**The Pennsylvania State University**
**The Graduate School**

# COMPROMISE-RESILIENT ANTI-JAMMING COMMUNICATION IN WIRELESS SENSOR NETWORKS

A Thesis in
Computer Science and Engineering
by
Xuan Jiang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2011

The thesis of Xuan Jiang was reviewed and approved* by the following:

Guohong Cao
Professor of Computer Science and Engineering
Thesis Advisor

Sencun Zhu
Associate Professor of Computer Science and Engineering

Jose A. Ventura
Professor of Industrial Engineering

Raj Acharya
Department Head and Professor of Computer Science and Engineering

---

*Signatures are on file in the Graduate School.

# Abstract

Jamming is a kind of Denial-of-Service (DoS) attack in which an adversary purposefully emits radio frequency signals to corrupt the wireless transmissions among normal nodes. Some research has been conducted on countering jamming attacks for wireless networks. Few works consider jamming attacks launched by insiders, where an attacker first compromises some legitimate sensor nodes to acquire the common cryptographic information of the sensor network and then jams the network through those compromised nodes. In this paper, we first retrospect research works for wireless networks from its feasibility to countermeasures. Then, we address the insider jamming problem in wireless sensor networks. In our proposed solutions, the physical communication channel of a sensor network is determined by the group key shared by all the sensor nodes. When insider jamming happens, the network will generate a new group key to be shared only by the non-compromised nodes. After that, the insider jammers are revoked and will not be able to predict the future communication channels used by the non-compromised nodes. Specifically, we propose two compromise-resilient anti-jamming schemes: the *split-pairing* scheme which deals with a single insider jammer, and the *key-tree-based* scheme which copes with multiple colluding insider jammers. We implement and evaluate the proposed solutions using Mica2 Motes. Experimental results show that our solutions have low recovery latency and low communication overhead, and hence they are suitable for resource constrained sensor networks.

# Table of Contents

# List of Figures

# Acknowledgments

First and foremost I want to thank my advisor Dr. Cao Guohong. It has been an honor to be his Master student. He has taught me, both consciously and unconsciously, how good research is done. I appreciate all his contributions of time, ideas, and funding to make me productive and stimulating. The joy and enthusiasm he has for his research was contagious and motivational for me.

I would also like to thank Dr. Zhu Sencun. This thesis would not have been possible without the guidance and the help of him. His suggestion and guidance improves the quality of this thesis. Without his idea, I would hardly imagine how this work would be excellent.

Prof. Ventura deserves a special thanks as my thesis committee member. His insights to optimization is second to none. Besides, he sets an example of a world-class researcher for his rigor and passion on research.

The members of the Dr. Cao's group have contributed immensely to my personal and professional time. The group has been a source of friendships as well as good advice and collaboration. I am especially grateful for the fun group of members who stuck it out in graduate school with me. I would like to acknowledge group member Wenhui Hu who contributes a great deal of time to help me in the experiment. We worked together day and night on the Mica 2 platform, and I very much appreciated his enthusiasm, intensity, willingness to do frequent changes and numerus measurements to ensure the work is persuasive and convincing.

Lastly, I would like to thank my family for all their love and encouragement. For my parents who raised me with a love of science and supported me in all my pursuits. Thank you all.

Chapter 1

# Introduction

The broadcast nature of wireless radio transmission makes it vulnerable to jamming-based Denial-of-Service (DoS) attacks in which an attacker purposefully launches signals to corrupt wireless communications. Wireless Sensor Networks (WSNs) are especially susceptible to jamming attacks since normal sensors have limited resources in computation, communication, storage and energy, and do not have complicated transceivers against the jammers.

Jamming models have been widely studied, classified and evaluated. For example, jammers can be classified in terms of capabilities (broadband or narrowband). In order to study the different attack philosophies, the authors classifies the jamming strategies based on its behaviors such as constant, deceptive, random, reactive in [1]. For a constant jammer, it continually emits a radio signal. Two typical implementations are a waveform generator which continuously sends a radio signal and a normal wireless device such as a Mica 2 Mote. For the second type, the constant jammer violates the underlying MAC protocol. It does not wait for the channel to become idle before transmitting so that it can prevent legitimate traffic sources from getting hold of channel and sending packets. Instead of sending out random bits, a deceptive jammer injects regular packets to the channel without any gap between subsequent packet transmissions. A normal communicator is then deceived to believe that the legitimate communication is ongoing and it keeps remaining in the receive state. For example, if a deceptive jammer sends a continuous stream of preamble bits (0xAA in TinyOS), the legitimate nodes will be deceived and keep in receive state even if the nodes have packets to send. For a random jammer, it switches between sleeping and jamming. During jamming, it can either behave like a constant jammer or a deceptive jammer. This jamming model takes the energy conservation into account which is important for those jammers that do not have unlimited power supply. The above three models are all active jammers which means that they try to interrupt communication without considering the ongoing traffic pattern. Active jammers are usually effective because they keep the channel busy all the time. However, they are easier to be detected. Compared with active jammers, the reactive jammer is more difficult to identify. The reactive jammers stay quiet when the channel is idle but starts transmitting a radio signal as soon as it senses ongoing communication. A reactive jammer aims to block the reception of a message. As a result, it is harder to detect. In addition to study the attack philosophies, [1, 2, 3, 4, 5, 6] study the working mechanism and categorize based on their protocol layers. Physical layer jammers directly emit energy on communication channels to interfere the reception of legitimate transmissions. MAC layer jammers can insert dummy packets or preambles to deceive the receivers. For cross-layer jammers, each layer can be decomposed into several manageable design problems related to jamming and sensing functions. For the design of a cross-layered attacker for the Transport/Network layer, by sensing the packet type and timing in physical layer, protocols introduce highly predictable timing that can be exploited. The limited information of packet size, timing, and sequence is therefore enough to accurately predict packet types. Using a combination of offline historical analysis of sequence to provide training data for the online models, a packet classifier was developed that adapts to variations across networks. As a result, wireless TCP/IP is significantly vulnerable.

Jamming cannot be adequately addressed by common security mechanisms such as confidentiality, authentication, and integrity, because jamming targets at the basic transmission and reception capabilities of the physical devices. Moreover, none of the cryptographic constructions such as encryption/decryption can be directly adopted to solve the problem. Thus, we have to seek new solutions to deal with this severe attack.

Most physical layer countermeasures rely on the spread spectrum technique [7, 8]. The idea of spread spectrum system is to transmit signal by spreading it over a wide frequency band, much wider than the minimum bandwidth required to transmit the signal. For instance, a spread spectrum system takes a narrow band signal with a bandwidth of a few kiloHertz and distributes it over a wide band that is many megaHertz by modulating it with a special sequential of noise-like signal structure, a pseudorandom sequence. To retrieve the original information signal, the receiver demodulates the received signals using the same sequence. As the result of spreading and de-spreading operation, the signal-to-nose ratio on the channel is enhanced and the effect of jamming is reduced. This is because the desponding is accomplished by correlating the received spread spectrum with e same sequence. When the two signals are matched, the desired signal collapses to its original bandwidth before spreading, whereas any unmatched input (the jamming signal) is spread to the wideband. A filter then rejects all but the desired narrow band signal. The enhanced signal-to-noise ratio on the channel is called process gain.

Depend on the methods that spread the narrow band signal, spread spectrum techniques can be divided into the following varieties: frequency-hopping spread spectrum (FHSS), direct-sequence spread spectrum (DSSS), time-hopping spread spectrum(THSS), chirp spread spectrum (CSS), and combinations of these techniques. Among them, FHSS, DSSS, and the combination those two provide jamming resistance.

*Frequency-hopping spread spectrum (FHSS).* Frequency hopping systems rapidly hop its carrier frequency among thousands of frequencies. The specific order of hopping frequencies is a pseudorandom sequence known to both sender and receiver. The rate of frequency hopping is a function of the information rate, the amount of redundance used, and the distance to nearest potential jammer. The challenge of FHSS is to synchronize the hopping sequence to both the sender and receiver. FHSS is robust to the narrowband jamming attack since the jamming signal will be effective only when the legitimate communication happens to hop to jammed frequency.

*Direct-sequence spread spectrum (DSSS).* Direct-sequence spread spectrum systems multiply the narrowband signal to be transmitted by a "noise" signal. This noise signal is a pseudorandom sequence of 1 and -1 values, at a frequency much higher than that of the original signal, thereby spreading the energy of the original signal into a much wider band. The modulated signal is like white noise. However, this noise-like signal can be used to exactly reconstruct the original data at the receiving end, by multiplying it by the same pseudorandom (PN) sequence. This process, known as "despreading", mathematically constitutes a correlation of the transmitted PN sequence with the PN sequence that the receiver believes the transmitted is using. For despreading to work correctly, the transmitted and received sequences must be synchronized. This requires the receiver to synchronize its sequence with the transmitter's sequence via some sort of timing search process. The despreading mechanism reduces the jamming signal power and makes the DSSS technique resistance to many types of jamming attacks. However, DSSS can only defense certain jamming attacks. A jammer that transmit continuous signals can defeat the DSSS system by increasing its jamming power high enough to overcome the processing gain in the receiver end. Alternatively, a jammer can emit pulse signals with the peak power much higher than the constant power, and adopt a 33% duty cycle pulse which is adequate for effective jamming.

*Time-hopping spread spectrum (THSS).* Time hopping systems use the pseudorandom sequence to control the transmitter on and off. This form of spread spectrum modulation is mainly applied in combination with frequency hopping. Typically, this sort of system is not effective in coping with jamming. It usually combines with FHSS to prevent single frequency jamming from causing significant communication interruption.

*Chirp spread spectrum (CSS).* A chirp is a sinusoidal-wave signal whose frequency increases or decreases with time. It is also known as sweep signal as its frequency scan through a predetermined frequency band. Modulating the narrowband signal with a chirp will spread the signal over a wider frequency band. Similar to frequency hopping technique, chirp spread spectrum can only reduce the narrowband jamming attack, since such jammer can only affect the chirp receiver for a small percentage of the time. Additionally, if the jammer can determine the tuning slope of the chirp frequency, it can effectively disable the chirp receiver by sweeping the frequency band at the same pace as the chirp. Since the chirp system requires both sender and receiver knows the chirp beforehand and chirp modulation in digital system is not flexible, modern communication systems rarely use CSS.

In modern physical layer countermeasures of jamming attacks, FHSS and DSSS are mostly used in which the sender and receiver share the same secret sequence. The secret sequence is usually determined and generated by a key and a pseudorandom function so that both sender and receiver can hop among channels or use the spreading sequence to evade the jamming attack. However, to successfully communicate under jamming attack, both sender and receiver need to know the same hopping or spreading sequence beforehand and keep it secret. [9, 10, 11, 12] studied the problem of key establishment without pre-shared secret under jamming. In [9], the uncoordinated frequency hopping (UFHSS) recovery scheme is proposed which enables the jamming-resistant one-to-one communication in the presence of a jammer without a pre-shared secret sequence. In the scheme, each message is broken into multiple segments and then sent out on random hopping frequencies chosen from a fixed frequency band. Like coordinated frequency hopping, UFHSS is based on the assumption that the attacker cannot jam all frequency channels on which the nodes communicate at the same time so that the sender and the receiver can still communicate through the remaining jam-free channels. However, in UFHSS, the sender and the receiver do not agree on a secret channel sequence but instead transmit and listen on randomly selected channels. The fundamental observation is that, with sufficient transmission attempts, the sender and receiver can send and listen on the same channels in a number of time slots, even if they did not agree on them beforehand. For example, given 500 channels and a sender hopping among the channels at a high rate 1500 Hz and a receiver hopping at a low rate 100 Hz, the receiver will be listening on the frequency where the sender is transmitting in average $1500/500 = 3$ times per second. Based on this observation, the UFHSS scheme is highly resistant to packet losses and active jamming attacks. It can be applied in settings where two nodes wish to establish an unanticipated and spontaneous communication without pre-shared secret sequence, which was so far not feasible using coordinated frequency hopping. Unfortunately, the UFHSS scheme deals with one-to-one communication and does not provide an efficient solution for broadcast communication. To fix this problem and support group communication, UDSSS [11] has been proposed for broadcast communication. In UDSSS, a public set of spreading sequences is stored and used by the sender and the receivers. The set is public and may be known to the attacker. To transmit a message, the sender randomly selects a spreading sequence from the set and spreads the message with this sequence. The receivers record the signal on the channel and despread the message by applying sequences from the set using a trial-and-error method. The receivers using UDSSS are not time-synchronized to the sender with respect to the spread signal. In order to compensate for this, the sender sends the message repeatedly and the receivers apply a sliding window approach to synchronize to the transmission. The efficiency of UDSSS is therefore determined by factors. One is that the time when the receivers need to find the right spreading code and its synchronization. The other is the probability of the attackers jamming success. Given that, the receivers need to search through the set and adjust the synchronization windows in order to despread the received message. UDSSS enables anti-jamming communication between nodes that are within each others transmission ranges without pre-share a secret sequence. Moreover, UDSSS supports broadcast anti-jamming communication for dynamic groups of untrusted receivers. However, as the cost of UDSSS, it requires the receivers to store the sequence set. It also requires the time to search the set and find the correct despreading sequence which defines the latency of the communication. From the above, we see that UFHSS, UDSSS and most of the physical layer anti-jamming approaches require sophisticated processing unit and storage device which are not applicable to sensor nodes. For example, Mica2 Mote does not support Direct-Sequence Spread Spectrum (DSSS).

Researchers studied jamming attacks on a broadcast control channel in [13, 14]. [13] considered an insider attacker in a centralized network with a cluster head who can compromise nodes to obtain the cryptographic information such as hopping sequences. A cluster head generates hopping sequences for each member in which some positions of the sequences share the same frequency bands for the control channel. The compromised node is identified by metrics such as the expected hamming distance. The cluster head then updates and redistributes the hopping sequence. To deal with a similar problem, [14] proposes a framework to control the channel access, using the random assignment of cryptographic keys to hide the location of the control channels. Both schemes, however, are centralized with the help of either cluster heads or trusted authorities.

For broadcast channel with insider receivers, tree-based and group-based schemes [15, 16, 17] have been proposed. [15] uses a balanced binary tree to allocate spectrum sequences. In this scheme, each node of the tree corresponds to a spread spectrum seqence. Each user in the broadcast group is assigned a leaf and can access to the sequences corresponding to that leaf and all its ancestors. The system transmits on a set of sequences such that all users can demodulate using exactly one sequence; such set is referred to as a disjoint cover. Besides transmitting on the disjoint cover, the system also transmits on a set of test sequences. A sequence in the disjoint cover is a detectable sequence if it is the ancestor of any of the test sequences. If any user receives message on a test sequence but not on the corresponding detectable sequence, then that user reports jamming on the detectable sequence, and the system responds by removing the detectable sequence from the cover and inserting its two children sequences in its place. In [16], group-based schemes have been proposed to combat insider jammers in broadcast systems. In the scheme, multiple receivers are organized into multiple broadcast groups and different groups use different channels. This ensures that a compromised receiver can only affect the members in the same group. A divide-and-conquer strategy is then used to isolate malicious receivers. However, the scheme requires the sender to send a separate copy of each broadcast message to every group, causing a lot of communication overhead. To deal with t compromised receiver, the sender needs to send at least 2t extra copies of messages for each broadcast. To compensate the communication overhead, [17] proposes a partial channel sharing solution. Instead of sending messages using one channel, the channel is divided into multiple smaller ones and let different groups partially share their channels. In this way, the data sent over the shared channels can reach more than one groups, saving substantial communication cost. In the performance analysis sections in [16, 17], we see that these schemes require a large number of available channels so that the attacks have low probability to jam the group-used communication channel. Otherwise, the compromised nodes could coordinate to jam all channels in a group which renders recovery failure for group-based schemes.

For WSNs, Xu et al. proposed to use channel surfing [2] to deal with a narrow-band and intermittent jammer. In this scheme, the jammed nodes immediately switch to another orthogonal channel and wait for opportunities to reconnect to the rest of the network. After the jammed nodes lose connectivity, their neighbors, the boundary nodes, will discover the disappearance of their jammed neighbor nodes and temporally switch to the new channel to search for them. If the lost neighbors are found on the new channel, the boundary nodes will participate in rebuilding the connectivity of the entire network. Specifically, the scheme consists of two different techniques for the boundary nodes to repair network connectivity. The first technique is called coordinated channel switching, in which the boundary nodes participate in switching the entire network to the new channel to rebuild total network connectivity on the new channel. The other technique is call spectral multiplexing, where boundary nodes multiplex between the old channel and the new channel, serving as a bridge that connects nodes operating on different channels. However, to avoid the jammer to predict the next channel, the channel selection uses a keyed pseudorandom generator. For example, in the coordinated channel switching, all nodes switch to a different channel $C(n+1) = F_K(C(n))$ to evade jamming after jamming is detected, where $K$ is a group key shared by all nodes, $F$ is a pseudorandom function and $C(n)$ is the original channel used before jamming. However, this technique is limited to outsider attacks and it does not work under node compromises since an insider attacker knows the group key $K$ and the function $F$. Additionally, Xu et al. propose the timing channel scheme for WSNs. The scheme is built as a low-rate physical layer overlay on top of the traditional physical/link-layers. The timing channel uses the detection and timing of failed packet receptions at the receiver, which we have shown is possible by time stamping CRC failures or by monitoring the signal strength. Then, the inter-arrival codes for building a single-sender, single-receiver timing channel is constructed. To address the multiple-sender, single-receiver cases, an asynchronous code and a decoding procedure that employs a bank of parallel correlators is constructed. To complete the scheme, an overlay data link layer, which provides framing, error detection/correction, and authentication, is proposed. However, the timing channel can only resume communications in a low data rate and the existing protocol should be modified or updated.

In this paper, we address the insider jamming problem in WSNs. Our solution includes two phases: the identification of malicious insiders and the recovery from the insider jamming. For the identification, our goals is to identify the malicious insider by all nodes. Specially, we split the network into two groups equally. Two nodes in different groups rotationally pair and verify each other by software attestation. In the recovery scheme, the physical communication channel is determined by the group key shared by all nodes. When recovery scheme starts, the network will generate a new group key to be shared only by the non-compromised nodes. After that, the insider jammers are revoked and will not be able to predict future communication channels used by the non-compromised nodes. To realize this idea, we address the following research challenges: *First, how can the non-compromised nodes agree on a new group key in a fully distributed way? Second, how do they distribute the new group key under the presence of one or multiple jammers.* Specifically, we propose two compromise-resilient anti-jamming schemes. The first scheme, called *split-pairing*, deals with a single insider jammer. Due to the channel switch delay, the insider jammer cannot jam two channels at the same time. By actively splitting non-compromised nodes into two or multiple groups using multiple channels, nodes communicating in jamming-free channels can first reestablish a new common group key, and then propagate this key to other non-compromised nodes. We further propose a *key-tree-based* scheme to cope with multiple colluding insider jammers. Our goal is to construct a logical key tree in a bottom-up manner under jamming so that all jammed nodes can finally share the root key to derive a common secret channel. Then, they can propagate the shared key to other non-compromised nodes.

We have implemented and evaluated the proposed solutions using Mica2 Motes. Experimental results show that our solutions have low recovery latency and low communication overhead, and hence they are suitable for resource constrained sensor networks.

The rest of this paper is organized as follows. Chapter 2 describes the system model and the design goal. The identification is addressed in chapter 3. The details of our recovery schemes are presented in Chapter 4 and Chapter 5. In Chapter 6, we present the performance evaluation results of our proposed schemes. Last, Chapter 7 concludes the paper.

Chapter 2

# System Model and Design Goal

## 2.1 Network Model and Assumptions

We assume each node in the network has multiple channels and can switch to different channels. For example, the Mica2 mote has 32 effective channels for radio transmission [18]. As our first step towards addressing the insider jamming problem, in this paper, we focus on a one-hop network in which each node can directly communicate with all other nodes. This model has been widely used and studied in recent works [9, 10, 11, 13, 14, 19].

For security purpose, we assume every pair of nodes share a pairwise key. For a static network, keying materials could be stored or hard-coded in non-volatile memory such as Flash memory. For a dynamic network, the issue of establishing pairwise keys has been well studied in wireless sensor networks. Many pairwise key establishment schemes [20, 21, 22] allow two nodes to establish a pairwise key on the fly as long as they know each other's id. In our work, we choose the Blundo scheme [23] since it provides clear security guarantee and simplifies our presentation. In the Blundo scheme, a bivariate symmetric polynomial $f(x, y)$ with degree of $t$ is chosen in advance and $f(i, y)$ is preloaded on sensor $i$. The pairwise key with node $j$ on $i$ can be generated by evaluating the function $f(i, j)$. The scheme provides unconditional secrecy if no more than $t$ nodes collude. For the storage cost, a node needs to store a univariate polynomial represented by $t + 1$ coefficients. The size of a coefficient is the same as that of a symmetric key. For example, if a sensor network wants to tolerate the compromises of tens of nodes, it needs to store tens of coefficients. The size of a typical key is 8 or 16 bytes [24]; hence, each node needs to store hundreds of bytes of keying material. This storage overhead is affordable for low-end sensor motes with 4KB RAM.

Last, we admit that jammer localization and identification for WSNs are still open issues. In this paper, we assume the software-based attestation technique [25, 26, 27, 28, 29, 30, 31, 32, 33] can identify node compromises although some challenges remain [29]. Basically, the attestation procedure applies a challenge-response exchange between two nodes. A verifier node sends a randomly-generated number as the challenge to an interrogated node. Based on this challenge, the interrogated node traverses its memory in a pseudorandom fashion while recursively computing a cryptographic checksum over traversed locations. It responds to the verifier with the final checksum. The verifier can validate the response since it knows the expected unchanged memory image and can locally precompute the correct checksum. The traversal algorithm is designed in a way that the compromised node responses with a wrong answer or returns the response too late. There are many other way to identify the compromised sensors. For example, RF fingerprinting for sensor nodes [34] and jammer localization [35].

## 2.2 Attacker Model

We assume that the attacker can compromise a few nodes to obtain confidential information such as group key which is used to derive the channel used by all the sensor nodes. We also assume that the attacker launches jamming through the compromised sensors. That is, the jammer has the same physical capability in terms of power and frequency band as the normal sensor. There are two reasons for this assumption. First, it is obvious that if the jammer is a high-power, broadband capable device, it is impossible to construct a jamming-resilient sensor network with the low-end sensors. Second, powerful jammers can be easily detected by defenders since they violate the normal communication rules. However, insider jamming is supposed to be more stealthy. Nevertheless, we assume the compromised sensors launch signals as strong as possible to maximize the attacker's damage.

In our attack model, the attacker has the following parameters:

- *Jamming Probability:* The attacker can jam up to $n$ channels with probability $p_i (1 \leq i \leq n)$ for channel $i$.
- *Channel Switch Latency ($t_l$):* The attacker needs time $t_l (t_l > 0)$ to switch from one channel to another. From our experiment in Chapter 6, the typical latency is 34 ms for Mica2 mote. For MicaZ mote [36], $t_l$ is 132 us. For 802.11 WiFi [37], the measurement result of $t_l$ for the Atheros chipset is 7.6ms.
- *Sensing and Jamming Duration:* We consider two types of jammers: active and reactive. For active jammers, attackers launch jamming signal immediately without sensing. We denote the jamming duration as $t_j$. For reactive attackers, attackers sense the traffic before jamming. Active attackers do not sense, so they may jam some channels that have no traffic. As such, active attacks have shorter response time but are not energy efficient; on the contrary, reactive attacks have longer response time but are more energy efficient.

## 2.3 Design Goal

Our goal is to design security mechanisms to minimize the damages caused by the insider jammer(s). More specifically, we consider a scenario where normal nodes could be compromised and deceived as malicious insider jammers. The attacker could use any cryptographic information known by the normal nodes to facilitate the jamming attack. For example, the jammer could always predict the next channel used for communication and launch jamming signals to block the eligible network traffic. In addition, we consider a more complicated case in which multiple nodes launch jamming attacks in a coordinated way. The goal of our proposed security mechanisms is to identify the compromised nodes, construct and propagate a new group key to all non-compromised nodes under the presence of one or even multiple jammers so that the new key can be used to establish a keyed secret channel which cannot be predicted by the insider attacks, thus excluding them from the network.

Chapter 3

# Compromised Node Identification

In this chapter, we present techniques to identify all the compromised insiders. We first split the network into two equal-size groups and two nodes in different groups rotationally pair and mutually attest each other via software-based attestation. Normal (i.e., good) nodes finally identifies the insiders by majority voting. The entire procedure includes two phases. Phase I is designed to ensure that the normal node could attest every node and identify any insiders in the opposite group by network splitting, node pairing and direct attestation. Phase II is designed to identify the insider in its own group by exchanging the attestation results between groups and majority voting.

## 3.1 Phase I: Direct Attestation

Suppose some nodes are compromised and begin to jam. After jamming is detected (e.g. based on [1]) , all $N$ nodes will be aware of it. Without loss of generality, let us denote the nodes' $id$s as $1, \cdots, N$. Our scheme starts with equally splitting the network into two groups with lower $id$s $\{1, \cdots, \lfloor \frac{N}{2} \rfloor\}$ and higher $id$s $\{\lfloor \frac{N}{2} \rfloor + 1, \cdots, N\}$. Since the pairwise secret between normal nodes in different groups is unknown to the attacker, we pair those two nodes and use their pairwise key to derive a keyed secret channel. Specifically, we pair the two nodes with lowest ids in each group, the second lowest and so on. The paired parties $i \in \{1, \cdots, \lfloor \frac{N}{2} \rfloor\}$ and $i + \lfloor \frac{N}{2} \rfloor$ switch to the keyed channel $C_{i,i+\lfloor \frac{N}{2} \rfloor} = H(K_{i,i+\lfloor \frac{N}{2} \rfloor}|0)$ where $H$ is a secure hash function preloaded into the sensor nodes. Since the attacker is unaware of the channel $C_{i,i+\lfloor \frac{N}{2} \rfloor}$, the best it can do is to scan all possible channels and jam those with traffic. We apply two techniques to avoid this scan-and-jam problem. First, we try to design short messages so that traffic is less likely to be detected or corrupted by jamming. Second, if the communication is jammed, the paired nodes $i$ and $i + \lfloor \frac{N}{2} \rfloor$ will switch to another secret channel $C'_{i,i+\lfloor \frac{N}{2} \rfloor} = H(K_{i,i+\lfloor \frac{N}{2} \rfloor}|1)$ which renders the attacker to rescan all channels again.

Next, the paired nodes mutually attest each other to verify its code integrity. We first assign node $i$ as the verifier and $i+\lfloor\frac{N}{2}\rfloor$ as the interrogated node and $i$ sends a 16-bit randomly-generated challenge to its paired party $i+\lfloor\frac{N}{2}\rfloor$ by

$$M_1 = i + \lfloor\frac{N}{2}\rfloor || E_{K_{i,i+\lfloor\frac{N}{2}\rfloor}}(T|i + \lfloor\frac{N}{2}\rfloor|challenge_{i+\lfloor\frac{N}{2}\rfloor})$$

where $T$ is a 8-bit timestamp to prevent the replay attack and $challenge_{i+\lfloor\frac{N}{2}\rfloor}$ is the challenge for node $i+\lfloor\frac{N}{2}\rfloor$. Upon successfully receiving and decrypting the challenge, sensor $i+\lfloor\frac{N}{2}\rfloor$ traverses its memory to compute the 8-bit checksum. Then it replies the response, switches its role to be a verifier and sends its challenge to node $i$ by

$$M_2 = i || E_{K_{i,i+\lfloor\frac{N}{2}\rfloor}}(T|i|checksum_{i+\lfloor\frac{N}{2}\rfloor}|challenge_i).$$

The verifier $i$ then compares the checksum with its precomputed version. A malicious insider is identified if the two checksums are not equal or $M_2$ could not be returned timely. Last, node $i$ switches its role to be interrogated, computes and replies the response by

$$M_3 = i + \lfloor\frac{N}{2}\rfloor || E_{K_{i,i+\lfloor\frac{N}{2}\rfloor}}(T|i + \lfloor\frac{N}{2}\rfloor|checksum_i)$$

After receiving $M_3$, node $i+\lfloor\frac{N}{2}\rfloor$ verifies node $i$ similarly.

If node $i$ is malicious, it may not initiate $M_1$ to avoid the attestation. Then, node $i+\lfloor\frac{N}{2}\rfloor$ will switch to be a verifier and launch $M_1$ after a timeout occurs. Note that a compromised node does not have to launch jamming to be detected. It can hide, for example, to steal any cryptographic information. However, the software attestation works as well since the code of this hidden insider is different from the normal sensor's.

In TinyOS 2.0.1, the MAC frame includes a data payload of 28bytes, a header of 10 bytes and a CRC of 2 bytes. Given the typical node ID of 1 byte, the challenge of 2 bytes, the checksum of 1 byte and the transmission rate of 19.2Kbps for Mica2, the transmission time for messages $M_1$, $M_2$ and $M_3$ are approximately 7.08ms, 7.5ms and 6.67ms. Since all these message exchanging times are less than $t_l$=34ms for Mica2, it makes the scan-and-jamming attack difficult.

After this initial mutual attestation, node $i$ pairs with the next node $i+1+\lfloor\frac{N}{2}\rfloor$ in the opposite group, and they two mutually attest each other again. After $\lceil\frac{N}{2}\rceil$ attestations, each node can meet and verify all nodes in the opposite group. Figure 3.1 illustrates the attestation phase for a network of 7 nodes.

## 3.2 Phase II: Result Sharing and Compromise Identification

In Phase I, normal nodes can meet and directly attest each node in the opposite group and therefore the insiders in their opposite group could be identified. It is possible that a node can attest all others in order to identify all the insiders. However, we noticed that the software-

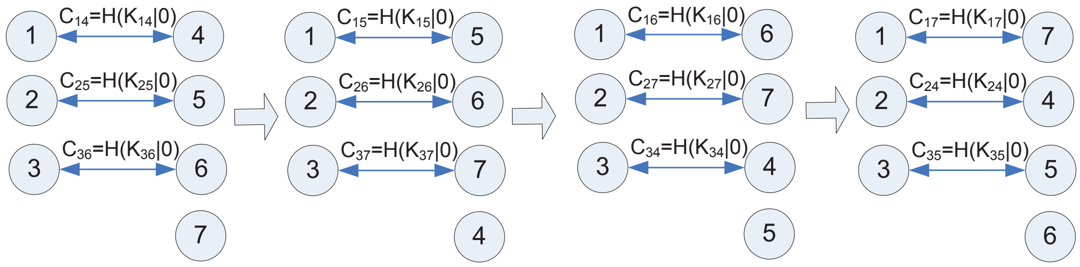

**Figure 3.1.** Attestation process for a network of 7 nodes

based attestation is both time and power consuming procedure. To avoid the attestation and still identify all insiders for every normal sensor, we design protocol in this phase to exchange those attestation results to identify any insiders in their own group. To achieve that, each node first requests the attestation results of its own group from the opposite. Then, it identifies the malicious insiders based on those results by majority voting.

Specifically, we apply the similar pairing scheme between two groups. The node $i$ pairs the node $i + \lfloor \frac{N}{2} \rfloor$ and sends its encrypted results by

$$M_4 = i + \lfloor \frac{N}{2} \rfloor || E_{K_{i,i+\lfloor \frac{N}{2} \rfloor}} (T|i + \lfloor \frac{N}{2} \rfloor | results_i)$$

where $results_i$ is the attestation results on node $i$. As a confirmation, node $i + \lfloor \frac{N}{2} \rfloor$ returns its own results by $M_5$

$$M_5 = i || E_{K_{i,i+\lfloor \frac{N}{2} \rfloor}} (T|i|results_{i+\lfloor \frac{N}{2} \rfloor})$$

Last, node $i$ replies a confirmation $M_6$

$$M_6 = i + \lfloor \frac{N}{2} \rfloor || E_{K_{i,i+\lfloor \frac{N}{2} \rfloor}} (T|i + \lfloor \frac{N}{2} \rfloor)$$

and pairs the next node in the opposite group.

For a network of 16 nodes with each group of 8 nodes, the attestation results could be encoded into 1 byte in which each bit represents one node. The transmission time for $M_4$, $M_5$ and $M_6$ are approximately 6.67ms, 6.67ms and 6.25ms for Mica2. The entire communication duration for one exchange is 19.59ms, which is smaller than $t_l$ for Mica2. Similarly, this makes the scan-and-jamming attack difficult.

Last, if every node can honestly report its results, the identification can stop and finish herein. Unfortunately, the attacker can forge the results. To tolerate those misleading information, each node applies the majority voting policy for the identification of malicious insiders. That is, each node identifies a malicious insider residing in its own group if more than half of the nodes in the opposite group accuse that node.

**Theorem 3.2.1.** *If the number of the compromised node is less than* $\lfloor \frac{N}{2} \rfloor / 2$*, our identification scheme can guarantee to detect all of them.*

*Proof.* In order to identify any insider in one's own group, the majority voting requires the number of normal nodes should be large than half of the group size for both two groups. Here, we consider the worst case in which all the insiders are split into one group. We further consider this group is the lower id group since it has fewer nodes when the network size is odd. To work with this worst case, we need more than half of the nodes in this group to be normal ($> \lfloor \frac{N}{2} \rfloor / 2$). In other words, less than $\lfloor \frac{N}{2} \rfloor / 2$ nodes could be the compromised insiders. So, our identification scheme could tolerate up to $\lfloor \frac{N}{2} \rfloor / 2 - 1$ compromised nodes.

□

The above identification scheme needs about $N$ rounds to complete. However, fast identification and low communication may be required for a dense network. In this case, the "mutual suicide" idea [38] could be applied here. Basically, we can pair and mutually attest paired parties only once. Then, the attestation results are exchanged in a similar way. Since the malicious insiders could accuse good nodes, a fast revocation mechanism is to mark both the accusor and the accused nodes as malicious when an accusation happens. In the other case, if paired nodes are both compromised, they may simply claim their peer as a normal sensor, leading to false negative. To deal with this issue, we can shift the order of pairing for attestation if jamming recurs. In this approach, we can greatly reduce the computation and communication overhead but at the cost of sacrificing the equal number of good sensors. We expect that the above solution is still affordable for dense networks.

Chapter 4

# The Split-Pairing Scheme for A Single Jammer

The basic idea of our scheme is to split the jammed nodes into two groups, each of which works on a different channel. At any given time the attacker can jam only one channel, or cannot jam any channel when it is switching channel. Since there is only one jammer, there must be a group which is free of jamming at any time, and nodes in this group can propagate a new group key. The scheme consists of three phases. Phase I deals with how to split the network into two groups and assign communication channels to them. Then, we design a protocol for intra-group key propagation in phase II to ensure that all nodes in one of the two groups will share the new key at the end of this phase. In phase III, nodes in two groups are paired to propagate the new key from one group to the other.

## 4.1 Phase I: Channel Splitting

Suppose all nodes work on channel $C_0$ originally and $r$ channels available to switch. Starting from time $t_0$, one node is compromised and it starts to jam channel $C_0$. After the jammer has been detected and identified, all $N$ non-compromised nodes will be aware of it. They will switch to new channels in a distributed way. Without loss of generality, let us denote their ids as $1, \cdots, N$. In this phase, nodes with lower $ids$ $\{1, \cdots, \lfloor \frac{N}{2} \rfloor\}$ switch to channel $C_1 = H(C_0|0)$, and nodes with higher $ids$ $\{\lfloor \frac{N}{2} \rfloor + 1, \cdots, N\}$ switch to channel $C_2 = H(C_0|1)$, where $H$ is a secure hash function preloaded in the sensor nodes and maps to one of $r$ channels.

The channel switching and splitting process is illustrated in Figure 4.1 and Figure 4.2. When node $A$ is identified as a compromised node, nodes with lower $ids$, i.e., nodes 1, 2 and 3, switch to channel $C_1 = H(C_0|0)$ and nodes in higher $ids$, i.e., nodes 4, 5, 6 and 7, switch to channel $C_2 = H(C_0|1)$.

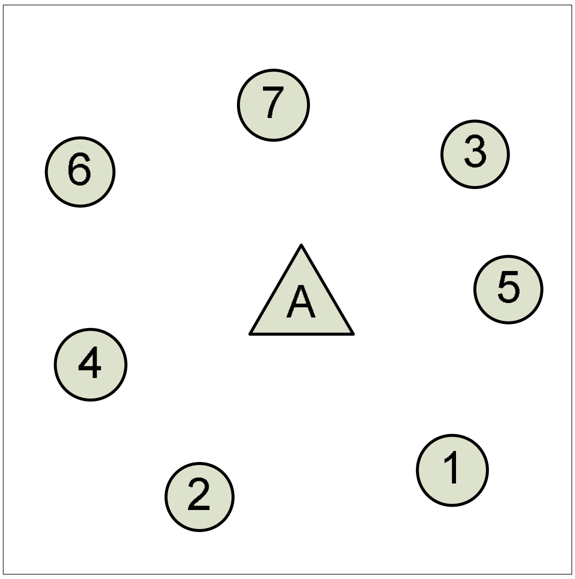

**Figure 4.1.** Network topology.

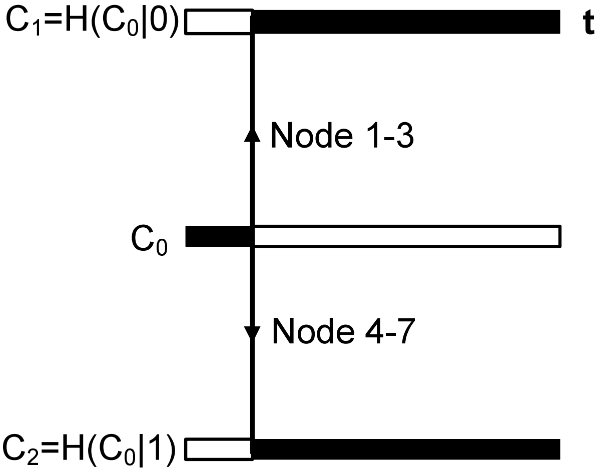

**Figure 4.2.** The illustration of channel switch for key reestablishment.

## 4.2 Phase II: Jamming and Key Propagation within A Group

Once channel splitting finishes, the node with the smallest *id* in each group acts as the group leader to generate a new group key, which is then propagated within each group. That is, node 1 is the group leader of the first group, and node $\lfloor \frac{N}{2} \rfloor + 1$ is the group leader of the second group. Then, the new group key $K$ is generated based on the pairwise key $K_{1,\lfloor \frac{N}{2} \rfloor+1}$ shared between two leaders by applying $K = F_{(K_{1,\lfloor \frac{N}{2} \rfloor+1})}(0)$, where $F$ is a pseudorandom function. The desirable advantage is that the new group key is generated without any communication and thus the jammer cannot interfere it. Since the key $K_{1,\lfloor\frac{N}{2}\rfloor+1}$ is unknown to the attacker, it cannot predict the new group key although the pseudorandom function $F$ is publicly known.

Once the group leaders have generated the same new key, they will only need to propagate the new key to all their group members. Clearly, the new key has to be encrypted to preclude the compromised attacker from eavesdropping. To propagate $K$, the simple solution is to let the group leader unicast the key to each group member. To save communication cost, we use reliable broadcast. Specifically, the group leader broadcasts the key to all group members and gets the acknowledgements (acks) from each of them. The group leader will retry if any acks are missing.

Specifically, in the broadcast message $M_1$, the new key $K$ is encrypted by different pairwise keys shared between the leader and each member. For group 1, node 1 broadcasts $M_1$ and starts a timer

$$M_1 = Mapping||E_{K_{1,2}}(T|2|K)||...||E_{K_{1,\lfloor\frac{N}{2}\rfloor}}(T|\lfloor\frac{N}{2}\rfloor|K).$$

where $T$ is the *timestamp* to prevent replay attacks. After successfully receiving and decrypting $M_1$, node $i$ sends back a confirmation message to the group leader 1 or $\lfloor\frac{N}{2}\rfloor + 1$. For group 1, node $i$ sends back

$$M_2 = E_{K_{1,i}}(T|i|K)||i.$$

If any confirmations are missing due to jamming or collision, a new key propagation message $M_1$ is reconstructed and sent out after timeout. Only unconfirmed nodes are required to send back confirmations to reduce the traffic and collision. This procedure continues until all confirmations are received by the leader.

In TinyOS 2.0.1, the MAC layer frame structure has a data payload of 28bytes. Given a typical key size of 8 bytes [24], one frame can include at most 3 encryptions of a group key. Also, node ID is 1 byte and encryption id is 1byte. For Mica2 with transmission rate of 19.2Kbps, the transmission time for $M_1$ with three encryptions (i.e., the subgroup size is 4 counting the leader) is $\tau_1 \approx \frac{(8Bytes\cdot 3)+1Byte)}{19.2Kbps} = 10.42ms$ and for $M_2$ is $\tau_2 \approx \frac{(8+1)Bytes}{19.2Kbps} = 3.75ms$, the one-round communication time will be $\tau_0 = \tau_1 + 3 \cdot \tau_2 = 21.67ms$. It is worth noting that the one-round time $\tau_0 < t_l$, where $t_l$=34ms for Mica2. That is, for a group of size 4, a keying message can be transmitted successfully before the jammer can switch to another channel which includes the following time: switching to another channel, jamming a minimal packet, and returning. If the group size is larger than 4, we have to embed multiple encryptions into two or more broadcast messages. Suppose that the key propagation time in one group without jamming is $T_{kr}$. Given the number of nodes in each group and the packet loss rate, we can compute the expected message transmission round $E[Y]$ based on [39]. Hence, $T_{kr} = \tau_0 E[Y]$.

Unfortunately, in practice the key propagation messages $M_1$ and $M_2$ can be corrupted by jamming and the actual key propagation needs more time. In order to estimate this time, we consider the optimal jamming strategies in which the attacker can maximize the total key propagation time for this phase. Since the hash function $H$ and the original channel $C_0$ are publicly known, the attacker knows channels $C_1$ and $C_2$ by computing the same hash values. However, the attacker has only one wireless interface and thus at any given time it can jam only one channel or neither of them when it is switching channel. This means that at least one of two groups are free of jamming at any time, and this group can execute the above key propagation protocol. In other words, the attacker cannot simultaneously prevent the key reestablishment for both groups and the best it can do is to prolong the key propagation time of phase II.

**Theorem 4.2.1.** *The optimal jamming strategy for a single jammer is to actively jam two channels with an equal probability.*

*Proof.* We denote $T_j$ as the overall jamming duration in phase II. The total key propagation time for Phase II is $T$. In our system model, $p_i$ is the probability for the attacker to launch jamming on channel $i$. For group $i$, the time it is free of jamming is $T_i = T - T_j p_i$. In order to maximize the key propagation time, an optimal attacker would minimize the maximum free-of-jamming time for all groups. Here we consider the case of two groups $i = 1, 2$. We formalize the optimization problem as follows:

$$\begin{aligned} &\min_{p_1,p_2} \max_{p_1,p_2} (T - T_j p_1, T - T_j p_2) \\ &s.t. p_1 + p_2 = 1 \\ &\quad p_{1,2} \geq 0 \end{aligned} \tag{4.1}$$

If $T - T_j p_1 \geq T - T_j p_2$, we have $p_1 \leq p_2$. Then, the problem is simplified to:

$$\min_{p_1 \geq 0, p_1 \leq p_2, p_1 + p_2 = 1} (T - T_j p_1) \tag{4.2}$$

The solution is $p_1 = p_2 = 0.5$. Similarly, we have the same result when $T - T_j p_2 \geq T - T_j p_1$. ☐

To estimate the key propagation time $T$ in one group, we consider a typical optimal case for the attacker where the attacker alternates between two channels and jams each channel for a period of $t_j$. If it starts with group 1, group 2 will be able to complete key propagation ahead of group one or at the same time as group one. We consider the worst case in which each jam leads to a retransmission. The number of retransmissions for one group due to jamming is $\frac{T}{2(t_j + t_l)}$ and the time for retransmission is $T_{jr} \approx \frac{T}{2(t_j + t_l)} \tau_0$. The finish time $T$ is

$$T \approx T_{kr} + T_{jr} = \frac{2(t_j + t_l)}{2(t_j + t_l) - \tau_0} T_{kr}. \tag{4.3}$$

## 4.3 Phase III: Key Propagation between Groups

After one group finishes the key propagation, this group excludes the attacker by the keyed secret channel. It is possible that the attacker chooses to jam group 2 all the way so that few nodes in group 2 can obtain the new group key. If so, nodes in group 1 can propagate the group key to nodes in group 2 by pairing one node in group 1 with another node in group 2. For simplicity, we pair the two nodes with the lowest *id*s in two groups, the second lowest and so on. If $N$ is

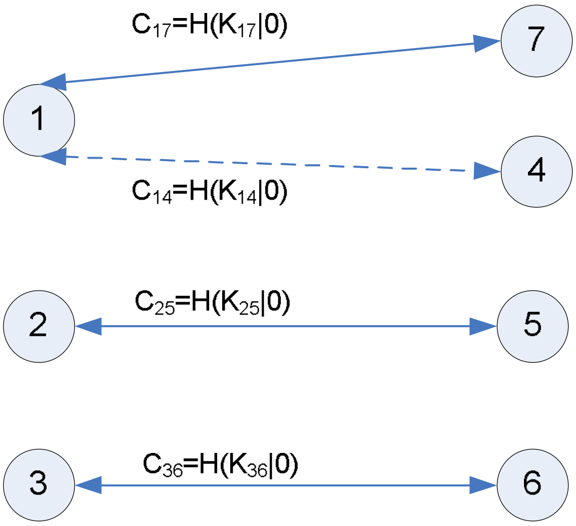

**Figure 4.3.** Pairing for key propagation between groups

odd, group 2 will have one more node left. We pair it to node 1 since the two lowest id nodes, 1 and $\lfloor \frac{N}{2} \rfloor + 1$, are group leaders they do not need to communicate in this phase. Therefore, node 1 is actually only responsible for that extra node. That is better than pairing this extra node to any other node in group 1, which is already paired. In Figure 4.1, we pair node 1, 4; 2, 5; 3, 6 and 1, 7, as shown in Figure 4.3.

In order to safely propagate the new group key from one group to the other, paired parties in different groups communicate in a keyed secret channel based on their pairwise key. Suppose node $i(1 \leq i \leq \lfloor \frac{N}{2} \rfloor)$ and $j(\lfloor \frac{N}{2} \rfloor + 1 \leq j \leq N)$ are paired and they share a pairwise key $K_{ij}$. Then, they switch to channel $C_{ij} = H(K_{ij}|0)$. In some rare cases, two or more pairs are hashed to the same channel due to the limited channel resource. We use random back-off mechanism to avoid collision.

After channel switching, all nodes that have received the new group key switch to the reception mode and wait for a request from their paired parties. For the key propagation, since phase II can guarantee that nodes in one group have correctly received the new group key, two cases may occur for the pair $i$ and $j$. First both $i$ and $j$ have correctly received the new group key. Then, $i$ and $j$ do not communicate to save energy and avoid unnecessary traffic and collision. Second, either $i$ or $j$ has received the new group key. Without loss of generality, we assume that $i$ has received the new key but $j$ has not. Then, $j$ initiates key reestablishment by sending a message $M_1$ to node $i$:

$$M_1 = T||j||MAC_{K_{ij}}(T|j).$$

where $T$ is a timestamp and MAC is a message authentication algorithm. Node $i$ replies to $j$ with message $M_2$:

$$M_2 = E_{K_{ij}}(T|i|K).$$

node $j$ decrypts $M_2$ to obtain $K$. Note that $M_2$ does not include a separate MAC because the knowledge of $T$ and $i$ serves as a way of (weak) authentication. At last, node $j$ returns a confirmation message $M_3$ to $i$:

$$M_3 = E_{K_{ij}}(T|j|K).$$

Given a typical size of 4-byte MAC [24] all three messages are short and the time for this exchange for the Mica2 mote is $\tau_3 < \frac{8Bytes \cdot 3}{19.2Kbps} = 10ms$, which can be completed within $t_l$. In other words, as long as the attacker is jamming a channel other than $C_{ij}$ at the beginning of this phase, inter-group communication of pair $ij$ can complete without jamming. To deal with some rare case that the attacker has chances to jam the communication on pair $ij$, paired nodes maintain a timer and the timeout can be set to $\tau_3$ or a bit more to tolerate lost time synchronization. Since nodes can detect failed packet [19], if any exchange message is detected to be failed, paired parties stop the exchange protocol and wait for a timeout. When a timeout occurs, they switch to another channel $C'_{ij} = H(K_{ij}|1)$, set timer and retry until one party can successfully propagate the new group key to the other.

After all nodes obtain the new group key $K$, they can start the legitimate communication on the secret channel $C_{new} = H(K|0)$. The attacker may compromise another node to obtain $K$. The above 3-phase procedure repeats to reestablish a new key and restore the network. Note that a revoked attacker may scan all the channels to discover and jam the new channel like an outsider jammer. In this case, all the nodes can switch to $C_{new} = H(K|1)$ (then $C_{new} = H(K|2)$ if it happens again). No group rekeying is necessary.

Chapter 5

# Tree-Based Scheme for Multiple Colluding Jammers

In this section, we describe our tree-based recovery scheme which can be used to deal with multiple colluding jammers.

## 5.1 Motivations and Overview

We consider $m$ malicious insiders where $m \geq 2$. Under a single jammer, the split-pairing scheme derives a new group key from a pairwise secret between two group leaders. In the multiple-jammer case, the split-pairing scheme can work successfully only if the network can be split into at least $m+1$ groups to ensure that one or more group(s) are free of jamming. Moreover, group leaders need to agree on the same group key without communication or interaction. This can be achieved if each node is preloaded with a $m$-variate symmetric polynomial [23]. However, each $m$-variate symmetric polynomial with a degree of $t \geq m$ has $\binom{t+m}{m}$ monomials; thus, the storage cost is $\binom{t+m}{m} \cdot 8$ bytes. For the case of 6 jammers, we need to split the network into at least 7 groups and each node should be preloaded with a 6-variate symmetric polynomial and the storage cost is at least 7KB, which is quite high to current sensors. In addition, the multiple jammers may coordinate together to jam multiple channels simultaneously, which makes it more difficult to recover.

Because of the above concerns, we propose a tree-based scheme to deal with multiple jammers. The tree-based scheme is more adaptive and can tolerate multiple colluding attackers without increasing the storage overhead. It borrows the idea from the logical key tree construction [40, 41] as in Figure 5.1. The root key located at level 0 is shared by all nodes and the lowest leaves are nodes at level $l = 3$. In our scheme, we use the binary key tree since establishing pairwise keys does not generate extra storage overhead and can be easily achieved by the Blundo scheme. Our goal is to construct the logical key tree in a bottom-up manner under jamming so that all

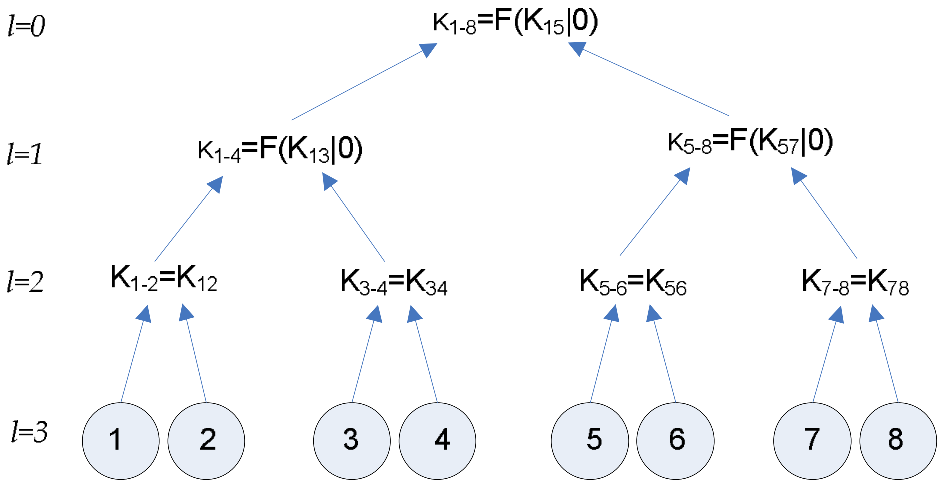

**Figure 5.1.** Key tree example.

jammed nodes can finally share the root key to derive a common secret channel. To achieve this, we first divide the network into subgroups, each of which consists of two nodes. Subgroup leaders generate subgroup keys and propagate them to their members on secret channels determined by the pairwise keys shared between the leaders and members. Then, two sibling subgroups are merged into a larger subgroup of four nodes and new keys are derived and propagated within each subgroup again. With the progress of this scheme, all nodes share a common key (root key) and work on the same secret channel. Although our scheme looks similar to to the one proposed for wired networks [40], there are two key differences. First, we do not assume the existence of secure channels between each pair of nodes; rather, we construct such channels based on pairwise keys that can be efficiently established. Second, since no secure channels exist between subgroups in advance, in our case subgroup leaders must derive subgroup keys individually without any communication or interaction.

## 5.2 The Protocol

To better understand the tree-based scheme, we first show an example for a network of 8 nodes as in Figure 5.2. To simplify our presentation, we define two terms.

- *Subgroup Key:* A key shared by subgroup members. We denote $K_{i-j}$ as a subgroup key shared by nodes whose ids are between $i$ and $j$. For generality, we may use the subgroup key notation $K_{i-(i+1)}$ to denote a pairwise key $K_{i,i+1}$.

- *Channel Key:* A key used to derive a secret channel. If a channel key is a pairwise key $K_{ij}$, the channel between node $i$ and $j$ is $C_{ij} = H(K_{ij}|0)$. If the channel key is a subgroup key $K_{i-j}$, the channel shared among nodes $i$ to $j$ is $C_{i-j} = H(K_{i-j}|0)$ with $H$ being a secure hash function for channel derivation.

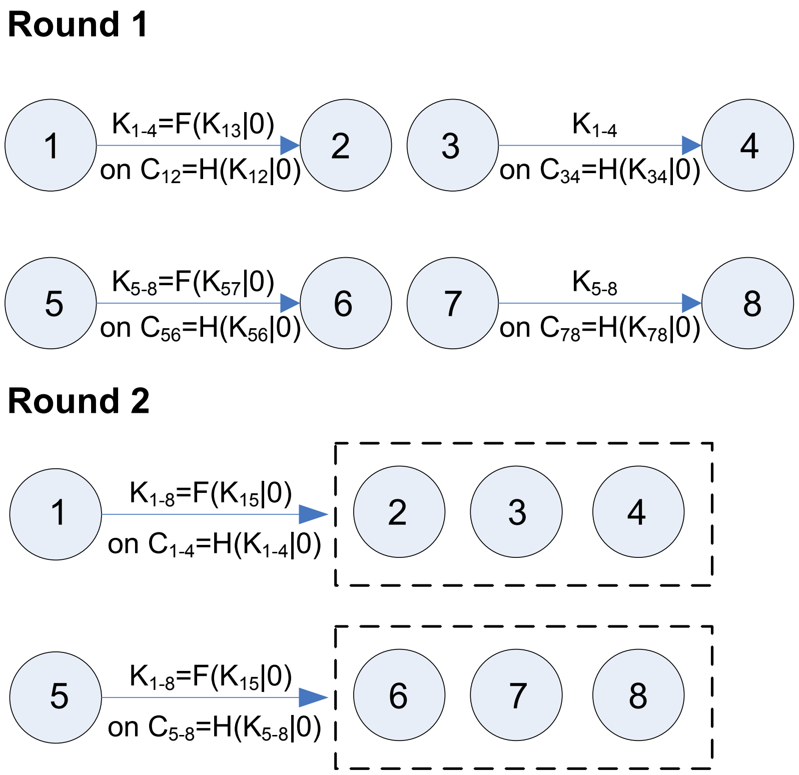

**Figure 5.2.** An example for our tree-based scheme.

Suppose all nodes are identified with ids $1, \cdots, N$ ($N = 8$ in our example) as before and a set of channels $\{C_1, C_2, \cdots, C_r\}$ are available to switch. Figure 5.2 shows how our tree-based scheme works and the details are as follows.

1. Members 1 and 2 agree on secret channel $C_{12}$ by channel key $K_{12}$.
   Members 3 and 4 agree on secret channel $C_{34}$ by channel key $K_{34}$.
   Members 5 and 6 agree on secret channel $C_{56}$ by channel key $K_{56}$.
   Members 7 and 8 agree on secret channel $C_{78}$ by channel key $K_{78}$.

2. Members 1 and 3 derive subgroup key $K_{1-4} = F_{K_{1,3}}(0)$ and distribute it to subgroup member 2 and 4, respectively.
   Members 5 and 7 derive subgroup key $K_{5-8} = F_{K_{5,7}}(0)$ and distribute it to subgroup member 6 and 8 respectively.

3. Members 1,2,3,4 agree on secret channel $C_{1-4}$ by channel key $K_{1-4}$.
   Members 5,6,7,8 agree on secret channel $C_{5-8}$ by channel key $K_{5-8}$.

4. Members 1 and 5 derive subgroup key $K_{1-8} = F_{K_{1,5}}(0)$ and distribute it to subgroup mem-

bers 2,3,4 and 6,7,8 respectively.

5. Members 1,2,3,4,5,6,7,8 agree on secret channel $C_{1-8}$ by channel key $K_{1-8}$.

**Algorithm 1** Key-Tree based Scheme

```
Input: a set nodes numbered $1, ..., N$ and their pairwise secret keys;
Procedure:
$k = 1$;
repeat
    Select Channel Key = $K_{(2^k \cdot i+1)-(2^k \cdot (i+1))}$ for $i = 0, 1, ..., \lfloor \frac{N}{2^k} \rfloor$;
    Switch to Channel $C_{(2^k \cdot i+1)-(2^k \cdot (i+1))}$;
    For node $2^k \cdot i + 1$, generate subgroup key $K_{(2^{k+1} \cdot i+1)-(2^{k+1} \cdot (i+1))} = F_{K_{2^{k+1} \cdot i+1, 2^k \cdot (2i+1)+1}}(0)$;
    For node $2^k \cdot i + 1$, propagates new subgroup key to nodes $2^k \cdot i + 2$ to $2^k \cdot (i + 1)$;
    $k$++;
until $k == \lceil \log_2 N \rceil$
```

The generation of all subgroup keys are offline (i.e., without interactions or communication between leaders). The details of our tree-based scheme are shown in Algorithm 1. The input includes all jammed nodes $1, \cdots, N$, the pairwise key $K_{ij}$ for $i, j \in \{1, \cdots, N\}$ and a set of channels $C = \{C_1, C_2, \cdots, C_r\}$. For round $k$, all nodes switch to channel $C_{(2^k \cdot i+1)-(2^k \cdot (i+1))}$ by using channel key $K_{(2^k \cdot i+1)-(2^k \cdot (i+1))}$. Then, the node with id $2^k \cdot i + 1$ generates the subgroup key and broadcasts it to all members. This procedure ends when all nodes receive the root key in the logical key tree as the new group key. In case $N \neq 2^j$ for some $j = 1, 2, ...$, some node may not have the channel key and it does not need to propagate the key in that round. For example, if $N = 9$, we need 3 rounds. Node 9 will not do channel switching or key propagation in all 3 rounds. It only generates root key at the last round by $F_{K_{1,9}}(0)$.

For key propagation, we apply a similar protocol as our split-pairing scheme. For the $k$th round, node $2^k \cdot i + 1$, $i = 0, 1..., \lfloor \frac{N}{2^k} \rfloor$ initiates key reestablishment by broadcasting message $M_1$ to members $2^k \cdot i + 2$ to $2^k \cdot (i + 1)$ on channel $C_{(2^k \cdot i+1)-(2^k \cdot (i+1))}$:

$$M_1 = Mapping || E_{K_{(2^k \cdot i+1)-(2^k \cdot (i+1))}}(T | K_{(2^{k+1} \cdot i+1)-(2^{k+1} \cdot (i+1))}).$$

Group members $i \in \{2^k \cdot i + 2, \cdots, 2^k \cdot (i + 1)\}$ reply to group leader $2^k \cdot i + 1$ with confirmation message $M_2$:

$$M_2 = E_{K_{(2^k \cdot i+1)-(2^k \cdot (i+1))}}(T | i | K_{(2^{k+1} \cdot i+1)-(2^{k+1} \cdot (i+1))}).$$

The group leader $2^k \cdot i + 1$ decrypts message $M_2$ to check which member has correctly received $M_1$. If some confirmation is not correctly received by the leader, the leader retransmits the subgroup key.

Finally, the subgroup key needs to be replaced if some member joins or leaves. We can apply the same approach used in [40]. Since the number of nodes in a one-hop network is not that large, based on the analysis of overhead below, the cost of rerunning our scheme is still affordable. In practice, we do not expect node compromise is a frequent event. By simply rerunning the scheme, some complicated problems such as tree rebalancing could be avoided.

## 5.3 Performance Analysis

- *Computation Cost.* We consider two computations, computing hashes $H$ and executing pseudorandom function $F$. For hashing, all nodes compute hash function $H$ for channel selection at most once in each round. The scheme can finish in $\lfloor \log N \rfloor$ round. The overall computation cost for hashing is $C_{hash} = O(N \log N)C_H$ where $C_H$ is the computation overhead for hashing once. For the pseudorandom function, approximately $\lceil \frac{N}{2^k} \rceil$ nodes involve the subgroup key generation in the $kth$ round. The overall computation cost for $F$ is $C_{PRF} = O(2^{\lceil \log_2 N \rceil})C_F \approx O(N)C_F$, where $C_F$ is the computation cost for executing the pseudorandom function once. Hence, the overall computation cost is $C_{hash} + C_{PRF} = O(N \log N)C_H + O(N)C_F$. Note that in our schemes, we implement the pseudorandom function with CBC MAC.

- *Communication Cost.* We denote $C_{broadcast}$ as the cost of the subgroup broadcast. In the tree-based scheme, the number of broadcasts is approximately the number of parent nodes in the logical key tree. Hence, the communication cost is $O(N)C_{broadcast}$.

- *Storage Overhead.* A regular sensor only needs to store its univariate polynomial with degree of $t$, i.e. $(t+1)$ coefficients. In addition, each node stores all ids in the network. Even for a dense network with tens of sensors within one-hop range, one byte is enough to represent an id. Hence, the storage overhead is the same as the split-pairing scheme.

Compared with the split-pairing scheme, it is worth noting that the tree-based scheme can tolerate multiple colluding insider attackers. Since channel keys are based on the secure pairwise keys or newly reestablished subgroup keys, colluding insider attackers cannot predict those keyed channels. Additionally, since the number of channels required in the tree-based scheme is $\lfloor \frac{N}{2} \rfloor$ at most, our scheme can tolerate up to $r - \lfloor \frac{N}{2} \rfloor$ attackers to launch jamming simultaneously with $r$ being the number of channels available. If multiple access techniques are used, we expect to use less channels and tolerate more attackers.

Chapter 6

# Performance Evaluations

In this section, we first describe our testbed, and then present the evaluation results of the identification scheme and the recovery scheme.

## 6.1 Testbed Configurations

The testbed consists of 19 Mica2 sensor motes [18] deployed at fixed locations in an indoor laboratory. Each sensor mote has a 902-928MHz Chipcon CC1000 radio [42], which is divided into 32 800KHz channels. Each mote is within the communication range of other motes and the data transmission rate is 19.2Kbps. All motes run TinyOS version 2.0.1 [43].

### 6.1.1 The Implementation of Channel Switching

In TinyOS 2.0.1, the module *CC1000ControlP* provides interface *CC1000Control* and command *tuneManual()* to control channel switching. Since Chipcon CC1000 uses a digital frequency synthesizer, a programmable register can be used to change frequency.

### 6.1.2 Implementation of the Jammer

In TinyOS 2.0.1, the implementation of the mote-to-mote communication depends on the radio chip. For Chipcon CC1000, the communication is implemented in two modules: *CC1000CsmaP* and *CC1000SendReceiveP* under directory *tinyos-2.x/tos/chips/cc1000. CC1000CsmaP* provides CSMA and low-power sensing logic, whereas *CC1000SendReceiveP* provides the send-and-receive logic for CC1000 radio. The send-and-receive logic includes Request-to-Send (RTS) and Clear-to-Send (CTS) commands. A node starts data transmission after receiving CTS. CSMA provides two mechanisms for media access control: random backoff and carrier sensing. The random backoff mechanism is used to reduce further collisions where the backoff delay is randomly set to [1,32] bytes initially. The sensing mechanism is used to determine if there is any ongoing communication on the channel. It requires the air interface to read received signal strength indication (RSSI) every 80 microseconds up to 5 readings. If all 5 readings are above a threshold, the backoff mechanism is activated. After each RSSI reading, the threshold is updated and thus it is is adaptively changed with the current channel condition.

Although a jammer can be a regular sensor mote, we have to make some changes in the implementation so that it can jam the communication channel. We disable the random backoff and the sensing mechanisms so that the jammer can send out packets arbitrarily to jam the channel. Specifically, we use command *disableCca()* provided by the *CsmaControl* interface in module *CC1000CsmaP* to bypass the media access control. We let the jammer's air interface stay in the transmission mode by using *enterTXState()*. We change the send-and-receive logic so that the jammer always receives CTS after sending a RTS.

In order to explore the impact of jamming duration, we bypass the MAC layer and directly use the command *writeByte()* provided by the interface *HplCC1000Spi*. In this way, the shortest jamming time can be as low as one byte ($t_j \approx 0.42ms$). For longer jamming duration, we have to increase the maximum message size defined in *message.h*, so that the jamming signal can last as long as 100ms.

#### 6.1.2.1 Implementation of the Software-based Attestation

To implement the software-based attestation, we need to randomly access the non-volatile storage for a Mica2 Mote. In TinyOS 2.0.1, the implementation depends on the flash chip. For Mica2 mote, it uses *AT45DB* chip and the corresponding read-and-write operations are implemented in the component *At45dbC* under the directory *tinyos-2.x/tos/chips/at45db*. For memory traversals, we randomly generate two numbers, one as a page number and the other as a page offset, by using the challenge as the seed. Then, we use the command *read()* provided by the interface *At45db* to access the memory and compute the 8-bit checksum by XOR operation.

### 6.1.3 Performance Metrics

Since we assume that the physical device has channel switching latency, we first measure the switching latency for Mica2 motes. For the identification phase, we focus on average number of retransmissions. To compare the performance under different cases, we measure the number of retransmissions for each node during one round. That is, we collect that overall retransmissions after the identification phase finished and average this number by nodes and rounds. For the recovery schemes, we measure the recovery latency instead. One reason we use different metrics is that recovery latency may not show the impact of jammer(s) since we need time synchronization in the identification phase. Another reason is that jamming different messages in recovery phase may result different number of retransmissions which will mislead our evaluation.

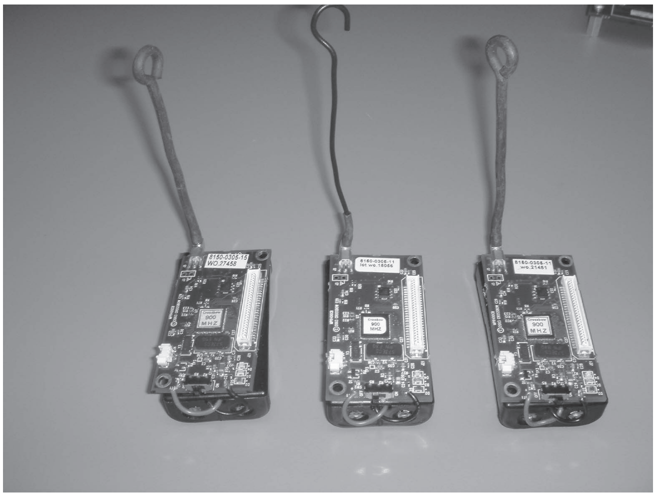

**Figure 6.1.** Three Mica2 motes are used for measuring the channel switching latency.

## 6.2 Channel Switching Latency

In our recovery scheme, we assume that the physical device has channel switching latency; thus, we first measure the switching latency for Mica2 motes.

In order to jam a communication channel, the attacker has to switch to that communication channel and send out at least a packet of 1 byte for the CC1000 chip. There is a minimum channel switching latency due to the limitations of the physical device. Three Mica2 motes as shown in Figure 6.1 are selected to measure this channel switching latency. We consider two switching modes: sequential switching and random switching. In the sequential switching mode, motes switch to one channel and send one minimum packet, then they switch to the next adjacent channel until all 32 channels are used. We consider two cases, ascendant and descendent. In the ascendant case, motes start from the lowest frequency channel to the highest, while the descendent case uses the reverse order. By running both cases 1000 times, we get the average and divide it by 32 to get the switching latency between two adjacent channels. In the random switching mode, motes randomly select the next channel. Similar to the sequential mode, we run the test 1000 times to get the switching latency between two arbitrary channels. As shown in Figure 6.2, the switching latency is independent of the channel switching mode, and it is around 34ms for all three motes.

## 6.3 The Performance of the Identification Phase

We conduct experiments to study the effectiveness of the identification phase described in Section 3. For the malicious insiders, they can either follow launch the jamming attack to corrupt the communication or follow the protocol to falsify the identification. In our implementation, we consider more advanced insiders in which each insider can both follow the protocol and jamming.

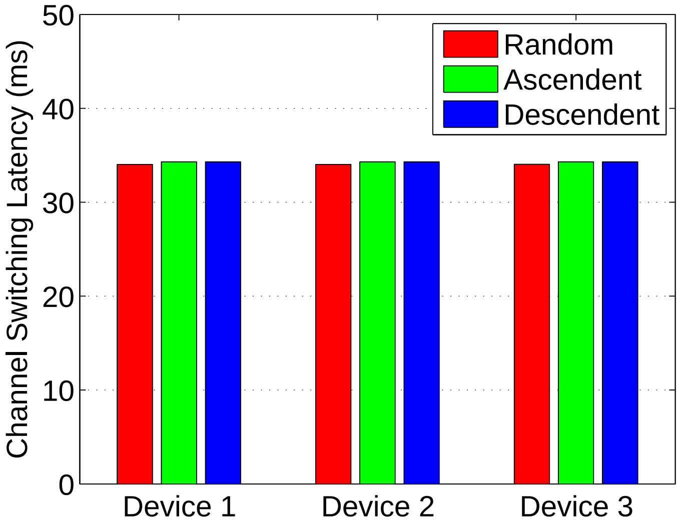

**Figure 6.2.** The channel switching latency for the three Mica2 motes.

We keep the most of normal sensor's code unchanged but only modify the code to exchange the forged results to falsify the identification. In addition, we put the same number of malicious insiders in the network dedicated for jamming. For example, for a network of 13 normal nodes and 3 malicious insiders. We put total 19 nodes in the network with 13 normal sensors, 3 insiders to falsify the identification and 3 dedicated jammers. We measure the impact of the following three parameters: the size of network, the number of jammers and the jamming duration.

### 6.3.1 The Impact of Network Size

We deploy 9, 13 and 17 nodes, in which one is selected to falsify the identification and one is the jammer, to simulate the network size of 8, 12 and 16 nodes. Since communication channels cannot be predicted, the duplicated jammer can only randomly jam one of 32 available channels at a time. We set the retransmission timeout to be 600ms since 100,000 times of memory access and the corresponding checksum response should be finished within this period [25]. Also, for the time synchronization purpose, we set the timeout for pairing once to be 4s so that the pair has a chance for three tries. For different network size, we measure the metrics by running and averaging 20 times.

For all runs, the identification rate is 100%, which indicates 100,000 times of memory access is enough for random memory traversal. As figure 6.3 shows, the average retransmission in all cases is less than one since the attacker cannot predict legitimate channels. However, the average retransmission increases with the network size since the jammer is more likely to successfully jam a legitimate communication when more nodes are involved. In addition, two or more pairs of nodes may switch to the same channel and the traffic collision occurs due to the limited number of channels. This is more severe with more sensors included when we notice the larger increase

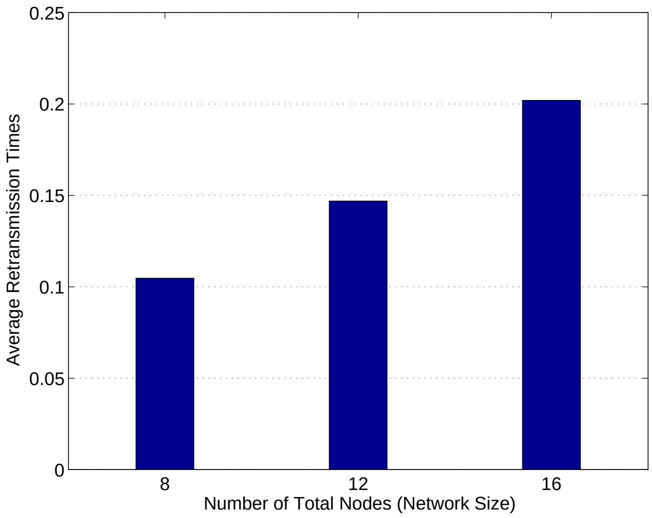

**Figure 6.3.** The average number of retransmissions for each node during one round for different network size.

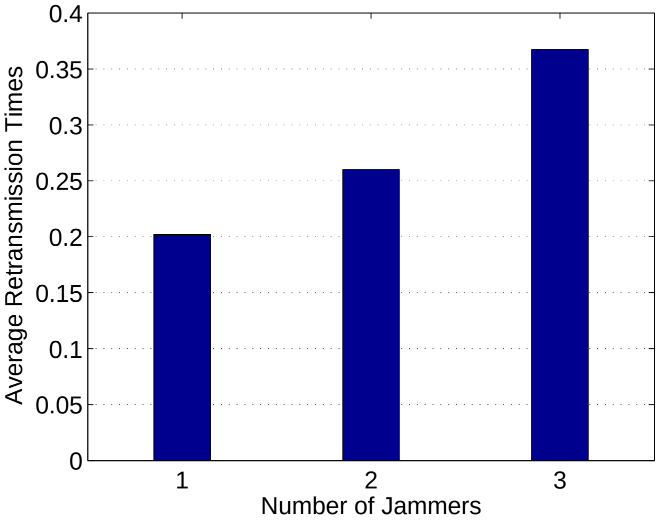

**Figure 6.4.** The average number of retransmissions for each node during one round for different jammers.

from size of 12 to 16 than 8 to 12.

### 6.3.2 The Impact of Number of Jammers

To evaluate the impact of different number of jammers, we deploy 16 to 19 nodes with one to three insiders and corresponding jammers to simulate a network of 16 nodes with 1 to 3 insiders. For one jammer, it sends a packet with one byte payload and then randomly selects and jams one of possible 32 channels. The jammer repeats this process until the identification completes. For the two or three colluding jammers, each is responsible for $\frac{1}{2}$ or approximately $\frac{1}{3}$ of the 32

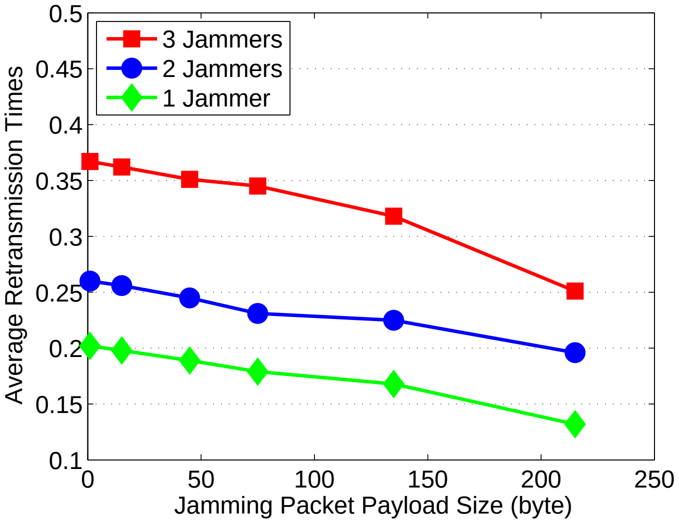

**Figure 6.5.** The average number of retransmissions for each node during one round under different number of jammers for different jamming durations.

channels. For example, for three jammers, jammer one, two and three are responsible for channel 1-11, 12-22, and 23-32 respectively.

For all runs, we find the identification rate is 100%. This is because the jamming does not influence the memory traversal and the forged results could be corrected by majority voting. Figure 6.4 shows the average number of retransmission under different number of jammers. We see the average retransmission is less than one even with 3 colluding jammers. The average retransmission increases with number of jammers and this trend becomes more intense when more jammers are involved. The reason is that the multiple colluding jammers can attack multiple channel simultaneously and more messages could be corrupted. Thus, more retransmissions are required to recover the jammed messages. Moreover, the collision of hashed channels needs the retransmission and the media access control mechanism to recover especially when the network size is large.

### 6.3.3 The Impact of Jamming Duration

To construct the different jamming durations, we add 1, 15, 45, 75, 135, and 215 bytes to the jamming packet. Similarly, we deploy 16 to 19 nodes with 1 to 3 jammers to simulate a network of 16 nodes with 1 to 3 insiders. For different number of jammers and jamming packet sizes, we measure the average retransmission 20 times and average them and show in Figure 6.5.

Since the jamming duration only impact the communication, we similarly find the 100% identification rate. In the Figure 6.5, the average retransmission becomes less when jamming duration increases. The major reason is the attacker can jam less channels if it resides in one channel more. This is more significant when more jammers are involved. In addition, we design our protocol to use short messages which reduces the probability of being jammed.

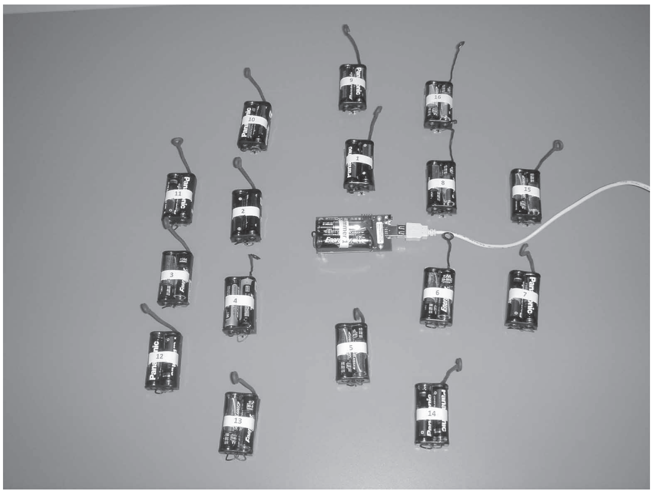

**Figure 6.6.** A network with 16 legitimate nodes and one jammer.

## 6.4 The Performance of the Split-Pairing Scheme

In this subsection, we conduct experiments to study the effectiveness of the split-pairing scheme described in Chapter 4, in which we consider a single jammer and the network is split into two groups. We measure the impact of the following two parameters: jamming probability and jamming duration.

### 6.4.1 The Impact of Jamming Probability

Since the jammer can only jam one channel at a time, it selects one of the two channels used for intra-group communication with some probability (the jamming probability) and sends a minimum size packet, then it repeats this process. We measure how the jamming probability affects the recovery latency.

We deploy 8, 12 and 16 nodes in the network and manually place a jammer in the center of the network to ensure that it can jam all the nodes. Legitimate nodes are split into two groups of 4, 6 and 8 nodes respectively. The network with 16 nodes and one jammer is shown in Figure 6.6. We set the retransmission timeout to be 250ms since one round of communication should be finished within 250ms. For different network size, we measure the recovery latency of the splitting phase for both groups by running our scheme 20 times and compute their average.

Figure 6.7 shows the average recovery latency of one group. As can be seen, the average latency increases with the jamming probability since nodes have to retransmit after the data is jammed. For a group of 4 nodes (the 8-node line in the figure considering there are two groups), the recovery latency does not change too much as the jamming probability increases from 0.1 to 0.3. This is because all versions of the group key can be embedded into one message which makes the key propagation message ($M_1$) less vulnerable of being jammed. However, when the group size increases to 6 or 8 nodes, different versions of the group key have to be split into

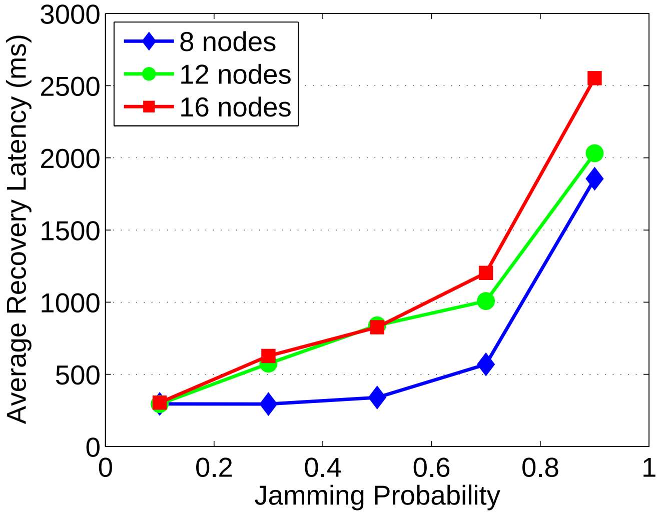

**Figure 6.7.** The recovery latency of the splitting phase (Phase I and II) for a single group.

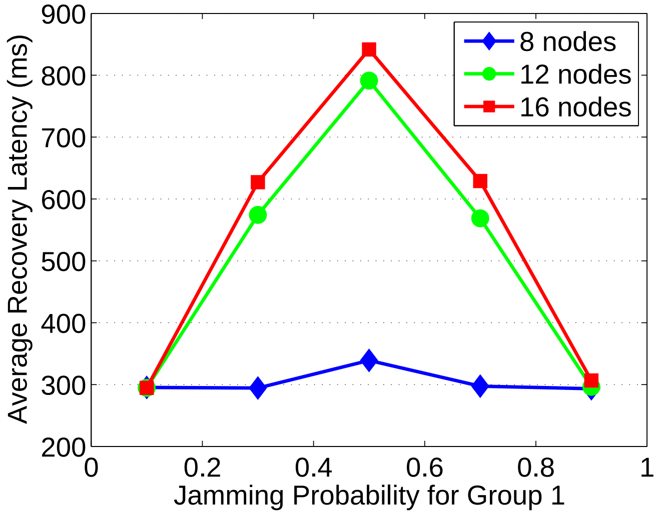

**Figure 6.8.** The recovery latency of the splitting phase (Phase I and II) which is the minimum latency of both groups.

two messages, and either one being jammed will lead to a retransmission, thus increasing the recovery latency. Moreover, as the network size increases, more confirmation messages ($M_2$) are required for key propagation and are more likely to be jammed, thus further increasing the recovery latency.

Figure 6.8 shows the recovery latency of the splitting phase in our scheme, which is the minimum of both groups. Since the jammer cannot jam two groups simultaneously, jamming one group always means free of jamming in the other group. After the jamming probability of group 1 is larger than 0.5, the minimum recovery latency should be the latency of group 2. This

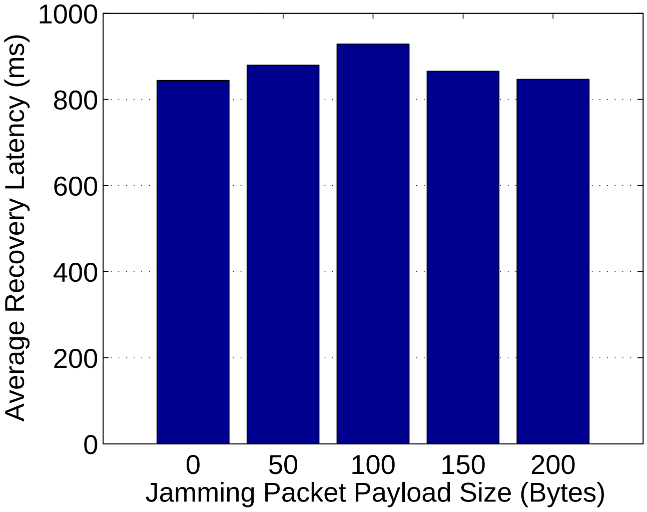

**Figure 6.9.** The recovery latency of the splitting phase (Phase I and II) under different jamming duration (Jamming probability=0.5, Network size=16 nodes).

explains why the recovery latency starts to decrease after the jamming probability is larger than 0.5. When the jamming probability is 0.5, the recovery latency reaches the highest point, which is consistent with our results on optimal jammer.

### 6.4.2 The Impact of Jamming Duration

In this subsection, we evaluate the impact of the jamming duration. We deploy a network of 16 nodes and fix the jamming probability to be 0.5. The retransmission time is set to be 250ms in Phase II and 70ms in Phase III. We add 0, 50, 100, 150 and 200 bytes to the jamming packet to construct different jamming durations.

Figure 6.9 shows the average recovery latency of the splitting phase by running our scheme 20 times. As can be seen, the recovery latency increases when the packet size increases from 0 to 100 bytes, and then decreases when the packet size increases from 100 bytes to 200 bytes. When the packet size increases from 0 to 100 bytes, the recovery latency is longer since the channel is jammed longer, and ongoing messages are more likely to be jammed and retransmitted. However, when the jammer stays in one group longer (100-200 bytes), the other group has larger chance to finish its intra-group communication. Since the recovery latency is the minimum key propagation time of both groups, the splitting phase completes as long as one group finishes the key propagation. Thus, jamming in one group longer gives the opportunity for the other group to finish earlier without any interruption, thus reducing the recovery latency.

Figure 6.10 shows the recovery latency of the split-pairing scheme including all three phases. Let $T_s$ denote the switching time from phase II to phase III. We consider two cases $T_s = 1000ms$ and $T_s = 1200ms$ due to the following reason. The splitting phase can be finished between 4 to 5 broadcast rounds. Since the retransmission timeout is 250 ms, the splitting phase should

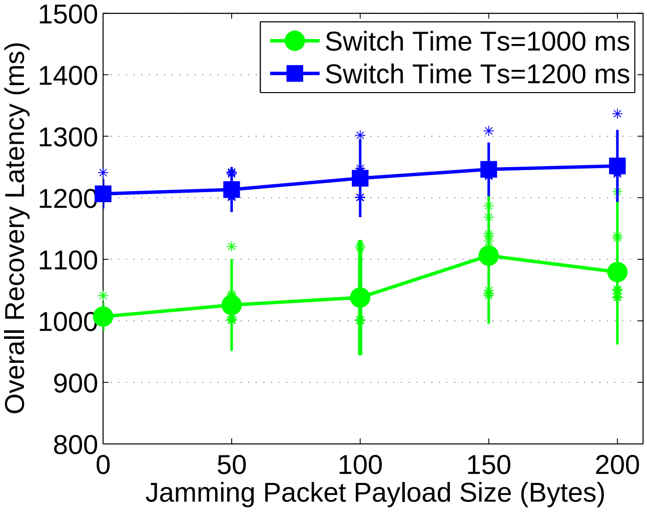

**Figure 6.10.** Recovery latency for the split-pairing scheme (including all 3 phases) under different jamming duration (Jamming probability=0.5, Network size=16 nodes).

be finished between time 250*4=1000 ms to 1250 ms. If we set $T_s$ smaller than 1000 ms, the splitting phase may not complete. If we set $T_s$ larger than 1250ms, both groups may have finished the key propagation and the pairing phase (Phase III) is not required any more. By setting the channel switching time to be 1000 ms and 1200 ms, we can investigate the impact of the jamming duration for both splitting phase (Phase II) and pairing phase (Phase III). For each jamming duration and switch time, we record the overall latency, and repeat the experiment 20 times. We also compute the mean and the 95% confidence interval shown as vertical bar in Figure 6.10.

For $T_s = 1200ms$, the latency does not change too much compared with the case of $T_s = 1000ms$. Given Figure 6.9, both groups have adequate time to finish the key propagation and therefore less communication is needed in the pairing phase. However, when the jamming duration increases, the latency slightly increases and the variability becomes larger. This is because the recovery difference between two groups in the splitting phase becomes more significant with longer jamming duration, thus more communications are needed in the pairing phase. This trend becomes more obvious with $t_s = 1000ms$. With $T_s = 1000ms$, the latency increases significantly between 100-150Bytes and declines between 150-200 Bytes. Since pairing in phase III needs more communication when the jamming duration increases, the random scan of the jammer in the pairing phase may have more chances to corrupt the communication and more messages are needed to be retransmitted. Therefore, the latency becomes larger. However, when the jamming duration increases, the jammer can scan less number of channels for a given period of time which reduces the chance of packets being jammed, thus the overall recovery latency becomes smaller.

## 6.5 The Performance of the Tree-based Scheme

The tree-based scheme can be used to deal with multiple colluding jammers. In our experiment, the number of jammers is between one and three. Since the jammers cannot predict the secret channels, they randomly select a channel and jam it. For one jammer, it is responsible for all 32 channels. For two or three jammers, each is responsible for $\frac{1}{2}$ or approximately $\frac{1}{3}$ of the 32 channels. For example, in the case of three jammers, one jammer is responsible for channels 1-11, the second jammer is responsible for channel 12-22, and the third jammer is responsible for channel 23-32.

### 6.5.1 Impact of the Jamming Duration

We conduct an experiment to study the impact of the jamming duration. In the experiment, we add 1, 16, 46, 76, 136, and 216 bytes to the jamming packet to construct different jamming durations. The number of legitimate nodes in the network is 8 as shown in Figure 6.11. For different numbers of jammers and jamming packet sizes, we measure the finish time 100 times and average them as the recovery latency shown in Figure 6.12.

The figure shows that the recovery latency is below one second in most cases. When the jamming packet size increases from 1 to 16 bytes, the recovery latency decreases quickly, but stops decreasing when the jamming packet size increases from 16 to 76 bytes. This is because the legitimate nodes have more delay since the channel is more likely to be occupied by the jamming signal, and more ongoing messages for the key propagation are corrupted by jammers and have to be retransmitted. However, beyond 76 bytes, the recovery latency begins to increase rapidly. The major reason is that legitimate nodes cannot transmit during jamming since each node always reads high signal strength. They can not seize the channel which results in longer recovery latency. From the figure, we can also see that the recovery latency increases with the number of jammers since more jammers can jam more channels.

### 6.5.2 The Impact of the Network Size

We set up another experiment to explore the impact of network size. In the experiment, the number of legitimate nodes in the network are 4, 8 and 16. One jammer is placed in the network with the jamming packet size of 150 bytes. The results are shown in Figure 6.13.

In the tree-based scheme, if groups on the same level of the tree can work in parallel, the recovery latency is proportional to the height of tree $log(n)$. However, in Figure 6.13, the recovery latency grows faster than $log(n)$. There are two explanations. First, more legitimate nodes require more channels at the beginning of the recovery. For instance, 8 nodes need 4 channels and 16 nodes need 8 channels. Therefore, the attacker are more likely to successfully jam legitimate packets which leads to more retransmissions. The second reason is related to collisions. For a large network, the probability of different nodes using the same channel is higher than a network with less number of nodes. When collision occurs during recovery, more retransmissions are

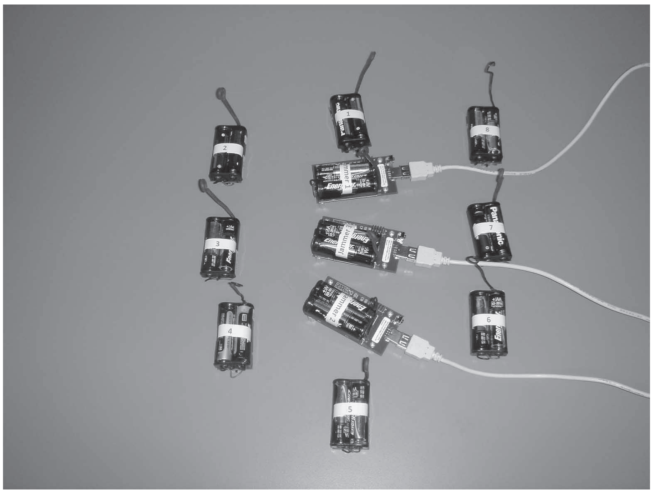

**Figure 6.11.** A network with 8 nodes and 3 jammers.

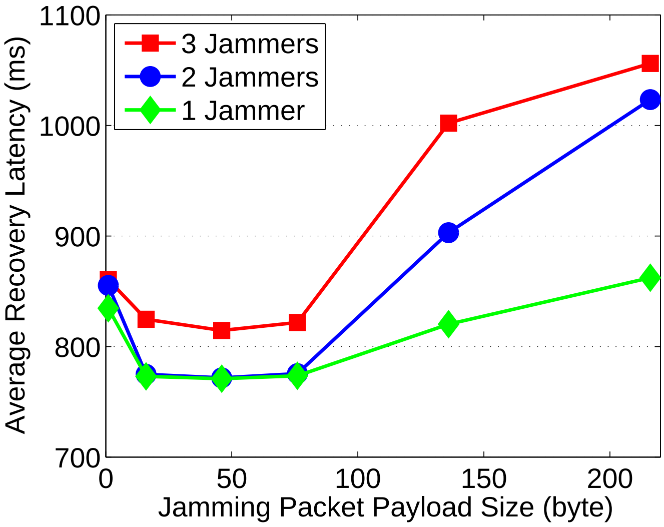

**Figure 6.12.** Recovery latency of the tree based scheme under different jamming packet size.

needed and the recovery latency is increased.

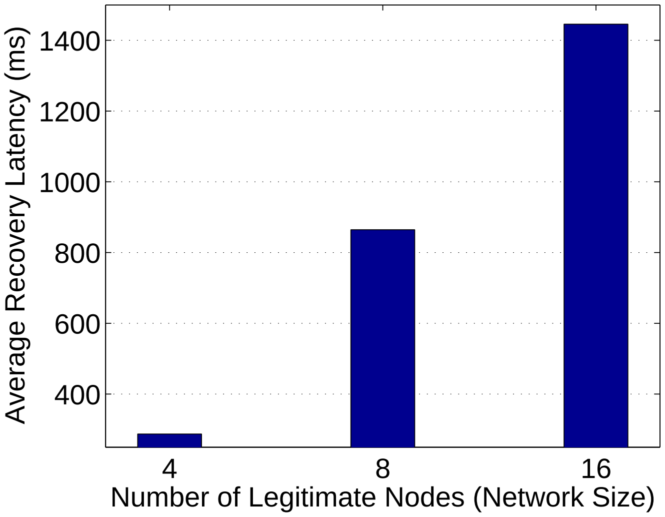

**Figure 6.13.** The recovery latency under different network size.

Chapter 7

# Discussions and Conclusions

## 7.1 Discussions

In this paper, we have focused on the Mica2 mote platform, but our scheme can also be applied to some other platforms. For example, Atheros 802.11 WiFi chipset has channel switching latency of 7.6ms. Given the transmission rate of WiFi 54Mb/s and a key size of 256 bits, more than 1500 keys can be transmitted within one channel switching. Thus, our scheme works much more effectively for the WiFi platform.

For the MicaZ sensor, the channel switching latency is 132 us and the minimum time for key propagation communication is 424 us [36]. It consists of the time for the jammer to leave the key propagation channel, send a minimum packet and then return. Given the transmission rate of 250 Kbps, only about 13 bytes could be transmitted. Considering the MAC frame header and key size, 13 bytes are not enough to transmit one key. To deal with this problem, we can apply the chained hash fragmentation technique [9]. The basic idea is to divide a large frame into small fragments. By hashing cyclically, fragments can be linked to reconstruct the original frame after receiving all of them.

## 7.2 Conclusions and Future Work

Wireless communication is susceptible to jamming attacks. Although some research has been conducted on countering jamming attacks, few works consider jamming attacks launched by insiders. In this paper, we proposed two schemes to address the insider jamming problem. In the split-pairing scheme, we exploit the fact that a single jammer can only jam one channel for any given time and nodes in other channels will be free of jamming and hence can start the recovery process. We further consider multiple colluding jammers in the tree-based scheme. Since attackers cannot compromise all pairwise keys, we use non-compromised pairwise keys at the beginning for deriving secret channels and propagating new keys. The new keys can be used to select channels and encrypt new keys again. This procedure continues until all non-compromised nodes share a common new key. Based on experimental results on Mica2 motes, we found that the split-pairing scheme is more efficient, but it can only deal with a single insider jammer. The tree-based scheme can cope with multiple colluding jammers, but it has higher message complexity and longer recovery time. As future work, we will focus on more advanced attacker models and we will extend our research to multi-hop networks.

# Bibliography

[1] Xu, W., W. Trappe, Y. Zhang, and T. Wood (2005) "The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks," in *ACM Mobihoc.*

[2] Xu, W., W. Trappe, and Y. Zhang (2007) "Channel Surfing: Defending Wireless Sensor Networks from Jamming and Interference," in *ACM IPSN.*

[3] Li, M., I. Koutsopoulos, and R. Poovendran (2007) "Optimal Jamming Attacks and Network Defense Policies in Wireless Sensor Networks," in *IEEE Infocom.*

[4] Wood, A. D., J. A. Stankovic, and S. H. Son (2003) "JAM: A Jammed-Area Mapping Service for Sensor Networks," in *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium.*

[5] Poisel, R. (2006) "Modern Communications Jamming Principles and Techniques," in *Artech House Publisher.*

[6] Brown, T., J. James, and A. Sethi (2006) "Jamming and sensing of encrypted wireless ad hoc networks," in *ACM Mobihoc.*

[7] Proakis, J. G. (2000) "Digital Communications, 4th edition," *McGraw-Hill.*

[8] Schleher, C. (1999) "Electronic Warfare in the Information Age," in *MArtech House.*

[9] Strasser, M., C. Popper, S. Capkun, and M. Cagalj (2008) "Jamming-resistant Key Establishment using Uncoordinated Frequency Hopping," in *IEEE Symposium on Security and Privacy*, pp. 64–78.

[10] Strasser, M., C. Popper, and S. Capkun (2009) "Efficient Uncoordinated FHSS Anti-Jamming Communication," in *ACM Mobihoc.*

[11] Popper, C., M. Strasser, and S. Capkun (2009) "Jamming-resistant Broadcast Communication without Shared Keys," in *USENIX Security Symposium.*

[12] Wang, Q., P. Xu, K. Ren, and X.-Y. Li (2011) "Delay-bounded Adaptive UFH-based Anti-jamming Wireless Communication," in *IEEE INFOCOM.*

[13] Lazos, L., S. Liu, and M. Krunz (2009) "Mitigating Control-Channel Jamming Attacks in Multi-Channel Ad Hoc Networks," in *ACM WiSec.*

[14] Tague, P., M. Li, and R. Poovendran (2009) "Mitigation of Control Channel Jamming under Node Capture Attacks," *IEEE Transactions on Mobile Computing.*

[15] Chiang, J. T. and Y.-C. Hu (2008) "Dynamic jamming mitigation for wireless broadcast networks," in *IEEE INFOCOM.*

[16] Dong, Q., D. Liu, and P. Ning (2008) "Pre-authentication filters: Providing DoS resistance for signature-based broadcast authentication in wireless sensor networks," in *ACM Conference on Wireless Network Security (WiSec), 2008.*

[17] Dong, Q. and D. Liu (2010) "Adaptive Jamming-Resistant Broadcast Systems with Partial Channel Sharing," in *International Conference on Distributed Computing Systems (ICDCS).*

[18] Datasheet, C. "Chipcon CC1000 Radio's Datasheet," *http://www.chipcon.com.*

[19] Xu, W., W. Trappe, and Y. Zhang (2008) "Anti-Jamming Timing Channels for Wireless Networks," in *ACM WiSec.*

[20] Chan, H., A. Perrig, and D. Song (2003) "Random Key Predistribution Schemes for Sensor Networks," in *IEEE Security and Privacy Symposim.*

[21] Liu, D., P. Ning, and R. Li (2003) "Establishing Pairwise Keys in Distributed Sensor Networks," in *ACM CCS.*

[22] Zhu, S., S. Setia, and S. Jajodia (2006) "LEAP+: Efficient Security Mechanisms for Large-scale Distributed Sensor Networks," in *ACM TOSN.*

[23] Blundo, C., A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung (1993) "Perfectly-secure key distribution for dynamic conferences," in *Advances in Cryptology, Proceedings of CRYPTO92*, pp. 471–486.

[24] Karlof, C., N. Sastry, and D. Wagner (2004) "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," in *ACM SenSys.*

[25] Seshadri, A., A. Perrig, L. van Doorn, and P. Khosla (2004) "SWATT: SoftWare-based ATTestation for Embedded Devices," in *IEEE Symposium on Security and Privacy.*

[26] Shaneck, M., K. Mahadevan, V. Kher, and Y. Kim (2005) "Remote software-based attestation for wireless sensors," in *ESAS.*

[27] Park, T. and K. G. Shin (2005) "Soft tamper-proofing via program integrity verification in wireless sensor networks," in *IEEE Transactions on Mobile Computing.*

[28] Yang, Y., X. Wang, S. Zhu, and G. Cao (2007) "Distributed software-based attestation for node compromise detection in sensor networks," in *IEEE SRDS.*

[29] Castelluccia, C., A. Francillon, D. Perito, and C. Soriente (2009) "On the difficulty of software-based attestation of embedded devices," in *ACM CCS.*

[30] Seshadri, A., A. Perrig, L. van Doorn, and P. Khosla (2004) "Swatt: Software-based attestation for embedded devices," in *IEEE Symposium on Security and Privacy.*

[31] Shaneck, M., K. Mahadevan, V. Kher, and Y. Kim (2005) "Remote software-based attestation for wireless sensors," in *ESAS.*

[32] Park, T. and K. G. Shin (2005) "Soft tamper-proofing via program integrity verification in wireless sensor networks," in *IEEE Transactions on Mobile Computing.*

[33] Yang, Y., X. Wang, S. Zhu, and G. Cao (2007) "Distributed software-based attestation for node compromise detection in sensor networks," in *IEEE SRDS.*

[34] Danev, B. and S. Capkun (2009) "Transient-based identification of wireless sensor nodes," in *ACM/IEEE IPSN*.

[35] Sun, Y. and X. Wang (2009) "Jammer localization in wireless sensor networks," in *Proc. of the 5th International Conference on Wireless communications, networking and mobile computing*.

[36] Wood, A., J. Stankovic, and G. Zhou (2007) "DEEJAM: Defeating Energy-Efficient Jamming in IEEE 802.15.4-based Wireless Networks," in *IEEE SECON*.

[37] Navda, V., A. Bohra, S. Ganguly, and D. Rubenstein (2007) "Using Channel Hopping to Increase 802.11 Resilience to Jamming Attacks," in *IEEE Infocom*.

[38] Reidt, S., M. Srivatsa, and S. Balfe (2009) "The fable of the bees: incentivizing robust revocation decision making in ad hoc networks," in *ACM CCS*.

[39] Towsley, D., J. Kurose, and S. Pingali (1997) "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols," in *IEEE Journal on Selected Areas in Communications (JSAC)*.

[40] Rodeh, O., K. Birman, and D. Dolev (2000) "Optimized Group Rekey for Group Communication Systems," in *Network and Distributed System Security Symposium (NDSS)*.

[41] Kim, Y., A. Perrig, and G. Tsudik (2004) "Tree-based Group Key Agreement," *ACM TISSEC*, **7**(1), pp. 60–96.

[42] Chipcon "CC1000 Single Chip Very Low Power RF Transceiver," *http://focus.ti.com/*.

[43] "Tinyos homepage," *http://webs.cs.berkeley.edu/tos/*.