output one. The advantages of network coding include (but not limited to) (1) maximizing network throughput, (2) reducing the number of forwarding messages, and (3) reducing the number of retransmissions. These nice properties make network coding technique widely applied in peer-to-peer systems, wireless networks and sensor networks. However, network coding poses new challenges to security. For example, because each forwarder (including compromised ones) decides how to generate its output messages independently, it is hard to detect if a message is a corrupted one or a valid one. Moreover, a smart adversary can generate *valid* messages that are linearly dependent on other previously transmitted ones. These *valid* messages are useless for decoding by receivers. Few research has been conducted on security issues in network coding. We identify *secure network coding* as a promising research area that involves a diversity of problems, which have not been discovered and solved.

1.6 Contributions of Research

From this research, we make the following contributions.

1. We classify the malicious attacks into different categories and provide a taxonomy of these attacks.

2. For enhancing confidentiality, we design a group-based key management scheme for wireless sensor networks to establish pair-wise keys between sensor nodes. Compared with others, our scheme has a stronger resilience against node capture, while achieving higher connectivity with lower memory cost and shorter transmission range.

3. For enhancing authenticity and availability, we propose a dynamic en-route scheme for filtering false data injection and DoS attacks in wireless sensor networks. Compared with others, our scheme offers higher filtering capacity with lower memory cost and can better deal with dynamic topology of wireless sensor networks.

4. For enhancing integrity, we present two schemes for securing network coding against pollution attacks, where one of our solutions is the first one addressing secure XOR coding problem. Compared with others, our schemes do not need any extra security channels and are two to three orders of magnitude faster. Hence, they are promising for wireless sensor networks.

1.7 Dissertation Organization

The rest of this dissertation contains 6 chapters that are organized as follows: In chapter 2, we provide a review of literature for the three problems targeted in our research. In chapter 3, we study how to provide efficient key management for wireless sensor networks for enhancing confidentiality and design a group-based key pre-distribution scheme using deployment knowledge for establishing pair-wise keys between sensor nodes. In chapter 4, we consider how to filter false data injection and DoS attacks in wireless sensor networks for enhancing authenticity and availability simultaneously. We propose a dynamic en-route scheme that filters false data injection and mitigates the impact of DoS attacks. In chapter 5 and 6, we research how to secure network coding against pollution attacks for enhancing integrity. We present a homomorphic signature scheme for securing normal network coding systems in chapter 5. In chapter 6, we further propose an efficient solution that is specifically designed for securing XOR network coding. In chapter 7, we summarize our research and discuss our future work.

CHAPTER 2 REVIEW OF LITERATURE

2.1 Key Management for Wireless Sensor Networks

2.1.1 Introduction

Wireless sensor networks may be deployed in uncontrolled or even hostile environments and hence are subject to various attacks. For example, an adversary can easily gain access to mission critical information by monitoring wireless communication among sensor nodes, or inject false messages into the networks through some compromised nodes. Therefore, it is crucial to deploy secret keys into wireless sensor networks to encrypt wireless communication or establish authentication among sensor nodes. The challenge is how to efficiently generate, distribute and maintain secret keys among sensor nodes. This problem is called key management problem for wireless sensor networks and can be solved by carefully designed key management schemes.

In this work, we survey existing key management schemes for wireless sensor networks and provide a taxonomy of them. Most of schemes deal with symmetric key management, while a few of them discuss public key management. Since public key algorithms are generally considered too expensive for the use in wireless sensor networks, we focus our discussion on symmetric key management schemes. We divide these schemes into pairwise key, group key and global key schemes depending on what kind of keys they distribute and manage, and category them into probabilistic, deterministic and hybrid depending on how possible the keys can be generated. We also classify these schemes depending on whether they exploit deployment knowledge or not. In the rest of this section, we first introduce threat model and goals of key management schemes, and then discuss the pairwise key schemes, the group key schemes and a global key scheme.

2.1.2 Threat Model and Goals

The adversaries can launch malicious attacks in passive and active modes. In passive mode, they can eavesdrop wireless communications among sensor nodes to capture mission critical or private information. In active mode, they may deploy their own malicious nodes or capture some sensor nodes in the networks. Once some nodes are captured and compromised, all secret keys or secret information stored in these nodes will be revealed and the adversaries may fully control these nodes. Through these malicious nodes or compromised nodes, the adversaries can inject false messages, and modify or drop the messages transmitted in the networks. Using the secret keys obtained from the compromised nodes, the adversaries could eavesdrop additional communications encrypted by these keys.

The purpose of key management for wireless sensor networks is to distribute and manage the secret keys of sensor nodes to secure wireless communications among sensor nodes and provide a basic mechanism for other security protocols. Specifically, key management schemes are expected to achieve the following goals:

1. *Connectivity*: The deployed sensor networks should be highly connected. Here, we emphasis *key connectivity*, which means two nodes are only considered as connected when two conditions are satisfied: (1) they are physical neighbors; and (2) they share at least one secret key.

2. *Resilience*: The sensor networks should be resilient against node capture. That is, the compromise of some secret keys (or secret information) would not reveal much information of additional communications or links.

3. *Efficiency*: The designed schemes should be efficient for sensor nodes with limited memory, computation and communication capacity.

4. *Scalability*: The schemes should be able to support large-scale sensor networks.

These goals may be contradictive to each other. Hence, we should make tradeoffs among these goals when designing key management schemes.

2.1.3 Taxonomy of Key Management Schemes

In general, key management process involves three phases, *key setup phase*, *key discovery phase* and *key update phase*. In key setup phase, secret keys or secret information are generated and carefully preloaded into sensor nodes. In key discovery phase, the deployed sensor nodes communicate with each other and exchange information to discover or generate secret keys. Moreover, two neighbors may establish secret keys through other discovered secure path(s) if they cannot find secret keys directly. In key update phase, compromised or obsolete keys are revoked and new keys are distributed or generated. We notice that key update (or rekeying) is a hard problem and has not been fully solved so far.

Key management schemes for wireless sensor networks can be classified into various categories. Table 2.1 provides a taxonomy of these schemes. First, depending on what kind of keys the schemes deal with, we divide these schemes into three classes, pairwise key schemes, group key schemes and global key schemes. These keys are used for encryption or authentication of communications between a pair of nodes, among a group (or cluster) of nodes and over the whole networks. We further classify the schemes into probabilistic, deterministic and hybrid, depending on whether the keys can be established with some probability or deterministically. Hybrid schemes apply some deterministic approach over a probabilistic one. Typically, we assume key distribution process is secure before sensor nodes are deployed. However, some researchers also suppose there exists a short security period even after the nodes are deployed to target field, because they believe adversaries need at least such a short period of time to find and compromise a sensor. Thus, we have two classes, pre-deployment and post-deployment security schemes. Depending on whether using deployment knowledge for the design schemes, we divide the schemes into two categories, with or without deployment knowledge. There are other metrics to classify the schemes, for example, (1) what kind of secret information is pre-loaded into sensors, secret key, key vector or polynomial; and (2) the pairwise key is pre-distributed into sensors or dynamically generated after sensors are deployed. Due to page limit, we do not discuss these classifications here.

Table 2.1 A Taxonomy of Key Management Schemes for Wireless Sensor Networks

	Pairwise Key				**Group Key**	**Global Key**
	Probabilistic	**Deterministic**		**Hybrid**	**Deterministic**	**Deterministic**
		Pre-deployment Security	**Post-deployment Security**			
No Deployment Knowledge	[25, 17, 97], [66, 37, 38]	[8], [9, 18]	[24, 96]	[22, 51]	[9, 96]	[63]
Deployment Knowledge	[23, 52], [53, 36]			[52, 88], [94, 95]		

2.1.4 Pairwise Key Management Schemes

2.1.4.1 Probabilistic Schemes without Using Deployment Knowledge

Eschenauer and Gligor [25] proposed the first probabilistic key pre-distribution scheme, called *basic scheme*. In key setup phase, each node is randomly preloaded with m keys from a global key pool of size M. Hence, a pair of nodes can establish a secure link with probability $p = 1 - \frac{\binom{M-m}{m}}{\binom{M}{m}}$. From random graph theory, the desired probability P_c that the entire network is connected can be achieved by choosing a proper p. The drawback of this scheme is that a pairwise key may be used by multiple pair of nodes. Given x nodes compromised, the adversaries can compromise an additional link with probability $1 - (1 - \frac{m}{M})^x$.

To improve resilience, Chan et al. [17] extended the basic scheme to *q-composite scheme*, which requires a pair to share at least $q > 1$ keys to establish a secure link. However, this scheme sacrifices the achievable connectivity. Hence, the authors proposed *random pairwise-key scheme*, which keeps the achievable connectivity when improving resilience. The idea is tat for each node, a set of m nodes are randomly chosen, and a unique pairwise key is assigned for this node and every node in the set. Since all pairwise keys are distinct, the scheme has a perfect resilience against node capture, but the size of network is limited by $\frac{m}{p}$.

In the basic scheme, each node is preloaded with m keys and has to broadcast all m key IDs to discover the shared key(s). Hwang et al. [37] proposed *cluster grouping scheme* to reduce the communication overhead of sensor nodes. In this scheme, the keys stored by each node are divided into c ($c < m$) equal-length clusters where each cluster has a start key ID. The remaining key IDs within the cluster are implicitly known from the start key ID. Hence, each node only needs to broadcast c start key IDs, instead of m key IDs. Zhu et al. [97] presented a pairwise key scheme which can reduce not only the communication overhead and but also the required storage. In this scheme, each sensor is assigned with a unique ID and a pseudo-random function f, where its key IDs can be determined by executing $f(ID)$. Thus, each node only needs to broadcast its IDs to discover the shared key(s). Similarly, Ren et al. [66] also proposed to use hash chains to construct the global key pool. Hence, for each chain assigned to a node, the node only needs to store the corresponding generation key and the hash function. In addition, Hwang and Kim [38] revisited the basic scheme and its derivatives, and proposed to reduce the number of keys stored by each node while still keeping a certain probability of sharing a key between two nodes. The basic idea is that probability is sufficient to guarantee the largest component, instead of the whole network, to be almost connected.

2.1.4.2 Deterministic Schemes

Blom [8] proposed a deterministic scheme to set up pairwise keys for a group of N nodes. First, a key distribution center constructs a $(t+1)\times(t+1)$ symmetric matrix D and a $(t+1)\times n$ public matrix G over a finite field, where $(DG)^T$ is called secret matrix. All pairwise keys of these N nodes are stored in a symmetric matrix $K = (DG)^T G$. Then, each node i is preloaded with the i-th row of secret matrix and the i-th column of public matrix. After deployment, any two nodes i and j can individually derive their pairwise key $k_{ij} = k_{ji}$ by only exchanging their columns. This scheme is called t-secure, that is, no additional key is revealed given that up to t nodes are compromised. However, when more than t nodes are compromised, the whole matrix is broken. Although we can improve the resilience of the scheme by increasing t, each node has to store more amount of secret information, which is $O(t)$.

Blundo [9] presented a *polynomial-based key management scheme*, which is a special case of Blom's scheme when matrix D is a Vandermonde matrix. The basic component of this scheme is t-degree bivariate polynomial $f(x,y) = \sum_{i=0}^{t}\sum_{j=0}^{t} a_{ij}x^i y^j$ over a finite filed, where we have $f(x,y) = f(y,x)$ by choosing $a_{ij} = a_{ji}$. Each node i is preloaded with a polynomial share $f(i,y)$. After deployment, any two nodes i and j can individually derive their pairwise key $f(i,j) = f(j,i)$ by evaluating their own share at the point of the peer's IDs. Similar to Blom's scheme, Blundo's scheme is also t-secure.

Chan and Perrig [18] introduced a deterministic scheme called *Peer Intermediaries for Key Establishment* (PIKE), in which all N nodes are organized into a two-dimensional space and each node has a unique coordinate (x,y), where $x,y \in [0,\sqrt{N}-1]$. Each node shares unique pairwise keys with $2(\sqrt{N}-1)$ nodes that have the same x or y coordinate. If two nodes with no common x or y coordinate, they need to choose an intermediate node that has common x

or y coordinate with both of them to help them establish a pairwise key securely. The problem of PIKE is its high communication overhead, because a node can only establish pairwise keys with $2(\sqrt{N}-1)$ nodes directly and needs to find multi-link path for any of other neighbors.

All deterministic schemes discussed above have a common assumption, that is, the adversaries can compromise sensor nodes as long as they are deployed. However, this assumption might be too strong. Although sensor nodes are not tamper-resist, the adversaries need at least some short period of time to find, break, and control a sensor node. We call that period *post-deployment security window*. Based on this weak threat model, key management schemes can be designed more efficiently and effectively. One solution is *Localized Encryption and Authentication Protocol* (LEAP) proposed by Zhu et al. Each node i is first preloaded with an initial key K_I and a pseudo-random function f, which can determine the node's master key $k_i = f_{K_I}(i)$. Within the security window after deployment, two nodes i and j exchange their IDs and can calculate the pairwise key $K_{ij} = f_{k_j}(i)$. At the end of security window, all nodes remove K_I from their memory, so the adversaries cannot calculate the pairwise keys of other links, even when they compromise some nodes. With this scheme, node addition is also simple. Each new node maintains K_I within the period of security window after deployment and hence it can calculate the pairwise keys with its neighbors. Moreover, Dutertre et al. [24] proposed a similar solution with the same post-deployment security assumption.

2.1.4.3 Hybrid Schemes

Deterministic schemes such as Blom's and Blundo's ones provide perfect resilience as long as the number of compromised nodes is below the threshold value. They can be applied over probabilistic key pre-distribution schemes to further improve the resilience of these schemes. Du et al. [22] combined the basic scheme and *Blom*'s scheme together and designed a *multi-space key pre-distribution scheme*. First, one public matrix G and ω symmetric matrices D_i (where $i = 1, ..., \omega$) are constructed. These matrices form ω spaces (D_i, G). Then, each node randomly selects τ spaces and stores the corresponding rows of spaces. After deployment, any two nodes can establish a pairwise key if they happen to select the same space. This scheme has a similar threshold property as that of original Blom's scheme, which is that no additional links will be compromised given the number of compromised nodes less than a threshold value. However, after that threshold value, the whole network will be quickly broken. Similarly, Liu and Ning [51] proposed to use Blundo's scheme to improve the security of the basic scheme. In one approach, each node randomly selects a subset of polynomials from a pool and stores the corresponding polynomial shares of the selected polynomials. In another approach, N nodes are organized into $m \times m$ grids, where each node is assigned with a coordinate (i, j) and each row i or column j of grids is associated with a polynomial $f_i^c(x, y)$ or $f_j^r(x, y)$. Each node then is preloaded with the polynomial shares corresponding to its row and column polynomials. Both approaches improve the resilience.

2.1.4.4 Probabilistic and Hybrid Schemes with Deployment Knowledge

Deployment knowledge is *a priori* information about the expected locations of sensor nodes or the distribution of nodes' location. It can tell us which sensor nodes are more likely to become neighbors and which local area a sensor node is more likely to reside. With the help of deployment knowledge, the schemes with better performance can be designed, because sensor nodes do not need to establish pairwise keys with those far away from them.

Liu and Ning [52] proposed *closest pairwise key scheme* in which each node shares pairwise keys with its c closest neighbors whose expected locations are closest to the node. In the extension version of this scheme, each node A has a unique key K_A. For node A and its c closest neighbors $B_1, B_2, ..., B_c$, the pairwise keys are $f(K_{B_i}|ID_A)$, where f is a pseudo-random function. Node A stores all c pairwise keys, while node B_i only stores its key K_{B_i} and f. This scheme has good resilience and connectivity and it also reduces the memory requirement of sensors. But its performance will be degraded with the growth of location error and it is hard to estimate sensors' expected locations accurately.

In [23], Du et al. first proposed the concept of deployment knowledge. They described a *group-based deployment model* in which the target filed is divided into square grids and nodes are categorized into groups. Each group of nodes pick their keys from a corresponding sub key pool and they are supposed to be deployed into a fixed grid. Sub key pools should be carefully designed. Specifically, the sub key pool of one group overlaps with the pools of the group's two horizontally and vertically neighboring groups with ratio α and with the group's four diagonally neighboring groups with ratio β, where $0 < \beta < \alpha < 1$. Since neighboring key pools overlap, the nodes from neighboring groups have higher probability to establish pairwise keys than those nodes whose groups are far from each other. This scheme outperforms the basic one in term of resilience. However, if smarter adversaries selectively compromise nodes within the same group, the scheme cannot keep the same good performance in resilience. That is, it cannot deal with selective node captures very well. In [53], the authors proposed a general *group-based key pre-distribution framework*. In the framework, the nodes from the same group establish pairwise keys using some existing approaches, while every group has an exact node to form cross-groups which are used to bridge neighboring groups. Whenever two nodes from different groups want to establish a pairwise key, they always need to exploit the bridging nodes. Hence, one problem of the framework is that those bridging nodes may consume up their energy earlier than other nodes. Huang et al. [36] introduced a location-aware key management scheme which is similar to the framework proposed by Liu and Ning.

We have already discussed some schemes that use deterministic approaches to improve resilience of probabilistic schemes, and also introduced some schemes that exploit deployment knowledge to improve the performance of schemes. Naturally, both deterministic approaches and deployment knowledge can be combined together to facilitate the design of key management schemes with better performance. Liu and Ning [52] integrated Blundo's polynomial-based technique with group-based deployment model and designed *location-based key pre-distribution using bivariate polynomials*. First, a bivariate polynomial is assigned to each grid of the target field. Then, for every node to be deployed into some grid, it is preloaded with the polynomial shares of polynomials from its own grid and four directly neighboring grids (i.e., two horizontally and two vertically neighboring grids). This scheme can easily achieve high connectivity with good resilience.

Similarly, Yu and Guan [88] depicted an approach to combine Blom's scheme with group-based deployment model. There are two significant differences between Yu's approach and other schemes. First, the authors divided the target field into hexagon grids, instead of square grids. Since hexagon grids are symmetric and have less neighboring grids, this scheme can achieve high connectivity with more efficient use of memory. Second, the authors proposed to use *geometric random graph* [61], instead of *(Bernoulli) random graph*, to model wireless sensor networks. The reason is that geometric random graph takes into consideration of the limited communication range of sensor nodes, but (Bernoulli) random graph does not. In addition, Zhou et al. [94, 95] discussed how to incorporate Blundo's polynomial-based technique and group-based deployment model with hexagon and triangle grids.

2.1.5 Group Key Management Schemes

Based on established pairwise keys, establishing a group key becomes straightforward. As depicted by Zhu et al. in [96], one node can directly send a group key to its neighbors through the links secured with pairwise keys.

Another approach is to exploit the pre-distributed polynomial shares of sensor nodes to generate a common group key. Blundo [9] proposed two models. The first model is non-interactive, where users compute a common key without any interaction. A random symmetric polynomial $f(x_1, \cdots, x_t)$ with t variables of degree λ is selected initially, where the coefficients come from a finite field $GF(q)$. Each user i receives share $f(i, x_2, \cdots, x_t)$. Users $j_1, \cdots, j_t$ can generate the conference key $K_{j_1,\cdots,j_t}$ by evaluating their polynomial shares. That is, each user j_i can obtain the conference key $K_{j_1,\cdots,j_t}$ independently by evaluating $f(j_i, j_1, \cdots, j_{i-1}, j_{i+1}, \cdots, j_t)$. In the second model, interactions are allowed in key computation. A symmetric polynomial $f(x, y)$ of degree $(\lambda + t - 2)$ is selected initially. Each user i receives share $f(i, y)$. Users $j_1, \cdots, j_t$ can establish the conference key K as follows: (1) the user with the largest identity, that is, user j_t, selects a random key K, (2) j_t calculates $K_{j_t,j_l} = f(j_t, j_l)$ for each $l = 1, \cdots, t-1$, (3) j_t sends $x_l = K_{j_t,j_l} \oplus K$ to each j_l, and (4) each j_l generates $K_{j_l,j_t} = f_{j_l}(j_t) = K_{j_t,j_l}$, and derives the secret $K = x_l \oplus K_{j_l,j_t}$. User j_t performs $(t-1)$ polynomial evaluations, and sends $(t-1)$ messages which carry a single x value to establish the group key.

2.1.6 Global Key Management Scheme

Perrig et al. [63] proposed μTESLA for authenticated broadcast and global key update. It requires the base station and sensor nodes to be loosely time synchronized. First, base station picks the last key K_n of a chain and generates the rest keys $K_0, K_1, \cdots, K_{n-1}$ of the chain using a one-way hash function H such that $K_i = K_{i+1}$. Given K_i, each node can generate the sequence $K_0, K_1, \cdots, K_{i-1}$, but not $K_{i+1}, \cdots, K_n$. At i-th time slot, base station broadcasts message M along with its message authentication code $MAC_{K_i}(M)$. Sensors need to store this message until base station discloses the authentication key K_i at $(i+1)$-th time slot. This is called delayed disclosure. Receiving $(i+1)$-th message, sensor nodes can verify the disclosed authentication key K_i by using the previous disclosed key K_{i-1} as $K_{i-1} = K_i$. This scheme requires sensor nodes to store a message until the authentication key disclosed. This brings communication delay, causes storage problem and may be exploited by the adversaries to launch DoS attacks. For example, the adversaries may jam key disclosure messages to saturate sensor nodes' storage.

2.2 Filtering False Data Injection and DoS Attacks in Wireless Sensor Networks

2.2.1 Introduction

Sensor nodes are not tamper-resistant, hence, the adversaries can easily compromise some nodes and launch various attacks through the compromised nodes. They can inject false data into the networks. These attacks not only cause false alarms in the base station, but also consume up the limited energy of forwarding nodes. Moreover, the adversaries can launch DoS attacks by selectively dropping some forwarding data, or intentionally contaminating the authentication information of data to make it dropped by other benign nodes if some authentication mechanisms are enforced.

False data injection is a violation of data authenticity and the DoS attacks cause data communication unavailable in wireless sensor networks. So, it is important to filter false data injection and DoS attacks in wireless sensor networks. The desired solutions should drop the false data as soon as possible, while still being efficient for wireless sensor networks in terms of communication overhead, memory cost and energy consumption.

In this work, we investigate existing solutions for filtering false data injection in wireless sensor networks and provide a survey of these solutions in the rest of the section. Most of these solutions do not consider DoS attacks. We also note that some of solutions assume a fixed path between some sensor nodes and the base station. We believe that this assumption is not appropriate for wireless sensor networks, because the topology of sensor networks may be changed frequently due to node failures or node changing their mode between active and sleeping ones.

2.2.2 Survey of Existing Solutions

Ye et al. proposed a statistical en-route filtering (SEF) scheme [86] based on probabilistic key distribution. In SEF, a global key pool is divided into n partitions, each containing m keys. Every node randomly picks k keys from one partition. When some event occurs, each sensing node (that detects this event) attaches to each report a MAC that is produced using one random key. The cluster head guarantees that a legitimate report contain T MACs generated using the keys from different partitions. Given that no more than $T-1$ nodes are compromised in the network, each node can detect a false report with probability proportional to $\frac{1}{n}$. The filtering capacity of SEF is independent of the network topology, but is constrained by the value of n. To increase the filtering capacity, we need a smaller value of n, which however makes it possible to break all partitions. In addition, since the keys are shared by multiple nodes, the compromised nodes can report forged or non-existent events that "occurs" within any clusters.

Zhu et al. designed an interleaved hop-by-hop authentication (IHA) scheme [98]. In this scheme, the base station periodically initiates an association process to make each node establish pairwise keys with others that are $t+1$ hops away, where t is a security threshold. In IHA, each sensing node generates a MAC using its multi-hop pairwise key, and a legitimate report should contain $t + 1$ distinct MACs. Since each multi-hop pairwise key is distinct, IHA can tolerate up to t compromised nodes in each cluster, instead of in the whole network as SEF. However, IHA requires the existence of a fixed path for transmitting control messages between the base station and every cluster head, which cannot be guaranteed when the network adopts some routing protocols such as GPSR [43] and GEAR [87]. In addition, the high communication overhead incurred by the association process makes IHA unsuitable for the sensor network whose topology is highly dynamic.

Yang et al. presented a commutative cipher based en-route filtering (CCEF) scheme [86]. In CCEF, each node is preloaded with a distinct authentication key. When a report is needed, the base station sends a session key to the cluster head and a witness key to every forwarding node along the path from itself to the cluster head. The report is appended with multiple MACs generated by sensing nodes and one generated by the cluster head using the session key. When the report is delivered to the base station along the same path, each forwarding node can verify the cluster head's MAC using the witness key. The MACs generated by sensing nodes are only verified by the base station. CCEE has several drawbacks. First, it has the same requirement for fixed paths as IHA. Second, it relies on expensive public-key algorithms to implement commutative ciphers. Third, it can only filter the false reports generated by a malicious node without the session key, excluding those generated by compromised cluster head or other sensing nodes.

Yang et al. further proposed a location based resilient security (LBRS) solution [85]. In LBRS, a sensor field is divided into square cells and each cell is associated with some cell keys that are determined from the cell's location. Each node stores two types of cell keys. One type contains the keys bounded to its sensing cells for authenticating its sensing reports. The other type contains the keys of some randomly chosen remote cells, which are very likely to forward their reports through the residing cell of the node. The authors define several types of report disruption attacks, in which the adversaries intentionally attach invalid MACs to reports to make them dropped by others. However, no concrete solutions are given in detail. In addition, LBRS suffers a severe drawback: It requires a short secure time slot within which all nodes can safely determine their locations and generate location-based keys. However, to the best of our knowledge, most practical sensor localization approaches [12, 33, 58] cannot be finished in such a short time slot, and the localization process itself is vulnerable to various attacks [14, 48, 49].

Recently, Ren et al. proposed a location-aware end-to-end data security (LEDS) scheme [66], which addresses both false report injection and some DoS attacks. Like LBRS, LEDS assumes that sensor nodes can generate location-based keys bounded to cells within a secure short time slot. LEDS provides end-to-end security by making sensing nodes encrypt their messages using cell key. A legitimate report contains T distinct shares produced from the encrypted message using nodes' secret keys, where the base station can always recover the original message from any t ($t < T$) valid shares. In LEDS, reports are forwarded through cells along report-auth route. Each node stores the authentication keys shared between its cell and others in its downstream report-auth area and on report-auth route. Each report contain $T + 1$ MACs generated by sensing nodes using authentication keys shared with the cells on report-auth route, and each forwarding node updates the MACs in the reports using its own authentication key. This process is similar to that in IHA. LEDS mitigates the impact of report disruption attacks by allowing t invalid MACs in reports, which invites adversaries to send false reports with less than t valid shares. In addition, LEDS addresses selective forwarding attacks by letting the whole cell of nodes to forward reports, which incurs high communication overhead.

2.3 Secure Network Coding

2.3.1 Introduction

2.3.1.1 Network Coding

Network coding [1, 50] is a new message forwarding technique that allows a forwarder to encode multiple input messages together to form an output one. Unlike the traditional approach that always duplicates every forwarding message, network coding is able to maximize the throughput of multicast networks. In 2003, Li et al. [50] further proved that linear network coding is sufficient to achieve the optimal throughput, which is the minimum of the max-flows from the single source to sinks. Because of this nice property, network coding has been widely used not only in wired networks [20, 29], but also in wireless networks [7, 16, 45, 46, 62, 67].

Figure 2.1 illustrates an example multicast network with linear network coding. In this network, source s can simultaneously send messages M_1 and M_2 to both sinks t_1 and t_2 through forwarders 1 to 4, as long as linear network coding is allowed at node 3. In this

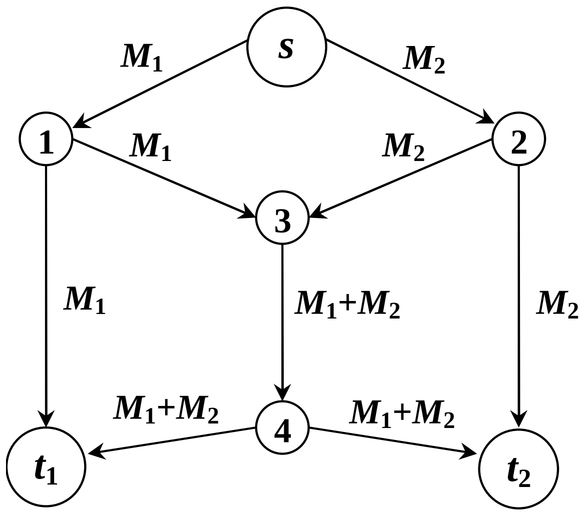


Figure 2.1 An example of linear network coding

example, the output message of node 3 is encoded as $M_1 + M_2$. In fact, other coding functions such as $aM_1 + bM_2$ and $M_1 \oplus M_2$ are also feasible, where a and b are two integer coefficients and $\oplus$ denotes exclusive OR (XOR). When a sink receives sufficient number of encoded messages, it can recover source messages by decoding, which is just solving a number of linear equations. Without network coding, node 3 has to transmit M_1 and M_2 separately, which not only increases the number of forwarding messages, but also lowers throughput, that is, two sinks cannot receive (and recover) both source messages simultaneously. We define encoding vector as a sequence of coefficients used to generate an encoded message. For example, the encoding vector of encoded message $M_1 + M_2$ is $(1, 1)$, while that of $aM_1 + bM_2$ is (a, b).

2.3.1.2 Secure Network Coding

The research on ”secure network coding” studies how to make network coding systems secure in the presence of various malicious attacks that are generally classified into passive ones and active ones. Precisely speaking, current research of secure network coding focuses on wiretapping attacks (out of passive ones) and pollution attacks (out of active ones).

In wiretapping attacks, the adversaries are assumed to be able to wiretap or eavesdrop on a subset of all the links of some network coding system and gain access to the information transmitted through the links. The problem is how to prevent the source information from leaking to the adversaries without using cryptographic mechanisms such as encryption. The basic idea to solve the problem is to introduce some random information into source messages and make the transmitted messages through the links ”randomized”. Several solutions based on this idea have been proposed and they mainly focus on how to transfer an existing feasible network coding scheme into a secure one and under what the condition that such a secure scheme exists.

We categorize these solutions into two classes, Shannon-secure ones and weakly-secure ones. Simply speaking, the difference between these two classes is that Shannon-secure does not allow the leakage of any information about the source, while weakly-secure disallows the leakage of any *meaningful* information. For instance, given two source messages M_1 and M_2, weakly-secure allows the adversaries to gain $M_1 + M_2$, because this information is not meaningful. However, Shannon-secure does not permit such leakage.

In pollution attacks, the adversaries may compromise some forwarders and modify (or pollute) their output messages. Pollution attacks not only prevent the sinks from recovering the source messages correctly, but also consume up the limited energy of the forwarders, which is especially harmful to resource-constrained wireless networks. Filtering pollution attacks is challenging in network coding systems, because traditional hashing or signature mechanisms no longer work. With traditional mechanisms, the source generates the hashes or signatures for all messages, which can be utilized by others to verify these messages. However, in network coding systems, the encoded messages are generated by the forwarders themselves and the source does not know how to produce the hashes or signatures for these encoded messages.

We classify the existing solutions for securing network coding against pollution attacks into two categories depending on whether the pollution attacks (or polluted messages) are filtered by the forwarders or only the sinks. Basically, the solutions of the first category utilize some homomorphic function and allow the forwarders to generate and verify the hashes or signatures of encoded messages without contacting the source. Similarly, the solutions of the second category create some error correction information for the source messages. This information is either appended to the source messages or sent to the sinks in advance, and will be used by the sinks to detect or filter polluted messages. Compared with those of the second category, the solutions of the first category save energy of the forwarders by filtering polluted messages as early as possible, however, they are typically much slower due to the heavy public-key operations for generating and verifying the hashes or signatures

2.3.2 Solutions to Wiretapping Attacks

2.3.2.1 Models and Goal

A network coding system can be represented by a tuple $(\mathcal{G}, \alpha, \mathcal{U})$. In the tuple, $\mathcal{G} = (V, E)$ is a directed acyclic graph, and V and E are the set of nodes and edges of $\mathcal{G}$. The capacity of each edge $e \in E$ denotes the maximum average rate of information that can be transmitted on this edge. α is the single source that generates and sends out a message vector $X = (x_1, x_2, \cdots, x_n)^T \in F_q^n$ every time unit. $\mathcal{U}$ is the set of users (can be more than one) that receive and recover the multicast information. In linear coding systems, the message (symbol) on any outgoing edge of a node is a linear combination of the messages (symbols) on its incoming edges, thus, the message (symbol) on any edge is eventually a linear combination of the source messages (symbols). We denote the message (symbol) transmitted on edge e by $\mathcal{T}_e X$, where $\mathcal{T}_e$, named the *global encoding vector* on edge e, is a row vector over F_q. (Note: When we discuss the solutions to wiretapping attacks, we may use *message* and *symbol* interchangeablly.)

In wiretapping attacks, adversaries can wiretap or eavesdrop on any of a collection of sets of edges, where the collection is denoted as $\mathcal{A} = \{A_1, A_2, \cdots, A_{|\mathcal{A}|}\}$ and $|\mathcal{A}|$ denotes the number of sets in the collection. Each A_i is a subset of all edges in the network. It is assumed that an eavesdropper can access any member but no more than one member of $\mathcal{A}$. Let M_i denote an eavesdropping matrix of dimension $k_i \times n$ for each A_i, where k_i is the maximum number of linearly independent encoding vectors of the edges in A_i. All the eavesdropping matrixes corresponding to $\mathcal{A} = \{A_1, \cdots, A_{|\mathcal{A}|}\}$ are $\mathcal{M} = \{M_1, \cdots, M_{|\mathcal{A}|}\}$. Hence, the information available to the eavesdropper is a set of linear equations $M_i X = B_i$ for any $M_i \in \mathcal{M}$. Some other model also assumes that the eavesdropper has access to at most k edges of the network. This is actually a special case of the first model in which the collection $\mathcal{A}$ contains all non-empty subsets of at most k edges of all the edges in the graph.

Suppose a feasible network coding solution already exists that enables the source to multicast the message vector $X = (x_1, x_2, \cdots x_n)^T$ to all receivers. The goal is to transform this insecure coding solution into a secure one that allows no information leakage to the eavesdropper. The solutions can be Shannon-secure and weakly-secure [6] depending on the extend to which the system can tolerate information leakage. Shannon-secure requires that for all $i \in \{1, \cdots, |\mathcal{A}|\}$, the eavesdropper gains no information about the source. However, weakly-secure requires that the eavesdropper receives no *meaningful* information about the source. For example, if the eavesdropper knows the sum of $x_1 + x_2$, where x_1 and x_2 are i.i.d. information symbols generated at the source, then she gets some information about the combinations of the source symbols, but no *meaningful* information about the value of either x_1 or x_2 alone. The system is weakly-secure but not Shannon-secure.

2.3.2.2 Shannon-secure Network Coding Schemes

Cai & Yeung [13] studied the problem of how to make a linear network coding system to transmit information "securely" in the presence of a wiretapper(or eavesdropper) who can eavesdrop on a bounded number of network links. They gave the definition of secure (i.e., Shannon-secure) network code, proposed a method to transform a given linear network code into a secure one, and presented the sufficient condition that guarantees the existence of such a secure transformation.

The basic idea of Cai & Yeung's method is to insert some random symbols into the message vector sent by the source, such that the symbols transmitted on all edges are "randomized", i.e., are the combination of the information symbols and the random symbols. More specifically, the input vector at the source is divided into two portions, the first $r = n - k$ symbols are information symbols and the remaining $k = max\{k_i\}$ symbols are random symbols chosen uniformly from F_q, i.e., $X = (S, W) = ((s_1, \cdots, s_r)^T, (w_1, \cdots, w_k)^T)$. When properly designed, a linear network code is "secure", i.e., the eavesdropper cannot eliminate the randomness or learn any combinations of solely the information symbols.

Definition of security: The secure network code defined by Cai & Yeung is Shannon-secure, that is,

$$H(S|M_iX = B_i) = H(S),\ \forall M_i \in \mathcal{M}\ . \tag{2.1}$$

This definition is equivalent to that the eavesdropper cannot learn any linear combinations of the first r information symbols, namely, $\forall(t_1, t_2, \cdots, t_{k_i}) \neq 0$ and $\forall(\beta_1, \beta_2, \cdots, \beta_r) \neq 0$,

$$(t_1, t_2, \cdots, t_{k_i})M_iX \neq (\beta_1, \beta_2, \cdots, \beta_r, 0 \cdots 0)X\ . \tag{2.2}$$

Cai and Yeung also proved that a network code is secure, if $\forall B_i \in F_q^{k_i}$ and $\forall S \in F_q^r$, there exists

$$|C(B_i)\overset{*}{\bigcap}(S)| = q^{k-k_i}\ , \tag{2.3}$$

where $C(B_i) = \{X \in F_q^n : M_iX = B_i\}, \forall b \in F_q^n$ and $C^*(S) = \{X \in F_q^n : X = (S, W)\}, \forall S \in F_q^r$.

Transformation matrix: Let C denote an $n \times n$ *secure transformation matrix*. Cai & Yeung's method is to apply this matrix to an insecure network code and transform it into a secure one. More specifically, for any edge $e \in E$ of encoding vector $\mathcal{T}_e$, the transformed encoding vector is $\mathcal{T}'_e = \mathcal{T}_eC$. Thus, the symbol transmitted on edge e becomes $\mathcal{T}'_eX = \mathcal{T}_eCX$. The properties that matrix C should satisfy such that the transformed network code is both feasible and secure, are given as follows:

Theorem 1 *The matrix C is full-rank* i.f.f. *the transformed network code is feasible, i.e., all receivers can recover the information symbols $S = (x_1, \cdots, x_r)^T$.*

Theorem 2 *The first r row vectors of matrix C^{-1} and all row vectors of matrix M_i are linearly independent* i.f.f. *the transformed network code is secure.*

Lower bound: To guarantee the existence of a secure transformation matrix C, the larger field size is required. Cai & Yeung proved that a lower bound of $q > |\mathcal{A}|$ is *sufficient* (but not *necessary*).

Feldman et al. [26] generalized and simplified Cai & Yeung's method, and showed that making a linear network code secure is equivalent to finding a code with certain generalized distance properties. They also presented a *necessary* (but not *sufficient*) condition that guarantees the existence of a secure transformation.

Definition of security: Feldman et al. defined that a secure network code should satisfy that $\forall B_i \in F_q^{k_i}$ and $\forall S, S' \in F_q^r$,

$$|R(S, B_i)| = |R(S', B_i)| \ , \tag{2.4}$$

where $R(S, B_i) = \{W \in F_q^k : M_i(S, W) = B_i\}, \forall S \in F_q^r$. $R(S, B_i)$ is the set of all possible random vectors W, such that when message vector $X = (S, W)$ is sent, the observed information on the linearly independent edges in A_i is B_i. This definition shows that when W is chosen randomly, whatever information B_i that an eavesdropper observes gives no information about the transmitted S.

Transformation matrix: Unlike Cai & Yeung, Feldman applied the transformation matrix C to the message vector sent by the source, while keeping the code unchanged. Thus, the message vector sent by the source is $X' = CX$. For any edge $e \in E$ of encoding vector $\mathcal{T}_e$, the symbol transmitted on edge e becomes $\mathcal{T}_e X' = \mathcal{T}_e CX$. This transformation is equivalent in power to that of Cai & Yeung, but it is simpler because it does not need to change the code.

Feldman et al. also proved that finding a secure transformation matrix is equivalent to solving some generalized coding problem. Let N be the number of all edges in the network. Let Z_G be an $n \times N$ matrix whose columns are the encoding vectors of all edges in the graph G. Since Z_G has rank n (i.e., the message vector can be decoded by the receivers), thus the null space of Z_G has rank $N - n$, which can be generated by an $(N - n) \times N$ matrix Z'_G.

Theorem 3 *Given an $(N-n) \times N$ matrix Z'_G, a $r \times N$ matrix F that satisfies the Hamming distance $\delta(Z'_G, F) > n-r$ exists* i.f.f. *an $n \times n$ secure transformation matrix C exists.*

Based on this theorem, the existence of matrix C is equivalent to the existence of matrix F, while finding the latter one is a generalized coding problem, i.e., Span Distance Problem. When solving the Span Distance Problem, the following result can be derived:

Theorem 4 *If $n = \frac{\log N}{\log q} - \frac{\log \mathrm{Vol}_q(d,N)}{\log q} + 2\log N + \log q + \log \ln q$ and $d \leq n-r$, then there is an $(N-n) \times N$ matrix Z'_G, such that for any $r \times N$ matrix F, the Hamming distance $\delta(Z'_G, F) > n-r$ cannot be satisfied.*

$\mathrm{Vol}_q(d, N)$ denotes the number of vectors in a *ball of radius* d around x, where given $x \in F_q^N$, the ball is the set of all vectors in F_q^N which differ from x in at most d coordinates. In this theorem, let $q = N^{\Omega(\sqrt{\frac{n-r}{logN}})}$. It can be seen that as long as r takes any value within the range $r < (N^{\Omega(\sqrt{\frac{n-r}{\log N}})} - 3)\log N$, the inequality $d \leq n-r$ is held.

Lower bound: Based on Theorem 3 and Theorem 4, the necessary lower bound of field size, i.e., $q > N^{\Omega(\sqrt{\frac{n-r}{logN}})}$, can be obtained for the existence of a secure transformation matrix C.

2.3.2.3 Weakly-secure Network Coding Schemes

Bhattad & Narayanan [6] observed that the security requirement [13] can be relaxed in practice. They believe that it is suffice if no *meaningful* information is leaked to the eavesdropper. For example, a network code is secure if the eavesdropper only get $x_1 + x_2$, but is unaware of x_1 and x_2, where x_1 and x_2 are two information symbols sent by the source. They defined a weakly-secure model, proposed a secure transformation to convert an insecure code into a secure one, and proved the sufficient condition for the existence of a secure transformation.

Definition of security: They defined that a weakly-secure network code should satisfy,

$$H(x_j | M_i X = B_i) = H(x_j) \ \forall x_j \in X \ . \tag{2.5}$$

Transformation matrix: Unlike the Shannon-secure code, a weakly-secure code does not require any random symbols inserted into the message vector. A secure transformation matrix C can be applied either to the message vector sent from the source or to the encoding vectors of all edges. After transformation, the message available to the eavesdropper is $M_i'X = M_iCX$ for all $M_i \in \mathcal{M}$.

Theorem 5 *The matrix C is full-rank* iff *the transformed coding solution is feasible; any row vector of the matrix C^{-1} and all row vectors of matrix M_i are linearly independent* i.f.f. *the transformed encoding solutions is weak-secure.*

Lower bound: Bhattad & Narayanan proved that if $q^n > |\mathcal{A}|q^k + q^{n-1}$, then the secure transformation matrix C must exist.

2.3.3 Solutions to Pollution Attacks

2.3.3.1 Filtering Pollution Attacks by Forwarders

Krohn et al. [47] proposed using a homomorphic hash function to verify the check blocks of a downloaded file in peer-to-peer systems, where the check blocks are linear combinations of original file blocks. Gkantsidis and Rodriguez [30] extended Krohn's approach and presented a homomorphic hashing scheme (called *GR's scheme* for short) for securing peer-to-peer file distribution systems with network coding against pollution attacks. Assuming multiple users want to download a file that is divided into n blocks $b_1, b_2, \cdots, b_n$. The source (and the system) transmits these blocks with linear network coding, that is, each forwarder transmits some encoded block $e = \sum_{i=1}^{n} c_i b_i \bmod q$, where $(c_1, c_2, \cdots, c_n)$ denotes the encoding vector and q is a prime. With a homomorphic hash function, the hash of this encoded block can be represented as $h(e) = \prod_{i=1}^{n} h^{c_i}(b_i) \bmod p$, where p is another prime. Hence, if a downstream node obtains the source blocks' hashes in advance, it is able to verify the encoded block e. However, GR's scheme has a severe drawback, i.e., it needs some extra secure channels for the source to transmit all of its hashes to the forwarders and sinks before sending the source blocks. Unfortunately, such secure channels do not exist in most networks. Otherwise, if we can find such a secure channel, we can directly send all the source blocks to the sinks and easily solve the pollution attack problem. So, the requirement of extra secure channels makes GR's scheme inapplicable or impractical in most cases. In addition, GR's scheme is based on heavy modular exponentiations and hence inefficient for wireless sensor networks.

Charles, Jain and Lauter [19] designed a new homomorphic signature scheme (called *CJL's scheme* for short) based on Weil pairing [54, 56] over elliptic curves. CJL's scheme utilizes a "linear" signature function based on some torsion points over elliptic curves, while the signature of an encoded message covers the contents of the message and the corresponding encoding vector. This allows forwarders to calculate the signatures of their encoded messages without contacting the source. Hence, CJL's scheme does not need any secure channels and can even provide source authentication. The main disadvantage of CJL's scheme is that its underlying pairing operations are extremely time-consuming. So, it is too slow to be used in wireless sensor networks.

Zhao et al. [93] studied the content distribution applications adopting network coding and proposed a signature scheme (called *Zhao's scheme*) that allows the forwarders to filter pollution attacks. They divided a source file into multiple vectors that span a subspace. In their scheme, the source calculates a signature of the spanned subspace, then broadcasts it to all the forwarders for them to verify if a received encoded vector is in that subspace or not. To verify one vector, a forwarder should calculate $m + n$ modular exponentiations, which is the same as GR's scheme, where m is the length of each vector and n is the total number of source vectors. The authors claimed that their approach does not need extra secure channels. However, the public keys and the signature used in Zhao's scheme are both related to the downloaded file. In the case that a lot of files should be downloaded continuously from the source, this scheme still requires secure channels to update the public keys or signature to all forwarders.

2.3.3.2 Filtering Pollution Attacks by Sinks

Ho et al. [35] studied *Byzantine modification attacks* in multicast networks and illustrated how randomized network coding can be utilized to detect these attacks without the use of cryptographic functions. In Ho's scheme, the source attaches each packet with a hash calculated from a polynomial hash function. If Byzantine modification attacks (i.e., pollution attacks) exist, a sink can detect inconsistency between the packets and corresponding hashes with a high probability, as long as the sink receives some unmodified packet whose content is unknown to the adversaries. The detection probability can be traded off against communication overhead and the number of unmodified packets. Ho's scheme is computationally efficient for the use of a simple polynomial hash function. However, it only allows the sinks, instead of the forwarders, to detect modification attacks. Hence, it cannot reduce the amount of energy consumed by the forwarders for transmitting the useless polluted packets. In addition, it can only detect (but not filter) polluted packets. So, it cannot help the sinks recover the source packets correctly, given that some polluted packets are detected.

Jaggi et al. [39] discussed how to build resilient network coding in the presence of *Byzantine adversaries*. Their idea is to append the source messages with extra parity information that can be used by the sinks to correctly recover the source messages even suffering *Byzantine attacks*. The tradeoff is the sacrifice of data transmission rate. They analyzed the optimal rate that network coding can achieve under different threat models and proposed some polynomial time algorithms to attain these optimal rates. Suppose the network capacity is C. When the adversaries can eavesdrop on all links and jam z_o links, their algorithm can achieve a rate of $C - 2z_o$. However, when the adversaries have limited snooping capabilities, their algorithm can achieve a higher rate of $C - z_o$. Similar to Ho's scheme, Jaggi's algorithms filter pollution attacks only by the sinks, so they cannot reduce the energy consumption of the forwarders for forwarding polluted information and are not efficient for wireless sensor networks.

CHAPTER 3 ENHANCING CONFIDENTIALITY: Providing Key Management for Wireless Sensor Networks

3.1 Introduction

Wireless sensor networks may consist of a large number of battery-powered sensor nodes, which are equipped with short-range radio, and only have constrained computation capability as well as limited memory space. These sensor networks pose security and privacy challenges when deployed in a hostile environment. For example, an adversary can easily gain access to mission critical or private information by eavesdropping on wireless communications among sensor nodes. Therefore, it is important to encrypt the wireless communication. However, as Chan et al. stated in [17], the challenge is how to bootstrap secure communications among sensor nodes, that is, how to set up secret keys among sensor nodes to allow them to establish secure links between each other.

Some general key distribution and management approaches are not suitable for wireless sensor networks. Firstly, trivially storing in each node a pairwise key for every other node poses a high memory requirement unaffordable for sensor nodes. Secondly, online key distribution and management offered by the base station is inefficient for wireless sensor networks due to high communication overhead. Thirdly, public-key algorithms such as RSA, Diffie-Hellman and Elliptic Curve Cryptography (ECC) are too expensive to current sensor nodes for high energy consumption and computation overhead. Experiment results of existing research [32, 76] show that the execution time of public-key based operations such as encryption and decryption is of the order of seconds or even ten seconds. Moreover, wireless sensor networks may not be able to provide the desired Public Key Infrastructure (PKI) for key distribution. We have to either distribute public keys into nodes through the base station online, which may cause high communication overhead, or pre-distribute public keys into nodes offline, which may need some scheme like what we propose in this work to improve its efficiency.

Fortunately, the bootstrapping problem can be solved by key pre-distribution schemes that pre-distribute secret information in nodes to help them establish secure links after deployment. Eschenauer and Gligor [25] proposed *basic scheme* by utilizing probabilistic key pre-distribution, which was improved by Chan et al. [17] and Du et al. [22]. Recently, Du et al. [23] and Liu and Ning [52, 53] independently proposed to make use of *deployment knowledge* for further improvement of the performance of key establishment. Different from all these schemes, *LEAP* [96] proposed by Zhu et al. assumes a weaker model, that is, there exists a short time interval within nodes can establish pairwise keys safely after deployment.

We propose a novel key management scheme by using deployment knowledge. In our scheme, a target field is divided into hexagon grids and sensor nodes are divided into the same number of groups as that of grids, where each group is deployed into a unique grid. Benefited from deployment knowledge, we can drastically reduce the number of potential groups from which a node's neighbors may come. Built on top of Blom's scheme [8], our scheme distributes secret information among nodes for them to generate pairwise keys. We first force each group of nodes to share the same secret matrix, hence, each pair of nodes from the same group are guaranteed to establish a pairwise key. Then, we assign some extra secret matrices to help the nodes from neighbor groups establish pairwise keys. By carefully arranging secret matrices for sensor groups, we achieve a probability approaching one for almost all the nodes to establish secure links with their neighbors, while the probability offered by other schemes is much less than one.

We study connectivity of sensor networks utilizing *geometric random graph* model [31, 60, 68] and derive the transmission range for achieving the desired connectivity based on sensor distribution. Compared with existing schemes, our scheme requires a *shorter transmission range* and achieves a *higher connectivity* even with a *lower memory requirement.* In addition, it has an interesting property: When the number of compromised nodes of a group is less than a threshold value, wireless communication between all the other nodes belonging to the same group is still secure. Simulation results also show that our scheme outperforms others in terms of resilience against node capture.

The rest of the chapter is organized as follows: In section 3.2, we discuss deployment model and define the key pre-distribution problem for wireless sensor networks. Then, we present our scheme in section 3.3. In section 3.4, we analyze connectivity of wireless sensor networks and study how to determine the transmission range for achieving the desired connectivity. In section 3.5, we evaluate security performance in term of resilience against node capture, while in section 3.6, we compare our scheme with others by simulation. Finally, we conclude in section 3.7.

3.2 Problem Statement

3.2.1 Deployment Model

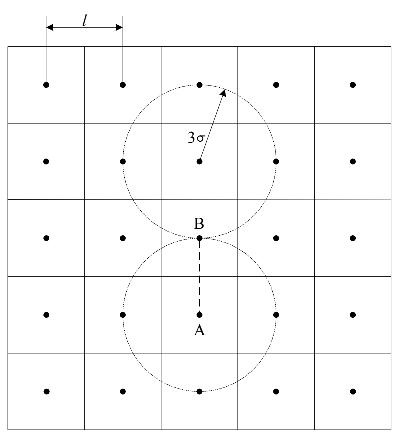
In the work, we assume that sensor nodes are stationary after deployment. The distribution of nodes can be determined from deployment model which shows how sensor nodes are deployed. A *general deployment model* states that N nodes are deployed into an arbitrary target field $\mathcal{S}_f$ and the location of each node i $(i = 1, \ldots, N)$ follows some distribution of probability density function (pdf) $f_i(x, y)$, where $(x, y) \in \mathcal{S}_f$ are the node's coordinates.

Except for deploying all nodes at once, it is also possible to deploy sensor nodes in groups, which leads to the following *group-based deployment model*:

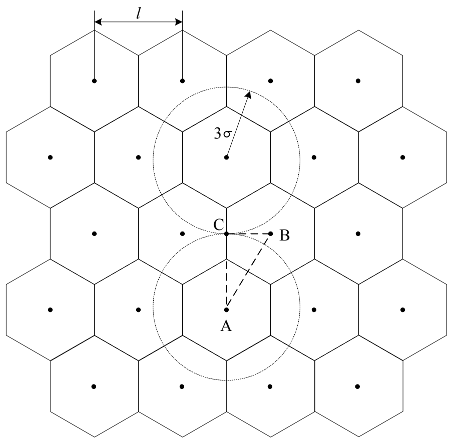
- An arbitrary target field $\mathcal{S}_f$ is divided into (and covered by) t grids equally.
- N nodes are also divided into t groups equally (and hence each group contains $n = \frac{N}{t}$ nodes). Each group of nodes will be deployed into a unique grid such that group i will deployed into grid i $(i = 1, \ldots, t)$, where i is called group (and grid) index.
- The center of each grid is called deployment point, which is the desired location of all nodes of corresponding group. Because of randomness of deployment process, a group of nodes may spread into a local area around the deployment point to which the group of nodes should be deployed. Hence, we assume the real location of each group of nodes

follows some distribution $f_i(x,y) = f(x,y,\mu_x,\mu_y)$, where $(\mu_x,\mu_y) \in \mathcal{S}_f$ is the coordinates of deployment point for the group.

Figure 3.1 depicts how to partition a target field into *square* and *hexagon* grids.



(a) Partition of square grids



(b) Partition of hexagon grids

Figure 3.1 A target field is partitioned into square or hexagon grids. l is the distance between two neighbor grids. σ denotes the variance of normal distribution of sensor nodes. A and B are two deployment points. C is the tangent point of two circles of radius 3σ and each circle is centered at a deployment point.

(Note: In the rest of chapter, we use terms *grid* and *group* interchangeably since they correspond to each other.)

In this work, we consider two popular distributions of deployed nodes, that is, uniform distribution and normal distribution.

For example, we can divide a target field into square grids and drop each group of nodes randomly in their grid. Thus, we can obtain such a uniform distribution:

$$f_i(x,y) = \frac{1}{l^2}\ , \tag{3.1}$$

where l is the distance between two neighbor deployment points, and $x \in [\mu_{x_i} - \frac{l}{2}, \mu_{x_i} + \frac{l}{2}]$, $y \in [\mu_{y_i} - \frac{l}{2}, \mu_{y_i} + \frac{l}{2}]$.

Another example might be to drop nodes from a helicopter. Each time when the helicopter is hanging above some deployment point, a group of nodes will be dropped. Due to randomness, each group of nodes may spread into a small circle area around their deployment point. The closer to the deployment point, the more nodes reside in the location. So, we may acquire a normal distribution as follows:

$$f_i(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-[(x-\mu_{x_i})^2+(y-\mu_{y_i})^2]}{2\sigma^2}}, \tag{3.2}$$

where σ^2 is the variance of distribution. This variance may be affected by various factors such as the height of helicopter and the weather when nodes are deployed. It can be measured by experiment. Here, we simply assume that it is already known before sensor nodes are deployed.

3.2.2 Threat Model

When designing our key management scheme, we consider the following threats:

- The adversaries can eavesdrop on wireless communication in sensor networks irrespective of whether it is encrypted or not.
- The adversaries can physically capture and compromise some sensor nodes in order to obtain the secret keys (or secret information) stored in those nodes.
- Having obtained the secret keys from the compromised nodes, the adversaries can decrypt or compromise all of the links secured with those keys. The compromised links include not only those directly connected to the compromised nodes, but also the additional ones that are established by non-compromised nodes using the same compromised keys.

3.2.3 Bootstrapping Problem

In this work, we define a link as a one-hop and bidirectional connection between a pair of neighbor nodes, where a pair of neighbor nodes are any two nodes whose physical distance is no more than their transmission range (suppose all nodes have the same transmission range). The authors of [96] have defined various types of keys such as *individual key*, *group key*, and *cluster key*[1]. However, in this work we focus on how to establish *pairwise keys* for neighbor

[1]In [96], individual key is defined as a unique pairwise key between each node and the base station; group key is defined as a globally shared key used by the base station to encrypt broadcast messages for the whole sensor network; and cluster key is defined as a key shared by a node and all its neighbors.

nodes.

(Note: *Cluster* has different meanings in [96] and in our work. In [96], a cluster includes a node and its neighbors. In our work, it is defined as one grid (or group) and its neighbor grids (or groups).)

Our purpose is to enable neighbor nodes to share some common key(s) that can be used to secure their communication. More precisely, we consider such a bootstrapping problem that how to distribute secret key(s) among sensor nodes while achieving the following goals:

- *Highly connected sensor network.* When measuring how sensor networks are connected, we only consider the secure links. That is, we do not allow any two neighbor nodes to be connected if they cannot find any shared key. So, our key management scheme poses a higher requirement for connectivity. (Note: We define connectivity as the probability that a deployed sensor network is connected.)

- *Strong resilience against node capture.* Resilience against node capture is defined as the fraction of links that the adversaries could compromise given that a certain number of nodes are compromised. The lower the fraction, the stronger the resilience.

- *Low memory requirement.* Memory requirement is measured by the number of secret keys stored in each node.

- *Short transmission range.* We assume that sensor nodes can adjust their transmission range by choosing different power levels of radio and this adjustment is done before deployment. After deployment, all the nodes use the same transmission range that is no longer changed. Obviously, choosing a shorter transmission range can save more energy.

These goals may cause conflicts. For example, to make more nodes connected, we may either store more secret keys in sensor nodes or increase their transmission range, hence, the first goal contradicts the third or the fourth one. When designing our scheme, we have to make tradeoffs among these goals.

3.3 Our Scheme

3.3.1 Background: Blom's Key Management Scheme

We briefly introduce Blom's key management scheme [8] here, as our scheme is built on top of it. (Interested reader may refer to [22] for more detailed explanation.)

Blom's scheme guarantees that any two nodes out of a group of n ones can always establish a pairwise key. It employs two basic components, a $(\lambda+1) \times (\lambda+1)$ symmetric matrix D and a $(\lambda+1) \times n$ public matrix G. We call $(DG)^T$ secret matrix and denote it as A or B, where T means transpose. In this scheme, all pairwise keys are arranged in an $n \times n$ symmetric matrix K, where $K = AG$ (or BG) $= (DG)^T G = K^T$. Each node i stores the i-th row of secret matrix and the i-th column of public matrix. To establish the pairwise key, two nodes, e.g., i and j, first exchange their columns of public matrix, then, each one can individually derive the key, e.g., $k_{ij} = k_{ji}$, that is the dot product of its own row and the column received from the other. Since the secret matrix (or rows) is never transmitted, no adversaries can get the key by eavesdropping on the communication between these two nodes. Moreover, no additional links will be revealed given that some node is compromised, because all the rows (or all the keys) are different. However, if the number of compromised nodes is greater than λ, the whole secret matrix can be computed (or broken) by the adversaries. This property is called λ-secure, where λ is called the security threshold.

3.3.2 Overview

Based on the group-based deployment model, we derive that each group of nodes reside only within a small local area, which implies that most neighbors of each node come from its own group and neighbor groups. Therefore, to achieve a highly connected network, the key point is to maximize the probability with which the nodes from the same group and neighbor groups can find some shared keys. For this purpose, we divide the links of sensor networks into two types, *in-group links* and *inter-group links*, depending on whether the involved nodes are from the same group or not. Accordingly, for these two types of links, we build two types of secret matrices, A and B, respectively.

Our scheme consists of tow phases, *key pre-distribution phase* and *key discovery phase.*

3.3.2.1 Key pre-distribution phase

In this phase, we generate a global public matrix G and a number of secret matrices A and B. All the groups share the global matrix G, that is, every node of a group picks a corresponding column from G. Meanwhile, each group is assigned a unique secret matrix A, that is, every node of a group picks a corresponding row from the unique matrix A assigned to its group. This way, we guarantee that any two nodes from the same group can always find a pairwise key.

Then, we assign each group some number of B matrices and guarantee *each pair of neighbor groups share at least one common B matrix.* More precisely, we first select some groups and assign each of them a distinct secret matrix B. These selected groups are called *basic groups*, while others are called *non-basic groups* or *normal groups.* Then, for each group (including basic and normal groups), we assign it all the B matrices that have been assigned to its *neighbor basic groups*, which are the basic groups among its neighbor groups. After that, each node picks the corresponding rows from some or all (depending on the different methods that we will discuss late) of the B matrices assigned to its group. Finally, we set all nodes the same transmission range and deploy them group by group.

3.3.2.2 Key discovery phase

After deployment, each node first probes its neighbors. Then, neighbor nodes exchange their group indexes, indexes of B matrices and columns of matrix G. If two neighbor nodes come from the same group, they can derive the pairwise key from the common matrix A and G. If they are not from the same group, but share one or more common B matrices, they can also find out the pairwise key from a shared matrix B and the common matrix G. Last, the neighbors establishing pairwise keys build the secure link between each other and start to transmit data securely. Those neighbors without pairwise keys will no longer communicate with each other.

(Note: The nodes without pairwise keys may still exploit other methods such as multi-hop path reinforcement to establish pairwise keys indirectly. However, this discussion is out of the scope of this work and we focus only on how to establish the pairwise keys using one-hop links.)

3.3.3 Detailed Procedures

We have different ways to assign B matrices to groups and allow nodes to pick their rows, which leads to a series of variants of our scheme. Each variant is a method identified by two parameters, b and w, where b is the maximum number of B matrices assigned to a group and w is the maximum number of rows picked by a node. (Note: We use *maximum* here, because our scheme does not guarantee every group (or node) has the same number of B matrices (or rows).) We denote each method as $(b = ?, w = ?)$, where "?" is some integer value. The value of b is 2, 3 or 7 and w may take a value no greater than that of b. Hence, we have totally $2 + 3 + 7 = 12$ slightly different methods. (Late on, we will discuss these methods in detail.) For example, for method $(b = 2, w = 2)$, $b = 2$ means each group will be assigned *at most* two B matrices, and $w = 2$ means each node will store *at most* two rows with each picked from a distinct matrix B. More precisely, if a group is assigned two B matrices, every node of this group will store two rows with each from a B matrix. If the group has only one B matrix, each of its nodes will store one row picked from the only matrix B.

In our scheme, each node stores one column of matrix G, one row of matrix A and at most w rows of B matrices. Each row has $(\lambda + 1)$ elements. Du et al. demonstrated in [22] that if the public matrix G is a Vandermonde matrix, each column can be derived from a single seed integer and hence memory consumption for columns can be ignored. Given the memory size of M for each node, the value of threshold λ of our scheme can be determined as follows:

$$M = (\lambda + 1)(w + 1) \implies \lambda = \frac{M}{w + 1} - 1 \, . \tag{3.3}$$

In our scheme, not every node is able to pick w rows from B matrices. So, equation (3.3) gives us a worst-case value of λ or a lower bound on λ.

Now, we present the procedures of our scheme in detail.

3.3.3.1 Key pre-distribution phase

- We generate a public matrix G to be shared by all the groups and a unique secret matrix A_i for each group i, where $i = 1, \ldots, t$ and t is the total number of groups. Each node j of group i picks the j-th row of A_i, where node index $j = 1, \ldots, n$.

- The target field is divided into $t = t_1 \times t_2$ grids. Hence, the coordinates of group i can be represented by a pair of row and column indexes (r_i, c_i), where $r_i = 1, \ldots, t_1$ and $c_i = 1, \ldots, t_2$.

- Now, we select some groups as basic groups and assign each basic group a distinct B matrix. More precisely,

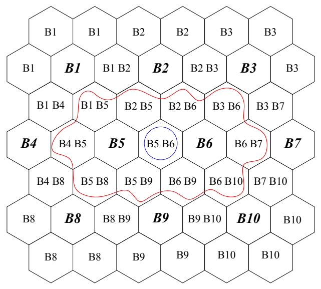
  - For the methods of $b = 2$, if the coordinates of group i satisfy "$r_i \bmod 2 = 0$ and $c_i \bmod 2 = 0$, but $r_i \bmod 4 \neq 0$" or "$r_i \bmod 4 = 0$ and $c_i \bmod 2 = 1$", this group is selected as a basic group and assigned a distinct matrix B, as shown in Figure 3.2(a), where the basic groups are labeled in Bold and italic font. We repeat this step until all the basic groups are found.

  - For the methods of $b = 3$, if the coordinates of group i satisfy "$r_i \bmod 2 = 1$ and $c_i \bmod 3 = 0$" or "$r_i \bmod 2 = 0$ and $c_i \bmod 3 = 2$", this group is selected as basic group and assigned a distinct matrix B, as shown in Figure 3.2(b). We repeat this step until all basic groups are found.

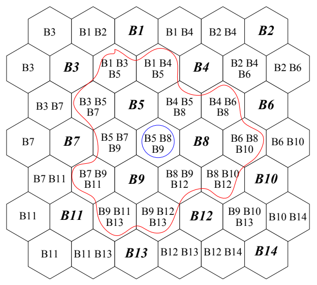
  - For the methods of $b = 7$, every group is a basic group. We assign each group i a distinct matrix B_i, as shown in Figure 3.2(c).

- Then, we assign B matrices to the normal groups for the methods of $b = 2$ or 3, and assign more B matrices to the basic groups for the methods of $b = 7$.

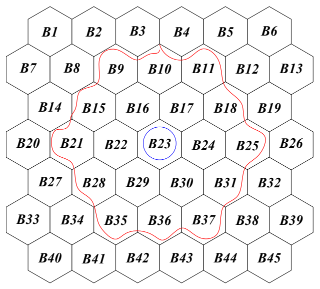
  - For the methods of $b = 2$ (or 3), we assign each normal group all the B matrices that are already assigned to its *neighbor basic groups*. Except for those at the edge of the target field, each normal group has two (or three) neighbor basic groups. Thus, these normal groups is eventually assigned two (or three) B matrices.


B1 B1 B2 B2 B3 B3
B1 B1 B1 B2 B2 B2 B3 B3 B3
B1 B4 B1 B5 B2 B5 B2 B6 B3 B6 B3 B7
B4 B4 B5 B5 B5 B6 B6 B6 B7 B7
B4 B8 B5 B8 B5 B9 B6 B9 B6 B10 B7 B10
B8 B8 B8 B9 B9 B9 B10 B10 B10
B8 B8 B9 B9 B10 B10

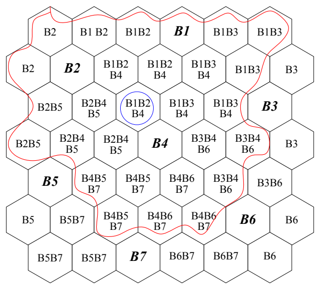
(a) Assignment of B matrices when $b = 2$. One cluster contains at most two basic groups.


B3 B1 B2 B1 B1 B4 B2 B4 B2
B3 B3 B1 B3 B5 B1 B4 B5 B4 B2 B4 B6 B2 B6
B3 B7 B3 B5 B7 B5 B4 B5 B8 B4 B6 B8 B6
B7 B7 B5 B7 B9 B5 B8 B9 B8 B6 B8 B10 B6 B10
B7 B11 B7 B9 B11 B9 B8 B9 B12 B8 B10 B12 B10
B11 B11 B9 B11 B13 B9 B12 B13 B12 B9 B10 B13 B10 B14
B11 B11 B13 B13 B12 B13 B12 B14 B14

(b) Assignment of B matrices when $b = 3$. One cluster contains at most three basic groups.



(c) Assignment of B matrices when $b = 7$. One cluster contains exact seven basic groups.


B2 B1 B2 B1B2 B1 B1B3 B1B3
B2 B2 B1B2 B4 B1B2 B4 B1B3 B4 B1B3 B3
B2B5 B2B4 B5 B1B2 B4 B1B3 B4 B1B3 B4 B3
B2B5 B2B4 B5 B2B4 B5 B4 B3B4 B6 B3B4 B6 B3
B5 B4B5 B7 B4B5 B7 B4B6 B7 B3B4 B6 B3B6
B5 B5B7 B4B5 B7 B4B6 B7 B4B6 B7 B6 B6
B5B7 B5B7 B7 B6B7 B6B7 B6

(d) Assignment of B matrices when $b = 1$. One cluster contains at most one basic group.

Figure 3.2 Different ways to assign B matrices, when the target field is partitioned into hexagon grids. The basic groups are labeled in bold and italic font. The compromised nodes are within the groups marked with a small (blue) circle and the groups affected by the compromised nodes are bounded by irregular (red) lines. In sub-figure (c), each group is actually assigned seven B matrices, where six matrices come from its neighbor groups and one is shown in the corresponding grid for the group.

 – For the methods of $b = 7$, all the groups are the basic groups and each group (except for those at the edge of the target field) has six neighbors. We further assign each group all the B matrices originally assigned to its neighbors. Thus, except for those at the edge of target field, each group is eventually assigned seven B matrices.

- After all the groups have their B matrices assigned, each node *tries to* select w rows from these matrices. Given a node of index i,
  – If its group is assigned more than w matrices, the node first randomly selects w matrices, then picks the i-th row from each selected matrix.
  – Otherwise, if its group has exactly or less than w matrices assigned, then the node directly picks the i-th row from each matrix. (That is why our scheme can not guarantee each node has exactly w rows picked from B matrices.)

- Finally, we set an identical transmission range r for all the nodes and deploy them into the target field group by group.

Figures 3.2(a), 3.2(b) and 3.2(c) show the different ways to assign B matrices, when $b = 2$, 3 and 7 and the target field is partitioned into hexagon grids. In these figures, the basic groups are labeled in bold and italic font. The common feature of these assignments is that any two neighbor groups share at least one common B matrix.

3.3.3.2 Key discovery phase

- After deployment, each node broadcasts its group index, the indexes of B matrices and the column of G, while receiving the same information from its neighbors.
- Then, each node checks if it has any index matches one of those received from its neighbors:
  - If two neighbor nodes have the same group index, then either of them can derive the pairwise key by computing the dot product of its own row of matrix A and the column received from the other.
  - If two neighbors share exactly one B matrix, then either of them can derive the pairwise key by computing the dot product of its row of that B matrix and the column received from the other.
  - If two neighbors share more than one B matrix, then they select the same matrix from the shared ones and derive the pairwise key based on the selected matrix B. To make agreement with the selected matrix, they can either negotiate with each other or (for example) simply select the matrix that has the smallest index.
  - If no matched index found, two neighbors will no longer communicate with each other.

3.3.4 Variants of Our Scheme

We have presented twelve variants of our scheme, but it is still not clear why we need these variants, what features they have, and whether there are other variants. In this sub-section, we try to answer these questions by studying three metrics, connectivity, memory usage, and security (i.e., resilience against node capture) of these variants.

- We evaluate connectivity by measuring the probability that two nodes from the same group and neighbor groups can establish a pairwise key, because the group-based deployment model indicates that most neighbor nodes are from the same group or neighbor groups.
- Memory usage is measured by the number of rows of B matrices stored in sensor nodes.
- To compare security of different variants, we introduce a new term, the *affected groups*. Since one matrix B may be shared by multiple groups, if the adversaries compromise some nodes of one group and obtain certain rows of the matrix B that is assigned to the group, they can compromise the additional links established by the nodes of other groups that share the same matrix B. We define the *affected groups* as, given a group with some nodes compromised, all other groups that share the same B matrices as the group. Hence, security of a variant can be roughly measured by the number of affected groups.

In our scheme, a group's B matrices come from its neighbor basic groups. We define a *cluster* as one (central) group along with its neighbor groups, and we study the variants of our scheme based on clusters, because we found that the different ways we assign basic groups in clusters directly form different variants. (Note: When nodes are deployed in hexagon, square or triangle grids, each cluster contains seven, nine or thirteen groups.)

The variants of our scheme can be categories into three classes depending on the value of b, which can only be 2, 3 or 7. Each class corresponds to a different assignment of B matrices, as shown in Figures 3.2(a), 3.2(b) and 3.2(c). We use methods $(b = 2, w = 2)$, $(b = 3, w = 3)$ and $(b = 7, w = 7)$ to represent these assignments respectively. They have almost the same performance in terms of connectivity, because they can all guarantee that any two nodes from the same group or neighbor groups establish a pairwise key. Thus, we only need to study these assignments in terms of memory usage and security.

When $b = 2$, we assign two basic groups in each cluster whose central group is a normal one. The two basic groups are located symmetrically with respect to the central one. In this assignment, each group is assigned at most two B matrices. Considering method $(b = 2, w = 2)$ of this assignment, each node stores *at most* two rows of B matrices and compromising nodes of one group can affect at most 13 groups. In Figure 3.2(a), 13 affected groups are bounded by irregular (red) lines, given that the compromised nodes are within group $B5B6$, which is marked with a small (blue) circle.

Similarly, when $b = 3$, we assign at most three basic groups in each cluster and each (normal) group is assigned at most three B matrices. This assignment is depicted in Figure 3.2(b). Considering method $(b = 3, w = 3)$, each node stores at most 3 rows of B matrices. As shown in Figure 3.2(b), given the compromised nodes within group $B5B8B9$ that is marked with a small (blue) circle, there are 16 affected groups that are bounded by irregular (red) lines. Clearly, method $(b = 3, w = 3)$ is worse than method $(b = 2, w = 2)$, because it consumes more memory of sensor nodes and produces more affected groups given that one group is compromised.

When $b = 4$, 5 and 6, we find that no explicit assignments can be constructed, because if we assign four, five or six basic groups in one cluster, the pattern of cluster cannot be repeated over the whole target field. However, it is possible to build an assignment when $b = 7$. As shown in Figure 3.2(c), each group is a basic one and hence a cluster contains seven basic groups. Considering method $(b = 7, w = 7)$, each node (except for those at the edge of field) has to store seven B matrices. This method produces 19 affected groups, given that some number of nodes within a group are compromised. Obviously, this methods is worse than methods $(b = 2, w = 2)$ and $(b = 3, w = 3)$ that are the representatives of other two assignments when $b = 2$ and 3, respectively.

When $b = 1$, we find that it is impossible to construct an assignment following the same rule as building other assignments. Otherwise, we have to assign exact one basic group in each cluster, which provides no guarantee to connectivity, that is, it is possible that two neighbor groups share no common matrix B. Hence, we have to modify the rule and build a new assignment, as shown in Figure 3.2(d). In this assignment, we have to assign each normal group (except for those at the edge of the field) three different B matrices, so each node should store at most three rows. When some nodes within a group are compromised, up to 43 groups might be affected. (Due to space limit, we do not show all the affected groups in Figure 3.2(d).) Compared with other assignments, this one is the worst in terms of memory usage and security and is eventually not selected as a possible solution to our bootstrapping problem.

So far, we have explained why there are only three possible assignments. For each assignment, we have multiple choices for sensor nodes to pick rows from B matrices. Let w denote the number of rows of B matrices picked by a node. w can take any value no more than that of b. Generally, given a fixed value of b, the smaller the value of w, the lower the connectivity, because it is less likely for neighbor nodes to find shared B matrices. Meanwhile, taking a smaller value of w leads to a higher resilience against node capture, since each matrix B is shared by less number of groups (or nodes). Similarly, if we fix the value of w, then the bigger the value of b, the lower the connectivity and the stronger the resilience, because the nodes have more choices to select B matrices and are less likely to find shared B matrices.

3.3.5 Shape of Grids

Only certain shapes of grids can be repeated to cover a continuous field. They are triangle, square (or rectangle) and regular hexagon. Figure 3.3 depicts the clusters of different shapes of grids (or groups). It shows that a triangle, square and hexagon grid have twelve, eight and six neighbors, respectively.

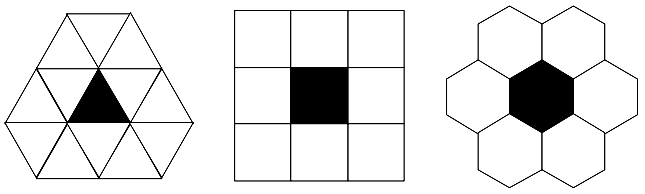


Figure 3.3 A cluster of triangle, square or hexagon grids (or groups). A triangle, square and hexagon grid have 12, 8 and 6 neighbors, respectively.

Same as to assign B matrices in hexagon grids, we can construct an assignment for square grids. As shown in Figure 3.4, we assign three basic groups in each cluster. A possible assignment for triangle grids is also to assign three basic group in each cluster, which has not been shown.

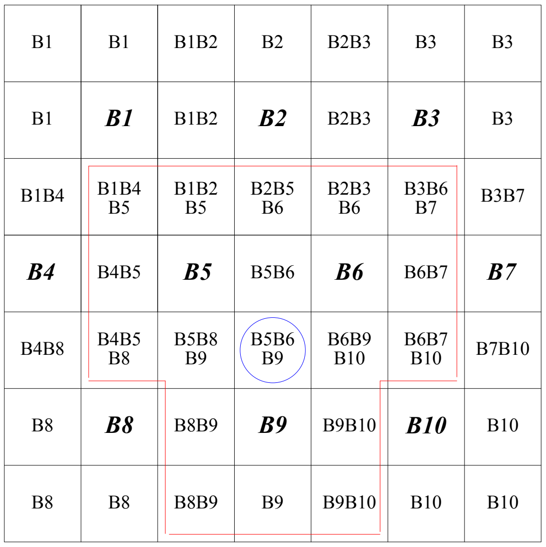

B1 B1 B1B2 B2 B2B3 B3 B3
B1 *B1* B1B2 *B2* B2B3 *B3* B3
B1B4 B1B4 B5 B1B2 B5 B2B5 B6 B2B3 B6 B3B6 B7 B3B7
B4 B4B5 *B5* B5B6 *B6* B6B7 *B7*
B4B8 B4B5 B8 B5B8 B9 B5B6 B9 B6B9 B10 B6B7 B10 B7B10
B8 *B8* B8B9 *B9* B9B10 *B10* B10
B8 B8 B8B9 B9 B9B10 B10 B10

Figure 3.4 Assignment of B matrices when a target field is partitioned into square grids. The basic groups are labeled in bold and italic font. The compromised nodes are within the group marked with a small (blue) circle and the affected groups are bounded by (red) lines.

Table 3.1 compares the assignments of B matrices given different shapes of grids, where a node picks the corresponding row from every matrix B assigned to its group. We list the

Table 3.1 Comparison among assignments given different shapes of grids

	Triangle	**Square**	**Hexagon**
Neighbor groups	12	8	6
Affected groups	31	21	13
Rows stored	4	4	3

number of neighbor groups for each group, that of affected groups given that some nodes are compromised with one group, and that of rows (from both A and B matrices) stored in nodes. It is clear that partitioning a target field into hexagon grids has the least number of affected groups and the least number of rows stored in nodes. This means that hexagon grid partitioning is the best in terms of security and memory usage, compared with square and triangle grid partitioning. Simulation results presented late will also prove this. (Triangle grid partitioning is apparently the worst and will no longer be studied.)

3.4 Connectivity Analysis

3.4.1 Grid Size Control

Let P_c denote connectivity, which is defined as the probability that a deployed sensor network is connected as the total number of nodes approaches infinity. Let p denote the probability that two neighbor nodes find at least one shared key. Obviously, connectivity grows up as p increases. Unlike the existing schemes that has a p value much smaller than one, most variants of our scheme can offer a much bigger p that approaches one. Four methods ($b = 2, w = 2$), ($b = 3, w = 3$) and ($b = 7, w = 6$ or 7) can even guarantee $p = 1$ for the neighbor nodes coming from the same group or neighbor groups. However, when the neighbor nodes are from non-neighbor groups, our scheme can only provide a very small or even zero p. Hence, to achieve a high connectivity, we need to control the size of grids to make the nodes from non-neighbor groups impossible to become neighbors.

When each group of nodes are uniformly distributed into a small local area, grid size control is easy, because we only need to make every grid cover the small area in which the corresponding group of nodes reside. However, it is not so straightforward when the location of nodes follows normal distribution. So, we focus on how to find a proper grid size under normal distribution. As shown in equation (3.2), the normal distribution of each group is identified by two parameters, the deployment point of the group and the variance σ. To measure grid size, we define the metric l as the distance between two neighbor deployment points. Our problem turns to: given some value of σ, how to set a proper value of l so that the nodes from non-neighbor groups are unlikely to become neighbors.

The property of normal distribution tells us that 99.87% nodes of a group would reside within a circle of radius 3σ that is centered at the group's deployment point. Given σ, we can use such a circle of radius 3σ to roughly represent a group of nodes. This representation can help us determine the value of l. First of all, l cannot be too big. Otherwise, when the value of σ is fixed and the size of grid is much larger than that of circle, all groups are separated from each other, which makes the deployed network completely partitioned. When we reduce the value of l, the deployment points are getting closer, which means that the circles (or groups) are moving to each other. Hence, the nodes should find more and more neighbors coming from their neighbor groups, instead of coming from their own groups, and the deployed network should be better connected. However, when the value of l becomes too small, e.g., the size of circle is even larger than that of a cluster of grids, the nodes of one group could spread into the grids that correspond to the non-neighbor groups, which definitely lowers the connectivity of deployed sensor network. Therefore, the value of l cannot be too big or too small. Our purpose is to choose the value of l as small as possible, but not allow the nodes from non-neighbor groups to become neighbors. In this way, we are able to maximize connectivity.

Let us consider the case that a target field is partitioned into hexagon grids as shown in Figure 3.1(b). In the figure, two circles of radius 3σ represent two groups whose grids are the nearest non-neighbor grid of each other. If the nodes from these two different groups are not able to meet each other, we claim that almost all the neighbors of one node should come from the node's own group or neighbor groups. If we further reduce the size of grid, those two circles become overlapping, which means that more and more nodes from non-neighbor groups become neighbors. Hence, the smallest grid size that prevents the nodes from non-neighbor groups from being neighbors, can be obtained when those two circles become tangent to each other. Observing triangle $\triangle ABC$ in Figure 3.1(b), we can easily find that $AB = l$, $AC = 3\sigma$ and $\angle CAB = 30°$. Thus, we have $AB = \frac{2}{\sqrt{3}}AC$, or equivalently, $l = 2\sqrt{3}\sigma$. Similarly, we can derive that $AB = l = 3\sigma$ in Figure 3.1(a), when the target field is partitioned into square grids. Given this setting of l, we conclude that each node has 99.87% probability to find the pairwise key with any of its neighbors.

3.4.2 Transmission Range Setup

Existing schemes [17, 22, 25] adopted *random (Bernoulli) graph* model [69] for connectivity analysis. However, this model does not consider transmission range of sensor nodes [68] and simply assumes any two nodes have the same probability p to establish a connection. In fact, when two nodes are out of each other's transmission range, p approaches zero.

To better model wireless sensor networks, we adopt *geometric random graph* [31, 68] for its consideration of nodes' transmission range. Given N nodes randomly placed in a unit target field $\mathcal{S}_f$, a geometric random graph $\mathcal{G}(N, r)$ is constructed in such a way that an edge between any two nodes exists if and only if they are within a distance of r from each other. Penrose [60] studied the longest edge of random Minimum Spanning Tree (MST). He proved that the longest edge M_N of an MST, whose N points are randomly and uniformly distributed in a unit square, satisfies

$$\lim_{N\to+\infty} Pr(N\pi M_N^2 - \ln N \leq \alpha) = e^{-e^{-\alpha}} , \tag{3.4}$$

for any real number α. Since a sensor network is always connected when $r \geq M_N$, if we set $N\pi r^2 = \ln N + \alpha$, then

$$\lim_{N\to+\infty} Pr(M_N \leq r) = \lim_{N\to+\infty} P_c = e^{-e^{-\alpha}} . \tag{3.5}$$

Equation (3.5) illustrates how to calculate the value of r by determining the value of α for achieving some desired connectivity P_c as N approaching infinity. However, we should also note that equation (3.5) has nothing to do with any finite value of N. We can only say that given α, if we always set $N\pi r^2 = \ln N + \alpha$, then connectivity of the deployed network approaches $e^{-e^{-\alpha}}$, when N is large enough. Hence, equation (3.5) can be used to determine the value of r for achieving the desired connectivity, when N is large enough. Although this result looks similar to that of Bernoulli graph [25], they are derived under different conditions, that is, transmission range has been taken into consideration in geometric random graph.

For those variants (or methods) providing $p = 1$ for the nodes from the same or neighbor groups, we can make use of the geometric random graph model to evaluate the required transmission range for achieving the desired connectivity. For example, in our scheme if we deploy

10^4 nodes uniformly into their grids over a $10^3 \times 10^3 m^2$ square field and require $P_c = 0.9999$, from equation (3.5) we can derive $\alpha \simeq 9.21$ and further obtain $r \simeq 24.22m$, which is the transmission range required to achieve the desired connectivity. However, if we adopt the basic scheme in the same condition, and set memory size $M = 200$ and key pool size $|S| = 10^5$ with $p = 0.33$, then we have to set $r \simeq 40m$ in each node to obtain a degree of 18 over 50 neighbors for achieving the same connectivity. Hence, our scheme requires a shorter transmission range than that of the basic scheme.

If nodes are not uniformly distributed, we cannot use equation (3.5) directly, because node density over the entire target field is not identical. It is easy to know that the area around a deployment point has higher node density than that around the intersection of every three neighbor hexagon grids (or every four neighbor square grids). If we assume the lowest node density over the entire target field, we can obtain an upper bound on transmission range in the worst case, because the lower the node density over the target field, the larger the transmission range required for achieving some desired connectivity. In practice, when we apply this larger transmission range, the nodes within the areas other than those intersection areas must be able to connect more neighbors. Hence, we can claim that the real connectivity over the entire target field would not be lower than the desired one given this new (larger) transmission range.

Figure 3.5 illustrates how to measure the lowest node density within a small circle area, when $l = 2\sqrt{3}\sigma$. This circle area has a radius of R and is centered at some intersection point. In this figure, we only show three neighbor groups (or grids), because the nodes from other groups do not reside in this circle area. For convenience, we represent normal distribution using polar coordinates and define n' as the number of nodes within the circle area. We have

$$n' = \frac{3n}{2\pi\sigma^2} \int_{2\sigma-R}^{2\sigma+R} h\,\theta(h)\, e^{\frac{-h^2}{2\sigma^2}}\, dh\ , \tag{3.6}$$

where $\theta(h) = 2\cos^{-1}(\frac{h^2+(2\sigma)^2-R^2}{4\sigma h})$. Thus, the lowest node density is $\frac{n'}{\pi R^2}$. Substituting $\frac{n'}{\pi R^2}$ for N in equation (3.5), we get

$$r = \sqrt{\frac{\ln(\frac{n'}{\pi R^2}) - \ln(-\ln(P_c))}{\frac{n'}{R^2}}}\ . \tag{3.7}$$

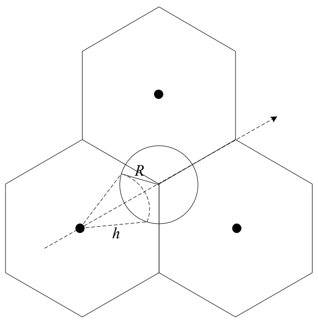


Figure 3.5 Computing the lowest node density within a circle area in polar coordinate system. The circle area has a radius of R and is centered at the intersection of three hexagon grids. The arrowed line is polar axis and h denotes the radial coordinate of some point.

For instance, if we deploy 10^4 nodes under normal distribution into a $10^3 \times 10^3 m^2$ square field with $R = 24.22m$ and $\sigma = 50m$, we get $r \simeq 31.25m$. That is, we only need to increase r from $24.22m$ under uniform distribution to $31.25m$ under normal distribution for achieving $P_c = 0.9999$. Compared with $r = 40m$ of the basic scheme under uniform distribution, our scheme requires a shorter transmission range for achieving the same connectivity even under normal distribution.

To see why our scheme requires a shorter transmission range, we study the number of links involved in the small circle area. In the basic scheme, when $n = 10^4$ nodes deployed in a $10^3 \times 10^3 m^2$ square field, each node needs around 50 neighbors for achieving $P_c = 0.9999$, when $r = 40m$ and $p = 0.33$. There are about $n' = \pi R^2 \frac{N}{10^3 \times 10^3} \simeq 18.4$ nodes in the circle. Hence, the number of links connected to nodes within this circle area is $p[(50-1)n' - \frac{n'(n'-1)}{2}] \simeq 247$, where $\frac{n'(n'-1)}{2}$ is the number of links whose both end nodes reside within the circle area and it is counted twice in $(50-1)n'$. On the other hand, simulation results show that our scheme generates 371 links in the same circle area, when $r = 40m$ and the target field is partitioned into hexagon grids. If we adopt square grids, there are even 483 links. Since our scheme achieves p approaching one and the basic scheme has only $p = 0.33$, it is easy to see why our scheme can generate more links with the same transmission range, which means our scheme can always achieve a connectivity higher than that of the basic scheme. In other words, for achieving the same connectivity, our scheme requires a shorter transmission range.

3.5 Security Analysis

3.5.1 Evaluation Metrics

We evaluate security, i.e., resilience against node capture, in terms of two metrics, *global security* and *local security*. The first one is measured as the fraction of links compromised, when adversaries randomly compromise some nodes over the whole target field. However, it might be easier for adversaries to capture nodes within a small local area. So, we also evaluate *local security*, which is defined as the fraction of links compromised, when compromised nodes are located within a grid. (For simplicity, we use a grid to simulate the small local area within which adversaries compromise nodes.) Obviously, local security metric is more stringent than global one, because breaking a secret matrix becomes easier for adversaries if compromised nodes concentrate within a small area.

Given that some nodes are already compromised, adversaries can compromise not only the links connected to these nodes directly, but also additional links between non-compromised nodes secured by the keys obtained from the compromised nodes. In our evaluation, we count all of the links that adversaries can compromise, but other schemes only consider the additional links. Therefore, our metrics are stricter than those of other schemes. In this work, we conduct theoretical analysis on local security, but study global security only by simulation because the theoretical analysis on global security is too complicated.

3.5.2 Theoretical Analysis of Local Security

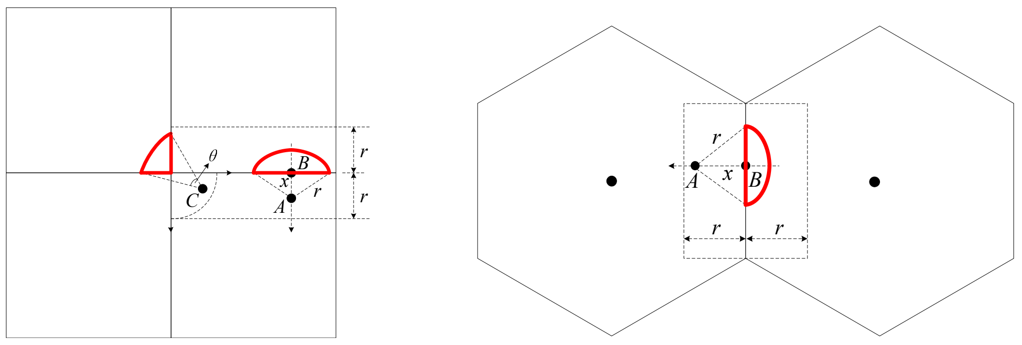
For simplicity, we consider uniform distribution in only two special cases: (1) exactly one node compromised, and (2) more than λ nodes compromised.

3.5.2.1 Exactly One Node Compromised

To compute local security, i.e., the fraction of links compromised given that exactly one node is compromised, we consider in-group links and inter-group links separately.

We first compute the number of in-group links that adversaries can compromise. Under uniform distribution, each node has about $\pi r^2 q - 1$ neighbors, where node density $q = \frac{N}{|\mathcal{S}_f|}$. Regardless of those nodes deployed at the edge of grids, the number of neighbors of a node is the same as that of in-group links this node has. For example, given $N = 10^4$ nodes uniformly deployed into a square field of $|\mathcal{S}_f| = 10^3 \times 10^3 m^2$ and transmission range $r = 24.22m$, each node has $\pi r^2 q - 1 \simeq 17.4$ neighbors, that is, about $\pi r^2 q - 1 \simeq 17.4$ in-group links will be compromised given one compromised node.

Then, we estimate the number of inter-group links that adversaries can compromise. When nodes are deployed into square grids shown in Figure 3.6(a), inter-group links can only be formed in a common area along the shared edge between two grids. This common area consists of two strip areas and the width of each strip area is r.



(a) Partition of square grids (b) Partition of hexagon grids

Figure 3.6 Computing the number of inter-group links compromised when a target field is partitioned into square or hexagon grids. A and C denote compromised nodes. x is the distance between A and B. The arrowed lines denote axes. Two arc areas and one quarter area are bounded by thick (red) lines.

We define L_{comm} as the average number of inter-group links compromised in a common area and show how to calculate L_{comm} that is twice of the number of inter-group links compromised in each strip area.

Let L denote the average number of inter-group links connected to a compromised node that happens to fall into a strip area. Equivalently, L is the average number of this node's neighbors coming from neighbor groups. As shown in Figure 3.6(a), these neighbors must be located in the arc area that is covered by the node's transmission region and within a neighbor grid. Hence, we have

$$L = \frac{q}{r} \int_0^r (r^2 \cos^{-1} \frac{x}{r} - x\sqrt{r^2 - x^2})\, dx \ , \tag{3.8}$$

where x is the distance between the node and the shared edge.

However, equation (3.8) counts twice the number of links formed with a diagonal neighbor group and hence should be subtracted from L. Observing the square grids in Figure 3.6(a), we find that only those nodes falling into the quarter area can form inter-group links with a diagonal neighbor group, where this quarter area is covered by the node's transmission region and within a diagonal neighbor grid. Let L' denote the average number of inter-group links formed when the node resides in the quarter area. We have

$$L' = \frac{4q}{\pi r^2} \int_0^r \int_0^{\sqrt{r^2-x^2}} [\frac{\theta \pi r^2}{2\pi} - \frac{r^2 \sin\theta}{2} + \frac{(\sqrt{r^2-x^2}+y)(\sqrt{r^2-y^2}-x)}{2}]\, dx\, dy \ , \tag{3.9}$$

where $\theta = \frac{\pi}{2} - \sin^{-1}\frac{x}{r} - \sin^{-1}\frac{y}{r}$. Hence, for square grids, if one node is compromised and resides in a strip area, $(L - L')$ inter-group links would be compromised. For hexagon grids, calculating L' is too complicated. So, we directly use L to estimate the number of inter-group links compromised, as shown in Figure 3.6(b).

The probability that a node falls into a strip area is the ratio of this strip area over the area of a grid. For hexagon grids, this probability is $\frac{r}{\frac{3\sqrt{3}l}{2}}$, and for square grids, it is $\frac{r}{l}$. Recalling that a common area consists of two strip areas, we get

$$\begin{aligned} L_{comm} &= \frac{4r}{3\sqrt{3}l} L && \text{(for hexagon grids)} \\ &\text{or } \frac{2r}{l}(L - L') && \text{(for square grids)} \ . \end{aligned} \tag{3.10}$$

To estimate the number of inter-group links the adversaries can compromise, we should further know how many common areas should be counted. Since a common area is only formed between two affected groups, we only need to know the number of affected groups. However, we should differentiate two cases $p = 1$ and $p \neq 1$ separately, when a compromised inter-group link is formed by nodes from neighbor groups.

Let us consider the case of $p = 1$ for nodes from neighbor groups. For example, for method $(b = 2, w = 2)$ with hexagon grid partitioning shown in Figure 3.2(a), compromising a node of normal group $B5B6$ would affect 13 groups that form 26 common areas. Therefore, $26L_{comm}$ inter-group links would be compromised. For method $(b = 3, w = 3)$ with square grid partitioning shown in Figure 3.4, there are totally 32 common areas formed among 21 affected groups. However, in some common areas, neighbor groups share two B matrices with only one compromised and a pairwise key is computed from a randomly chosen matrix. For instance, given that one node of group $B5B6B9$ is compromised, neighbor nodes from groups $B1B4B5$ and $B1B2B5$ can establish a pairwise key (or a secure link) using either $B1$ or $B5$. In this case, only the key (or the link) established based on $B5$ will be compromised, which means in these areas only half of inter-group links would be compromised. Since there are 16 such common areas, the number of inter-group links compromised in square grids is $(\frac{1}{2} \times 16 + 16)L_{comm} = 24L_{comm}$. (Note: The value of L_{comm} is different for hexagon and square grids.)

Because each node has $\pi r^2 q - 1$ neighbors, there are totally $\frac{N}{2}(\pi r^2 q - 1)$ links in the whole network. Considering both in-group links and inter-group links, we compute *local security given that exactly one node compromised* as follows:

$$\begin{aligned} & \frac{(\pi r^2 q-1)+26L_{comm}}{\frac{N}{2}(\pi r^2 q-1)} \quad \text{(for hexagon grids)} \\ \text{or} \quad & \frac{(\pi r^2 q-1)+24L_{comm}}{\frac{N}{2}(\pi r^2 q-1)} \quad \text{(for square grids)}\ . \end{aligned} \tag{3.11}$$

In the case of $p < 1$ for nodes from neighbor groups, compromising one node may not always cause inter-group links of some common area compromised, because two nodes from neighbor groups may not establish a secure link. For example, for method $(b = 3, w = 2)$ with hexagon grid partitioning shown in Figure 3.2(b), one node of group $B5B8B9$ is compromised. (For simplicity, we assume each of three matrices $B5$, $B8$ and $B9$ has a row compromised.) Now, let us consider two nodes such as node i from group $B5B8B9$ and node j from $B5B7B9$ become neighbors. When node i picks rows from matrices $B5B8$ and node j picks rows from $B7B9$, or node i from $B8B9$ and node j from $B5B7$, they cannot establish a secure link. Hence, in this case we should not count this non-existing link. In fact, node i has probability $\frac{1}{3}$ to select

$B5B8$ or $B8B9$, and node j is similar. So, they have only probability $\frac{7}{9}$ to establish a secure link. In general, for each common area i, we define p_i as probability that an inter-group link can be established for nodes within the common area, and $L_{comm,i}$ is defined for this common area similarly. Thus, for any method of hexagon grid partitioning, we compute local security given one compromised node as follows:

$$\frac{(\pi r^2 q - 1) + \sum_{i=1}^{26} p_i L_{comm,i}}{\frac{N}{2}(\pi r^2 q - 1)} . \tag{3.12}$$

Local security for methods of square grid partitioning can be calculated similarly.

3.5.2.2 More Than λ Nodes Compromised

If more than λ nodes of a group are compromised, the matrix A and some B matrices will be broken, as if all n nodes of the group were compromised. However, we cannot simply calculate the fraction of links compromised as n multiples of the value of (3.12), because in that way we would count each compromised link twice due to its both end nodes compromised. Because either end of each compromised link involves a compromised row, we count the number of compromised links twice. Thus, the true value is $\frac{n}{2}$ multiples of that of (3.12), which is

$$\frac{n(\pi r^2 q - 1) + n \sum_{i=1}^{26} p_i L_{comm,i}}{N(\pi r^2 q - 1)} . \tag{3.13}$$

Similar result can be obtained when nodes are deployed into square grids.

Our theoretic calculation based on expressions (3.11) and (3.13) matches simulation result. The error between theoretic calculation and simulation result is no more than 5% for hexagon grids and 3% for square grids, which has not been shown here.

3.6 Simulation Study

3.6.1 Simulation Setup

In this section, we perform simulation study. We compare our scheme with others in terms of security and connectivity, and study the impact of grid size and estimation error on the performance of our scheme. Without specification, we use method $(b = 2, w = 2)$ shown in

Figure 3.2(a) and method ($b = 3, w = 3$) shown in Figure 3.4 to represent our scheme, when nodes are deployed into hexagon and square grids.

We define connectivity as the probability that a deployed network is connected when the total number of nodes approaches infinity. However, it is impossible to measure in simulation. So, we adopt an alternative definition used in [23], which measures connectivity as the fraction of the size of the largest component of the deployed network, where a component is a connected subgraph.

We list our simulation setup in Table 3.2, where "Du's scheme" means *Du's deployment knowledge scheme.* In simulation, we assume that 10^4 nodes are deployed into a $10^3 \times 10^3 m^2$ square field with the desired connectivity $P_c = 0.9999$. In our scheme, we set $\sigma = 33.3m$ and $l = 3\sigma$ (or $2\sqrt{3}\sigma$) for square (or hexagon) grids. For Du's deployment knowledge scheme, we choose $\sigma = 50m$ and $l = 2\sigma$. Hence, both schemes have similar grid size. When testing normal distribution in our scheme, we compute transmission range dynamically based on the lowest node density that is measured online.

Table 3.2 Simulation Setup

	Our Scheme		**Du's**	**Basic**
	Square	**Hexagon**	**Scheme**	**Scheme**
N	10^4			
$\mathcal{S}_f$	$10^3 \times 10^3 m^2$			
P_c	0.9999			
t	100	104	100	–
n	100	96	100	–
l	$100m$	$115m$	$100m$	–
σ	$33.3m$	$33.3m$	$50m$	–
r	$24.22m$ (Uniform) dynamic (Normal)		$40m$	
M	100		100 & 140	100 & 200
λ	24	32	–	

We set $\sigma = 33.3m$ in our scheme, instead of $\sigma = 50m$ as in Du's deployment knowledge scheme. Otherwise, the number of groups of our scheme is only about half of that of Du's scheme. Since our scheme guarantees that the nodes from the same group are always connected, the smaller the number of groups, the higher the connectivity that our scheme can achieve. To be fair to Du's scheme, we decide to set $\sigma = 33.3m$ in our scheme in order to keep the number of grids approximately the same in both schemes.

3.6.2 Simulation Study on Local Security

Figure 3.7 depicts the fraction of links compromised in our scheme as a function of the number of nodes compromised, where these nodes are all located in the same group. In the figure, "Our (Sqr-Uni, M=100)" denotes our scheme under uniform distribution with square grids and memory of size 100. Other labels have similar meaning, for example, "Hex" denotes hexagon grids.

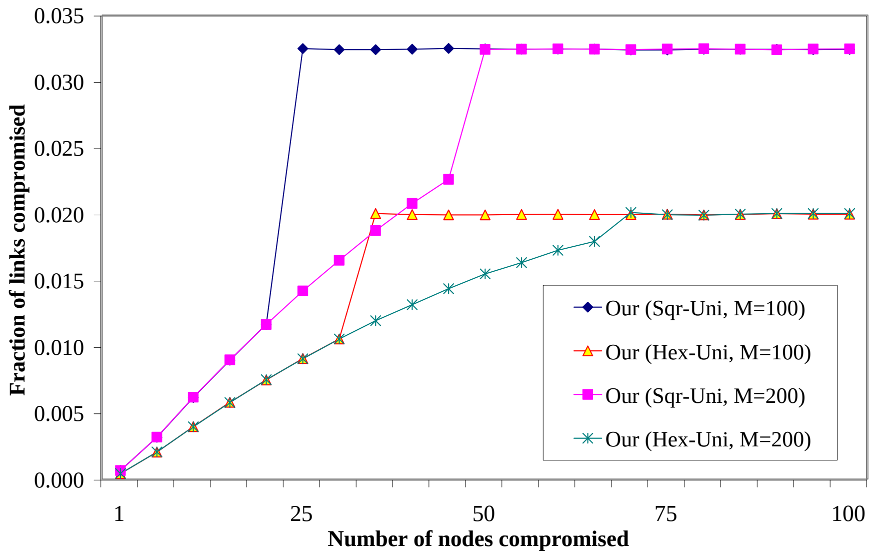


Figure 3.7 Local security: the fraction of links compromised as a function of nodes compromised, when all of the compromised nodes are located in the same group. "Our" is the short term of *our scheme*. "Sqr"/"Hex" denotes *Square/Hexagon* grids. "Uni" means *Uniform distribution*.

Observing the curves shown in Figure 3.7, we find that when the number of nodes compromised exceeds some threshold value, the fraction of links compromised no long increases. For example, for the curve labeled as "Ours (Sqr-Uni, M=100)", i.e., nodes are uniformly deployed in square grids with $M = 100$, when more than 25 nodes are compromised, the fraction of links compromised goes up to its highest value 0.032. It is due to λ-secure property of Blom's scheme we adopt. In this case, the number of nodes compromised is greater than the security threshold, which is $\lambda = \frac{100}{4} - 1 = 24$ (following equation (3.3)), so all secret matrices of the compromised group are broken.

From Figure 3.7, we also find that the curves of $M = 200$ are lower than those of $M = 100$. It demonstrates that our scheme performs better in local security given more memory space. It can be explained using equation (3.3), which shows that the larger the value of M, the greater the threshold λ and hence the better the performance of local security.

Further, we observe that the curves of "Hex" have better performance in local security than those of "Sqr", which means that partitioning a target field into hexagon grids is better than into square ones. There are two reasons to explain why hexagon partition is better. First, the length of each common area between hexagon grids is much shorter than that between square ones ($66m$ vs $100m$). Meanwhile, both kinds of partitions generate almost the same number of common areas (26 vs 24 shown in expression (3.11)). So, a node is less likely to fall into a common area in hexagon deployment, which in turn provides stronger resilience against node capture than square deployment. Second, a hexagon grid has less neighbors than a square one does. Thus, a node in hexagon deployment needs to store less rows, or equivalently, each row stored by a node in hexagon deployment is longer than that stored in square deployment. Recalling the length of row is $\lambda + 1$, we can see hexagon deployment offers a bigger λ and hence performs better in local security.

3.6.3 Simulation Study on Global Security

Figure 3.8 shows the comparison among various schemes in terms of global security, where "Du's deployment" represents Du's deployment knowledge scheme [23] and "Du's pairwise" stands for Du's pairwise key scheme [22]. Here, we choose the curve $\underline{(\tau = 5, p = 0.42),\ \textit{Simulation}}$ shown in Figure 3 of [22] as the representative of Du's pairwise key scheme. It achieves a connectivity approaching 0.9999 and requires $M = 200$.

Figure 3.8 illustrates that our scheme is always better than others (except for Du's pairwise scheme), for achieving the same connectivity with even a smaller memory space. For example,

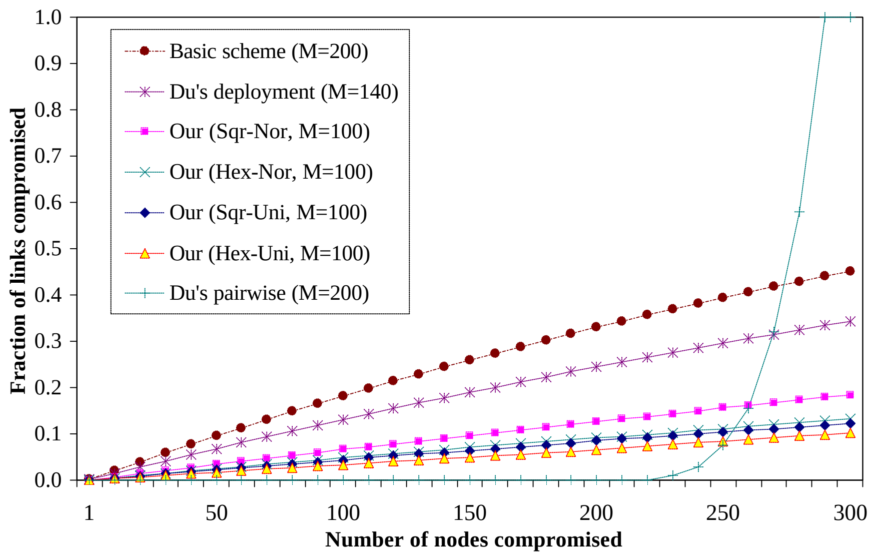


Figure 3.8 Global security: the fraction of links compromised as a function of nodes compromised, when the compromised nodes are distributed over the whole network. "Du's deployment" and "Du's pairwise" are the short term of *Du's deployment knowledge scheme* and *Du's pairwise key scheme*. "Nor" means *Normal* distribution.

given 200 compromised nodes, our scheme reveals at most 12.69% of links, compared with 24.44% in Du's deployment knowledge scheme and 32.99% in the basic scheme. At the same time, our scheme only has $M = 100$, but Du's deployment knowledge scheme requires $M = 140$ and the basic scheme $M = 200$. Firstly, we should thank Blom's scheme, which benefits our scheme in two aspects: (1) all in-group links are distinct, so in our scheme no additional in-group links exist no matter how many nodes are compromised; and (2) given that the number of compromised nodes is smaller than a threshold value, no additional inter-group links will be revealed. Hence, our scheme outperforms others in terms of global security. Secondly, by utilizing deployment knowledge, we drastically reduce the potential number of neighbors for each node. Therefore, in our scheme nodes can use their memory more efficiently such as to achieve high connectivity without storing too much secret information. Since the amount of secret information stored in nodes is reduced, our scheme reveals less additional links given that the same number of nodes are compromised.

Figure 3.8 also plots the curve of Du's pairwise key scheme [22], which achieves perfect security when no more than 250 nodes are compromised. However, if more than 300 nodes are compromised, almost all links are revealed. We did not plot the curve of Liu's polynomial-based key pre-distribution scheme [52, 53], because the authors did not provide data of their scheme for achieving the same connectivity as in our simulation. However, we find that this scheme has a threshold property similar to that of Du's pairwise key scheme. Observing the curve of $\underline{L = 2.5}$ (we use the same cell-size/transmission range ratio in our simulation) shown in Figure 6(b) of [52], we see that all links are revealed when more than 500 nodes are compromised. Compared with Du's and Liu's schemes, our scheme is also based on Blom's scheme, (where the bivariate polynomials used in Liu's scheme is a special form of Blom's scheme), but our scheme has a smoother curve in terms of security, that is, it never allows the whole network to be compromised.

Figure 3.8 also shows that our scheme under normal distribution is not as good as under uniform distribution. Under normal distribution, more nodes spread into neighbor grids and form more inter-group links, which increases the fraction of inter-group links compromised. Although nodes under uniform distribution perform better in terms of security, they generate less inter-group links. It is not good for routing data across groups, because less nodes are used to forward inter-group communications and their energy will be consumed up very quickly. More severely, adversaries are able to break the whole network more easily by compromising all nodes responsible for inter-group communications. Thus, deploying nodes under normal distribution is still useful.

3.6.4 Simulation Study on Connectivity

We list the connectivity of the variants of our scheme and others in Table 3.3. In our scheme, we assume that sensor nodes are deployed in hexagon grids. In this table, the variants are listed in the increasing order of connectivity. The general observation of these variants is that the larger the value of w and the smaller the value of b, the higher the connectivity provided by our scheme. Explanation to this observation can be found in section 3.3.4.

Table 3.3 also shows that our scheme outperforms others in terms of connectivity with even

Table 3.3 Comparison of connectivity among various schemes

Scheme	**Connectivity** (P_c)
q-composite scheme ($q = 2, M = 200$)	0.9358
Basic scheme ($M = 100$)	0.9867
Basic scheme ($M = 200$)	0.9999
Du's deployment knowledge scheme ($M = 100$)	0.9988
Du's deployment knowledge scheme ($M = 140$)	0.9999
Our scheme ($b = 7, w = 1, M = 100$)	0.9969
Our scheme ($b = 3, w = 1, M = 100$)	0.9977
Our scheme ($b = 2, w = 1, M = 100$)	0.9978
Our scheme ($b = 7, w = 2, M = 100$)	0.9993
Our scheme ($b = 3, w = 2, M = 100$)	0.9996
Our scheme ($b = 2, w = 2, M = 100$)	0.9999
Our scheme ($b = 7, w = 3, M = 100$)	0.9999
Our scheme ($b = 3, w = 3, M = 100$)	0.9999
Our scheme ($b = 7, w = 4 \sim 7, M = 100$)	1

smaller memory space. For example, to achieve $P_c = 0.9999$, we only need to set $M = 100$ for variants of our scheme, whereas the basic scheme and Du's deployment knowledge scheme require $M = 200$ and $M = 140$. q-composite scheme produces the lowest connectivity even with memory of size $M = 200$. (Note: Our simulation (which is not listed in the table) even shows that when $q = 2$ and $M = 100$, q-composite scheme has a connectivity less than 0.01, which means the deployed network is completely partitioned.)

3.6.5 Impact of Grid Size

In this section, we study impact of grid size on the variants of our scheme in terms of global security. In this study, sensor nodes are also deployed in hexagon grids. We define $l = a \cdot \sigma$, that is, given σ, a determines the grid size in our deployment.

Figure 3.9 plots the fraction of links compromised as a function of a, given that 200 nodes are compromised. In the legend of this figure, the variants are listed in the decreasing order of the fraction. For example, method ($b = 7, w = 7$) is on the top of the list, so it has the highest fraction and hence performs worst in terms of global security. We observe that the order of variants listed in the legend of Figure 3.9 is reverse in Table 3.3, which implies that

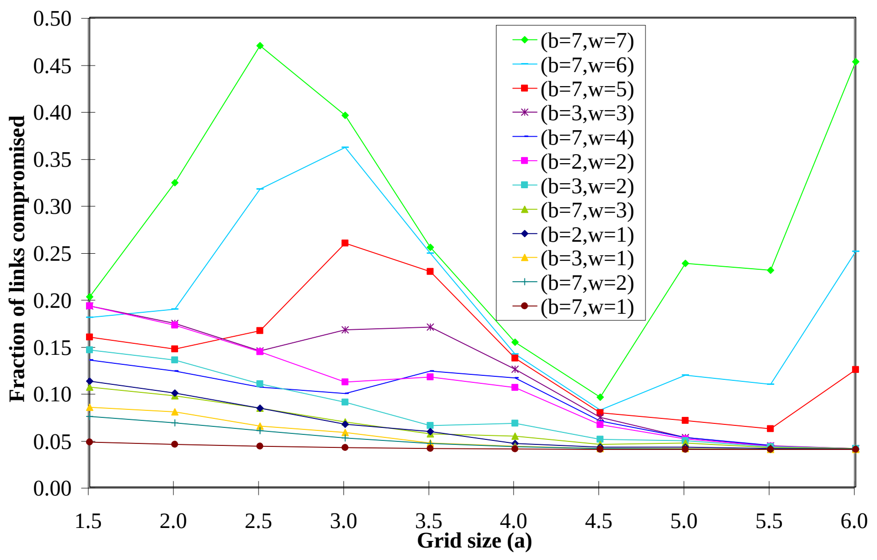


Figure 3.9 Global security: the fraction of links compromised as a function of grid size, with 200 compromised nodes randomly distributed over the whole network. In our scheme, we set $M = 100$. In the legend, the variants of our scheme are listed in the decreasing order of the fraction.

connectivity and global security conflict with each other.

Observing the variants in Figure 3.9, we can divide them into three categories:

1. Method $(b = 3, w = 2)$ to $(b = 7, w = 1)$, the last six variants on the name list of Figure 3.9. In the variants of this category, fraction of links compromised decreases gradually as the value of a is getting bigger.

2. Methods $(b = 3, w = 3)$, $(b = 7, w = 4)$ and $(b = 2, w = 2)$, the forth to sixth variant on the name list of Figure 3.9. Their fraction first increases when a is not too big, then goes down gradually with the increase of a.

3. Method $(b = 7, w = 7)$, $(b = 7, w = 6)$ and $(b = 7, w = 5)$, the first three variants on the name list of Figure 3.9. Their fraction grows up rapidly on both ends of curve, when the value of a is too small or too big.

In the following paragraphs, we explain why these categories perform so differently.

3.6.5.1 Why fraction decreases gradually

Typically, fraction of links compromised decreases with a increasing. It happens in all variants, especially in those of the first category. Our explanation is as follows: Under normal distribution, most nodes of each group reside within a circle area of radius 3σ around their deployment point. When σ is fixed and a becomes bigger, every grid is getting larger. Especially, when a approaches 6, most nodes of a group are located in their own grid and the deployed network becomes partitioned. In this process, more and more in-group links are formed, while inter-group links become contrary. In our scheme, in-group links are distinct and compromised links only come from inter-group ones, therefore, fraction of links compromised decreases gradually.

3.6.5.2 Why fraction increases with a

In the second and third categories, fraction of variants grows up when the value of a increases, as long as it is not too big. This is due to more and more B matrices broken.

Let us choose method $(b = 7, w = 7)$ as example for demonstration. In Figure 3.10, we plot the number of B matrices broken as well as that in total, with various values of a and 200 nodes compromised. It shows that when a is not too big, i.e., less than 3, more and more B matrices are broken with the increase of a. First, let us see why some B matrices are broken. For example, when $a = 2$, we have 126 groups in total. Given that 200 nodes are compromised, each group will have $\frac{200}{126} \simeq 1.5$ nodes compromised in average. Since each matrix B may be shared by up to 7 groups, it will have $1.5 \times 7 \simeq 10.5$ rows compromised in average. Meanwhile, we calculate security threshold from equation (3.3) as $\lambda = \frac{100}{7+1} - 1 \simeq 11$. This value of λ (i.e., 11) is quite close to the average number (i.e., 10.5) of rows compromised in each matrix B, thus, a matrix is very possible to be broken. As a result, some B matrices get broken. Then, if a becomes bigger, we will have less groups and hence each group will have more nodes compromised. Equivalently, each matrix B will have more rows compromised and more B matrices will be broken. That is why fraction of links compromised grows up with a. Further increasing the value of a has two results: (1) almost all B matrices will be broken; and

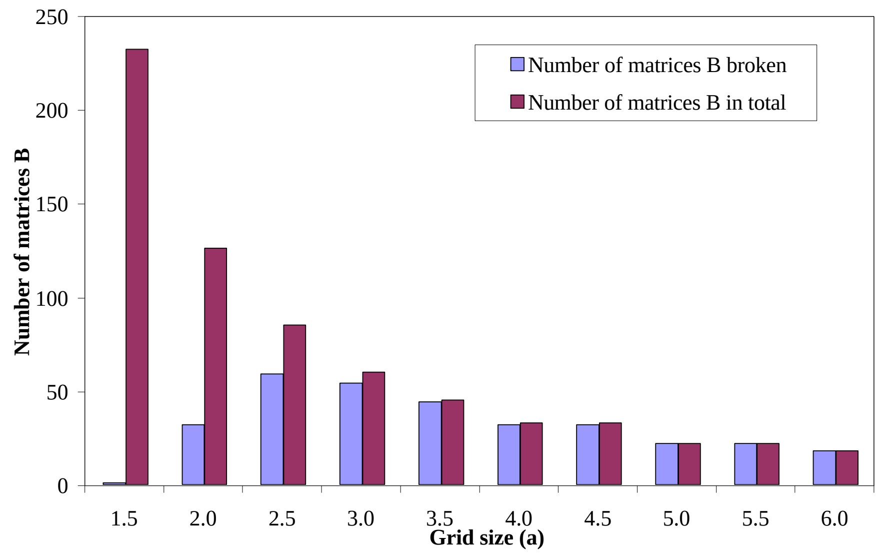


Figure 3.10 Comparison between the number of matrices B broken and the total number of matrices B for our method ($b = 7, w = 7$) as grid size increases. Sensor nodes (including 200 compromised nodes) are deployed into hexagon grids, and the compromised nodes are randomly distributed over the whole network.

(2) the total number of B matrices will decrease. Consequently, inter-group links are less and less, which keeps on decreasing fraction of links compromised.

Carefully observing the curves of the second and third categories in Figure 3.9, we can see impact of w on the fraction. Given a fixe value of b (i.e., 7), when w becomes bigger, the fraction increases earlier (i.e., when a is smaller) and more rapidly. From equation (3.3), we know that the bigger the value of w, the smaller the value of λ. Hence, a matrix B is more likely to be broken. In other words, more matrices B will be broken. So, the fraction of the variants of a smaller w increases earlier and more rapidly.

3.6.5.3 Why fraction grows up again with a

In the variants of the third category, when a is large enough (i.e., greater than 4.5), fraction grows up again. This is because more and more matrices A will be broken.

In this scenario, almost all matrices B have been broken, but we can carry out similar analysis to matrices A. Let us still consider method ($b = 7, w = 7$). For example, we have

22 groups in total when $a = 5$. So, each group will have $\frac{200}{22} \simeq 9$ nodes compromised in average. Since each node contains only one row from matrix A, the average number of rows compromised of each matrix A is also 9, which is close to $\lambda = 11$ of this variant. Therefore, some matrices A are likely to be broken. when a is getting bigger, more and more matrices A will be broken. This is why fraction of the variants of the third category grows up with a rapidly, when a is sufficiently big. For those variants of other categories, threshold λ is too big for some matrix A to be broken, given that only 200 nodes are compromised. So, we cannot observe fraction in those categories increase twice, even when a becomes very big.

3.6.6 Impact of Estimation Error

We assume that nodes' location satisfies a normal distribution, for example, when the nodes are deployed from a helicopter. It may be very hard to obtain an accurate estimation to the distribution. Here, we study the impact of error in the estimation of distribution parameter such as variance. Let σ denote the true value of variance and σ' denote the estimated value. We define $\sigma' = e\sigma$, where e determines the amount of estimation error. We consider $e \in [0.5, 1.5]$, which implies there exists up to 50% of error in estimation.

First, we study the impact on global security. Assume sensor nodes are deployed into hexagon grids. As we already discussed, the grid size in this case should be set as $l = 2\sqrt{3}\sigma$. However, due to the estimation error, we get $l = 2\sqrt{3}\sigma' = 2\sqrt{3}e \cdot \sigma$. Meanwhile, we already define $l = a \cdot \sigma$, so we derive that $a = 2\sqrt{3}e$. Thus, studying the impact of estimation error with $e \in [0.5, 1.5]$ is equivalent to studying the impact of grid size with $a = 2\sqrt{3}e \in [1.7, 5.2]$, which has been shown in Figure 3.9. We find that even in the worst case ($e = 0.5$), our scheme (method $(b = 2, w = 2)$) allows only 18% of links compromised, which is better than 24% of Du's deployment knowledge scheme and 32% of the basic scheme.

Figure 3.11 depicts the connectivity of our scheme as a function of estimation error. We only show the results of three variants that include methods $(b = 7, w = 1)$, the worst one to achieve some desired connectivity. Figure 3.11 shows that even when we under-estimate the distribution variance by 50%, method $(b = 7, w = 1)$ only causes a decrease in connectivity by

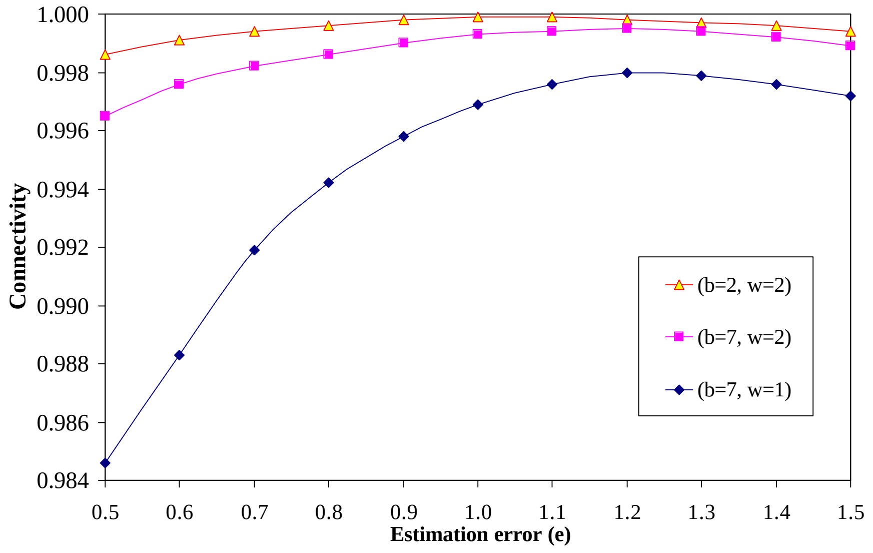


Figure 3.11 Connectivity of some variants of our scheme as a function of estimation error, when sensor nodes are deployed into hexagon grids. In our scheme, we set $M = 100$.

at most 1.4%. Meanwhile, other variants perform much better. So, we conclude that estimation error has little impact on our scheme in terms of connectivity. Moreover, we observe that the connectivity does not go down as quickly as we thought, even when we over-estimate the distribution variance by 50%. It is because we ignore the impact of transmission range when deriving $l = 2\sqrt{3}\sigma$. In fact, the value of transmission range is close to that of σ. Thus, when we have a large grid size due to over-estimating the variance, the nodes of each group can still establish secure links with a lot of others from neighbor groups.

3.7 Conclusion

We propose a key management scheme using deployment knowledge for establishing pairwise keys between sensor nodes. We study network connectivity based on geometric random graph model and show how to compute the required transmission range for achieving some desired connectivity. Simulation results show that our scheme outperforms others in terms of resilience against node capture. Meanwhile, it achieves a higher connectivity with a shorter transmission range and a lower memory requirement.

CHAPTER 4 ENHANCING AUTHENTICITY AND AVAILABILITY: Filtering False Data Injection and DOS Attacks in Wireless Sensor Networks

4.1 Introduction

Wireless sensor networks consist of a large number of small sensor nodes equipped with limited computation capacity, restricted memory space, limited power resource and short-range radio communication device. In military applications, sensor nodes may be deployed in a hostile environment such as battlefield to report the activities of enemy forces to the base station. However, they suffer various malicious attacks. One is *false report injection attack* [98], in which an adversary can inject false data reports containing non-existent events or faked readings via compromised nodes. It not only causes false alarms at the base station, but also drains out the limited energy of forwarding nodes. On the other hand, the adversary may also launch various DoS attacks to legitimate reports. In *selective forwarding attacks* [66], they selectively drop the reports. In *report disruption attacks* [85], they intentionally contaminate the authentication information (e.g., MACs) in the reports to make them filtered out by other nodes. Therefore, it is important to design a dynamic quarantine scheme to detect and filter these attacks or mitigate their impact to the functionalities of wireless sensor networks.

Recently, several schemes such as SEF [86], IHA [98], CCEF [84], LBRS [85] and LEDS [66] have been proposed to combat false report injection attacks and/or DoS attacks. However, they have different limitations. SEF is independent of network topology, but it has limited filtering capacity and cannot prevent compromised nodes from impersonating others. IHA requires that sensing nodes periodically establish multi-hop pairwise keys with others. Moreover, it needs a fixed path between the base station and any cluster head for transmitting messages in both directions, which cannot be guaranteed by some routing protocol such as GPSR [43] or the dynamic network topology. CCEF requires the fixed paths as IHA does, and even expensive public-key operations. Most severely, it does not support en-route filtering. LBRS and LEDS both utilize location-based keys for filtering false reports. LBRS introduces *report disruption attacks*, but does not give any concrete solution. LEDS tries to address selective forwarding attacks by allowing a whole cell of nodes to forward one report, which incurs high communication overhead. In addition, both schemes require that sensor nodes determine their locations in a short secure time slot. However, this assumption is not practical, because many localization approaches [12, 33, 58] take quite long and are not secure [14, 48, 49].

In this work, we propose a dynamic en-route scheme for filtering false report injection attacks and DoS attacks in wireless sensor networks. In our scheme, sensor nodes are organized into clusters and a legitimate report is validated by multiple message authentication codes (MACs) that are produced by sensing nodes using their own authentication keys. The authentication keys of each node form a hash chain. Before sending reports, nodes disseminate their keys to forwarding nodes using *Hill Climbing* approach. Then, they send the reports in rounds. In each round, every sensing node endorses the reports using a new key, and then disclose the key to forwarding nodes. Using the disseminated and disclosed keys, the forwarding nodes can verify the validity of reports. Our scheme allows each node to monitor its downstream nodes by overhearing their broadcast, which prevent the reports from being modified. Report forwarding and key disclosure are repeated by the forwarding node at each hop, until the reports are dropped or delivered to the base station.

Compared with existing ones, our scheme has the following advantages:

- Each node has its own authentication keys, which makes the uncompromised node not to be impersonated. The compromised nodes can only report fake or non-existent events occurring in the clusters that they belong to. Once they are detected, the base station can easily quarantine the infected clusters.

- We design *Hill Climbing* approach for key dissemination, which achieves that the nodes closer to clusters hold more authentication keys than those closer to the base station do.

This approach not only balances the memory requirement among nodes, but also makes false reports dropped as early as possible.

- Keys are disseminated to the forwarding nodes along multiple paths from a cluster to the base station, which not only reduces the cost to maintain key information in forwarding nodes in highly dynamic networks, but also mitigates the impact of selective forwarding attacks.
- We exploit the broadcast nature of wireless communication and let each node monitor its downstream nodes or neighbors. This prevents compromised nodes from launching DoS attacks by intentionally contaminating the reports or other control messages.

Simulation results show that, when compared with existing ones, our scheme drops false reports earlier even with a lower memory requirement, and can better deal with the dynamic sensor networks.

The rest of the chapter is organized as follows: We introduce some routing protocols designed for sensor networks in section 4.2 and define system model and goals in section 4.3. Then, we present our scheme in section 4.4, analyze its performance in section 4.5, and discuss simulation results in section 4.6. Finally, we summarize the advantages and the limitations of our scheme in section 4.7.

4.2 Routing Protocols for Sensor Networks

Before discussing our scheme, we first introduce some routing protocols designed for wireless sensor networks, because these protocols determine how sensor nodes exchange and distribute information and greatly affect the design of our scheme for filtering false reports.

Several distributed distance-vector based routing protocols [79] have been designed and implemented within TinyOS [73]. In these protocols, each node periodically broadcasts its routing cost to the sink, e.g., the base station, and builds a routing table according to the information received from its neighbors. Route is selected based on the routing metrics such as hop count or link quality.

GPSR [43] and GEAR [87] are location-aware routing algorithms, which assume each node knows its location. Route is determined as the neighbor with the shortest distance to the sink. If all neighbors are farther than itself, the forwarding node would use a right-hand rule to select the route. In GEAR, the energy level of each neighbor is also taken into consideration for route selection. (Note: in GPSR/GEAR, the path between two nodes is not bidirectional, i.e., the reports from node i to j may choose a different path from that used by the reports from node j to i.)

Braginsky et al. proposed Rumor [10] routing protocol, in which when a sensing node detects some event, it may create and send out an agent that is a message containing routing information about the event. The agent follows a straight path leaving the sensing node and is associated with a maximum TTL. Each node passed by the agent learns the route to the event. When the reports about some interested event are needed, the base station sends out a query message. The movement pattern of a query message is similar to that of an agent. When the query message is delivered to a node who knows the route to the event, a path between the base station and the sensing node (the event) can be established.

Note: Our scheme is not limited to these routing protocols and can take advantage of others for wireless sensor networks.

4.3 Problem Statement

4.3.1 System Model

We model the communication region of wireless sensor nodes as a circle of radius r that is called *transmission range*. We also assume that the links between neighbor nodes are bi-directional. (Note: If some links are not bi-directional, sensor nodes just ignore them.) That is, if the distance between two nodes is no more than r, they are the neighbor of each other and can communicate with each other.

Wireless sensor nodes may be deployed into some target field to detect the events occurring within the field. For example, in a military application, they are deployed to a battle field to detect the activities of enemy forces. We assume that sensor nodes form a number of clusters after deployment, each containing at least n nodes. In each cluster, one node is randomly selected as *cluster head*. To balance energy consumption, all nodes within a cluster take turns to serve as the cluster head. (Note: there is no difference between a cluster head and a normal node physically. A cluster head also senses events as a normal node.)

Given that an event occurs, e.g., tank movement, we assume that at least t nodes can detect it simultaneously, where t is a pre-defined system parameter. These nodes detecting the event are called *sensing nodes*. They generate *sensing reports* and broadcast to the cluster head. Then, the cluster head aggregates these sensing reports into *aggregated reports*, and forward through some forwarding nodes to the *base station*.

Figure 4.1 illustrates the organization of sensing nodes in wireless sensor networks. In the figure, CH and BS denote *Cluster Head* and *Base Station* respectively. $u_1 \sim u_5$ are forwarding nodes, and $v_1 \sim v_8$ are sensing nodes (they can also serve as forwarding nodes for other clusters). The black dots represent compromised nodes, which are located either in the clusters or en-route.

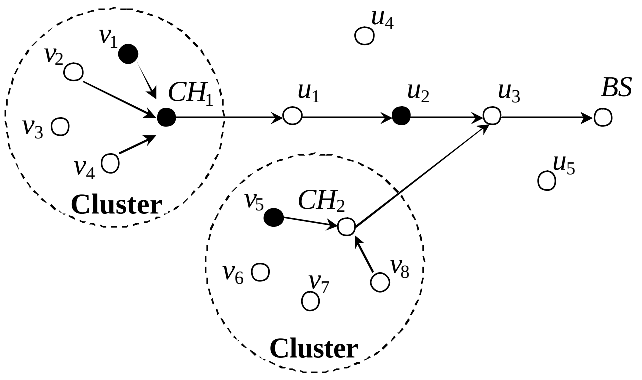


Figure 4.1 Sensor nodes are organized into clusters. The big dashed circles outline the regions of clusters. CH and BS denote *Cluster Head* and *Base Station* respectively. $u_1 \sim u_5$ are forwarding nodes, and $v_1 \sim v_8$ are sensing nodes (they can also serve as forwarding nodes for other clusters). The black dots represent the compromised nodes, which is located either within the clusters or en-route.

Note: we regard data reporting as a high layer application and ignore the impact of link quality to the delivery of reports. We assume that there exist some lower layer protocols such as routing or MAC layer protocols, which are able to handle the failures or collisions in wireless communication by utilizing the mechanisms of acknowledgement and retransmission.

We consider the case that the topology of wireless sensor networks is highly dynamic, because the sensor nodes are prone to failures and need to switch their state between active mode and sleeping mode for saving energy. Thus, two messages generated by the same cluster may be delivered along different paths to the base station. Moreover, we assume the messages transmitted from a cluster head to the base station and those from the base station to the cluster head do not necessarily follow the same path, because the underlying routing protocols such as GPSR [43], GEAR [87] or Rumor[10] that are designed for sensor networks cannot make this guarantee.

4.3.2 Threat Model

Typically, sensor nodes are not tamper-resistant and can be compromised by adversaries. We assume that each cluster contains at most $t-1$ compromised nodes, which may collaborate with each other to generate false reports by sharing their secret key information. Here, t is a pre-determined system parameter, which implies the extent of security that a filtering scheme can provide.

In this work, we consider the following attacks that the adversaries can launch through the compromised nodes.

- *False report injection attacks*: The compromised nodes can send false reports to the base station by pretending to observe some forged or non-existent events within the clusters that they belong to. Moreover, given sufficient secret information, they may even impersonate some uncompromised nodes of other clusters and report the forged events "occurring" within those clusters. These false reports not only cause false alarm at the base station, but also drain out the limited energy of forwarding nodes.

- *DoS attacks*: The compromised nodes can prevent the legitimate reports from being delivered to the base station, by either selectively dropping some reports (called *selective forwarding attacks* [66]), or intentionally inserting invalid authentication information into the reports to make them filtered by other forwarding nodes (called *report disruption attacks* [85]).

4.3.3 Goals

We require that each report be attached with t message authentication codes (MACs) produced by different sensing nodes with their authentication keys. A false report is defined as one that contains less than t valid MACs. Here, the selection of t determines a tradeoff between security and overhead. To tolerate more compromised nodes, we have to increase the length of reports.

As we discussed, the adversaries can launch either false report injection attacks or DoS attacks. Our objective is to design a scheme to detect these attacks and/or mitigate their impact to wireless sensor networks. Compared to existing schemes, we expect our scheme to achieve the following goals:

1. It can offer a higher filtering capacity and drop false reports earlier with acceptable memory requirement, where the filtering capacity of our scheme is defined as the average number of hops that a false report is allowed to travel.

2. It can address the report disruption attacks or mitigate the impact of the selective forwarding attacks.

3. It can accommodate highly dynamic sensor networks and does not require frequent path establishment or reparation.

4. It should not rely on any fixed paths between the base station and cluster heads for transmitting messages in both directions.

5. It should prevent the uncompromised nodes from being impersonated. So, when the compromised nodes are detected, the infected clusters can be easily quarantined by the base station.

4.4 Our Scheme

4.4.1 Overview

When some event occurs with some cluster, the cluster head collects *sensing reports* from the sensing nodes and aggregates them into *aggregated reports*. Then, it and forwards the aggregated reports to the base station through forwarding nodes. In our scheme, each sensing report contains one MAC that is produced by a sensing node using its authentication key (called *auth-key* for short), while each aggregated report contains t distinct MACs, where t is the maximum number of compromised nodes existing in each cluster.

In our scheme, each node possesses a sequence of auth-keys that form a hash chain. Before sending the reports, the cluster head disseminates the first auth-key of all nodes to the forwarding nodes along multiple paths to the base station. The reports are organized into rounds, each containing a fixed number of reports. In every round, each sensing node chooses a new auth-key to authenticate its reports. To allow forwarding nodes to verify the reports, the sensing nodes discloses their auth-keys in each round. Meanwhile, to prevent the forwarding nodes from abusing the disclosed keys, a forwarding node can receive the disclosed auth-keys, only after its upstream node overhears its broadcast of the reports. Receiving the disclosed keys, each forwarding node verifies the validity of the reports, and informs its next-hop node to forward or drop the reports based on the verification result. If the reports are valid, it discloses the keys to its next-hop node after overhearing. The process of verification, overhearing and key disclosure is repeated by the forwarding node at every hop, until the reports are dropped or delivered to the base station.

Specifically, our scheme can be divided into three phases, *key pre-distribution phase*, *key dissemination phase* and *report forwarding phase*. In the *key pre-distribution phase*, each node is preloaded with a distinct seed key from which it can generate a hash chain of its auth-keys. In the *key dissemination phase*, the cluster head disseminates each node's first auth-key to the forwarding nodes, which allows them to be able to filter false reports later. In the *report forwarding phase*, each forwarding node verifies the reports using the disclosed auth-keys and disseminated ones. If the reports are valid, the forwarding node discloses the auth-keys to its next-hop node after overhearing that node's broadcast. Otherwise, it informs the next-hop node to drop the invalid reports. This process is repeated by every forwarding node until the reports are dropped or delivered to the base station.

Figure 4.2 demonstrates the relationship between the three phases of our scheme. *Key pre-distribution* is performed before the nodes are deployed, e.g., it can be done offline. *Key dissemination* happens before the sensing nodes begin to send the reports. It may be executed periodically depending on how dynamically the topology is changed, and each time the latest (unused) auth-key of sensing nodes will be disseminated. *Report forwarding* occurs at each forwarding node and in each round.

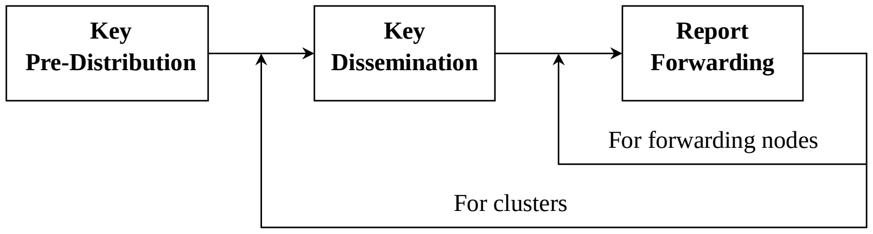


Figure 4.2 The relationship between three phases of our scheme. Key pre-distribution is preformed only once. Key dissemination is executed by clusters periodically. Report forwarding happens at each forwarding node in every round.

4.4.2 Detailed Procedures

In the section, we discuss the procedure of each phase in detail.

4.4.2.1 Key Pre-Distribution Phase

Key pre-distribution needs to be performed only once. It consists of two steps:

Step1: Each node is preloaded with a distinct seed key. From the seed key, it can generate a sequence of auth-keys using a common hash function h. Thus, each node's auth-keys form a hash chain. Let m denote the length of hash chain. Given node v_i and seed key $k_m^{v_i}$, its auth-keys are calculated as follows:

$$\begin{aligned} k_{m-1}^{v_i} &= h(k_m^{v_i}), \\ k_{m-2}^{v_i} &= h(k_{m-1}^{v_i}) = h^2(k_m^{v_i}), \\ &\vdots \\ k_1^{v_i} &= h^{m-1}(k_m^{v_i}), \end{aligned} \tag{4.1}$$

where v_i is the node's index, and $h^2(k_m^{v_i})$ means hashing $k_m^{v_i}$ twice. The first key of the chain is $k_1^{v_i}$, which should also be used the first, although it is the last one generated from the seed key. We assume that the base station is aware of each node's seed key, so that the adversaries cannot impersonate the uncompromised nodes.

Step2: Besides the seed key, each node is also equipped with $l+1$ secret keys, where l keys (called y-keys) are randomly picked from a global key pool (called y-key pool) of size v, and the rest one (called z-key) is randomly chosen from another global key pool (z-key pool) of size w. *Among n nodes of a cluster, we assume that there are at least t nodes each having a distinct z-key.*

Figure 4.3 shows the auth-keys and secret keys possessed by sensor nodes. For example, node v_i's auth-keys are $k_1^{v_i}, \cdots, k_m^{v_i}$, and its secret keys are $y_1^{v_i}, \cdots, y_l^{v_i}$ and z^{v_i}. If v_i has sufficient memory, it can store all of its auth-keys in memory. Otherwise, it only stores the seed key and generates an auth-key every time when necessary.

4.4.2.2 Key Dissemination Phase

In our scheme, the cluster head discloses the sensing nodes' auth-keys after sending the reports of each round. However, it is vulnerable to such an attack that a malicious node can pretend to be a cluster head and inject arbitrary reports followed by falsified auth-keys. To prevent this attack, we enforce *key dissemination*, that is, the cluster head should disseminate the *first* auth-keys of all nodes to the forwarding nodes before sending the reports in the first round. By using the disseminated keys, the forwarding nodes can verify the authenticity of the disclosed auth-keys, which is further used to check the validity and integrity of the reports.

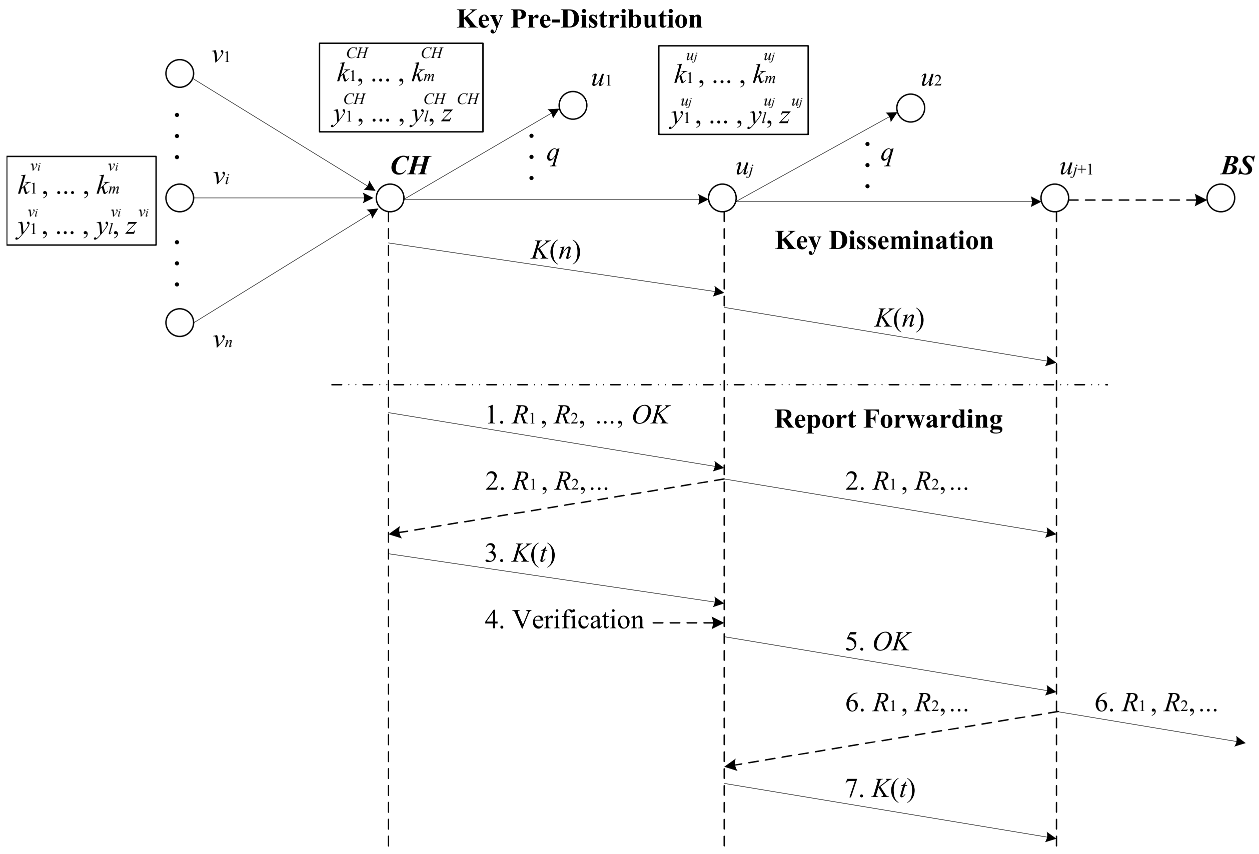


Figure 4.3 The detailed procedure of each phase. In the *key pre-distribution phase*, each node is preloaded with $l+1$ secret keys $y_1, \cdots, y_l$, and z, and generates a hash chain of auth-keys $k_1, \cdots, k_m$ from the seed key k_m. In the *key dissemination phase*, the cluster head disseminates the auth-keys of all nodes through message $K(n)$ to q downstream neighbor nodes. Every downstream node may decrypt and obtain some auth-keys from $K(n)$, then, it forwards $K(n)$ to q more downstream neighbor nodes, which repeat the same decrypting and forwarding operations. In the *report forwarding phase*, each forwarding node en-route performs the following steps: (1) It receives the reports from its upstream node. (2) If it receives confirmation message OK, then forwards the reports to its next-hop node. Otherwise, it discards the reports. (3) It receives the disclosed auth-keys within message $K(t)$ and verifies the reports using the disclosed keys. (4) It informs its next-hop node the verification result.

Key dissemination should be performed periodically in case that some forwarding nodes aware of the disseminated keys become failed, especially when the network topology is highly dynamic. In this case (of re-dissemination), the *first unused*, instead of the *first*, auth-keys will be disseminated. The first unused auth-key of a node is called the *current auth-key* of that node. When none of a node's auth-keys has ever been used, the current auth-key is just the first auth-key of its hash chain.

The detailed procedure of the key dissemination phase is as follows:

Step1: Each node constructs an *Auth* message, which contains $l+1$ copies of its current auth-key, each encrypted using one of its secret keys. For example, given node v_i, its *Auth* message is:

$$\begin{aligned} Auth(v_i) \;=\; \{\; & v_i,\; j_i,\;\; id(y_1^{v_i}),\; \{id(y_1^{v_i}), k_{j_i}^{v_i}\}_{y_1^{v_i}}, \\ & \cdots,\;\; id(y_l^{v_i}),\; \{id(y_l^{v_i}), k_{j_i}^{v_i}\}_{y_l^{v_i}}, \\ & id(z^{v_i}),\; \{id(z^{v_i}), k_{j_i}^{v_i}\}_{z^{v_i}}\;\}, \end{aligned} \tag{4.2}$$

where j_i is the index of its current auth-key. (Note: in the first dissemination, $j_i = 1$.) $id(y_1^{v_i})$ denotes the index of $y_1^{v_i}$ within the y-key pool, and $\{\cdot\}_{y_1^{v_i}}$ means an encryption operation using key $y_1^{v_i}$. In equation (4.2), the purpose of encrypting the index of a secret key along with an auth-key is to guarantee the correct decryption.

Step2: The cluster head collects the *Auth* messages from all nodes and aggregates them into message $K(n)$:

$$K(n) = \{\; Auth(v_1), \cdots, Auth(v_n)\;\}, \tag{4.3}$$

where $v_1, \cdots, v_n$ are all nodes of the cluster.

Step3: The cluster head chooses q ($q > 1$) forwarding nodes from its neighbors and forwards them message $K(n)$. These q nodes may be selected based on different metrics such as the distance to the base station, the link quality, the amount of energy available, the speed of energy consumption, or a combination of all. How to select the metric is application specific and out of the scope of this work. Anyway, the purpose is to find those nodes that can best forward the reports to the base station. Thus, when some downstream neighbor node dies, the reports can be easily switched to another node without re-disseminating $K(n)$.

Step4: When a forwarding node receives $K(n)$, it performs the following operations:

1. It verifies the validity of $K(n)$ to see if $K(n)$ contains at least t distinct indexes of z-keys. If not, this $K(n)$ is assumed to be forged and should be dropped.

2. It checks the indexes of secret keys in $K(n)$ to see if it has any shared key. If finding a shared secret key, it decrypts the corresponding auth-key using that key and stores the auth-key in memory. Obviously, it must assure that the decryption key is the correct one by checking the index encrypted along with the auth-key. Otherwise, it discards $K(n)$.

   Note: we have difference concerns to the use of y-keys and z-keys, although forwarding nodes can use both of them to decrypt auth-keys from $K(n)$. In our scheme, y-keys are mainly used to control how many auth-keys a forwarding node can obtain. Meanwhile, each node can use the only z-key it possesses to verify the validity of $K(n)$. Since each node has multiple y-keys, the number of y-keys shared by several compromised nodes can easily overwhelm that of distinct y-keys each report may have. Hence, y-keys cannot be used for verification.

3. $K(n)$ does not need to be disseminated to the base station. We define h_{max} as the maximum number of hops that $K(n)$ needs to be disseminated. (Note: we will study how to determine h_{max} in section VI.) If the forwarding node finds that $K(n)$ has already been disseminated by h_{max} hops, it discards $K(n)$. Otherwise, it forwards $K(n)$ to other q downstream neighbor nodes, which are selected using the same metric as the cluster head uses.

Each node receiving $K(n)$ repeats these operations, until $K(n)$ gets to the base station or has been disseminated h_{max} hops. Figure 4.3 illustrates the dissemination of $K(n)$ from the cluster head to forwarding nodes.

When the sensing reports are generated continuously and the network topology is highly dynamic, key dissemination should be performed periodically in case that all of q selected downstream nodes of some node die or fail. This period is determined based on the frequency of topology changing. The more frequently the topology changes, the more often the cluster head should disseminate auth-keys. We do not discuss how to determine the period here, because it is out of scope of this work.

4.4.2.3 Hill Climbing

We propose two important observations. First, when multiple clusters disseminate keys at the same time, some forwarding nodes may need to store the auth-keys for different clusters. *The nodes closer to the base station need to store more auth-keys than others (typically those closer to clusters) do*, because they are usually the hot spots and have to serve more clusters. For example, in Figure 4.1, u_3 serves two clusters and u_1 serves only one, so u_3 has to store more auth-keys. Second, *the false reports are mainly filtered by the nodes closer to clusters, while most nodes closer to the base station have no chance to use the auth-keys they stored for filtering.* If we could let the nodes closer to clusters hold more auth-keys, the false reports can be dropped earlier. Therefore, to balance the memory requirement of nodes and provide a higher filtering capacity, we propose *Hill Climbing* approach, which achieves that the nodes closer to clusters hold more auth-keys than those closer to the base station do.

Hill Climbing contains two variants, one for the key pre-distribution phase and the other for the key dissemination phase.

The first variant is: In Step2 of the key pre-distribution phase, instead of picking y-keys from a global key pool, each node selects each of its y-keys randomly from an independent hash chain. Specifically, the original y-key pool is partitioned into l equal-sized hash chains, each containing $\frac{v}{l}$ keys that are generated from a distinct seed key. As shown in Figure 4.4, the first hash chain contains keys $y_1^1, \cdots, y_u^1$, where $u = \frac{v}{l}$ and y_u^1 is the seed key. Similarly, $y_1^l, \cdots, y_u^l$ belong to the last chain. Node v_i chooses each of its y-keys $y_1^{v_i}, \cdots, y_l^{v_i}$ from a corresponding chain, and so does node u_j.

It is easy to know that a forwarding node holding a larger index y-key can always decrypt a sensing node's auth-key from $K(n)$, as long as the sensing node's y-key has a smaller index. Inspired by this, we propose the second variant: In Step4 of the key dissemination phase,

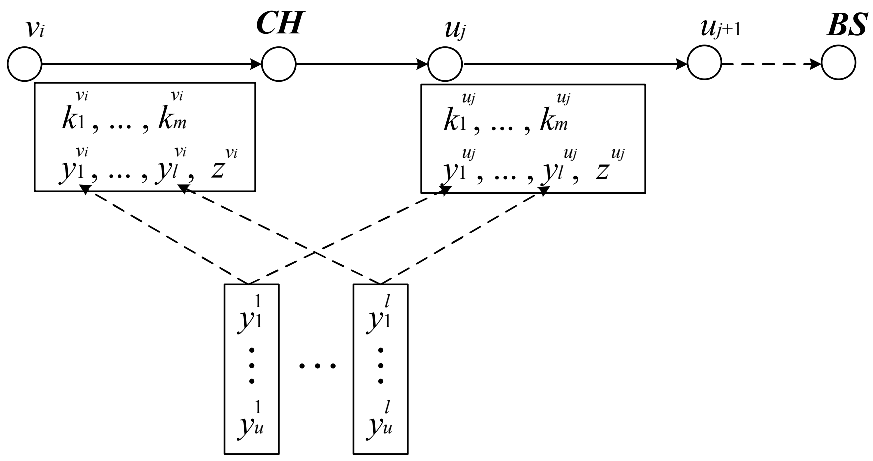

v_i
CH
u_j
u_{j+1}
BS
$k_1^{v_i}, \ldots, k_m^{v_i}$
$y_1^{v_i}, \ldots, y_l^{v_i}, z^{v_i}$
$k_1^{u_j}, \ldots, k_m^{u_j}$
$y_1^{u_j}, \ldots, y_l^{u_j}, z^{u_j}$
y_1^1
y_u^1
y_1^l
y_u^l

Figure 4.4 Key pre-distribution of *Hill Climbing*. Every node picks each of its y-keys randomly from a distinct hash chain whose length is $u = \frac{v}{l}$, while the z-key is still selected from the global key pool, instead of from a hash chain.

after a forwarding node decrypts an auth-key from $K(n)$, it updates $K(n)$ by encrypting the auth-key using its own y-key and then forwards the updated $K(n)$ to its downstream neighbor nodes. For example, if $K(n)$ contains $\{\ id(y_1^{v_i}), k_{ji}^{v_i}\ \}_{y_1^{v_i}}$ (as shown in equation (4.2)) and $id(y_1^{u_j}) > id(y_1^{v_i})$, u_j substitutes $\{\ id(y_1^{u_j}), k_{ji}^{v_i}\ \}_{y_1^{u_j}}$ for $\{\ id(y_1^{v_i}), k_{ji}^{v_i}\ \}_{y_1^{v_i}}$ and then forwards the new $K(n)$ to q downstream neighbor nodes. By enforcing this substitution at every forwarding node, the indexes of y-keys contained in $K(n)$ will be increased gradually, just like climbing hill. It becomes harder and harder for the nodes closer to the base station to decrypt the auth-keys from $K(n)$. Consequently, the nodes closer to clusters store more auth-keys, which makes the false reports dropped earlier.

A simpler way to make downstream nodes obtain fewer auth-keys is to discard the auth-keys obtained by forwarding nodes with a gradually increasing probability as they approach to the base station. However, *Hill Climbing* has two advantages: (1) It makes upstream nodes get more auth-keys than no only downstream nodes but also the upstream nodes at the same positions without using *Hill Climbing*. (2) It eliminates redundant decryptions and verifications, because when an auth-key has been decrypted by a upstream node, it is not necessary for a downstream node to decrypt that auth-key (or use that auth-key to verify a report) again.

4.4.2.4 Report Forwarding Phase

In this phase, sensing nodes generate sensing reports in rounds. Each round contains a fixed number of reports, e.g., 10 reports, where this number is pre-determined before nodes are deployed. In each round, every sensing node chooses a new auth-key, i.e., the node's current auth-key, to authenticate its reports.

Given node v_i, its sensing report $r(v_i)$ is:

$$r(v_i) = \{\ E, v_i, j_i, MAC(E, k_{j_i}^{v_i})\ \}, \tag{4.4}$$

where E denotes the event information, j_i is the index of v_i's current auth-key, and $MAC(E, k_{j_i}^{v_i})$ is the MAC generated from E using key $k_{j_i}^{v_i}$.

In each round, the cluster head generates the aggregated reports and forwards them to next hop, i.e., one of its q selected downstream forwarding nodes. Then, it discloses the sensing nodes' auth-keys after overhearing the broadcast from the next-hop node. The reports are forwarded hop-by-hop to the base station. At every hop, a forwarding node verifies the validity of reports using the disclosed keys and informs its own next-hop node the verification result. The same procedure is repeated at each forwarding node until the reports is dropped or delivered to the base station.

Figure 4.3 depicts the detailed procedure, which consists of the following steps:

Step1: In each round, the cluster head collects the sensing reports from all sensing nodes and generates a number of aggregated reports such as $R_1, R_2, \cdots$. It sends these aggregated reports and an OK message to next hop, u_j. *Each aggregated report should contain t MACs, each from a sensing nodes that has a distinct z-key.* For example, an aggregated report R looks as follows:

$$R = \{\ r(v_{i_1}), \cdots, r(v_{i_t})\ \}, \tag{4.5}$$

where $v_{i_1}, \cdots, v_{i_t}$ denote t sensing nodes whose z-keys are different. Since every sensing node reports the same event information E, only one copy of E is kept in the aggregated report R.

Step2: Receiving the aggregated reports and the OK message, u_j forwards the aggregated reports to next hop, u_{j+1}. The cluster head overhears the broadcast of aggregated reports from u_j.

Step3: Overhearing the broadcast from u_j, the cluster head discloses the auth-keys to u_j by message $K(t)$,

$$K(t) = \{\ Auth(v_{i_1}), \cdots, Auth(v_{i_t})\ \}. \tag{4.6}$$

$K(t)$ contains the auth-keys of $v_{i_1}, \cdots, v_{i_t}$. It has the same format as $K(n)$, but only t auth-keys.)

Step4: Receiving $K(t)$, u_j first checks the authenticity of the disclosed keys using the disseminated ones that it decrypted from $K(n)$ before. Then, it verifies the integrity and validity of the reports by checking the MACs of reports using the disclosed keys. The verification process is as follows:

1. To verify the validity of $K(t)$, u_j checks if $K(t)$ is in correct format and contains t distinct indexes of z-keys. If not, it drops $K(t)$.

2. To verify the authenticity of the auth-keys in $K(t)$, u_j checks if each auth-key it stored can be generated by hashing a corresponding key in $K(t)$ with certain number of times. For example, suppose u_j has already stored node v_i's auth-key $k_\alpha^{v_i}$, and v_i discloses a new key $k_\beta^{v_i}$ in $K(t)$. u_j checks if $\beta > \alpha$ and $k_\beta^{v_i} = h^{\beta-\alpha}(k_\alpha^{v_i})$. If not, $k_\beta^{v_i}$ is either replayed or forged, and $K(t)$ should be dropped. If $K(t)$ is valid, u_j stores $k_\beta^{v_i}$, instead of $k_\alpha^{v_i}$, in its memory.

3. To verify the integrity and validity of reports $R_1, R_2, \cdots$, u_j checks the MACs in these reports using the disclosed auth-keys that it decrypts from $K(t)$.

Step5: If the reports are valid, u_j sends an OK message to u_{j+1}. Otherwise, it informs u_{j+1} to drop invalid reports. (The negative message is not shown in Figure 4.3).)

Step6: Similar to Step2, u_{j+1} forwards the reports to next hop.

Step7: Similar to Step3, after overhearing the broadcast from u_{j+1}, u_j discloses $K(t)$ to u_{j+1}.

Every forwarding node repeats Step4 to Step7 until the reports are dropped or delivered to the base station.

We exploit the broadcast nature of wireless communication. In our scheme, each node monitors its next-hop node to assure no message is forged or changed intentionally.

4.5 Performance Analysis

4.5.1 Filtering capacity

Filtering capacity of our scheme is defined as the average number of hops that a false report can travel. It is determined by the probability that a false report can be detected by the forwarding node at every hop. For simplicity, we consider the worst case in which each false report contains exactly one forged MAC. We define *detecting probability* as the probability that a forwarding node has the valid auth-key to detect the forged MAC. Since a forwarding node should decrypt the auth-key from $K(n)$ or $K(t)$, the detecting probability is equivalent to the probability that a forwarding node has at least one shared secret key to decrypt the auth-key.

Without using *Hill Climbing* in our scheme, each node randomly picks l y-key from a pool of size v and one z-key from a pool of size w. The probability that two nodes share at least one common y-key is $1-\frac{\binom{v-l}{l}}{\binom{v}{l}}$ and that of sharing the same z-key is $\frac{1}{w}$. Therefore, the detecting probability is

$$\begin{aligned} p &= 1-\frac{\binom{v-l}{l}}{\binom{v}{l}}+\frac{1}{w}-\left(1-\frac{\binom{v-l}{l}}{\binom{v}{l}}\right)\frac{1}{w} \\ &\simeq 1-\frac{\binom{v-l}{l}}{\binom{v}{l}}, \end{aligned} \tag{4.7}$$

when $\frac{1}{w}$ is much smaller than $1-\frac{\binom{v-l}{l}}{\binom{v}{l}}$. For example, if $l=2$ and $v=w=20$, then $p \simeq 0.275$. Meanwhile, we also know that a forwarding node can decrypt and store np auth-keys for each cluster in average.

When using *Hill Climbing*, each node picks l y-key from different hash chains. To decrypt an auth-key of some sensing node, a forwarding node should have a shared z-key or at least one y-key whose index is larger than that of the sensing node's y-key. The probability of having a y-key of larger index is $\frac{1}{2}$. However, if the forwarding node is i hops away from a cluster, to decrypt an auth-key of some sensing node, it should have a y-key with its index greater than that of no only the sensing node, but also other $i-1$ upstream forwarding nodes. This happens with probability $1-(1-(\frac{1}{2})^i)^l$. Therefore, when using *Hill Climbing*, the detecting probability of a forwarding node i hops away from the cluster is

$$\begin{aligned} p_i &= 1-(1-(\frac{1}{2})^i)^l + \frac{1}{w} - (1-(1-(\frac{1}{2})^i)^l\frac{1}{w} \\ &\simeq 1-(1-(\frac{1}{2})^i)^l, \end{aligned} \tag{4.8}$$

when $\frac{1}{w}$ is small. Averagely, the forwarding node stores np_i auth-keys for the cluster. If $l=2$, $v=w=20$ and $i=1$, then $p_i = p_1 \simeq 0.775$, which is much larger than that without using *Hill Climbing*.

Let $P(h)$ denote the probability of filtering a false report within h hops, which is also the fraction of false reports that can be filtered within h hops. When using *Hill Climbing*, we have

$$P(h) = 1 - \prod_{i=1}^{h}(1-p_i). \tag{4.9}$$

Without using *Hill Climbing*, $P(h) = 1-(1-p)^h$ because p_i is always equal to p.

Let H_{avg} denote the average number of hops that a false report can travel. When using *Hill Climbing*, we derive that

$$H_{avg} = \sum_{i=1}^{\infty} ip_i \prod_{j=1}^{i-1}(1-p_j), \tag{4.10}$$

where $p_i \prod_{j=1}^{i-1}(1-p_j)$ denotes the probability that a false report is dropped at exactly the i-th hop. Without using *Hill Climbing*, the forwarding node at every hop has the same detecting probability p. In this case, we have

$$H_{avg} = \sum_{i=1}^{\infty} ip(1-p)^{i-1} = \frac{1}{p}. \tag{4.11}$$

It indicates that a false report can travel averagely $\frac{1}{p}$ hops.

4.5.2 Energy Savings

To compare energy savings of various schemes, we adopt the same energy consumption model used in SEF [86]. Let L_r denote the length of a normal report without using any filtering scheme and L'_r denote the length of an authenticated report attached with MACs.

The amount of legitimate reports and false reports is 1 and β. Assume each report travels H hops without using any filtering scheme and each node has a detecting probability p when using a filtering scheme. As shown in SEF, the energy consumption for transmitting a normal report is

$$e = L_r H(1+\beta), \tag{4.12}$$

where a normal report is 24-byte long, that is,

$$L_r = 24 \times 8 = 192 \text{ bits.} \tag{4.13}$$

The energy consumption for transmitting an authenticated report is

$$\begin{aligned} E &= L'_r[H + \beta \sum_{h=1}^{H} hp(1-p)^{h-1}] \\ &\simeq L'_r(H + \beta \frac{1}{p}). \end{aligned} \tag{4.14}$$

In SEF, a key index is 10-bit long and the Bloom filter is 64-bit long. So,

$$L'_r = 306 \text{ bits.} \tag{4.15}$$

In our scheme, each forwarding node forwards not only the report R, but also control messages $K(n)$, $K(t)$ and OK. We have

$$L'_r = L_R + \gamma(L_{K(t)} + L_{OK}) + \delta L_{K(n)}, \tag{4.16}$$

where $L_R, L_{K(n)}, L_{K(t)}$ and L_{OK} are the length of aggregated reports and that of the corresponding control messages. γ denotes the ratio of the number of messages $K(t)$ (or OK) over that of reports, and and δ is the ratio for $K(n)$. Assume each sensing node generates 10 reports in each round and key dissemination is carried out every 10 rounds, then $\gamma = \frac{1}{10}$ and $\delta = \frac{1}{100}$. let us further assume that each MAC and secret key are 64-bit long, and OK message, node index and key index are 8-bit long. If $l = 2$, $t = 5$, $v = w = 20$ and $n = 10$, then $L_R = 592$ bits, $L_{K(t)} = 1160$ bits, $L_{K(n)} = 2320$ bits, and $L_{OK} = 8$ bits. Following equation (4.16), we get

$$L'_r \simeq 732 \text{ bits.} \tag{4.17}$$

We set when not using *Hill Climbing*.

Let $\beta = 10$, $p = 0.05$ for SEF and $p \simeq 0.275$ for our scheme. From equations (4.12) to (4.17), we derive that

$$e = 2112H, \tag{4.18}$$

$$E_{Our} \simeq 732(H+36), \tag{4.19}$$

$$E_{SEF} = 306(H+200), \tag{4.20}$$

where E_{Our} and E_{SEF} denote the energy consumption in our scheme and SEF.

When compared with SEF, our scheme saves $1 - \frac{E_{Our}}{E_{SEF}} \simeq 50\%$ of energy when $H = 10$, and 40% when $H = 20$. As compared with the case of not using any filtering scheme, our scheme consumes more energy, about $\frac{E_{Our}}{E_e} - 1 \simeq 50\%$ when $H = 10$. It starts to save energy after 20 hops. However, 20 hops may seem to be too large for real sensor networks. To make our filtering scheme more practical, we turn to *Hill Climbing*, which can save more energy by dropping the false reports earlier.

When using *Hill Climbing*, the forwarding nodes have a higher detecting probability. For example, the fist forwarding node of a cluster has a detecting probability of 0.775, which is much larger than 0.275 when not using *Hill Climbing*. Assume the average detecting probability at each hop is 0.5. Then, the energy consumption of our scheme becomes $E_{Our} \simeq 732(H + 20)$, which means that our scheme starts to save energy after 10 hops and hence becomes more practical. In addition, this comparison is based on a static network. When the network is highly dynamic, our scheme can drop the false reports much earlier than other schemes do (see our simulation results), which indicates that our scheme is more efficient and practical for dynamic environments.

However, we also notice that this comparison is not quite fair for other schemes, because our scheme involves extra overhead caused by control message $K(n)$, which is 4 times longer than an authenticated report and has to be disseminated at most h_{max} hops with q nodes each hop. Although we try to reduce the frequency for disseminating $K(n)$ and choose a small value of h_{max} and q (see our simulation results), this overhead may be still high. In addition, $K(n)$ has more than 2000 bits and needs to be transmitted in multiple messages (the payload of each message in TinyOS is 29 bytes.), which increases the overhead of our scheme. We do not measure the extra overhead caused by extra control messages, but we do agree that it is a main drawback of our scheme.

4.5.3 Filtering DoS Attacks

Except for false reports injection attacks, adversaries may launching DoS attacks such as *report disruption attacks* and *selective forwarding attacks* to prevent legitimate reports from being delivered to the base station.

In *report disruption attacks*, compromised sensing nodes intentionally insert invalid MACs into the reports to make them dropped by others. Most existing schemes are vulnerable to these attacks. LEDS [66] mitigates the impact of these attacks by allowing a few invalid MACs in the reports, but it does not solve the problem completely. On the contrary, our scheme can easily combat these attacks. In our scheme, each sensing node discloses its auth-key following the reports. Its neighbors can overhear the node's reports and disclosed auth-key, and verify the validity of its MACs. Since the node is monitored by its neighbors, it is deterred from launching the attacks.

In *selective forwarding attacks*, compromised forwarding nodes selectively drop the legitimate reports. These attacks are very hard to detect. In our scheme, each forwarding node disseminates $K(n)$ to q downstream neighbor nodes that are awarded the filtering capacity. To deal with the attacks, we can require the forwarding node send the reports to all of these q nodes. Unless all of them are compromised, the legitimate reports can always be delivered to the base station. However, this solution incurs high communication overhead. An alternative way is that in each round the forwarding node sends the reports to a randomly chosen node out of those q ones. Given that only x nodes are compromised, the impact of selective forwarding attacks will be mitigated into $\frac{x}{q}$.

4.5.4 Filtering Other Attacks

Our scheme introduces new control messages and is a little bit complicated. It may suffer some attacks that are specific for itself. Here, we discuss how to deal with these attacks.

Attack1: A cluster head is compromised. In our scheme, normal nodes take turns to act as cluster head, so there is no difference between a cluster head and a norma node. It means that a cluster head may be easily compromised or any compromised node can claim to be a cluster head.

A compromised cluster head can disseminate a forged $K(n)$ and then inject false reports arbitrarily. Our scheme offers two countermeasures to prevent this attack. First, any node including the compromised node is monitored by other nodes of the same cluster. When any node overhears a forged $K(n)$, it can easily detect it by checking its auth-key that is supposed to be contained in the $K(n)$. Thus, it knows that the cluster head is compromised and can report to the base station to revoke that node. (Note: how to revoke a node is out of the scope of our work.) Second, each $K(n)$ must contain t distinct z-key. Since we assume that each cluster has at most $t-1$ compromised nodes, there must be at least one invalid z-key in the forged $K(n)$. The size of global z-key pool is w, thus, each forwarding node can filter the forged $K(n)$ with a probability $p = \frac{1}{w}$, or equivalently, the forged $K(n)$ can travel $\frac{1}{p} = w$ hops following equation (4.11).

Similarly, when a compromised cluster head generates forged $K(t)$ or false reports, these two countermeasures are still useful. The forged message and reports are not only monitored by other sensing nodes, but also filtered by forwarding nodes.

Attack2: Consecutive compromised nodes collaborate. If two or more consecutive nodes are compromised and collaborate with each other, they can share the auth-keys they decrypt from $K(t)$ to generate false reports without being monitored.

Assume x consecutive nodes are compromised, where $x \leq (t-1)$. They can decrypt xpt auth-keys from a valid $K(t)$, where p (as computed in equation (4.7)) is the probability that one can decrypt an auth-key from $K(n)$. So, they can generate a false report containing only $(t - xpt)$ forge MACs. Since p is also the probability that a forged MAC can be detected, the false report can be detected by an uncompromised node with the probability $\frac{(1-xp)t}{p}$ and travel $\frac{p}{(t-xpt)}$ hops. Obviously, this analysis makes sense only when $t - xpt > 0$. Given $x = (t-1)$ and $t = 5$, we have $p < \frac{1}{t-1} = 0.25$. It means when $p < 0.25$, $t-1$ consecutive compromised nodes cannot gain sufficient auth-keys to produce a false report without being filtered. We can achieve this by adjusting the values of l, v and w. For example, when $l = 2$ and $v = w = 20$, we can easily get $p = 0.23$ following equation (4.7).

Attack3: Compromised forwarding nodes abuses OK message. The OK message can be abused in either negative or positive ways. First, a compromised forwarding node can always or selectively send negative messages to make the reports dropped by its next-hop node. This is actually a selectively forwarding attack caused by the abuse of OK message. It can be addressed with the solutions we discussed above. Second, using OK message, a compromised forwarding node can cheat its next-hop node to forward false reports one more hop. Given at most $t-1$ compromised nodes en-route, the worst case is that every two compromised nodes are separated by a uncompromised one. Therefore, in the worst case, these $t-1$ compromised nodes can make false reports forwarded at most $2t-2$ hops.

Attack4: The compromised nodes use invalid node index. A false report containing unknown node indexes can escape the detection from forwarding nodes, because they thought that they do not have the corresponding auth-keys for those unknown nodes.

It is not possible to make the forwarding nodes be aware of the indexes of all nodes in a cluster, because nodes are randomly deployed. Our solution is to re-assign each node a continuous index from $[1, n]$. One way is to let nodes finish this job by themselves. In neighbor discovery phase, each node can overhear the indexes of others within the same cluster. So, it can sort these indexes in some order and re-assign itself a new index. Another way is to let each forwarding node hash the nodes indexes into $[1, n]$. However, different indexes may be hashed into the same value, which causes false positive errors. We can deploy more than n nodes into each cluster and only allow one node to send reports when hash collision happens among several nodes. The benefit is that we always have enough working nodes for each cluster, even when some node dies.

Attack5: In Hill Climbing, a large index y-key is compromised. In *Hill Climbing*, y-keys of nodes are picked from hash chains. When a node having a large index y-key is compromised, all y-keys of smaller indexes can be derived. In fact, the compromise of those y-keys of smaller indexes is not an attack to our scheme. In *Hill Climbing*, y-keys are only used by forwarding nodes to decrypt auth-keys from $K(n)$. When an adversary compromises a large index y-key, he could not get more auth-keys by deriving other y-keys of smaller indexes within the same hash chain.

However, when a forwarding node containing a large index y-key is compromised, other downstream nodes having y-keys of smaller indexes can no long decrypt the corresponding auth-key, so they cannot detect a false report which happens to contain a forged MAC using the corresponding auth-key. The filtering of this false report has to be delayed until it is received by another forwarding node whose has a y-key of an even larger index. This is an attack caused by compromising a large index y-key at early hop, which may allow a false report to travel more hops. However, our simulation results show that *Hill Climbing* is much better than other schemes even facing the impact of this attack.

4.6 Simulation Study

We study the performance of our scheme by simulation and also compare our scheme with others such as SEF, IHA, and CCEF in terms of filtering capacity, fraction of false reports filtered, and memory requirement in different environments.

4.6.1 Simulation Setup

- 10^3 nodes are randomly deployed to a $10^3 \times 10^3$ m^2 square field with the base station located at the center. The transmission range of each node is 50 m. These nodes are divided into 100 clusters, where each cluster contains exactly $n = 10$ nodes.

- Each node picks $l = 2$ y-keys and one z-key, where the size of y-key pool and z-key pool is $v = w = 20$.

- The size of memory used by each node is denoted as *mem*, and measured by the number of keys that each node stores. Typically, $mem = 50$. In our simulations, each cluster head disseminates auth-keys to forwarding nodes. One node may need to store the auth-keys from different clusters. When the memory is full, each node equally assigns its memory to each cluster that it serves.

- Each node forwards $K(n)$ to $q = 2$ selected downstream neighbor nodes, until $K(n)$ reaches the base station or has been forwarded h_{max} hops. Typically, $h_{max} = 10$.

- Each aggregated report contains $t = 5$ MACs, and there are at most $t - 1 = 4$ compromised nodes in each cluster. The compromised nodes from the same cluster collaborate with each other to share the compromised secret keys.

- To simulate the dynamic topology, we apply a simple ON/OFF operation model in which each node switches its state between ON and OFF periodically. The duration of ON and OFF states satisfies an exponential distribution. We define the percentage of OFF time as *network churn rate*, which indicates the extend of topology changing.

- The routing protocol adopted in our simulations is GPSR. However, IHA and CCEF are not suitable for GPSR, because they both require the existence of a fixed path between each cluster head and the base station for transmitting control messages in both directions. To make them work on top of GPSR, we revise them accordingly and define a *revised IHA* and a *revised CCEF*. Moreover, in the revised CCEF, we let each forwarding node always keep on forwarding the reports for which it has no witness key. This is different from the original CCEF in which those reports are always discarded.

4.6.2 Simulation Results

Comparing our scheme with others by simulation, we obtain the following results:

1. Our scheme drops false reports earlier even with a lower memory requirement. In some scenario, it drops the false reports within 6 hops with only 25 keys stored in each node, but other scheme requires 12 hops even with 50 keys stored.

2. Our scheme can better deal with the dynamic topology of sensor networks. It achieves a higher filtering capacity and filters out more fraction of false reports than others do in a dynamic network.

3. *Hill Climbing* increases the filtering capacity of our scheme greatly and balances the memory requirement among sensor nodes.

4.6.2.1 Fraction of False Reports Filtered vs Number of Hops Traveled

We first consider the case that $t-1$ compromised nodes are within the same cluster. We assume a static environment in which all nodes are in ON state.

Figure 4.5 illustrates how the fraction of false reports that are filtered increases as the number of hops that they travel grows. In our scheme, $K(n)$ is disseminated at most $h_{max} = 10$ hops and each node stores at most $mem = 50$ keys. In SEF, each node picks $k = 50$ keys from one of $n = 10$ partitions and each partition contains $m = 100$ keys. Hence, our scheme and SEF are subject to the same memory constraint.

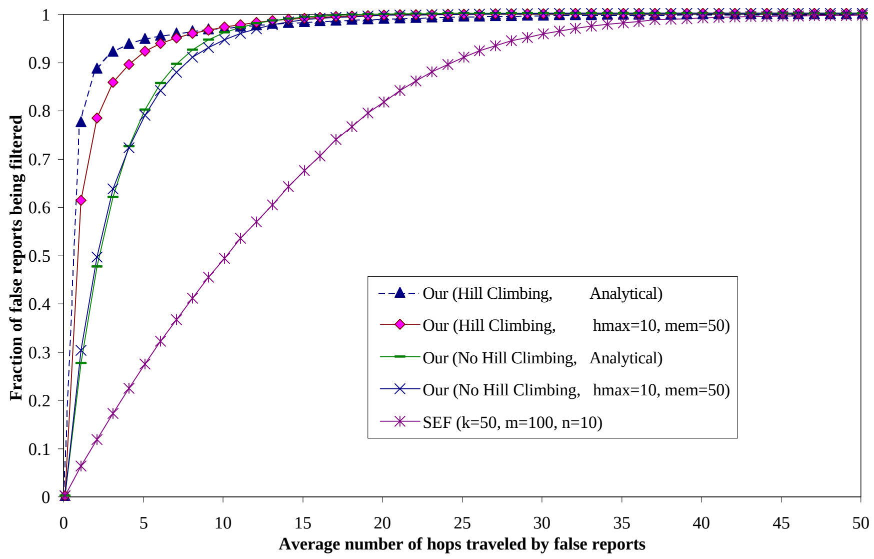


Figure 4.5 Fraction of false reports being filtered as a function of the number of hops that they traveled (In our scheme $q = 2$)

The simulation results in Figure 4.5 show that our scheme can drop 90% of false reports within 5 hops (when using *Hill Climbing*) or 10 hops (without *Hill Climbing*), while SEF needs about 25 hops to achieve the same performance. The reason is that our scheme offers a higher detecting probability than SEF does. For example, when not using *Hill Climbing*, our scheme achieves $p \simeq 0.275$ that is much larger than $p = 0.05$ in SEF. If using *Hill Climbing*, the detecting probability of the first forwarding node is up to $p_1 = 0.775$, about 15 times larger than that in SEF. This also proves that *Hill Climbing* can greatly improve the filtering capacity of our scheme.

We also plot the analytical results of our scheme in Figure 4.5. We observe that the simulation result is a little worse than the analytical one in the case of using *Hill Climbing*. This is due to the limit of memory size ($mem = 50$) that we set in the simulation.

4.6.2.2 Filtering Capacity vs Memory Size

Figure 4.6 depicts how the filtering capacity varies with the different memory size, where the filtering capacity is measured as the average number of hops that a false report can travel.

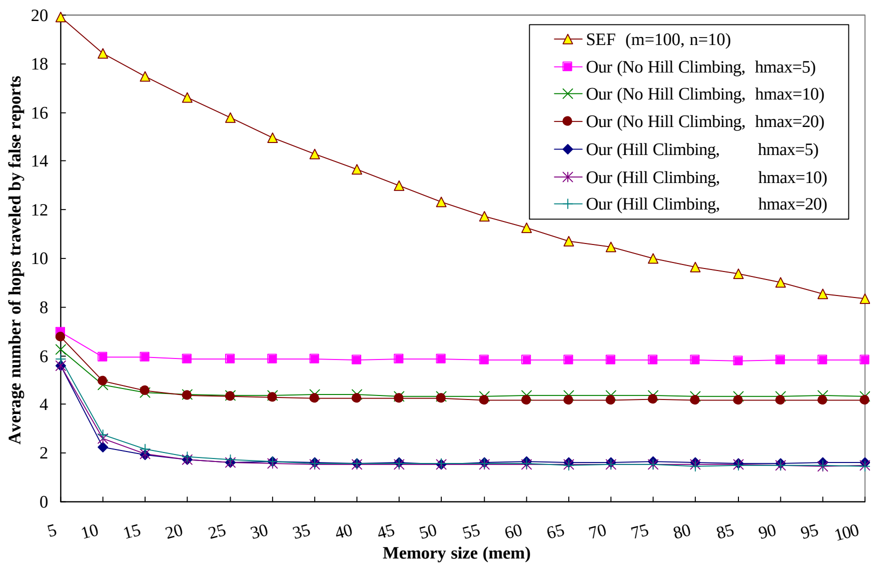


Figure 4.6 The average number of hops traveled by false reports as a function of the size of memory for each node. (In our scheme, we set $q = 2$.)

The smaller the number of hops, the higher the filtering capacity. In SEF, when each node picks all of 100 keys from a partition, a false report can still travel more than 8 hops. On the contrary, in our scheme a false report can travel no more than 6 hops even when each node stores at most $mem = 25$ keys. So, our scheme outperforms SEF even with a much lower memory requirement. Intuitively, a larger memory size can increase the filtering capacity. However, we do not observe too much improvement in our scheme when $mem > 25$, which implies that storing 25 keys in each node is sufficient for our scheme to filter most false reports.

4.6.2.3 Filtering Capacity vs Maximum Number of Hops for Key Dissemination

Figure 4.7 shows the impact of h_{max} to the filter capacity of our scheme. Typically, disseminating auth-keys farther gives more nodes the auth-keys and enables them to filter false reports. On the contrary, the limited memory size forces each node discard more auth-keys for each cluster in order to accommodate more clusters. Hence, it is not always helpful to increase the value of h_{max}. Figure 4.7 indicates that the best value of h_{max} is between 5 to 10, because 90% of false reports have been dropped within 10 hops, as shown in Figure 4.5.

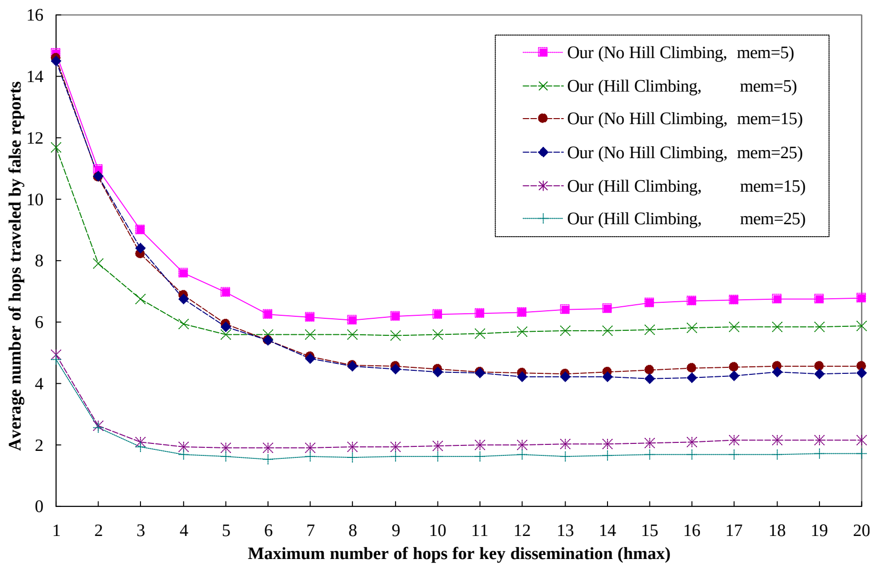


Figure 4.7 The average number of hops traveled by false reports as a function of the maximum hops for key dissemination. (In our scheme, we set $q = 2$.)

4.6.2.4 Filtering Capacity in Dynamic Environments

Sensor nodes are prone to failures and may also turn off their radio and CPU to save energy, which makes the topology of sensor networks highly dynamic. We simulate this by applying an ON/OFF model and use *network churn rate* to measure the extend of the topology changing. We still assume $t-1$ compromised nodes are within the same cluster.

In Figure 4.8, we compare the filtering capacity of different schemes in dynamic environments of various network churn rate.

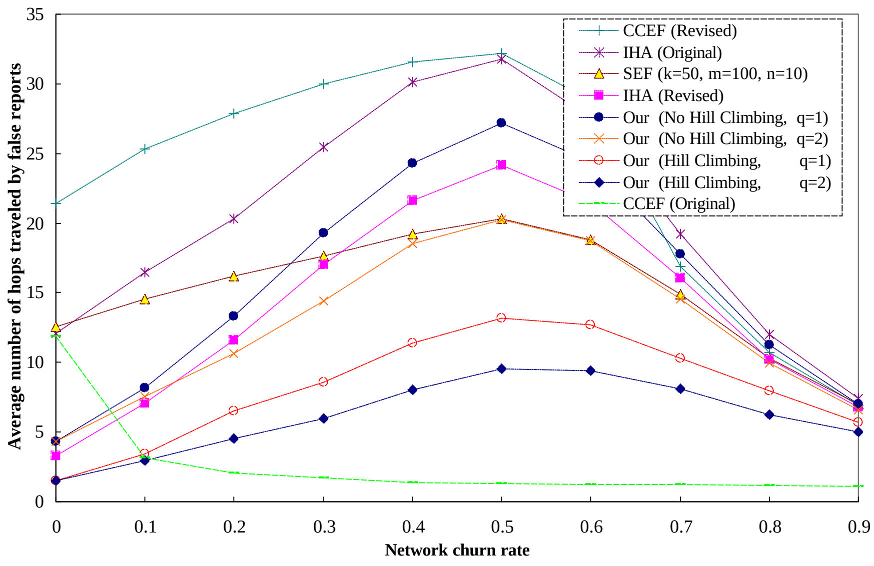


Figure 4.8 The average number of hops traveled by false reports as a function of the network churn rate. (In our scheme, we set $h_{max} = 10$ and $mem = 50$.)

We distinguish the following two cases:

- *The network churn rate is greater than 0.4*: In this case, the deployed network is severely partitioned, and a lot of repots are dropped for path broken. Our experiments (not plotted in Figure 4.8) show that around 10% of reports are dropped for path broken when the network churn rate is 0.4, while more than 60% are dropped when the rate is 0.9. It implies that when the rate is bigger than 0.4, more and more false reports are dropped due to network partition.

- *The network churn rate is smaller than 0.4*: In this case, partition is not a big problem, and the filtering capacity is mainly determined by the detecting probability of nodes and the length of paths from cluster heads to the base station. When the network churn rate increases, the paths become longer and hence the false reports can travel more hops. We focus on this case in the following discussion.

Figure 4.8 shows that our scheme has a higher filtering capacity than most of other schemes have in dynamic environments. The only exception is the original CCEF, which seems to outperform our scheme. However, it has a severe drawback, that is, a lot of legitimate reports are dropped, because the forwarding nodes lack the corresponding witness keys in dynamic environments. In addition, the original CCEF is not as good as our scheme in a static situation, because it cannot filter the false reports en-route that are generated by compromised cluster heads. Let us check it by simple calculations. When the network churn rate is zero, the average length of a path is 27. Each cluster head has a probability $\frac{t-1}{n} = 40\%$ to be compromised, in which case a false report can travel 27 hops. When the cluster head is not compromised, a false report can go only one hop. Hence, the filtering capacity of CCEF is $27 \times 40\% + 1 \times 60\% = 11.4$ hops, which is consistent with our simulation results and much greater than that of our scheme.

In Figure 4.8, we observe that the curves of SEF and our scheme are relatively flat, which means these two schemes are more independent of the topology changes than others. SEF achieves this from the use of probabilistic key pre-distribution, however, it has to make a tradeoff for a low filtering capacity. Our scheme achieves this by choosing q downstream neighbor nodes for each forwarding node in key dissemination. We studied the impact of q when choosing different values from 1 to 6. Intuitively, a larger value of q makes more nodes receive the necessary key information for filtering false reports, and hence is more suitable for dynamic networks. However, the limited size of memory forces the nodes to reduce the stored auth-keys and lowers the filtering capacity. So, the value of q should be properly selected and not be too large or small. Our experiments show that the curves for $q = 2$ to 6 are close to each other, although $q = 4$ is slightly better than others. Considering the high overhead incurred by choosing a large value of q, we decide to set $q = 2$ in our scheme. To make the figure clear, we only plot the curves for $q = 1$ and 2. For example, let us observe the curves of our scheme with *Hill Climbing*. The average number of hops traveled by false reports is 3.4618 when $q = 1$ and 2.9469 when $q = 2$ in the case that the network churn rate is 0.1, and that number is 11.3745 when $q = 1$ and 8.0412 when $q = 2$ in the case that the rate is 0.4. So, choosing $q = 2$ can improve the filtering capacity by $1 - \frac{2.9469}{3.4618} \simeq 15\%$ when the rate is 0.1, and $1 - \frac{8.0412}{11.3745} \simeq 29\%$ when the rate is 0.4. This means that our scheme is more adaptive to dynamic environments when choosing $q > 1$ (e.g., $q = 2$), and it earns more benefit when the topology is highly dynamic.

Figure 4.9 depicts the fraction of that false reports that reach the base station as a function of the network churn rate. When the network churn rate is greater than 0.4, the fraction goes down quickly because the network is severely partitioned. When the network churn rate is

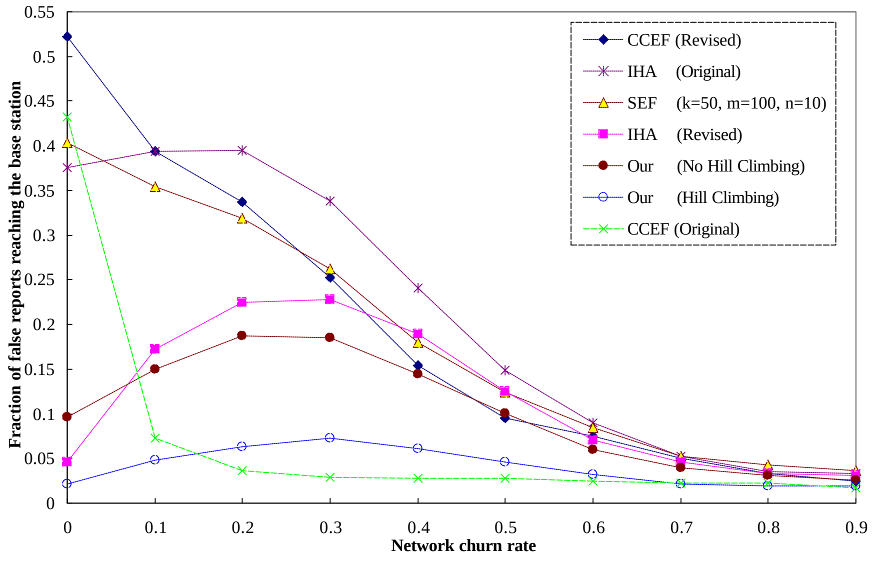


Figure 4.9 The fraction of the false reports that reach the base station as a function of the network churn rate. (In our scheme, we set $h_{max} = 10$, $mem = 50$ and $q = 2$.)

smaller than 0.4, the fraction is mainly determined by the detecting probability of nodes. The simulation results in Figure 4.9 show that our scheme can drop more fractions of false reports than others, except for the original CCEF (which drops even lots of legitimate report when lacking a witness key.). As the network churn rate becomes larger (from 0 to 0.4): (1) The fractions of our scheme and IHA go up gradually, because more false reports are forwarded by the nodes that have no corresponding auth-keys. (2) The fractions of SEF and the revised CCEF decrease quickly, because the paths become longer, which makes the false reports subject to the verification of more nodes and hence more likely to be filtered.

4.6.2.5 Filtering Capacity when Forwarding Nodes are Compromised

Besides sensing nodes, forwarding nodes may also be compromised. We assume that a forwarding node does not directly collaborate with compromised sensing nodes by sharing their keys. However, they may collaborate indirectly. That is, when a forwarding node is compromised, it never filters out the false reports. Moreover, once it detects a forged MAC within a false report, it may even replace the forged MAC by a correct MAC generated using its own key. In these simulations, we assume there are exactly $t-1$ compromised nodes including one compromised cluster head. We further differentiate two scenarios: (1) The compromised nodes are randomly distributed over the whole network. (2) They are located along the same path from the compromised cluster head to the base station.

Table 4.1 shows the comparison of filtering capacity among various schemes in these two scenarios. We set $h_{max} = 10$ and $mem = 25$ in our scheme, and do not test CCEF because it cannot filter the false reports generated by the compromised cluster head.

Table 4.1 The average number of hops traveled by false reports.

	Compromised nodes	
	in network	**along path**
Our (Hill Climbing)	1.0294	1.6976
Our (No Hill Climbing)	1.3702	2.9176
SEF	3.2918	6.664
IHA (Revised)	2.5451	7.5769
IHA (Original)	9.1472	13.773

The results in Table 4.1 demonstrates that our scheme outperforms others and can drop false reports earlier in both scenarios. The advantage of our scheme comes from two reasons: (1) Our scheme offers a higher detecting probability to nodes. (2) Our scheme allows each node to monitor its downstream nodes, which prevents the compromised forwarding nodes from replacing forged MACs.

4.7 Conclusion

In this work, we propose a dynamic en-route quarantine scheme for filtering false data injection attacks in wireless sensor networks. In our scheme, each node uses its own auth-keys to authenticate the reports, while a legitimate report should be endorsed by t nodes. The auth-keys of each node form a hash chain, and are updated in each round. The cluster head disseminates the first auth-keys of all nodes to forwarding nodes and then sends the reports followed by disclosed auth-keys. The forwarding nodes verify the authenticity of the disclosed keys by hashing the disseminated ones, and further check the integrity and validity of the reports with the disclosed keys. Then, they inform the next-hop nodes to drop or keep on forwarding the reports according to the verification results. This process is repeated at the forwarding node at every hop.

Our scheme has several advantages: (1) Compared with others, our scheme can drop false report much faster even with a smaller size of memory. (2) The nodes that are not compromised would not be impersonated because of the distinct auth-keys that they own. So, if compromised nodes could be detected, the infected clusters can be easily quarantined. (3) The *Hill Climbing* key dissemination approach greatly improves the filtering capacity of our scheme and keeps a balance of memory requirement among nodes. (4) Each node has multiple downstream nodes that possess the necessary key information and are capable of filtering false reports. This not only makes our scheme adaptive to highly dynamic networks, but also mitigates the impact of attacks in which compromised nodes selectively drop legitimate reports. (5) Each node monitors the broadcast of its downstream nodes or neighbors, which prevents the compromised nodes from contaminating the legitimate reports intentionally or generating false control messages.

However, our scheme achieves these advantages with some tradeoffs: (1) Compared with SEF, our scheme is quite complicated. It introduces extra control messages such as $K(n)$, $K(t)$

and OK, which not only increases the complexity of operations, but also incurs extra overhead, as we discussed in section 4.5.2. (2) Like any normal reports, the control messages can also be abused, e.g., they also suffer forgery and DoS attacks. (Note: We have already discussed how to prevent the abuse of control messages in section 4.5.4. For example, a forged $K(n)$ can be filtered within w hops.) (3) The introducing of extra control messages increases the delay in delivering reports. (4) Our scheme requires sensor nodes to monitor their downstream nodes or neighbors, which can be achieved by using only bidirectional links. So, sensor nodes have to discard all directed links. (5) In our scheme, each node uses the same auth-key to authenticate all of its reports of the same round. So, this auth-key can only be disclosed after each forwarding node forwards all the reports to next hop, which poses a high memory requirement to forwarding nodes due to the storing of all the reports of each round. (6) It is hard to make our scheme cooperate with other energy saving protocols, because each node has to be awake until it overhears the broadcast from its next-hop node. We leave this as our future work.

CHAPTER 5 ENHANCING INTEGRITY: Securing Network Coding against Pollution Attacks

5.1 Introduction

Network coding is a new forwarding technique which receives various applications in traditional computer networks, wireless sensor networks [62] and peer-to-peer systems [29]. It was first proposed by Ahlswede et al. [1] in order to maximize the throughput of multicast networks, in which a source intends to send its messages to multiple sinks simultaneously. Using network coding, a node (including the source and forwarders) can encode its input messages to generate an output one. This technique is different from the traditional approach which requires duplicating every input message. In 2003, Li et al. [50] further proved that linear network coding is sufficient to achieve the optimal throughput in multicast networks, which is the minimum of all max-flows from the source to every sink.

However, network coding poses new challenges for security. For example, the applications built on top of network coding are vulnerable to *pollution attacks*, in which the compromised forwarders can intentionally pollute the transmitted messages or inject the forged messages into networks. These attacks prevent the sinks from recovering the source messages correctly. A more severe problem is pollution propagation That is, even a small number of polluted messages can quickly propagate into the networks and infect a large proportion of nodes, because each polluted message can be used by all downstream nodes. Therefore, the polluted messages should be detected and filtered as early as possible.

Traditional signature approaches based on hash functions such as SHA or MD5 are not suitable for network coding, because the encoding process carried out by each forwarder can destroy the source's signatures. Recently, several novel hashing or signature schemes have been proposed to address the pollution attacks against networks coding applications. Gkantsidis and Rodriguez proposed a *homomorphic hashing scheme* [30] (called *GR's scheme*) based on Krohn's work [47], and Charles, Jain and Lauter designed a new *homomorphic signature scheme* [19] (called *CJL's scheme*. However, GR's scheme relies on extra secure channels to transmit message hashes from the source to each node, while CJL's scheme is built on top of expensive Weil pairing operations [54, 56] over elliptic curves. Ho et al. [35] proposed to use a simple polynomial hash function to detect polluted messages, and Jaggi et al. [39] discussed the optimal rate that network codes can achieve under different threat models. Unfortunately, Ho's and Jaggi's approaches can only detect or filter polluted messages at the sinks, rather than at the forwarders.

In this work, we propose an efficient signature-based scheme against pollution attacks on linear network coding systems. In our scheme, the source signs its messages using its private key, while other nodes verify the received messages using the source's public key. Our scheme utilizes a novel homomorphic signature function, which allows forwarders to compose the signatures for their output messages from those of input messages using the similar way that the output messages are composed from the input messages. Since each node appends the signatures to its output messages, its downstream nodes can verify the received messages effectively and discard the polluted or forged ones. We prove that finding a hash-collision message in our scheme is equivalent to solving a hard discrete logarithm problem. Experimental results show that our scheme is ten times faster than some existing one. In addition, we present an alternate lightweight scheme based on a much simpler linear signature function. This alternate scheme further improves computation efficiency and is more suitable for resource-constrained networks such as wireless sensor networks. However, it introduces a trade-off between efficiency and security.

Our contribution is to propose an efficient signature-based scheme for addressing pollution attacks. Our scheme allows the source to delegate its signing authority to the forwarders. That is, the forwarders can generate the signatures for their output messages without contacting the source, but they cannot create the valid signatures for polluted or forged messages. Our scheme does not need any extra secure channels, and can provide source authentication and batch verification. Most importantly, it is much more efficient than existing ones.

The rest of this chapter is organized as follows: In section 5.2, we define system model, threat model, and our goal. Then, we present our scheme in section 5.3 and provide security analysis in section 5.4. We further introduce an alternate lightweight scheme in section 5.5 and explain experimental results in section 5.6. In section 5.7, we present an example application of our schemes in wireless sensor networks. Finally, we conclude in section 5.8.

5.2 Problem Statement

5.2.1 System Model

Network coding has been used in many networking systems such as wireless sensor networks where some sensing nodes intend to send data to multiple sinks, or peer-to-peer file sharing systems where multiple users want to download a file from a server.

In this work, we consider a general multicast network as shown in Figure 5.1. It consists of a source s, multiple sinks $t_1, t_2, \cdots, t_k$ and a number of forwarders. In this network, s wants to send n source messages $M_1, \cdots, M_n$ to all the sinks, while the forwarders use linear network coding to generate their output (or encoded) messages, which are typically denoted as E. (In this work, we use the terms *output message* and *encoded message* interchangeably. They are essentially the messages generated and transmitted by the forwarders.)

We follow the same settings adopted in [19] and [30]. That is, each message is divided into m codewords each randomly picked from a finite field $\mathbb{Z}_q$, where prime q is a pre-determined security parameter. So, each source message M_i for $i = 1, \cdots, n$ can be regarded as a row vector such as

$$M_i = (m_{i,1}, m_{i,2}, \cdots, m_{i,m}) \ , \tag{5.1}$$

where $m_{i,j} \in \mathbb{Z}_q$ for $j = 1, \cdots, m$ denote the codewords. Similarly, an encoded message E can be represented as

$$E = (e_1, e_2, \cdots, e_m) \ , \tag{5.2}$$

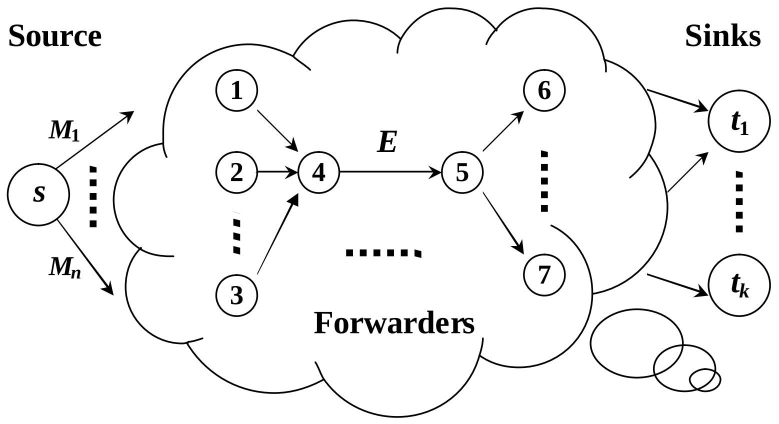


Figure 5.1 A general multicast network that adopts network coding. In this network, a singe source s simultaneously transmits n messages $M_1, \cdots, M_n$ to k sinks $t_1, \cdots, t_k$ through forwarders, which are represented by nodes 1 to 7. The encoded messages are denoted as E.

where $e_j \in \mathbb{Z}_q$ denote the codewords of E.

In linear network coding, each forwarder encodes its input messages into output message E, which is a linear combination of input messages and can be eventually regarded as a linear combination of source messages. Because of this, we can write E as

$$\begin{aligned} E &= (\alpha_1 \ \cdots \ \alpha_n) \times \begin{pmatrix} M_1 \\ \vdots \\ M_n \end{pmatrix} \bmod q \\ &= \sum_{i=1}^{n} \alpha_i M_i \bmod q \\ &= \left(\sum_{i=1}^{n} \alpha_i m_{i,1}, \cdots, \sum_{i=1}^{n} \alpha_i m_{i,m} \right) \bmod q \ , \end{aligned} \tag{5.3}$$

where $(\alpha_1 \ \cdots \ \alpha_n)$ is called encoding vector and used by the sinks to recover the source messages.

Encoding vectors can be either randomly generated as described in [34] or pre-determined based on the topology of networks. We assume that each message is appended with its encoding vector in order to facilitate the decoding at the sinks. (This approach was described in [20].)

Here, we augment each source message M_i and encoded message E respectively and obtain

$$\begin{aligned}\tilde{M}_i &= (m_{i,1}, m_{i,2}, \cdots, m_{i,m}, \underbrace{0, \cdots, 0}_{i-1}, 1, \underbrace{0, \cdots, 0}_{n-i}) \\ &= (\tilde{m}_{i,1}, \tilde{m}_{i,2}, \cdots, \tilde{m}_{i,m}, \tilde{m}_{i,m+1}, \cdots, \tilde{m}_{i,m+n}) \qquad (5.4)\end{aligned}$$

and

$$\begin{aligned}\tilde{E} &= (e_1, e_2, \cdots, e_m, \alpha_1, \cdots, \alpha_n) \\ &= (\tilde{e}_1, \tilde{e}_2, \cdots, \tilde{e}_m, \tilde{e}_{m+1}, \cdots, \tilde{e}_{m+n})\ . \qquad (5.5)\end{aligned}$$

$\tilde{M}_i$ or $\tilde{E}$ is called the augmented messages, which consists of the contents of original message and corresponding encoding vector. It is easy to verify that $\tilde{E} = \sum_{i=1}^{n} \alpha_i \tilde{M}_i \bmod q$, while each codeword of $\tilde{E}$ can be expressed as the same linear combination of the corresponding codewords of augmented source messages. (Note: In the rest of the chapter, when we mention a message (including the source message and encoded one), we always mean the augmented version of the message.)

We further assume that the source (not only the forwarders) can also encode its messages, i.e., the output messages from the source are also the encoded ones. This prevents the adversaries from recovering the source messages without collecting the sufficient number of encoded messages.

5.2.2 Threat Model

We assume that the source is always secured, but the forwarders are not trustable. The adversaries can fully control the compromised forwarders and launch the *pollution attacks* from them. In such attacks, the adversaries may intentionally pollute the output messages (augmented ones) of the compromised nodes, or directly inject the forged messages into the networks through these nodes. Formally speaking, we identify message $\tilde{E}$ as polluted or forged if and only if it looses the consistency between the contents of original one and the appended encoding vector. In that case,

$$\tilde{E} \neq (\tilde{e}_{m+1} \ \cdots \ \tilde{e}_{m+n}) \times \begin{pmatrix} \tilde{M}_1 \\ \vdots \\ \tilde{M}_n \end{pmatrix}, \tag{5.6}$$

where $(\tilde{e}_{m+1} \ \cdots \ \tilde{e}_{m+n})$ is the encoding vector embedded into $\tilde{E}$. Suffering from such attacks, the sinks may not be able to recover the source messages correctly.

More severely, the applications built on top of network coding may suffer *pollution propagation*, which means that a small number of polluted messages can easily infect a large proportion of the networks. Once a forwarder received a polluted message, all its output messages become polluted. When these polluted messages are further used by downstream nodes, more and more messages are polluted. In this way, a small number of polluted messages can quickly propagate into a large proportion of the networks.

5.2.3 Goal

In this work, we address the problem of securing network coding systems against the pollution attacks. Our goal is to design an efficient scheme for filtering pollution attacks in network coding systems, especially in those resource-constrained networks such as wireless sensor networks that adopt network coding. The scheme should enable the forwarders to detect and filter polluted messages as early as possible and must be highly efficient for the use in resource-constrained networks in terms of computation. In addition, it does not require any extra secure channels.

5.3 Our Scheme

5.3.1 The Framework of Hashing or Signature Schemes

Before introducing our scheme, we first propose a framework of the hashing or signature schemes include our scheme and existing ones [19, 30] that address the pollution attacks for linear network coding systems. In this framework, we roughly divide the schemes into three phases:

- *Parameter setup phase*: In this phase, the source chooses security parameters, calculates its private and public keys, and determines the hash or signature function.
- *Hash or signature calculation phase*: In this phase, the source calculates the hashes or signatures for its messages. These hashes or signatures may be securely transmitted to forwarders and sinks, or directly attached to the messages.
- *Message verification phase*: In this phase, the forwarders and the sinks verify the received messages. Verification is based on the encoding vectors embedded into the messages, the hashes or signatures of the messages, and the source's public keys. If verification succeeds, the received messages will be accepted and used for further encoding or decoding. Otherwise, they are discarded.

The first phase can be done offline, but the other two must be executed online. So, the efficiency of schemes is mainly determined by the computation overhead occurring in the second and third phases.

Note: when a forwarder detected some polluted messages, it can either stop generating its output messages, or only encode those unpolluted messages using new encoding vectors. However, how to process after the pollution attacks are detected is out of the scope of our work and will not be discussed here.

5.3.2 Overview and Detailed Procedures

In our scheme, the source generates the signature for each message using RSA private key and appends the signature to the corresponding message. Other nodes (including both forwarders and sinks) verify the received messages using the source's public key. Our scheme is based on a homomorphic signature function, which guarantees that any encoded message's signature can be composed from those of input messages (that are used to generate the encoded message). Thus, a forwarder can generate the valid signatures for its output messages even without knowing the source's private key, as long as every node appends the signatures to its output messages. The nice property of our scheme is that the forwarders have no ability to create the valid signature for any polluted or forged message.

The details of three phases of our scheme are as follows:

Parameter setup phase: In this phase, the source chooses the following security parameters and keys.

- Two primes p and q satisfying $q|(p-1)$. Typically, q is 257-bit long and p is 1024-bit long.

- $(m+n)$ different elements $g_1, \cdots, g_{m+n}$. Each element is randomly picked from $\mathbb{Z}_p$ and has order q.

  That is, for $j = 1, \cdots, (m+n)$, each g_j has the form as $s^{(p-1)/q} \bmod p$, where some arbitrary $s \in \mathbb{Z}_p$ and $s \neq 1$. Thus, we have

$$g_j^{tq} \bmod p = 1 \quad \text{for any integer } t \; . \tag{5.7}$$

- RSA public key (r, e) and private key d. We have $r = uv$, where u and v are two primes that are only of the half length of r. Typically, r is 1024-bit long.

  Let $\phi = (u-1)(v-1)$, then e and d are chosen such that $\gcd(e, \phi) = 1$ and $ed \equiv 1 \bmod \phi$. Thus, we have

$$t^{ed} \equiv t \bmod r \quad \text{for any integer } t \in \mathbb{Z}_r \; . \tag{5.8}$$

We assume all nodes know security parameters p, q and $g_1, \cdots, g_{m+n}$ as well as the source's public key (r, e).

Signature calculation phase: In this phase, the source calculates the signatures for its messages $\tilde{M}_1, \cdots, \tilde{M}_n$, where n is the total number of messages that the source can transmit at the optimal rate in each unit of time.

Let h denote the signature function. For each message $\tilde{M}_i$, the source calculates the signature

$$h(\tilde{M}_i) = \left(\prod_{j=1}^{m+n} g_j^{\tilde{m}_{i,j}} \bmod p \right)^d \bmod r \tag{5.9}$$

and append $h(\tilde{M}_i)$ to $\tilde{M}_i$.

Similarly, the signature of encoded message $\tilde{E}$ can be calculated as

$$h(\tilde{E}) = \left(\prod_{j=1}^{m+n} g_j^{\tilde{e}_j} \bmod p \right)^d \bmod r \ . \tag{5.10}$$

In our scheme, each forwarder is required to append $h(\tilde{E})$ to its output message $\tilde{E}$. But it does not need to calculate $h(\tilde{E})$ as shown in equation (5.10), because the signature function is homomorphic. That is, given an output message $\tilde{E} = \sum_{i=1}^{n} \alpha_i \tilde{M}_i \bmod q$, its signature can be calculated as

$$h(\tilde{E}) = \prod_{i=1}^{n} h(\tilde{M}_i)^{\alpha_i} \bmod r \ . \tag{5.11}$$

This is because

$$\begin{aligned}
h(\tilde{E}) &= h\left(\sum_{i=1}^{n} \alpha_i \tilde{M}_i \bmod q \right) \\
&= h\left(\left(\sum_{i=1}^{n} \alpha_i \tilde{m}_{i,1}, \cdots, \sum_{i=1}^{n} \alpha_i \tilde{m}_{i,m+n} \right) \bmod q \right) \\
&= \left(\prod_{j=1}^{m+n} g_j^{\sum_{i=1}^{n} \alpha_i \tilde{m}_{i,j} \bmod q} \bmod p \right)^d \bmod r \\
&= \left(\prod_{j=1}^{m+n} g_j^{\sum_{i=1}^{n} \alpha_i \tilde{m}_{i,j}} \bmod p \right)^d \bmod r \qquad (\because (5.7)) \\
&= \left(\prod_{j=1}^{m+n} \prod_{i=1}^{n} g_j^{\alpha_i \tilde{m}_{i,j}} \bmod p \right)^d \bmod r \\
&= \prod_{i=1}^{n} \left(\left(\prod_{j=1}^{m+n} g_j^{\tilde{m}_{i,j}} \bmod p \right)^d \bmod r \right)^{\alpha_i} \bmod r \\
&= \prod_{i=1}^{n} h(\tilde{M}_i)^{\alpha_i} \bmod r \ .
\end{aligned} \tag{5.12}$$

Equation (5.12) shows that the signature of an output message can be easily composed by raising the signature of each input message to the power of corresponding coefficient. So, each forwarder can generate the valid signatures for its output messages from the signatures of input ones, without knowing the source's private key. That why our scheme does not need any extra secure channels to broadcast the source messages' signatures into the whole network.

Moreover, in our scheme, the forwarders have no ability to generate the valid signatures for any polluted or forged messages (we will prove this later).

Message verification phase: In this phase, the forwarders or sinks verify the received messages based on the source's public keys and the messages' signatures. To verify message $\tilde{E}$, a forwarder (or a sink) simply checks whether

$$h(\tilde{E})^e \bmod r = \left(\prod_{j=1}^{m+n} g_j^{\tilde{e}_j} \bmod p \right) \bmod r \ , \tag{5.13}$$

which can be derived from equations(5.8) and (5.10). If equation (5.13) is satisfied, the message is accepted. Otherwise, it is assumed to be polluted.

5.3.3 Batch Verification

Our scheme supports batch verification [5], which can further reduce computation overhead and speed up message verification. For instance, suppose a forwarder receives three messages $\tilde{E}_a$, $\tilde{E}_b$ and $\tilde{E}_c$. It can randomly select three coefficients β_a, β_b and β_c and generate a new message $\tilde{E} = \beta_a \tilde{E}_a + \beta_b \tilde{E}_b + \beta_c \tilde{E}_c$. Benefiting from the homomorphic signature function, it can obtain the signature of this new message as

$$h(\tilde{E}) = h(\tilde{E}_a)^{\beta_a} \times h(\tilde{E}_b)^{\beta_b} \times h(\tilde{E}_c)^{\beta_c} \ , \tag{5.14}$$

and then verify $\tilde{E}$ using $h(\tilde{E})$ as normal.

If the new message passes verification, all three input messages are accepted. Otherwise, one or more messages must be polluted. In this case, further verification should be carried out to find the malicious one(s). The node needs to re-check each input message individually or use batch verification repeatedly on subsets of input messages. For example, we can use binary-checking that is similar to binary-searching algorithm, to speed up re-checking. Binary-checking rules out a half of input messages at each step. That is, we encode and check each half of input messages separately. If pass, that half of messages will be accepted. Otherwise, two sub-halves of the suspected half will be re-checked. This binary-checking process can be iteratively carried out until all polluted messages are found.

5.4 Security Analysis

We define a message as polluted using inequality (5.6), that is, the message's contents are inconsistent with the encoding vector transmitted along with the message. To prevent the adversaries from polluting the encoding vector, we require that each signature be calculated based on the augmented message that includes both the contents of original message and the appended encoding vector. If signature calculation does not cover the encoding vector, the adversaries can easily transmit a message (original one) along with a valid signature, but append it with an incorrect encoding vector.

The purpose of an adversary is to generate a polluted message that can pass verification. He has two choices. First, he may try to generate a valid signature for a polluted or forged message. However, this requires he obtain the source's private key. Second, he may try to find a hash-collision message $\tilde{E}'$, which is different from a valid message $\tilde{E}$, but has the same signature as that of $\tilde{E}$. That is, $\tilde{E}' \neq \tilde{E}$, but $h(\tilde{E}') = h(\tilde{E})$.

In our scheme, determining the source's private key from its public key is equivalent to solving an integer factorization problem, which is a well known hard problem and its hardness is determined by the length of r, typically 1024 bits.

The adversary may brute force search for a hash-collision message. In our scheme, each signature is a random element of finite field $\mathbb{Z}_r$, so the probability to find a hash-collision message with brute force is as low as $\frac{1}{r} \simeq \frac{1}{2^{1024}}$. However, a smart adversary may try to find a hash-collision message $\tilde{E}'$ by exploiting a valid message $\tilde{E}$ and its signature $h(\tilde{E})$. We prove that this is equivalent to solving a discrete logarithm problem.

Proposition: *Given a valid message $\tilde{E}$ along with its signature $h(\tilde{E})$, generating a hash-collision message $\tilde{E}'$ from $\tilde{E}$, where $\tilde{E}' \neq \tilde{E}$, is equivalent to solving a discrete logarithm problem.*

Proof: (Because of page limit, we only give the basic idea of proof here.)

First, we consider a special case when $(m+n) = 2$. Thus, the signature of $\tilde{E}$ is

$$h(\tilde{E}) = (g_1^{\tilde{e}_1} \times g_2^{\tilde{e}_2} \bmod p)^d \bmod r \ . \tag{5.15}$$

The purpose of the adversary is to generate a hash-collision message $\tilde{E}' = (\tilde{e}'_1, \tilde{e}'_2)$ such that

$$(g_1^{\tilde{e}'_1} \times g_2^{\tilde{e}'_2} \bmod p) \bmod r = (g_1^{\tilde{e}_1} \times g_2^{\tilde{e}_2} \bmod p) \bmod r \ , \tag{5.16}$$

where $\tilde{e}'_1 \neq \tilde{e}_1$ and $\tilde{e}'_2 \neq \tilde{e}_2$. Let us fix $\tilde{e}'_1$ and define $x = \tilde{e}'_2$. The problem becomes to determine x such that

$$\begin{aligned} & g_1^{\tilde{e}'_1} \times g_2^{x} = g_1^{\tilde{e}_1} \times g_2^{\tilde{e}_2} \\ \Longrightarrow \quad & g_2^{x} = g_1^{\tilde{e}_1 - \tilde{e}'_1} \times g_2^{\tilde{e}_2} \ . \end{aligned} \tag{5.17}$$

This is a discrete logarithm problem, that is, determining x given (g_2, g_2^x) over some group $\mathbb{Z}_p$ (or $\mathbb{Z}_r$).

Similarly, this idea can be extended to the case when $(m+n) > 2$, which we do not discuss in detail. In conclusion, we proved that finding a hash-collision message in our scheme is equivalent to solving a hard discrete logarithm problem. ■

We emphasize that a message whose contents are consistent with appended encoding vector can always pass verification and will never be considered as polluted or forged, although the message may not be as exactly as what we designed. For example, some forwarder is supposed to transmit message $\tilde{E}$, but it actually outputs another message $\tilde{E}' = \alpha\tilde{E}$ and appends a new signature $h(\tilde{E})' = h(\tilde{E})^{\alpha}$, where α is an arbitrary integer. In our scheme, although $\tilde{E}'$ is a new message, it is still consistent and will be considered as a valid one, because in this new message, the contents (i.e., codewords) of original message and coefficients of encoding vector are both multiplied by α and it would not cause any problem for decoding at the sinks.

5.5 An Alternate Lightweight Scheme

5.5.1 Detailed Procedures

In some scenarios such as resource-constrained wireless sensor networks, computation efficiency is even more important than security. To be adaptive to these special scenarios, we propose an alternate lightweight scheme based on a simpler linear signature function. Compared with our homomorphic signature scheme, this alternate scheme improves signature calculation efficiency up to a hundred times. However, it has to sacrifice desired security level to such a extent that, adversaries may be able to derive the source's private keys or find some hash-collision message if they cab capture sufficient number (i.e., (m+n) or log$q$) of linearly independent messages.

The details of this alternate scheme are as follows:

Parameter setup phase: In this phase, the source chooses

- two primes p and q such that $(q|p-1)$. Typically, q is 257-bit long and p 1024-bit.
- an order-q element g randomly picked from $\mathbb{Z}_p$, where $g^{tq} \bmod p \equiv 1$ for any integer t.
- $(m+n)$ distinct private keys $x_1, x_2 \cdots, x_{m+n}$, where $x_j \in \mathbb{Z}_q$ for $j = 1, \cdots, (m+n)$.
- and corresponding public keys $y_1, y_2 \cdots, y_{m+n}$, where $y_j = g^{x_j} \bmod p$ for $j = 1, \cdots, (m+n)$.

We assume all nodes know the security parameters p, q and g as well as the source's public keys.

Signature calculation phase: In this phase, the source calculates the signatures for its messages $\tilde{M}_1, \cdots, \tilde{M}_n$, and appends the signatures to corresponding messages.

Let h denote the signature function. For each message $\tilde{M}_i$ where $i = 1, \cdots, n$, the source calculates its signature

$$h(\tilde{M}_i) = \sum_{j=1}^{m+n} x_j \tilde{m}_{i,j} \bmod q \ . \tag{5.18}$$

Similarly, the signature of encoded message $\tilde{E}$ is

$$h(\tilde{E}) = \sum_{j=1}^{m+n} x_j \tilde{e}_j \bmod q \ . \tag{5.19}$$

However, a forwarder does not need to calculate the signature for its output message $\tilde{E}$ this way. Since the signature function is linear, an encoded message's signature can be composed from those of input messages with the same linear relationship that the encoded message itself is composed from those input ones. That is, given $\tilde{E} = \sum_{i=1}^{n} \alpha_i \tilde{M}_i \bmod q$, we have

$$h(\tilde{E}) = \sum_{i=1}^{n} \alpha_i h(\tilde{M}_i) \bmod q \ . \tag{5.20}$$

Thus, a forwarder can generate valid signatures for its output messages by simply encoding those of its input ones. So, this scheme does not require any extra secure channel.

Message verification phase: In this phase, the forwarders or sinks verify received messages based on the source's public keys and the messages' signatures. Assume a forwarder receives message $\tilde{E}$ along with its signature $h(\tilde{E})$. It simply checks whether

$$g^{h(\tilde{E})} \bmod p = \prod_{j=1}^{m+n} y_j^{\tilde{e}_j} \bmod p \ . \tag{5.21}$$

This verification should work, because for a valid message $\tilde{E}$,

$$\begin{aligned} g^{h(\tilde{E})} \bmod p &= g^{\sum_{j=1}^{m+n} x_j \tilde{e}_j \bmod q} \bmod p \\ &= \prod_{j=1}^{m+n} y_j^{\tilde{e}_j} \bmod p \ . \end{aligned} \tag{5.22}$$

If equation (5.21) is satisfied, the message will be accepted. Otherwise, it is discarded.

5.5.2 Security Analysis

In this scheme, calculating the source's private key x_j from corresponding public key g^{x_j} is equivalent to solving a discrete logarithm problem [70]. However, an adversary may try to learn the private keys from transmitted messages, because each message $\tilde{E}$ represents a linear equation with $(m+n)$ unknown private keys $x_1, x_2, \cdots, x_{m+n}$. That is,

$$h(\tilde{E}) = (x_1\tilde{e}_1 + x_2\tilde{e}_2 + \cdots + x_{m+n}\tilde{e}_{m+n}) \bmod q \ . \tag{5.23}$$

If the adversary has collected $(m+n)$ linearly independent messages, it can derive the private keys by solving the corresponding linear equations. Thus, this scheme poses a upper bound of the number of messages that the source can transmit.

The probability that the adversary brute force a hash-collision message is $\frac{1}{q} \simeq \frac{1}{2^{256}}$, since each signature is a random element picked from finite field $\mathbb{Z}_q$. However, a smart adversary may exploit the linear property of our signature function to find a hash-collision message. In this scheme, each message can be regarded as a vector of length $(m+n)$ and each signature can be represented by a vector of $\log q$ bits, where typically $\log q = 256$. Hence, the linear signature function establishes a map from $(m+n)$-dimension message space to $\log q$-dimension signature spaces. Ideally, $\log q$ linearly independent messages will be mapped to the same number linearly independent signatures without collision. However, if the adversary picks one more message which is linearly independent of all previous ones, this new message will be mapped to a signature which is linearly dependent on previous signatures, since we already have $\log q$ linearly independent signatures and any new signature must be linearly dependent on these $\log q$ ones. Thus, the adversary successfully finds out two messages mapped to the same signature. To prevent this attack, the source should generate no more than $\log q = 256$ linearly independent messages. If the source wants to send more messages, it has to either choose a larger q or update its private keys after producing $\log q - 1$ linearly independent messages. Certainly, the second method will increase communication overhead.

5.6 Experimental Results

5.6.1 Experiment Setup

We compare CJL's scheme [19], GR's scheme [30] and our schemes in experiments. (We do not study Ho's scheme [35], since it can only detect polluted messages at sinks, instead of filtering them at forwarders.) The implementation of these schemes is built on top of software package MIRACL [57] and tested on a Pentium-4 3.00 GHz Linux machine.

For CJL's scheme, we choose a supersingular elliptic curve

$$E(\mathbb{F}_p):\ y^2 = x^3 + 1 \tag{5.24}$$

with $p \equiv 2 \mod 3$ and $p \equiv 3 \mod 4$, where p is a 1024-bit prime. We test this scheme based on the Tate pairing with q-torsion points, where q is a 160-bit prime. (We select Tate pairing instead of Weil pairing [41], because it is more computationally efficient for cryptographic operations, as shown in [28].) For GR's scheme and our schemes, we set p a 1024-bit long prime and q a 256-bit long prime. Typically, we choose the total number of messages $n = 128$ and each message containing $m = 256$ codewords. In our experiments, we always let $n = m/2$.

5.6.2 Computation Overhead

We study computation overhead of these schemes in term of running time of each phase with $m = 256$ and $n = 128$. Table 5.1 shows that, CJL's scheme is the slowest and the alternate scheme is the fastest, especially in terms of signature calculation. Our scheme has similar performance in computation efficiency to GR's scheme.

Table 5.1 Computation overhead of different schemes on Pentium-4 computer (m: 256-bit and n: 128-bit)

	CJL's Scheme	**GR's Scheme**	**Our Scheme**	**Alternate Scheme**
Parameter Setup (n *messages*)	10.65 s	2.85 s	2.87 s	1.55 s
Sig/Hash Calculation (*per message*)	5.37 s	0.96 s	1.42 s	0.01 s
Message Verification (*per message*)	16.54 s	1.43 s	1.44 s	1.43 s

In parameter setup phase, CJL's scheme must generate $(m+n+1)$ q-torsion points. GR's scheme should choose m order-q elements. Our scheme has to select $m+n$ order-q elements and a RSA private key, while the alternate scheme needs to generate $(m+n)$ pairs of private key and corresponding public key. Table 5.1 shows that choosing q-torsion is the most time-consuming, which takes $4\times$ the time for choosing $(m+n)$ order-q elements for our scheme and $7\times$ the time for choosing $(m+n)$ private and public keys for the alternate scheme.

To calculate the signature for a message, our scheme (or GR's scheme) spends $1.42s$ (or $0.96s$) on $(m+n)$ (or m) modular exponentiations, while the alternate scheme spends only $0.01s$ on $(m+n)$ linear operations. Signature calculation of CJL's scheme is based on $(m+n)$ q-torsion points and extremely time-consuming, which takes $5.37s$. (Note: Time values of signature or hash calculations shown in the table are based only on one message. To send n messages, the source should spend much more time to calculate signatures or hashes, which may cause a large transmission delay at the source.)

The main task of forwarders is to verify messages. Verification should be done as fast as possible. Otherwise, it becomes the bottleneck of whole network and prevents the source from sending messages at the optimal rate. Hence, verification speed is the most important metric for evaluating performance of schemes. Table 5.1 shows that our scheme and the alternate one have similar verification efficiency to GR's scheme and are much faster than CJL's scheme. In addition, Figure 5.2 depicts how verification overhead of different schemes increases linearly as m grows from 16 to 1024. Clearly, our scheme, the alternate one and GR's scheme are much faster than CJL's scheme in message verification.

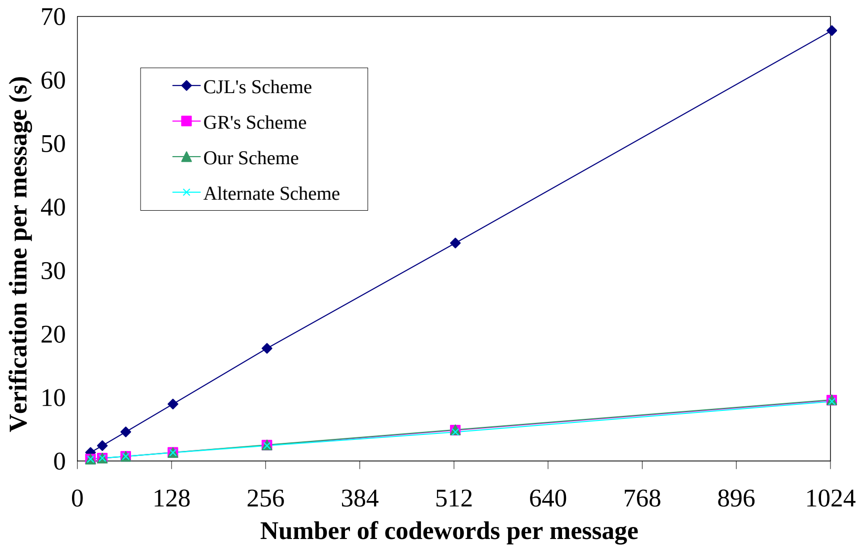


Figure 5.2 Comparison of computation efficiency among different schemes in terms of verification time (per message). It shows that GR's scheme, our scheme and alternate scheme preform similarly in message verification.

From equations (5.13) and (5.21), we can see that our scheme and the alternate one require $(1 + m + n)$ modular exponentiations on the signature of received message, m codewords and n encoding vector elements. GR's scheme needs $(m + n)$ ones, where m operations are used for calculating the hash of received message and n ones for the hashes of source messages. Compared to other operations such as modular additions and modular reductions, modular exponentiations dominant the message verification phase. So, the ratio of verification overhead of our scheme (or the alternate one) over that of GR's scheme is $\frac{1+m+n}{m+n} \simeq 1$. That is why Figure 5.2 shows these schemes have almost the same performance on verification efficiency. Although GR's scheme has comparable efficiency to our scheme in terms of verification, it requires an extra secure channel. In addition, our scheme can provide source authentication, since it is based on a signature function, instead of a hash function as in GR's scheme.

The verification process of CJL's scheme is similar to that of our scheme and the alternate one. The only difference is that CJL's scheme is based on pairing operations, while our scheme and the alternate one are based on modular exponentiations. Hence, the difference of verification efficiency between our scheme (or the alternate one) and CJL's scheme is mainly determined by efficiency of pairing operations and modular exponentiations. Figure 5.2 shows that CJL's scheme is *ten times slower* in message verification than our scheme (or the alternate one), because the pairing computation is extremely time-consuming. So, our scheme is much more efficient than CJL's scheme, although they both base their security on the discrete logarithm problem.

5.7 Application to Wireless Sensor Networks

Wireless sensor networks consist of a number of resource-constrained nodes with limited power resource, memory space, computation and communication capacity. Maximizing network throughput with network coding is very important for wireless sensor networks, because it can reduce communication overhead and hence save energy for sensor node. Moreover, since sensor nodes are prone to failure, applying network coding technique in wireless sensor networks can make wireless communications between sensor nodes more robust by reducing the need of frequent retransmission (as long as sinks can receive sufficient number of messages for decoding).

Let us consider an example data-centric storage application [65] for wireless sensor networks. In this application, sensing data are organized by keys, and each sensing node wants to store their data into multiple storage nodes responsible for some particular key. In this scenario, network coding technique can be used for maximizing data rate of sensing nodes and providing robust communications. However, sensor nodes can be easily compromised and adversaries could launch pollution attacks from these compromised nodes. Hence, we need an efficient and effective scheme for addressing such pollution attacks. In wireless sensor networks, there is no secure channel between an arbitrary sensing node and storage nodes. (Someone may claim to use a trusted based station to forward secure information between the sensing node and storage nodes. However, it incurs high communication overhead and in some cases the trusted base station may even not exist when needed.) Thus, GR's scheme is not applicable in this scenario. Since wireless sensor nodes are extremely constrained in terms of computation capacity and power resource, computation efficiency is the main consideration for choosing a proper scheme to address pollution attacks. Compared with CJL's scheme, our scheme and the alternate one take much less time in signature calculation and message verification, so they significantly reduces energy consumption in computation and hence is more suitable for wireless sensor networks.

We have implemented GR's scheme, our scheme and the alternate one on MicaZ mote. (The implementation of CJL's scheme on MicaZ mote is still in progress.) MicaZ mote is only equipped with an 8-bit microprocessor ATmega128, 4K bytes memory (RAM) and 128K bytes program flash memory (ROM). Since it is extremely resource-constrained, we have to relax our security requirement by choosing some smaller security parameters. In our implementation, we set p as 256-bit prime, q as 128-bit prime, the number of codewords m=16 and the number of source messages n=8. Our implementation is based on the software package provided by Wang and Li [76] and experimental results are shown in Table 5.2. From the table, we can see that all schemes have to spend around 150s to verify one message, which implies that modular exponentiation is still time-consuming for wireless sensor nodes. Table 5.2 also shows that the alternate scheme is much more efficient in signature calculation than other two, where it takes only 0.12s to generate a signature, compared to almost 100s for GR's scheme and around 147s for our scheme. If we emphasize verification speed, we should choose the alternate scheme. If we value high security as well as efficiency, our scheme is the best, instead of GR's scheme.

(Note: Although our scheme and the alternate one still need long time for message verification, we hope technical advance in electronics and better software implementation would make them more practical for resource-constraint networks. We are glad to see that more and more researches [32, 72, 76] are being conducted on efficient implementation of public key

Table 5.2 Computation overhead of different schemes on wireless sensor nodes (p: 512-bit, q: 128-bit, m=16 and n=8)

	GR's Scheme	**Our Scheme**	**Alternate Scheme**
Parameter Setup (n *messages*)	1.56 s	1.61 s	1.39 s
Sig/Hash Calculation (*per message*)	99.79 s	147.36 s	0.12 s
Message Verification (*per message*)	149.70 s	155.14 s	151.28 s

crypto-systems on sensor nodes.)

5.8 Conclusion

In this work, we proposed an efficient signature-based scheme against pollution attacks for securing linear network coding. Our scheme utilizes a novel homomorphic signature function and allows a source to delegate its signing authority to forwarders, which means that the forwarders can generate the signatures for their output messages without contacting the source. This property allows the forwarders to verify received messages, but prevent them from creating the valid signatures for polluted or forged ones. Thus, the pollution attacks can be efficiently and effectively filtered out at the forwarders. Our scheme does not need any extra secure channels, and can support source authentication and batch verification. Experimental results show that our scheme can improve verification efficiency up to 10 times compared to some existing one. In addition, we presented an alternate lightweight scheme which utilizes a simpler signature function. This scheme is much faster and more suitable for resource-constrained networks such as wireless sensor networks. However, it introduces a trade-off between computation efficiency and security.

In this work, we assume that the source is always benign, but only the forwarders can be compromised. In future, we will study how to detect and filter forged messages injected by adversaries via the compromised sources. In addition, we will implement CJL's pairing-based signature scheme on sensor nodes and conduct experimental evaluation.

CHAPTER 6 ENHANCING INTEGRITY: Securing XOR Network Coding against Pollution Attacks

6.1 Introduction

Unlike the traditional message forwarding approaches that always duplicate the forwarding messages, network coding [1, 50] allows forwarders to combine multiple input messages into one or more output (or encoded) ones. This technique is promising to maximize network throughput and to reduce the number of retransmissions in both wired networks [20, 29] and wireless ones [44, 62]. In these applications, network coding is normally operated over large finite fields, so we term it *normal network coding*. Recently, a special network coding based only on XOR operations (i.e., over a field of size 2), has gained an increasing number of applications [45, 81, 92], especially in wireless networks, due to its simplicity. We call this special network coding *XOR network coding*, which is the focus of our research.

Both normal and XOR network coding systems are vulnerable to *pollution attacks*. In such attacks, adversaries inject polluted messages into the systems via the compromised forwarders. These attacks not only prevent the sinks from recovering the source messages, but also drain out the energy of the forwarders. Clearly, they are a big threat to resource-constrained wireless networks such as wireless sensor networks. Therefore, it is crucial to filter the polluted messages in network coding systems as early as possible.

So far, a number of schemes [19, 30, 35, 39, 47, 91, 93] have been proposed for addressing pollution attacks against network coding systems. These schemes can be categorized into two classes: (1) filtering the polluted messages only at the sinks, such as [35, 39]; and (2) filtering the polluted messages at the forwarders (including the sinks), such as[19, 30, 47, 91]. However, these schemes all base their security on the size of underlying fields, so none of them could be used to secure XOR network coding systems.

In this work, we propose the first scheme (to the best of our knowledge) for securing XOR network coding systems against pollution attacks. Our scheme allows the polluted messages to be filtered at the forwarders, and it works not only for XOR network coding, but also for for normal network coding.

Our scheme exploits probabilistic key pre-distribution and message authentication codes (MACs). In our scheme, the source generates multiple MACs for each message using its secret keys, where each MAC can authenticate only a part of the message and the parts authenticated by different MACs are overlapped. Every encoded message is attached with the MACs of the source messages from which it is constructed. Therefore, multiple downstream forwarders can collaboratively verify different parts of the encoded message using the MACs and their own shared keys. By carefully controlling the overlapping between the parts authenticate by different MACs, our scheme can filter polluted messages in a few hops with a high probability. Experimental results show that it is 200 to 1000 times faster than existing ones, hence, it is particularly suitable for resource-constrained wireless networks.

The rest of chapter is organized as follows: In section 6.2, we discuss system model and threat model, and define the problem. Then, we explain in section 6.3 the symbols we use in the work. We propose our scheme in section 6.4 and analyze its performance in section 6.5. We further explain experimental results in section 6.6. Finally, we conclude in section 6.7.

6.2 Problem Statement

6.2.1 System Model

In this work, we consider a general multicast network in which there are one source, multiple sinks and a number of forwarders. The source sends its messages to all of the sinks at the optimal rate that the network can support, while the forwarders use XOR network coding technique to generate and forward the output (or encoded) messages. (Note: we use terms *output message* and *encoded message* interchangeably in this work.)

Let s denote the source, and $M_1, \cdots, M_n$ denote the source messages, where n is the number of messages that s can transmit per unit of time in its optimal rate. Here, we assume that the source can generate messages continuously. That is, in every unit of time, s generates n messages and the forwarders transmit them using the same network code. We claim that our model is more generalized compared with some commonly used file-distribution models, which consider distributing a single file mainly. In those models, the security parameters are calculated from the content of the distributed file. So, once a new file is to be downloaded, the source has to re-broadcast the security parameters. We believe this design is very cumbersome so we consider a more generalized model.

We denote the encoded messages as E. In XOR coding, an encoded message can be represented as

$$E = \alpha_1 M_1 \oplus \alpha_2 M_2 \oplus \cdots \oplus \alpha_n M_n \ , \qquad (6.1)$$

where $\alpha_i \in \{0, 1\}$ for $i = 1, \cdots, n$. The bit string $(\alpha_1 \ \cdots \ \alpha_n)$ is called the encoding vector of E. For example, if $\alpha_1 = \alpha_2 = 1$ and other coefficients are 0, we have $E = M_1 \oplus M_2$. We assume a randomized network code, which generates encoding vectors randomly and transmits them along with the corresponding encoded messages. So, the forwarders (and the sinks) can use the encoding vectors to verify the received messages.

We adopt the model used in [30] and divide each message into m codewords of the same length. Typically, each codeword is 256-bit long. Most of existing schemes regard each codeword as a random element over a finite field of size q and encode the messages over the same field, so in those scheme q is a 256-bit prime. However, our scheme partitions codewords only for constructing message authentication codes (MACs), so the field used for partitioning the codewords is different from that over which the codewords are operated. For XOR coding, our scheme encodes the codewords over a field of size 2, although it still divides the codewords into 256-bit long.

From the perspective of codewords, each source message M_i for $i = 1, \cdots, n$ can be expressed as a row vector

$$M_i = (m_{i,1}, m_{i,2}, \cdots, m_{i,m}) \ , \qquad (6.2)$$

where $m_{i,j}$ for $j = 1, \cdots, m$ denote codewords. Similarly, an encoded message E can also be regarded as

$$E = (e_1, e_2, \cdots, e_m) \ , \tag{6.3}$$

where e_j are the codewords for $j = 1, \cdots, m$.

We further assume that all of the nodes have been assigned some random secret keys using the probabilistic key pre-distribution schemes such as [25, 90]. In particular, we assume that each node picks a fixed number of keys randomly from a large global key pool. By carefully controlling the key pool size and the number of keys that each node picks, we assure that any two nodes have certain probability to find some shared keys. The source uses its keys to generate message authentication codes (MACs) for its messages, while the forwarders verify the MACs of received messages using their shared keys with the source.

6.2.2 Threat Model and Goal

We assume that the source is always trusted, but the forwarders can be compromised. The adversaries can fully control the compromised forwarders and launch *pollution attacks*. In such attacks, they may either pollute the output messages of the compromised nodes, or inject the forged messages into systems. Formally speaking, we identify that an encoded message E has been polluted or forged, if and only if its content is not consistent with its encoding vector, that is,

$$E \neq \alpha_1 M_1 \oplus \alpha_2 M_2 \oplus \cdots \oplus \alpha_n M_n \ . \tag{6.4}$$

The pollution attacks not only prevent the sinks from recovering the source messages, but also drain out the limited energy of the forwarders, especially in resource-constrained wireless networks.

More severely, network coding systems (including XOR and normal coding) suffer from *pollution propagation*, i.e., a small number of polluted messages can quickly propagate in the systems and infect a large proportion of nodes and their messages. When a forwarder receives a polluted message, all of its output messages will be polluted. Then, these polluted messages are further used by downstream forwarders for encoding, thus, more and more messages will be polluted. So, it is necessary to filter the polluted messages as early as possible.

Our goal is to design an efficient scheme that can filter pollution attacks for the systems adopting XOR network coding. We are particularly interested in the resource-constrained wireless networks such as wireless sensor networks, which can be greatly benefited from the use of XOR network coding. We expect that our scheme can make the forwarders to detect and filter the polluted messages as early as possible, while it is still highly efficient in terms of computation overhead. In addition, the scheme should not rely on any extra secure channels.

6.3 Notation

In Table 6.1, we explains the symbols used in the paper.

6.4 Our Scheme

6.4.1 The Framework for Securing Network Coding against Pollution Attacks

Before introducing our scheme, we first propose a framework that generalizes all of the schemes for securing network coding systems against pollution attacks. Within this framework, we roughly divide these schemes into three phases:

- *Parameter setup phase*: The source determines security parameters, chooses its keys including secret keys or public and private keys, and selects its hash or signature function.
- *MAC (hash or signature) calculation phase*: The source calculates the authentication information such as the hashes, MACs or signatures of its messages. This information is either securely transmitted to the forwarders and sinks, or directly attached to the original messages.
- *Message verification phase*: The forwarders and sinks verify received messages. Verification is based on encoding vectors, authentication information, shared secret keys or the source's public keys. If verification succeeds, the received messages are accepted and will be used for further encoding or decoding. Otherwise, they are discarded.

Table 6.1 Notation

Symbol	**Explanation**
$M_i, m_{i,j}$	i-th source message and its j-th codeword
E, e_j	encoded message and its j-th codeword
n	the number of source messages transmitted
m	the number of codewords of each message
t	the number of random keys each node has
u	the number of codewords hashed in each MAC
$w_{i,j}$	message M_i's hash embedded in its j-th MAC
$\mathcal{K}, \|\mathcal{K}\|$	global key pool and its size
$k_{s,i}$	i-th key of the source
$id(k_{s,i})$	$k_{s,i}$'s index in key pool
$\{x\}_{k_{s,i}}$	encrypting x with random key $k_{s,i}$
r_i	random seed used to generate hash chain for i-th MAC
$r_{i,j}$	j-th element of hash chain computed from r_i

The first phase can be done offline, but the other two must be executed online. Hence, the second and third phases mainly determine the efficiency of schemes.

Note: Once a forwarder detects a polluted message, it may either encode other unpolluted messages by selecting a new encoding vector, or ask its upstream node to send the message again, because the pollution may be due to transmission error. Of course, the number of retransmissions should be pre-defined. We do not discuss this issue here, because it is out of the scope of our work.

6.4.2 The Detailed Procedure of Our Scheme

We assume that each node can randomly pick up a number of secret keys from a global key pool, utilizing some probabilistic key pre-distribution approaches such as [25, 90]. Thus, any two nodes have certain probability to share a common secret key. The source generates the same number of MACs for each message using its random keys. Each MAC is calculated based on some codewords randomly selected from the message, hence, it can authenticate those codewords of the message. In this way, each forwarder sharing some secret key(s) with the source can verify the corresponding codewords of an input message by checking the MACs using the shared key(s).

However, this shared-key based verification has a vulnerability. That is, a compromised forwarder who has a shared key is aware that which codewords have been used to generate an MAC. Then, it can pollute the corresponding codewords of messages without being detected, although it is unable to pollute the codewords authenticated by other MACs for which it has no shared keys. To address this vulnerability, we choose to overlap the codewords authenticated by any two MACs for the same message. By carefully controlling the overlapping ratio, we assure that a polluted message can be detected within certain hops with a high probability. We describe the detailed procedure of each phase of our scheme in the rest of this section.

Parameter setup phase: In this phase, the source first chooses the following security parameters, functions and secret keys:

- Two parameters t and u, where t is the number of MACs attached to each source message, and u is the number of codewords used to generate a MAC. These two parameters are public.
- t random integers $r_1, \cdots, r_t$, where each $r_j \in [1, m]$ for $j = 1, \cdots, t$. Each integer will be embedded into an MAC for identifying the indexes of codewords based on which the MAC is generated.
- A pseudo-random permutation function $f : [1, m] \rightarrow [1, m]$, where f is public and any node can compute a hash chain from a given seed r_j using this function.
- A hash function $h : \mathbb{Z}_q^u \rightarrow \mathbb{Z}_q$, where $\mathbb{Z}_q$ constrains the range of codewords and h is public. Using h any node can generate a hash from u codewords, where the length of the hash is the same as that of codewords.
- t random keys $k_{s,1}, \cdots, k_{s,t}$ from a global key pool $\mathcal{K}$, where s is the index of the source. The index of each key $k_{s,i}$ in the key pool for $i = 1, \cdots, t$ is denoted as $id(k_{s,i})$.

Note: We suppose that each node picks t random keys from $\mathcal{K}$. The keys of node j are denoted as $k_{j,1}, \cdots, k_{j,t}$.

MAC calculation phase: In this phase, the source attaches t MACs to each message M_i for $i = 1, \cdots, n$, where n is the total number of source messages. Each MAC is calculated by encrypting the hash of u randomly selected codewords using a random key. For XOR network coding, a hash is simply an XOR of the selected codewords, whereas for normal network coding, the hash is a random linear combination of the selected codewords.

More precisely, message M_i is attached with t MACs $\mathrm{MAC}_{i,1}, \cdots, \mathrm{MAC}_{i,t}$ as well as the corresponding indexes of the random keys that are used to generate MACs. Thus, in our scheme, the source actually generates and transmits

$$M_i, id(k_{s,1}),\ \mathrm{MAC}_{i,1}, \cdots, id(k_{s,t}),\ \mathrm{MAC}_{i,t}\ . \tag{6.5}$$

For $j = 1, \cdots, t$, we define

$$\mathrm{MAC}_{i,j} = \{id(k_{s,j}), r_j, h_{i,j}\}_{k_{s,j}}\ , \tag{6.6}$$

where $\{\cdot\}_{k_{s,j}}$ denotes encryption using key $k_{s,j}$, and $h_{i,j}$ is the hash of u randomly selected codewords of message M_i. The indexes of these codewords are determined by a hash chain that is computed from a seed r_j using function f. For $v = 1, \cdots, u$, let $r_{j,v}$ denote each element of the hash chain. Then, we have

$$r_{j,v} = f(r_{j,v-1})\ , \tag{6.7}$$

where $r_{j,0} = r_j$. Here, $r_{j,1}, \cdots, r_{j,u}$ are the indexes of selected codewords.

Once u codewords are selected, the source can generate the hash from these codewords using function h. For XOR network coding, the hash is

$$h_{i,j} = m_{i,r_{j,1}} \oplus \cdots \oplus m_{i,r_{j,u}}\ . \tag{6.8}$$

Note: we also take the consideration of normal network coding. In this case, the hash becomes

$$\begin{aligned} h_{i,j} &= \beta_1 m_{i,r_{j,1}} + \cdots + \beta_u m_{i,r_{j,u}} \bmod q \\ &= \sum_{v=1}^{u} \beta_v m_{i,r_{j,v}} \bmod q\ , \end{aligned} \tag{6.9}$$

where the coefficients $\beta_v \in \mathbb{Z}_q$ for $v = 1, \cdots, u$ are randomly generated to combine these codewords. A simple way to generate these coefficients is to let them form a hash chain, which is generated from the seed r_j using another pseudo-random permutation function f'. (We do not discuss f' here.)

Finally, each source message is attached with t MACs, and each MAC is computed from u codewords. Or equivalently, each MAC authenticates u codewords of a message. We emphasize that the codewords authenticated by different MACs may overlap, that is, the same codeword may be used to generate different MACs. Averagely, each codeword is authenticated by $\frac{t \times u}{m}$ MACs.

In our scheme, when each forwarder generates its output message, it always attaches the MACs of all source messages from which this output message is produced. For example, when a forwarder generates $E = M_i \oplus M_j$, it will attach $\text{MAC}_{i,1}, \cdots, \text{MAC}_{i,t}$ and $\text{MAC}_{j,1}, \cdots, \text{MAC}_{j,t}$ to its output message E. We observe that the source generates the MACs for different messages using the same set of random keys, so the indexes of keys such as $id(k_{s,j})$ in equation (6.6) do not need to be transmitted multiple times.

Message verification phase: In this phase, each forwarder or sink verifies its input messages based on the MACs for which it has the shared key(s) with the source. When receiving a message along with the MACs of all source messages from which this message is encoded, the node processes as follows:

1. It first checks the indexes prefixed to each MAC to see if it has any shared key with the source.

2. Once finding a shared key, it decrypts the corresponding MACs of source messages and generates the indexes of u codewords from the seed embedded into the MACs.

3. For normal network coding, it also needs to generate the coefficients used to combine the codewords.

4. After identifying the indexes of codewords, it takes the corresponding codewords out of the received message and calculates the hash of these codewords following equation (6.8) for XOR coding (or equation (6.9) for normal linear coding).

5. It further takes out the hashes embedded into the decrypted MACs of source messages and encodes them using the encoding vector transmitted along with the received message.

6. Finally, it checks if the hash of the received message (calculated in step 4) equals the combination of the hashes embedded in corresponding MACs (obtained in step 5). If equals, the verification succeeds. Otherwise, the received message is assumed to be polluted and will be discarded.

For example, if a node receives $E = M_i \oplus M_j$, and finds that it can decrypt $\text{MAC}_{i,l}$ and $\text{MAC}_{j,l}$ of messages M_i and M_i. From the decrypted MACs, it further knows that the MACs are calculated from the codewords of indexes x, y and z, then it checks if

$$e_x \oplus e_y \oplus e_z = h_{i,l} \oplus h_{j,l} \ , \tag{6.10}$$

where e_x, e_y and e_z are the corresponding codewords of message E, and $h_{i,l}$ and $h_{j,l}$ are the hashes encrypted in $\text{MAC}_{i,l}$ and $\text{MAC}_{j,l}$. When equation (6.10) is satisfied, the node accepts message E. Otherwise, it discards E.

6.4.3 Batch Verification

Our scheme supports batch verification, which can further reduce computation overhead and speed up message verification. Suppose a node receives three messages E_a, E_b and E_c. For XOR network coding, it generates a new message $E = E_a \oplus E_b \oplus E_c$. For normal network coding, it first chooses three random coefficients γ_a, γ_b, and γ_c, then, generates a new message $E = \gamma_a E_a + \gamma_b E_b + \gamma_c E_c$. The node further calculates E's encoding vector as an XOR or linear combination (with coefficients γ_a, γ_b, and γ_c) of those of messages E_a, E_b and E_c. And it also attaches to E all unique MACs appended to E_a, E_b and E_c. Finally, it verifies E as normal.

If the new message passes verification, all the input messages are accepted. Otherwise, one or more messages must have been polluted. In this case, further verification should be carried out to find the malicious one(s). The node needs to re-check each input message individually or use batch verification repeatedly on the subsets of input messages. For example, we can speed up re-checking by using binary-checking, that is similar to binary-search algorithm. Binary-checking rules out a half of input messages at each step. That is, we encode and check each half of input messages separately. If pass, that half of messages will be accepted. Otherwise, two sub-halves of the suspected half will be re-checked. This binary-checking process can be iteratively carried out until all polluted messages are found.

6.5 Performance Analysis

6.5.1 Threat Analysis

A polluted message defined in inequality (6.4) is one whose contents are not consistent with its encoding vector. It can be generated by an adversary in different ways. We analyze these ways and discuss possible countermeasures as follows:

- If the adversary has no shared keys with the source, it may randomly pollute a message (or the MACs attached to the message). This pollution can be easily detected, since the adversary does not know how to generate valid MACs without any shared keys.

- If the adversary has one shared key, it knows what codewords are authenticated by the corresponding MAC. Then, it may pollute those codewords of a message and generate a false MAC matching the polluted codewords. For XOR coding, a smarter adversary can even simply pollute a message by exchanging the positions of two codewords without the need of generating a false MAC. Since our scheme allows one codeword to be authenticated by multiple MACs, this pollution can be detected from another unpolluted MAC that happens to authenticate only one exchanged (or polluted) codeword. One exception is that if the unpolluted MAC happens to authenticate both (exchanged) codewords, it cannot detect the pollution. But the possibility of this exception can be reduced by carefully controlling the codewords in each MAC. In addition, this exception does not exist in the case of normal network coding, because the same codewords are combined with different coefficients in different MACs.

- An adversary may try to replace all the MACs of a polluted message with those MACs generated using its own keys. However, this pollution can be detected by comparing the key indexes contained in the polluted message with those in a unpolluted message, because all messages should contain the same key indexes.

- A more severe attack is collusion. Colluded adversaries have more knowledge of the shared keys. Once they find all of the source keys, no polluted message can be detected. The only way for addressing collusion attacks is to increase the number of MACs attached to each message. Following the assumption adopted in [90], we assume that there exists a upper bound of the number of source keys known by colluding nodes and it is never greater than the total number of source keys.

6.5.2 Security Analysis

In the section, we study the performance of our scheme in terms of the probability that a polluted message can be detected and the average number of hops that the polluted message can travel. For convenience of analysis, we simply assume that each message is attached with only t MACs. This makes sense, because all the MACs attached to a message are encrypted using only t keys, irrespective of how many MACs that the message really has. From the perspective of encryption/decryption, all MACs encrypted with the same key are actually as one MAC.

To simplify our analysis, we further assume that the adversary generates a polluted message by exchanging only the positions of exactly two codewords of a normal encoded message. We call those two codewords *polluted codewords*. More specifically, given a normal message with t MACs, the adversary first checks out how many shared keys he has and identifies the corresponding MACs that he can decrypt. We call these MACs *revealed MACs* and others *unrevealed MACs*. Then, the adversary chooses two codewords in such a way that no revealed MACs need to be modified, when the positions of these two codewords are exchanged. For example, he can choose two codewords that both occur in some revealed MACs, but neither of them occurs in other revealed ones. In a word, to analyze the security performance of our scheme, we consider such an extreme case that the adversary makes the least modification to the normal message in order to generate the polluted one.

In the extreme case, the polluted message can only be detected by using the unrevealed MACs. In fact, only those unrevealed MACs that contain exactly one polluted codeword are useful for detection. We call those useful unrevealed MACs *effective MACs*. Our purpose is to calculate the number of hops that the polluted message can travel based on the number of effective MACs it has and the probability that it has that number of effective MACs.

Suppose that the adversary compromises one forwarder. He can obtain t keys from the compromised forwarder, where these keys are randomly picked from a global key pool of size $|K|$. Given a certain source key, let p_k denote the probability that any forwarder (including the compromised one) shares this key with the source, where subscript $_k$ implies *shared keys*. We have

$$p_k = \frac{t}{|\mathcal{K}|} \ . \tag{6.11}$$

Equivalently, p_k is also the probability that the adversary can decrypt exactly one MAC of the normal message, or the probability that one MAC of the message becomes a revealed one. Considering totally t MACs of the message, the probability that the message has exactly x revealed MACs is

$$P_{rev}(x) = \begin{pmatrix} t \\ x \end{pmatrix} p_k^x (1-p_k)^{t-x} \ , \tag{6.12}$$

where subscript $_{rev}$ implies *revealed MACs*, $\begin{pmatrix} t \\ x \end{pmatrix}$ denotes the combination of choosing x MACs from t MACs and $x \in [0, t]$.

Given x revealed MACs (or $(t-x)$ unrevealed MACs equivalently), we can calculate how many effective MACs that the message can have and what the probability is for the message to have that number of effective MACs. In our scheme, each MAC authenticates u codewords out of m ones of the message. That is, each codeword can occur in an MAC with probability $\frac{u}{m}$. Given two polluted codewords, the probability that an MAC contains exactly one polluted codeword is

$$p_e = 2\frac{u}{m}(1-\frac{u}{m}) \ . \tag{6.13}$$

p_e is also the probability that one unrevealed MAC can become an effective one, where subscript $_e$ implies *effective MACs*.

Now, let $P_{eff|rev}(x,y)$ denote the probability that the message has exactly y effective MACs, given that it already has x revealed MACs, where subscript $_{eff|rev}$ implies *effective MACs given revealed MACs* and $y \in [0, t-x]$. $P_{eff|rev}(x,y)$ can be calculated as

$$P_{eff|rev}(x,y) = \begin{pmatrix} t-x \\ y \end{pmatrix} p_e^y (1-p_e)^{t-x-y} \ . \tag{6.14}$$

If the polluted message has one effective MAC, it can be detected by a receiving node with probability p_k, which is the probability that the receiving node can decrypt the effective MAC. Let $P_{det|eff}(y)$ denote the probability that the polluted message can be detected, given that it has exactly y effective MACs, where subscript $_{det|eff}$ implies *detection probability given effective MACs*. We can easily find that

$$P_{det|eff}(y) = 1 - (1-p_k)^y \ . \tag{6.15}$$

Now, we are able to calculate the average number of hops that the polluted message can travel. We define H_{max} as the length of the longest path from the source to a sink, and simply assume that the polluted message can always travel H_{max} hops, when it is not detected. Let $H_{avg|eff}(y)$ denote the average number of hops that the polluted message can travel, given that it has exactly y effective MACs, where subscript $_{avg|eff}$ implies *average number of hops given effective MACs*. We derive that

$$\begin{aligned} H_{avg|eff}(y) &= \sum_{z=1}^{H_{max}} z\, P_{det|eff}(y) \left(1 - P_{det|eff}(y)\right)^{z-1} \\ &+ \left(1 - P_{det|eff}(y)\right)^{H_{max}} H_{max} \ . \end{aligned} \tag{6.16}$$

In this equality, $P_{det|eff}(y) \left(1 - P_{det|eff}(y)\right)^{z-1}$ denotes the probability that the polluted message can travel exactly z hops, and $\left(1 - P_{det|eff}(y)\right)^{H_{max}}$ denotes the probability that none of H_{max} nodes can detect the polluted message.

Then, we define $H_{avg|rev}(x)$ as the average number of hops that the polluted message can travel, given that it has exactly x revealed MACs, where subscript $_{avg|rev}$ implies *average number of hops given revealed MACs*. It is easy to know that

$$H_{avg|rev}(x) = \sum_{y=0}^{t-x} P_{eff|rev}(x,y) H_{avg|eff}(y) \ . \tag{6.17}$$

Hence, when x takes all values from 0 to t, we can derive H_{avg}, the average number of hops that the polluted message can travel, as follows

$$H_{avg} = \sum_{x=0}^{t} P_{rev}(x) H_{avg|rev}(x) \ . \tag{6.18}$$

When the adversary compromises more nodes, he can obtain more keys, instead of only t keys. We can perform security analysis almost the same way, as long as we change (i.e., increase) probability p_k in equation (6.11) according to the actual number of keys that the adversary obtains.

6.5.3 Analysis of Communication Overhead

As shown in equation (6.5), our scheme attaches t MACs and indexes of encryption keys to an original message, which increases communication overhead, especially for some resource-constrained networks such as wireless sensor networks. To study the communication overhead, we compare the bit-length of the attached MACs with that of the original message.

Let us first consider a source message, which has m codewords with each of $\log_2 q$ bits. So, the bit-length of a source message is $m \log_2 q$. With our scheme, the source message is attached with t MACs. Each MAC contains: (1) one index of random key, which is of $\log_2 |\mathcal{K}|$ bits; (2) one random seed used to identify the indexes of codewords, which is of $\log_2 m$ bits; and (3) the hash which has the same length as a codeword, that is, $\log_2 q$ bits. Our scheme also attaches the indexes of t keys to each source message and each index is of $\log_2 |\mathcal{K}|$ bits. To further reduce the length of a message, we can attach only one index of key, instead of t indexes to the message, where other $(t-1)$ indexes form a hash chain that can be generated from the attached index. Hence, our scheme attaches $t(\log_2 |\mathcal{K}| + \log_2 m + \log_2 q) + \log_2 |\mathcal{K}|$ bits to each source message and introduces communication overhead

$$\frac{t(\log_2 |\mathcal{K}| + \log_2 m + \log_2 q) + \log_2 |\mathcal{K}|}{m \log_2 q} \ . \tag{6.19}$$

Typically, $t = 5 \sim 10$, where $|\mathcal{K}|$ and m are both less than 256 and q is 256-bit long. $\log_2 |\mathcal{K}|$ and $\log_2 m$ are negative when compared to $\log_2 q$. Hence, the communication overhead for source messages can be reduced to $\frac{t}{m} \simeq 2\% \sim 4\%$.

However, the communication overhead for an encoded message may be much higher than that for a source message. If an encoded message is computed from x source messages, it will be attached with all MACs of those source messages. Hence, the communication overhead is x times higher. When $x = 10$, the overhead is $\frac{xt}{m} \simeq 20\% \sim 40\%$. This is a drawback of our scheme.

6.6 Experimental Results

6.6.1 Experiment Setup

We evaluate the performance of our scheme in experiments that consist of two parts. First, we study the impact of different parameters such as t, u and $\mathcal{K}$ on filtering capacity of our scheme. Second, we compare our scheme with others in terms of computation overhead. We test various schemes on both PC and MicaZ sensor node, where the implementation of these schemes is built on top of software package MIRACL [57] (on PC) and Wang's software package [76] (on MicaZ). For CJL's scheme, we implement the Tate pairing [28] over a super-singular elliptic curve $E(\mathbb{F}_p): y^2 = x^3 + 1$. For our scheme, we choose SHA-1 for hash chain generation, and AES (on PC) and SkipJack (on MicaZ) for encryption/decryption.

6.6.2 Detection Capability

We measure the detection capability of our scheme by H_{avg}, the number of hops a polluted message travels. In our experiments, the polluted message is generated by exchanging the positions of two codewords. Given $H_{max} = 20$ (i.e., the polluted message can travel at most 20 hops), $m = 256$ (i.e., each message contains 256 codewords), H_{avg} can be affected by parameters t, u and $|K|$. We study the impact of these parameters separately.

Figure 6.1 illustrates H_{avg} as a function of various u. The best detection capability is achieved when $u \simeq 128 = \frac{m}{2}$, which means the polluted message can be detected the most rapidly as each MAC authenticates about half number of codewords of the message. For example, given $t = 10$, the polluted message can travel around 3 hops when $u = 128$. However, when u is either 16 or 240, the message will not be detected within 11 hops. This makes

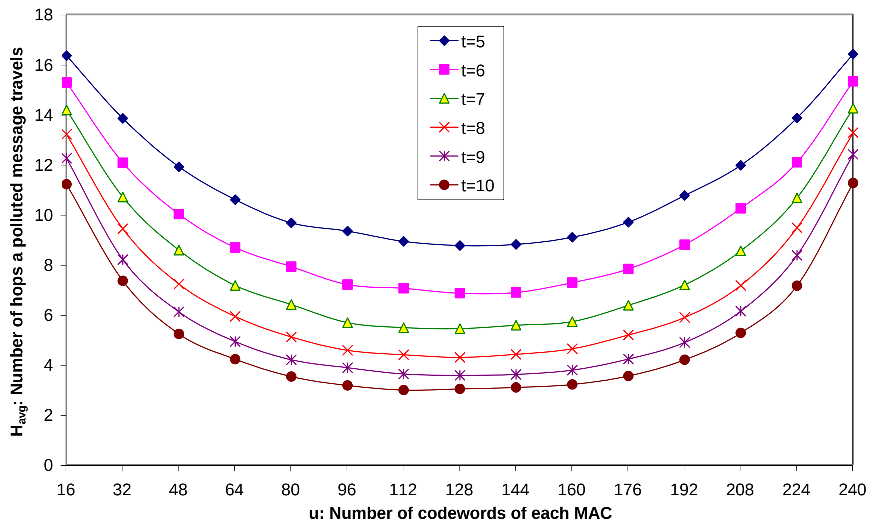


Figure 6.1 Impact of u to H_{avg}, where $H_{max} = 20$, $m = 256$ and $|K| = 100$.

sense, because: (1) when u is too small, a unrevealed MAC is less likely to authenticate a polluted codeword and the message can travel longer; and (2) when u is too big, both polluted codewords are more likely to be authenticated in the same unrevealed MACs and hence prevent the message from being detected.

In Figure 6.1, we can observe that the bigger the value of t, the faster the polluted message can be dropped, while this trend is more clearly shown in Figure 6.2. For example, when

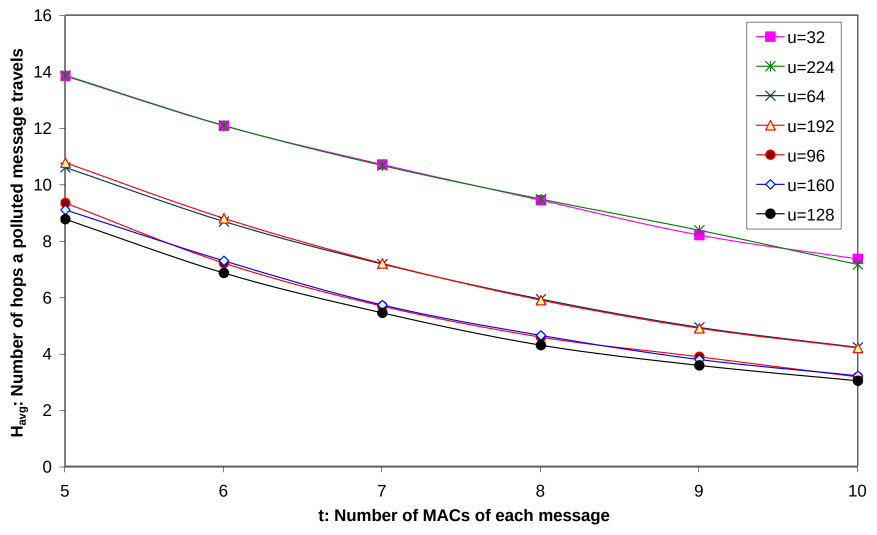


Figure 6.2 Impact of t to H_{avg}, where $H_{max} = 20$, $m = 256$ and $|\mathcal{K}| = 100$.

$u = 128$, the polluted message can travel more than 8 hops in the case of $t = 5$, however, it will be dropped after traveling 3 hops when $t = 10$. It is obviously that the detection capability can be improved by attaching more MACs to the transmitted messages.

The size of key pool $|K|$ also affects the detection capability deeply. Basically, the smaller the size of key pool, the easier a forwarder to find out some shared keys with the source, hence, the sooner the polluted message to be detected. Figure 6.3 illustrates the impact of $|K|$ to

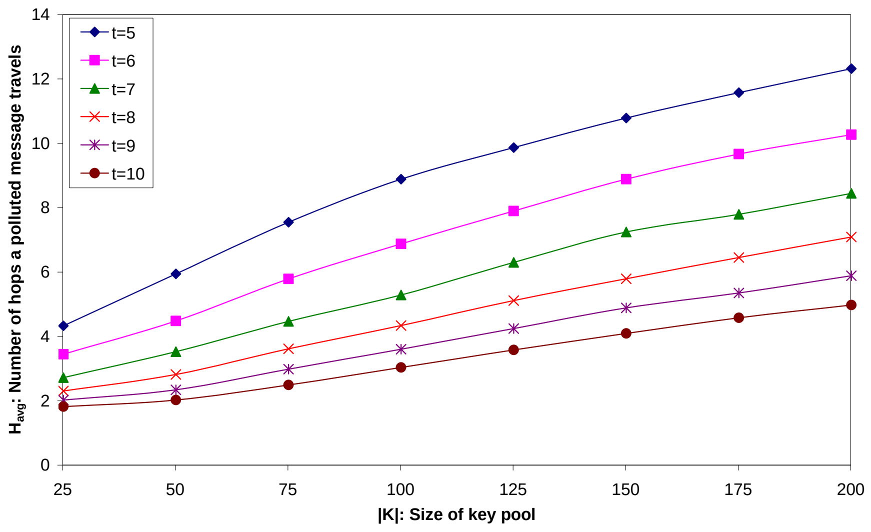


Figure 6.3 Impact of $|K|$ to H_{avg}, where $H_{max} = 20$, $m = 256$ and $u = 128$.

detection capability, where we always set $u = 128$. As shown in Figure 6.3, when $t = 5$, the polluted message can travel around 4.3 hops in the case of $|K| = 25$. However, when $|K| = 200$, the message can travel more than 12 hops. Generally, a smaller key pool performs better in terms of detection capability. However, we do not prefer a key pool which is too small, because the adversary otherwise is able to obtain all the keys from the pool by compromising a few nodes, which makes a polluted message undetectable.

Moreover, it is not always true that a smaller key pool is better. Obviously, when the key pool size is close to t, almost all keys are revealed to the adversary. Hence, no node can detect the polluted message. Figure 6.4 shows this trend. We can see that when the key pool size is smaller than 15, the detection capability becomes worse as the size decreases. Combining the resultsof Figure 6.3 and Figure 6.4, we find that the optimal key pool size (for achieving the

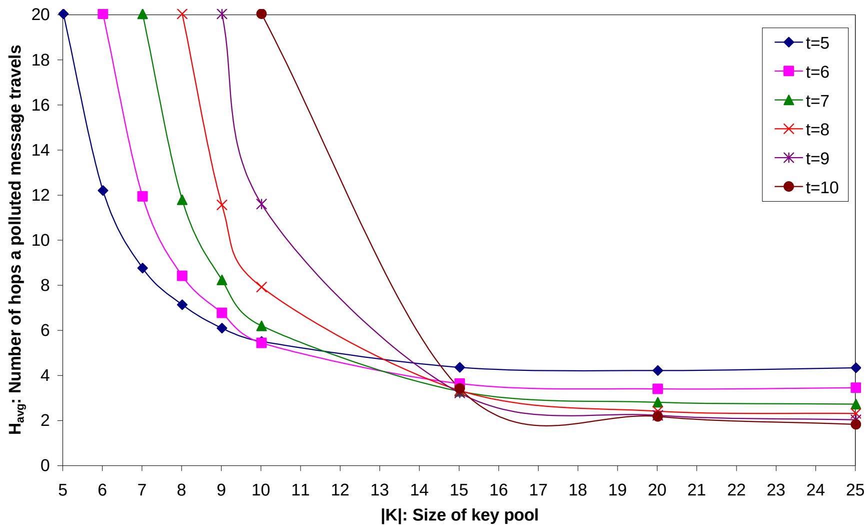


Figure 6.4 Impact of $|K|$ to H_{avg}, where $H_{max} = 20$, $m = 256$ and $u = 128$.

best detection capability) is between 15 and 20. However, as we already discuss, we cannot choose such a small key pool. So, in the rest experiments, we set $|K| = 100$.

In our scheme, the indexes of codewords authenticated by each MAC are calculated from a hash chain in order to reduce communication overhead. We also test a variant of our scheme, that is, we generate each index randomly and embed the whole sequence of indexes into each MAC. In this variant, we do not consider communication overhead and only study if a hash chain is as random as a normal random sequence. We find that the variant has similar performance to our original scheme in terms of detection capability. For space limit, we do not show experimental results here.

6.6.3 Computation Overhead

We compare the computation overhead of CJL's scheme, Yu's scheme, GR's scheme, and our scheme. The overhead is measured by running time of each phase of these schemes. First, we run our test on a Pentium-4 3.0 GHz Linux PC, and set $m = 256$, $n = u = 128$ and $t = 5$ for the schemes. Experimental results are shown in Table 6.2. For our scheme, we test two variants depending on whether the indexes of codewords in MACs are a random sequence or a hash chain, where these two variants are denoted denoted as *Random Sequence* and *Hash*

Chain in Table 6.2. Since CJL's scheme, Yu's scheme, and GR's scheme can only deal with normal linear coding, rather than XOR coding, we test our scheme in both scenarios.

Experimental results in Table 6.2 show that, CJL's scheme is 5 to 10 times slower than Yu's scheme and GR's scheme, while our scheme is 200 to 1000 times faster than others. Generally speaking, the reason is that CJL's scheme, Yu's scheme and GR's scheme are based on expensive public-key operations, but our scheme only adopts symmetric-key operations.

In parameter setup phase, CJL's scheme must generate $(m+n+1)$ q-torsion points, GR's scheme should choose m order-q elements, and our scheme only need to generate t random sequences or hash chains, where each sequence or chain has u elements. (To see the time difference between random sequences and hash chains generation, we move them from the second phase to the first one.) Table 6.2 shows that choosing q-torsion is the most time-consuming. We can also see that in our scheme, generating t hash chains takes $20\times$ the time for creating t random sequences. Since adopting hash chain is useful for reducing the length of messages, it provide us a tradeoff between computation efficiency and communication overhead.

In the second phase, to calculate signature or hash or MAC of one message, CJL's scheme takes $5.37s$ on linear combinations of $(m+n)$ q-torsion points. Yu's scheme and GR's scheme spend $1.42s$ and $0.96s$ on $m+n+1$ and m modular exponentiations, respectively. Meanwhile, our scheme only needs $5.56ms$ for $t \cdot u$ modular multiplications as well as t AES encryptions. Compared to that of other schemes, the computation time of our scheme is negative.

In the third phase, to verify one message, CJL's scheme needs $16.54s$ on $(m+n)$ paring operations. Yu's scheme and GR's scheme require $1.44s$ and $1.43s$ for $(m+n+1)$ and $(m+n)$ modular exponentiations, respectively. In our scheme, we assume an encoded message is computed from $\frac{n}{2}$ source messages and the receiving node has only one shared key. So, our scheme must conduct $(u+\frac{n}{2})$ modular multiplications, $\frac{n}{2}$ AES decryptions, and one hash chain generation, which takes only $2.74ms$ in total. These results prove that pairing operation is extremely time-consuming.

We test Yu's scheme, GR's scheme and our scheme on MicaZ mote. (The implementation of CJL's scheme on MicaZ mote is still in progress.) MicaZ mote is only equipped with an

Table 6.2 Computation overhead of different schemes on Pentium-4 3.0GHz PC (p: 1024-bit, q: 256-bit, m=256, n=u=128 and t=5)

	CJL's Scheme	**Yu's Scheme**	**GR's Scheme**	**Our Scheme**			
				Hash Chain		**Random Sequence**	
				Linear	**XOR**	**Linear**	**XOR**
Parameter Setup *(per n messages)*	10.65 s	2.87 s	2.85 s	0.81 ms		0.04 ms	
Hash Calculation *(per message)*	5.37 s	1.42 s	0.96 s	5.50 ms	0.33 ms	5.56 ms	0.36 ms
Message Verification *(per message & per key)*	16.54 s	1.44 s	1.43 s	2.74 ms	1.83 ms	2.55 ms	1.66 ms

8-bit microprocessor ATmega128, 4K bytes memory (RAM) and 128K bytes program flash memory (ROM). Since it is extremely resource-constrained, we have to relax our security requirement. In our implementation, we set p as 256-bit prime, q as 128-bit prime, m=16 and n=8. Experimental results in Table 6.3 show that Yu's scheme and GR's scheme need around 100s to generate a hash for a message and 150s to verify one message. It indicates that the public-key based approaches are not affordable for resource-constrained wireless networks. On the contrary, our scheme takes at most 32ms to generate MACs for a source message and less than 20ms to verify a message, which means our scheme is quite suitable for wireless sensor networks. Moreover, Table 6.3 shows that XOR coding is more efficient than normal linear coding, so it is promising for using in wireless sensor networks. Since only our scheme can address pollution attacks to XOR coding, it further proves that our scheme is especially suitable for resource-constrained networks such as wireless sensor networks.

Note: the verification phase of Zhao's scheme requires the same number of modular exponentiations as GR's scheme does. So, although we do not test Zhao's scheme on PC and MicaZ, we can conclude that our scheme is much more efficient than Zhao's scheme.

6.7 Conclusion and Future Work

In this work, we propose an efficient scheme against pollution attacks for securing the systems adopting XOR (and normal linear) network coding. In our scheme, the source generates MACs for each message using its random keys, and the forwarders attach to each encoded message the MACs of the source messages that are used to encode the message. Since each MAC authenticates some number of codewords of the message, a forwarder with some shared keys can verify an encoded message using the corresponding MACs. To prevent the compromised forwarders from polluting the codewords using the revealed MACs, we let each codeword authenticated by multiple MACs. By carefully controlling the overlapping codewords authenticated by different MACs, we achieve that a polluted message can be detected within several hops with high probability. To the best of our knowledge, our scheme is the first one that address pollution attacks for XOR coding. It does not rely on any extra secure channels and

Table 6.3 Computation overhead of different schemes on MicaZ sensor node (p: 512-bit, q: 128-bit, m=16, n=u=8 and t=5)

	Yu's Scheme	**GR's Scheme**	**Our Scheme**			
			Hash Chain		**Random Sequence**	
			Linear	**XOR**	**Linear**	**XOR**
Parameter Setup *(per n messages)*	1.61 s	1.56 s	45.94 ms		0.32 ms	
Hash Calculation *(per message)*	147.36 s	99.79 s	31.20 ms	4.57 ms	31.71 ms	4.68 ms
Message Verification *(per message & per key)*	155.14 s	149.70 s	18.75 ms	12.50 ms	9.42 ms	3.10 ms

is extremely efficient. The experimental simulation results show that it is 200 or even 1000 times faster than existing ones, so it is parti suitable for resource-constrained networks such as wireless sensor networks.

Our scheme falls into the category that allows the forwarders to filter polluted attacks. On the contrary, Jaggi's scheme can recover the source messages at the sinks without the help of the forwarders. Although Jaggi's scheme introduce no interference with the forwarders, it has several disadvantages. First, it makes a great sacrifice in network throughput, because it has to tolerate pollution propagation. In our scheme, if a forwarder finds a pollution message, it can drop the message and encode the unpolluted messages using a new encoding vector. Ho et al. [34] proved that a random generated network code allows the sinks to recover the source messages with a high probability, so our scheme can keep a higher source rate. Second, Jaggi's scheme does not take the forwarders' energy consumption into consideration, while our scheme is more suitable for resource-constrained wireless networks by filtering the polluted messages as early as possible. Third, the success of Jaggi's scheme is determined by the estimation to the power of adversaries. However, it is very hard to make an accurate estimation. So, it has to reduce source data rate by overestimating the adversarial capability. Otherwise, the whole scheme becomes failed.

In this work, we assume that the source is never compromised. In the future, we will study how to detect the forged messages injected by false or compromised source(s). In addition, we will implement CJL's signature scheme on MicaZ mote and conduct experimental evaluation.

CHAPTER 7 SUMMARY

7.1 Conclusion

In this research, we focus on enhancing information assurance for resource-constrained wireless networks such as wireless sensor networks. Particularly, we study three important problems: (1) key management for wireless sensors networks, (2) filtering false data injection and DoS attacks in wireless sensor networks, and (3) secure network coding. Solving these problems provide enhancements for several aspects of information assurance such as confidentiality, authenticity, availability and integrity.

We investigate various malicious attacks in wireless sensor networks and propose a number of practical solutions for wireless sensor networks for establishing pairwise keys between sensor nodes, detecting and filtering false data injection and DoS attacks, and securing network coding against pollution attacks. Our solutions are efficient for wireless sensor networks in terms of transmission range, memory cost, computation and communication overhead.

In summary, our contributions from this research are fourfold.

1. We classify the malicious attacks into different categories and provide a taxonomy of these attacks.

2. For enhancing confidentiality, we design a group-based key pre-distribution scheme using deployment knowledge for wireless sensor networks to establish pair-wise keys between sensor nodes. Compared with others, our scheme achieves higher connectivity with lower memory cost and shorter transmission range. It also outperforms others in terms of resilience against node capture attacks.

3. For enhancing authenticity and availability, we propose a dynamic en-route scheme for

filtering false data injection in wireless sensor networks. Compared with others, our scheme offers higher filtering capacity with lower memory cost. It can effectively mitigate the impact of DoS attacks and better deal with dynamic topology of wireless sensor networks.

4. For enhancing integrity, we present several schemes for securing network coding against pollution attacks. Our solution is the first one that addresses security problem for XOR network coding systems. Compared with others, our schemes do not need any extra security channels and can improve computation efficiency by two to three orders of magnitude. Hence, our solutions are promising for resource-constrained wireless networks.

7.2 Future Work

To provide efficient key management for wireless sensor networks, we propose a solution for establishing pairwise key between sensor nodes. However, within this research area, there are many other problems that have not been well solved. Some of the problems that we want to study in the future include *end-to-end key establishment*, *group key management* and *key revocation*.

- An end-to-end key is required when two nodes far away from each other want to exchange their own information. It may be established by exploiting the pairwise keys of multiple hops or with the help of the base station. However, either approach involves high communication overhead and is head to be applied into wireless sensor networks.

- Group key management is another challenging problem. Group keys are highly desired in wireless sensor networks, because sensor nodes are often organized into groups or clusters to detect the events occurring locally. However, sensor nodes may frequently join and leave their groups due to unexpected failures, temporary disconnection, or node renew. Therefore, it is very hard to maintain group keys with low communication overhead while keeping strong resilience to key disclosure in wireless sensor networks. In addition, group keys are often required to be self-healing for accommodating unreliable wireless communication, which greatly increases the complexity of group key management.

- Another untouched problem in key management is the revocation of compromised keys (or nodes). Majority voting is a promising solution to this problem. However, it cannot be applied trivially, because the compromised nodes whose keys should be revoked can also make use of the votes to attack benign nodes. A carefully designed voting scheme is necessary to tackle the key revocation problem.

For filtering false data injection and DoS attacks in wireless sensor networks, we come up with a solution that allows neighbor nodes to mutually monitor each other by exploiting the broadcast nature of wireless communication. However, this mutual monitoring approach brings two problems:

- Wireless communication is essentially unreliable and may not be bi-directional, which pose challenges to the implementation of mutual monitoring. We may enforce retransmission to overcome unreliability or eliminate directional links. However, this incurs high communication overhead and is inefficient for wireless sensor networks.

- Mutual monitoring requires that sensor nodes be awake always or within a long period. This causes conflicts with many energy-efficient approaches that turn off sensor nodes frequently to save energy of sensor nodes.

We will further investigate these two problems and keep on improving our proposed solution.

For defending network coding against pollution attacks, we design a homomorphic signature scheme for securing normal network coding and further present a MAC-based approach for XOR network coding (including normal network coding). In the future, we would investigate other malicious attacks against network coding systems and study how to address them in the environments of wireless sensor networks. Some possible attacks that we have identified are as follows:

- So far, we assume there exists a single source within the networks. However, in wireless sensor networks, it is quite likely that multiple sensor nodes may broadcast their information simultaneously. In a network coding system with multiple sources, existing solutions addressing pollution attacks are no longer applicable, because the signatures (or the MACs) generated by different sources using different keys (or using different codewords) cannot be encoded together. So, multi-source poses new challenge to securing network coding against pollution attacks.

- Besides pollution attacks, the adversaries have other ways to prevent the sinks from recovering source messages. They can either selectively drop some messages, or generate new messages that are linearly dependent on previous ones. The purpose of both ways is to reduce the number of linearly independent messages received by the sinks. Introducing redundancy of source messages can solve the problem, however, it reduces the throughput of networks.
- The adversaries may even insert some false sources into network coding systems, which violates the security goal of authenticity. Our homomorphic signature scheme provides authenticity, however, our solution for XOR coding based on symmetric keys does not. So, how to achieving authenticity and integrity in XOR coding systems simultaneously is still an unsolved problem.

BIBLIOGRAPHY

[1] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network Information Flow", *IEEE Transactions on Information Theory*, Vol. 46, No. 4, pp. 1204–1216, 2000.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks", *IEEE Communications Magazine*, Vol. 40, No. 8, pp. 102–114, 2002.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless Sensor Networks: A Survey", *Computer Networks*, Vol. 38, No. 4, pp. 393–422, 2002.

[4] P. Barreto, H. Kim, B. Lynn, and M. Scott, "Efficient Algorithms for Pairing-Based Cryptosystems", in *Proc. 22nd Annual International Cryptology Conference on Advances in Cryptology*, LNCS, Vol. 2442, pp. 354–368, 2002.

[5] M. Bellare, J. Garay, and T. Rabin, "Fast Batch Verification for Modular Exponentiation and Digital Signatures", in *Proc. Advances in Cryptology (EUROCRYPT'98)*, LNCS, Vol. 1403, pp. 236–250, 1998.

[6] K. Bhattad and K. Narayanan, "Weakly Secure Network Coding", in *Proc. 1st Workshop on Network Coding, Theory, and Applications (NetCod)*, 2005.

[7] S. Biswas and R. Morris, "ExOR: Opportunistic MultiHop Routing for Wireless Networks" in *Proc. ACM SIGCOMM*, pp. 133–143, 2005.

[8] R. Blom, "An optimal class of symmetric key generation systems", in *Proc. EUROCRYPT, Advances in Cryptology*, LNCS, Vol. 209, Springer, pp. 335–338, 1985.

[9] C. Blundo, A. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, “Perfectly-secure key distribution for dynamic conferences”, in *Proc. CRYPTO, Advances in Cryptology*, LNCS 740, Springer, pp. 471–486, 1993.

[10] D. Braginsky, and D. Estrin, “Rumor Routing Algorithm for Sensor Networks”, in *Proc. WSNA*, pp. 22–31, 2002.

[11] BTnode, http://www.btnode.ethz.ch/.

[12] N. Bulusu, J. Heidemann, and D. Estrin, “GPS-less Low Cost Outdoor Localization for Very Small Devices”, *IEEE Personal Communications Magazine*, Vol. 7, No. 5, pp. 28–34, 2000.

[13] N. Cai and R. Yeung, “Secure Network Coding”, in *Proc. International Symposium on Information Theory (ISIT)*, 2002.

[14] S. Capkun, and J. Hubaux, “Secure Positioning of Wireless Devices with Application to Sensor Networks”, in *Proc. IEEE INFOCOM*, 2005.

[15] D. Carman, B. Matt, and G. Cirincione, “Energy-efficient and Low-latency Key Management for Sensor Networks”, in *23rd Army Science Conference*, 2002.

[16] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, “Trading Structure for Randomness in Wireless Opportunistic Routing”, in *Proc. ACM SIGCOMM*, pp. 169–180, 2007.

[17] H. Chan, A. Perrig, and D. Song, “Random key predistribution schemes for sensor networks”, in *Proc. IEEE Symposium on Security and Privacy*, pp. 197–213, 2003.

[18] H. Chan, and A. Perrig, ”PIKE: Peer Intermediaries for Key Establishment in Sensor Networks”, in *IEEE INFOCOM*, 2005.

[19] D. Charles, K. Jian, and K. Lauter, “Signature for Network Coding”, Technique Report MSR-TR-2005-159, Microsoft, 2005.

[20] P. Chou, Y. Wu, and K. Jain, “Practical Network Coding”, in *Proc. Allerton Conference on Communication, Control, and Computing*, 2003.

[21] Committee on National Security Systems (CNSS) Instruction No. 4009, http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf.

[22] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks", in *Proc. ACM CCS*, pp. 42–51, 2003.

[23] W. Du, J. Deng, Y. S. Han, S. Chen, and P. K. Varshney, "A key management scheme for wireless sensor networks using deployment knowledge", in *Proc. IEEE INFOCOM*, 2004.

[24] B. Dutertre, S. Cheung, and J. Levy, "Lightweight Key Management in Wireless Sensor Networks by Leveraging Initial Trust", SRI Int., Tech. Rep. SRI-SDL-04-02, 2004.

[25] L. Eschenauer, and V. D. Gligor, "A key-management scheme for distributed sensor networks", in *Proc. ACM CCS*, pp. 41–47, 2002.

[26] J. Feldman, T. Malkin, C. Stein, and R. Servedio, "On the Capacity of Secure Network Coding", in *Proc. 42nd Annual Allerton Conference on Communication, Control, and Computing*, 2004.

[27] C. Fragouli, J. Boudec, and J. Widmer, "Network Coding: An Instant Primer", *ACM SIGCOMM Computer Communication Review*, Vol. 36, No. 1, 2006.

[28] S. Galbraith, K. Harrison, and D. Soldera, "Implementing the Tate Pairing", in *Proc. 5th Algorithmic Number Theory Symposium (ANTS V)*, LNCS, Vol. 2369, pp. 324–337, 2002.

[29] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale File Distribution", in *Proc. IEEE INFOCOM*, 2005.

[30] C. Gkantsidis and P. Rodriguez, "Cooperative Security for Network Coding File Distribution", in *Proc. IEEE INFOCOM*, 2006.

[31] P. Gupta, and P. R. Kumar, "Critical power for asymptotic connectivity in wireless networks", in *Stochastic Analysis, Control, Optimization and Applications: A Volume in Honor of W. H. Fleming*, pp. 547–566, 1998.

[32] N. Gura, A. Patel, A. Wander, H. Eberle, and S. Shantz, "Comparing elliptic curve cryptography and RSA on 8-bit CPUs", in *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, LNCS, Vol. 3156, Springer, pp. 119–132, 2004.

[33] T. He, C. Huang, B. Blum, J. Stankovic, and T. Abdelzaher, "Range- Free Localization Schemes in Large Scale Sensor Network", in *Proc. ACM MobiCom*, pp. 81–95, 2003.

[34] T. Ho, R. Koetter, M. Médard, R. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting" in *Proc. International Symposium on Information Theory (ISIT)*, 2003.

[35] T. Ho, B. Leong, R. Koetter, M. Méard, M. Effros, and D. Karger, "Byzantine Modification Detection in Multicast Networks Using Randomized Network Coding", in *Proc. 2004 IEEE International Symposium on Information Theory (ISIT)*, 2004.

[36] D. Huang, M. Mehta, D. Medhi, and L. Harn, "Location-aware Key Management Scheme for Wireless Microsensor Networks", in *ACM SASN*, 2004.

[37] D. Hwang, B. Lai, and I. Verbauwhede, "Energy-memory-security Tradeoffs in Distributed Sensor Networks", in *ADHOC-NOW*, LNCS, Vol. 3158, 2004.

[38] J. Hwang, and Y. Kim, "Revisiting Random Key Pre-distribution for Sensor Networks", in *ACM SASN*, 2004.

[39] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Méard, "Resilient Network Coding in the Presence of Byzantine Adversaries", in *Proc. IEEE INFOCOM*, 2007.

[40] K. Jain, "Security Based on Network Topology against the Wiretapping Attack", *IEEE Wireless Communications*, Vol. 11, No. 1, pp. 68-71, 2004.

[41] A. Joux, "The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems", in *Proc. 5th Algorithmic Number Theory Symposium (ANTS V)*, LNCS, Vol. 2369, pp. 20–32, 2002.

[42] C. karlof, and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures", in *Proc. 1st IEEE Int. Workshop on Sensor Network Protocols and Applications*, pp. 113–127, 2003.

[43] B. Karp, and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks", in *Proc. ACM MobiCom*, pp. 243–254, 2000.

[44] S. Katti, D. Katabi, W. Hu, H. Rahul, and M. Médard, "The Importance of Being Opportunistic: Practical Network Coding for Wireless Environments", in *Allerton*, 2005.

[45] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "XORs in The Air: Practical Wireless Network Coding", in *Proc. ACM SIGCOMM*, pp. 243–254, 2006.

[46] S. Katti, S. Gollakota, and D. Katabi, "Embracing Wireless Interference: Analog Network Coding", in *Proc. ACM SIGCOMM*, pp. 397–408, 2007.

[47] M. Krohn, M. Freeman, and D. Mazieres, "On-the-fly Verification of Rateless Erase Codes for Efficient Content Distribution", in *Proc. IEEE Symposium on Security and Privacy*, 2004.

[48] L. Lazos, and R. Poovendran, "SeRLoc: Secure Range-Independent Localization forWireless Sensor Networks", in *Proc. ACM WiSe*, pp. 21–30, 2004.

[49] L. Lazos, R. Poovendran, and S. Capkun, "ROPE: Robust Position Estimation in Wireless Sensor Networks", in *Proc. IPSN*, pp. 324–331, 2005.

[50] S. Li, R. Yeung, and N. Cai, "Linear Network Coding", *IEEE Transactions on Information Theory*, Vol 49, No. 2, pp. 371–381, 2003.

[51] D. Liu, and P. Ning, "Establishing pairwise keys in distributed sensor networks", in *Proc. ACM CCS*, pp. 52–56, 2003.

[52] D. Liu, and P. Ning, "Location-based pairwise key establishment for static sensor networks", in *ACM SASN*, pp. 72–82, 2003.

[53] D. Liu, and P. Ning, “Improving key pre-distribution with deployment knowledge in static sensor networks”, *ACM Transactions on Sensor Networks (ToSN)*, Vol. 1, No. 2, pp. 204–239, 2005.

[54] A. Menezes, T. Okamoto, and S. Vanstone, “Reducing Elliptic Curve Logorithms to Logorithms in a Finite Field”, *IEEE Transactions on Information Theory*, Vol 39, No. 5, pp. 1639–1646, 1993.

[55] MICAz, http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.

[56] V. Miller, “Short Programs for Functions over Curve”, unpublished manuscript, *http://crypto.stanford.edu/miller/miller.pdf*, 1986.

[57] MIRACL, Multiprecision Integer and Rational Arithmetic C/C++ Library, http://www.shamus.ie/, Shamus Software Ltd.

[58] R. Nagpal, H. Shrobe, and J. Bachrach, “Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network”, in *Proc. IPSN*, LNCS, Vol. 2634, pp. 333–348, 2003.

[59] D. Nicolescu, and B. Nath, “Ad-Hoc Positioning Systems (APS)”, in *Proc. IEEE GLOBECOM*, Vol. 5, pp. 2926–2931, 2001.

[60] M. D. Penrose, “The longest edge of the random minimum spanning tree”, *The Annals of Applied Probability*, Vol. 7, No. 2, pp. 340–361, 1997.

[61] M. D. Penrose, *Geometric Random Graphs*, Oxford University Press, 2003.

[62] D. Petrovic, K. Ramchandran, and J. Rabaey, “Overcoming Untuned Radios in Wireless Networks with Network Coding”, *IEEE Transactions on Information Theory*, Vol. 52, No. 6, pp. 2649–2657, 2006.

[63] A. Perrig, R. Szewczyk, V. Wen, D. Culer, and J. Tygar, “SPINS: Security Protocols for Sensor Networks”, in *Proc. ACM MobiCom*, pp. 189–199, 2001.

[64] B. Przydatek, D. Song, and A. Perrig, "SIA: Secure Information Aggregation in Sensor Networks", in *Proc. ACM SenSys*, pp. 255–265, 2003.

[65] S. Ratnasamy, B. karp, L. Yin, F. Yu, D. Estrin, R. Govindan and S. Shenker, "GHT: A Grographic Hash Table for Data-Centric Storage", in *Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications, (WSNA'02)*, pp. 78–87, 2002.

[66] K. Ren, W. Lou, and Y. Zhang, "LEDS: Providing Location-aware End-to-end Data Security in Wireless Sensor Networks", in *Proc. IEEE INFOCOM*, 2006.

[67] S. Sengupta, S. Rayanchu, and S. Banerjee, "An Analysis of Wireless Network Coding for Unicast Sessions: The Case for Coding-Aware Routing", in *Proc. IEEE INFOCOM*, pp. 1028–1036, 2007.

[68] S. Shakkottai, R. Srikant, and N. Shroff, "Unreliable sensor grids: coverage, connectivity and diameter", in *Proc. IEEE INFOCOM*, 2003.

[69] J. H. Spencer, *The Strange Logic of Random Graphs*, Springer, 2001.

[70] W. Stallings, Cryptography and Network Security: Principles and Practice, Second Edition, Prentice Hall.

[71] TelosB, http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.

[72] A. Liu, P. Kampanakis, and P. Ning, TinyECC: Elliptic Curve Cryptography for Sensor Networks, http://discovery.csc.ncsu.edu/software/TinyECC/.

[73] TinyOS Community Forum, Available: http://www.tinyos.net.

[74] Ubiquitous Computing, http://www.ubiq.com/hypertext/weiser/UbiHome.html.

[75] D. Wang, D. Silva, and F. Kschischang, "Constricting the Adversary: A Broadcast Transformation for Network Coding", in *Proc. 45th Annual Allerton Conference on Communication, Control and Computing*, 2007.

[76] H.Wang, and Q. Li, "Efficient implementation of public key cryptosystems on mote sensors", in *Proc. ICICS*, LNCS, Vol. 4307, pp. 519–528, 2006.

[77] R. Want, A. Hopper, V. Falcão, and J. Gibbons, "The Active Badge Location System", *ACM Transactions on Information Systems (TOIS)*, Vol. 10, No. 1, pp. 91–102, 1992.

[78] Mark Weiser, http://www.ubiq.com/hypertext/weiser/.

[79] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks", in *Proc. ACM SenSys*, pp. 14–27, 2003.

[80] IEEE 802.15 Working Group for Wireless Personal Area Networks (WPAN), http://ieee802.org/15/.

[81] Y. Wu, P. Chou, and S. Kung,"Information Exchange in Wireless Networks with Network Coding and Physical-Layer Broadcast", in *Proc. 39th Annual Conference on Information Sciences and Systems (CISS)*, 2005.

[82] Xerox PARCTAB, http://www.ubiq.com/parctab/.

[83] Xerox Pad, http://www.ubiq.com/weiser/testbeddevices.htm.

[84] H. Yang, and S. Lu, "Commutative Cipher Based En-route Filtering in Wireless Sensor Networks", in *Proc. IEEE VTC*, Vol. 2, pp. 1223–1227, 2004.

[85] H. Yang, F. Ye, Y. Yuan, S. Lu, and W. Arbaugh, "Toward Resilient Security in Wireless Sensor Networks", in *Proc. ACM MobiHoc*, pp. 34–45, 2005.

[86] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-route Detection and Filtering of Injected False Data in Sensor Networks", in *Proc. IEEE INFOCOM*, 2004.

[87] Y. Yu, R. Govindan, and D. Estrin, "Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks", Computer Science Department, University of California, Los Angeles, Tech. Rep. UCLA-CSD TR-01-0023, May 2001.

[88] Z. Yu, and Y. Guan, “A key pre-distribution scheme using deployment knowledge for wireless sensor networks”, in *Proc. IEEE/ACM IPSN*, 2005, pp. 261–268.

[89] Z. Yu, and Y. Guan, “A robust group-based key management scheme for wireless sensor networks”, in *Proc. IEEE WCNC*, Vol. 4, pp. 1915–1920, 2005.

[90] Z. Yu, and Y. Guan, “A Dynamic En-Route Scheme for Filtering False Data Injection in Wireless Sensor Networks”, in *Proc. IEEE INFOCOM*, 2006.

[91] Zhen Yu, Yawen Wei, Bhuvaneswari Ramkumar, and Yong Guan, “An Efficient Signature-based Scheme for Securing Network Coding against Pollution Attacks”, in *Proc. IEEE INFOCOM*, 2008.

[92] S. Zhang, S. Liew, and P. Lam, “Hot Topic: Physical-Layer Network Coding”, in *Proc. MobiCom*, pp. 358–365, 2006.

[93] F. Zhao, T. Kalker, M. Médard, and K. J. Han, “Signatures for Content Distribution with Network Coding”, in *Proc. International Symposium on Information Theory (ISIT)*, pp. 556–560, 2007.

[94] Y. Zhou, Y. Zhang, and Y. Fang, ”LLK: A Link-layer Key Establishment Scheme in Wireless Sensor Networks”, in *IEEE WCNC*, 2005.

[95] Y. Zhou, Y. Zhang, and Y. Fang, ”Key Establishment in Sensor Networks based on Triangle Grid Deployment Model”, in *IEEE MILCOM*, 2005.

[96] S. Zhu, S. Setia, and S. Jajodia, “LEAP: efficient security mechanisms for large-scale distributed sensor networks”, in *Proc. ACM CCS*, 2003, pp. 62–72.

[97] S. Zhu, S. Xu, S. Setia, and S. Jajodia, “Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach”, in *Proc. IEEE ICNP*, 2003, pp. 326–335.

[98] S. Zhu, S. Setia, S. Jajodia, and P. Ning, “An Interleaved Hop-by-Hop Authentication Scheme for Filtering of Injected False Data in Sensor Networks”, in *Proc. IEEE Symposium on Security and Privacy*, pp. 259–271, 2004.