Wireless Netw (2011) 17:1513–1527
DOI 10.1007/s11276-011-0361-8

Compromise-resilient anti-jamming communication in wireless sensor networks

Xuan Jiang · Wenhui Hu · Sencun Zhu · Guohong Cao

Published online: 18 June 2011

Abstract Jamming is a kind of Denial-of-Service attack in which an adversary purposefully emits radio frequency signals to corrupt the wireless transmissions among normal nodes. Although some research has been conducted on countering jamming attacks, 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 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.

X. Jiang (✉) · W. Hu · S. Zhu · G. Cao
Department of Computer Science and Engineering,
The Pennsylvania State University, University Park,
PA 16802, USA
e-mail: xjiang@cse.psu.edu

Keywords Compromise-resilient · Jamming · Sensor networks

1 Introduction

Wireless communication is 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 due to resource constraints on computation, communication, and energy.

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.

Many existing countermeasures against jamming focus on spread spectrum [20, 22] in which the sender and receiver hop among channels or use different 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. Although uncoordinated frequency hopping (UFHSS) and direct spread spectrum (UDSSS) [19, 25, 26, 31] have been proposed to enable key establishment between a pair of nodes without a pre-shared secret under a jammer, these approaches are typically not applicable to WSNs since they are designed for one-to-one communication or require sophisticated wireless interface to support direct spread spectrum.

For broadcast communication, group-based schemes [4, 7, 8] have been proposed. The idea is to divide receivers into multiple broadcast groups and different groups use predefined different channels. A compromised receiver can only jam the communication in the same group. Then, a divide-and-conquer strategy is applied to remove malicious receivers. However, these schemes require a large number of available channels. Otherwise, the compromised nodes could coordinate to jam all channels in a group.

For WSNs, Xu et al. proposed to use channel surfing [35] to deal with a narrow-band and intermittent jammer. Their basic idea is to let sensor nodes switch channels in a way that the jammer cannot predict them. For example, 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.

In this paper, we address the insider jamming problem in WSNs. In our solution, the physical communication channel is determined by the group key shared by all 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 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. Section 2 describes the system model and the design goal. The details of our recovery schemes are presented in Sects. 3 and 4. In Sect. 5, we present the performance evaluation results of our proposed schemes. Section 6 presents related work and Sect. 7 concludes the paper.

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 [6]. 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 [12, 19, 25], 26, 28, 36].

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 [3, 14, 41] 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 [1] 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 [10]; hence, each node needs to store hundreds of bytes of keying material. This storage overhead is affordable for low-end sensor motes with 4 KB RAM.

To simplify our presentation, we assume that legitimate nodes have detected and identified the jammer. Jammer localization and identification for WSNs are still open issues, although recently some efforts have been made towards addressing them, for example, RF fingerprinting for sensor nodes [5], jammer localization [27] and software-based attestation [16, 23, 24, 38]. Nevertheless, the focus of this paper is on recovering from insider jamming attacks.

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 Sect. 5, the typical latency is 34 ms for Mica2 mote. For MicaZ mote [33], t_l is 132 us. For 802.11 WiFi [15], the measurement result of t_l for the Atheros chipset is 7.6 ms.
- *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 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. Note that in the case of multiple compromised nodes, a compromised node may pariticipate in the network benignly and never jam. As a result it may still know the new group key after rekeying. While we acknowledge this is a valid arguement, our mechanism for rekeying under multi-jammers still bears its value on its own. On one hand, in any group key management protocols [17, 32, 39, 40], a default assumption is that somehow the ids of the compromised nodes are already known. The problem of how to detect compromised nodes was not addressed and considered as a separate issue. This type of stealthy attack is in principle valid to any group rekeying mechanisms. In our setting, we might not rely on a jammer detection algorithm to detect any compromised nodes which do not jam at all. But we can deploy a node compromise detection mechanism instead. For example, software attestation techniques [16, 23, 38] have been shown to be very powerful to detect node compromises by detecting the change of code in a sensor node. Second, even if compromised nodes are not detected, our rekeying mechanism at least will not introduce additional security problems into the network.

Next, we propose two compromise-resilient anti-jamming schemes: a split-pairing scheme to deal with single insider jammer and a tree-based scheme to handle multiple colluding attackers.

3 The split-pairing scheme

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.

3.1 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 $1, \ldots, N$. In this phase, nodes with lower *id*s $\{1, \ldots, \lfloor \frac{N}{2} \rfloor\}$ switch to channel $C_1 = H(C_0|0)$, and nodes with higher *id*s $\{\lfloor \frac{N}{2} \rfloor + 1, \ldots, N\}$ 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 Fig. 1(a, b). When node A is identified as a compromised node, nodes with lower *id*s, i.e., nodes 1, 2 and 3, switch to channel $C_1 = H(C_0|0)$ and nodes in higher *id*s, i.e., nodes 4, 5, 6 and 7, switch to channel $C_2 = H(C_0|1)$.

3.2 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.

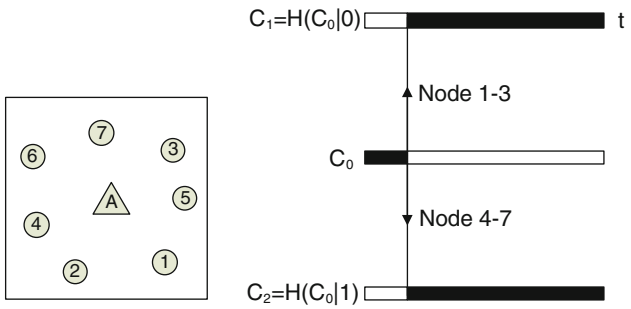

Fig. 1 **a** Network topology, **b** the illustration of channel switch for key reestablishment

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

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

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 28 bytes. Given a typical key size of 8 bytes [10], 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 19.2 Kbps, the transmission time for M_1 with three encryptions (i.e., the subgroup size is 4 counting the leader) is $\tau_1 \approx \frac{(8\text{ bytes}\cdot 3)+1\text{ bytes})}{19.2\text{ Kbps}} = 10.42\text{ ms}$ and for M_2 is $\tau_2 \approx \frac{(8+1)\text{ bytes}}{19.2\text{ Kbps}} = 3.75\text{ ms}$, the one-round communication time will be $\tau_0 = \tau_1 + 3 \cdot \tau_2 = 21.67\text{ ms}$. It is worth noting that the one-round time $\tau_0 < t_l$, where $t_l = 34$ ms 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 [30]. 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 3.1 *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:

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

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:

$$\min_{p_1 \geq 0, p_1 \leq p_2, p_1 + p_2 = 1} (T - T_j p_1) \tag{2}$$

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

$$T \approx T_{kr} + T_{jr} = \frac{2(t_j + t_l)}{2(t_j + t_l) - \tau_0} T_{kr}. \tag{3}$$

3.3 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 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 Fig. 1(a), we pair node 1, 4; 2, 5; 3, 6 and 1, 7, as shown in Fig. 2.

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:

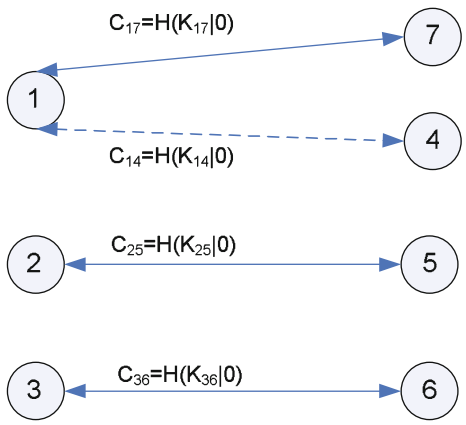

Fig. 2 Pairing for key propagation between groups

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

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 [10] all three messages are short and the time for this exchange for the Mica2 mote is $\tau_3 < \frac{8\text{ bytes}\cdot 3}{19.2\text{ Kbps}} = 10\text{ ms}$, 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 τ_3 or a bit more to tolerate lost time synchronization. Since nodes can detect failed packet [36], 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.

4 Tree-based scheme

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

4.1 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 [1]. However, each m-variate symmetric polynomial with a degree of $t \geq m$ has $\binom{t+m}{m}$ monomials; thus, the storage cost is $\binom{t+m}{m}$ 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 7 KB, 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 [11, 21] as in Fig. 3. 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 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 the one proposed for wired networks [21], there are two key differences. First, we do not assume the

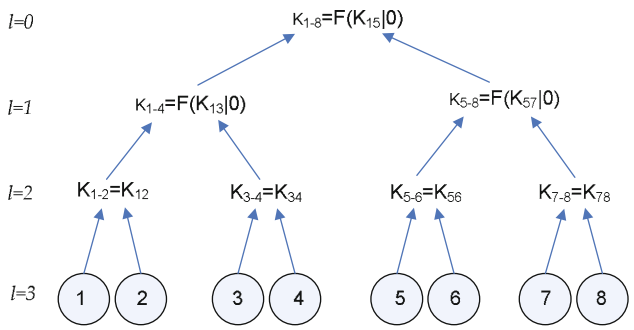

Fig. 3 Key tree example

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.

4.2 The protocol

To better understand the tree-based scheme, we first show an example for a network of 8 nodes as in Fig. 4. 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 $C_{i\text{-}j} = H(K_{i-j}|0)$ with H being a secure hash function for channel derivation.

Suppose all nodes are identified with ids $1, \ldots, N$ ($N = 8$ in our example) as before and a set of channels $\{C_1, C_2, \ldots, C_r\}$ are available to switch. Figure 4 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}.

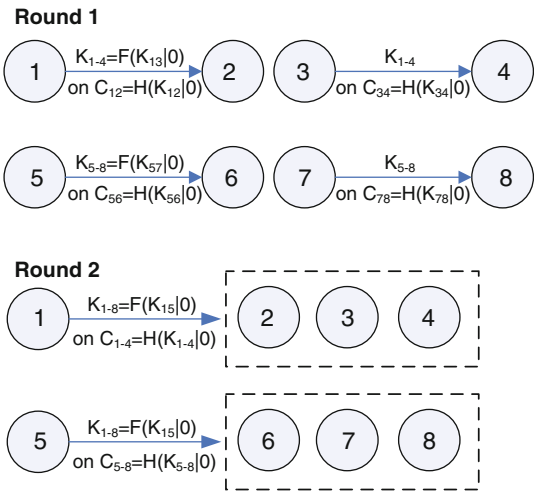

Fig. 4 An example for our tree-based scheme

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 members 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}.

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 $1, \ldots, N$, the pairwise key K_{ij} for $i, j \in \{1, \ldots, N\}$ and a set of channels $C = \{C_1, C_2, \ldots, C_r\}$. 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 $j = 1, 2, \ldots$, 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 kth round, node $2^k \cdot i + 1, i = 0, 1 \ldots, \lfloor \frac{N}{2^k} \rfloor$ 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 $i \in \{2^k \cdot i + 2, \ldots, 2^k \cdot (i + 1)\}$ reply to group leader $2^k \cdot i + 1$ with confirmation messageM^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 [21]. Since the number of nodes in a one-hop

Algorithm 1 Key-tree based scheme

Input: a set nodes numbered 1, …, N and their pairwise secret keys;

Procedure:

1: $k = 1$;
2: **repeat**
3: Select Channel Key = $K_{(2^k \cdot i+1)-(2^k \cdot (i+1))}$ for $i = 0, 1, \ldots, \lfloor \frac{N}{2^k} \rfloor$
4: Switch to Channel $C_{(2^k \cdot i+1)-(2^k \cdot (i+1))}$;
5: For node $2^k \cdot i + 1$, generate subgroup key $K_{(2^{k+1} \cdot i+1)-(2^{k+1} \cdot (i+1))} = F_{K_{2^{k+1} \cdot i+1, 2^k \cdot (2i+1)+1}}(0)$;
6: For node $2^k \cdot i + 1$, propagates new subgroup key to nodes $2^k \cdot i + 2$ to $2^k \cdot (i + 1)$;
7: k++;
8: **until** $k == \lceil \log_2 N \rceil$

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.

4.3 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.

5 Performance evaluations

In this section, we first describe our testbed, and then present the evaluation results of the split-pairing scheme and the tree-based scheme.

5.1 Testbed configuration

The testbed consists of 17 Mica2 motes [6] deployed at fixed locations in an indoor laboratory. Each sensor mote has a 902–928 MHz Chipcon CC1000 radio, which has 32 800 KHz channels. Each mote is within the communication range of other motes and the transmission rate is 19.2 Kbps. All motes run TinyOS version 2.0.1 [29].

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 the frequency and then achieve channel switching.

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). 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] initially. The sensing mechanism is used to determine if there is any ongoing communication on the channel. It requires the wireless interface to read received signal strength indication (RSSI) every 80 ms 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 adaptively changed with the current channel condition.

We modify the TinyOS source code to implement the jammer. 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 wireless 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 ($t_j \approx 0.42$ms). 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 100 ms.

5.2 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 Fig. 5(a) 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 1,000 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 1,000 times to get the switching latency between two arbitrary channels. As shown in Fig. 5(b), the switching latency is independent of the channel switching mode, and it is around 34 ms for all three motes.

5.3 The performance of the split-pairing scheme

In this subsection, we conduct experiments to study the effectiveness of the split-pairing scheme described in Sect. 3, 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.

5.3.1 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 Fig. 6(a). We set the retransmission timeout to be 250 ms since one round of communication should be finished within 250 ms. 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(b) 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 two messages, and either

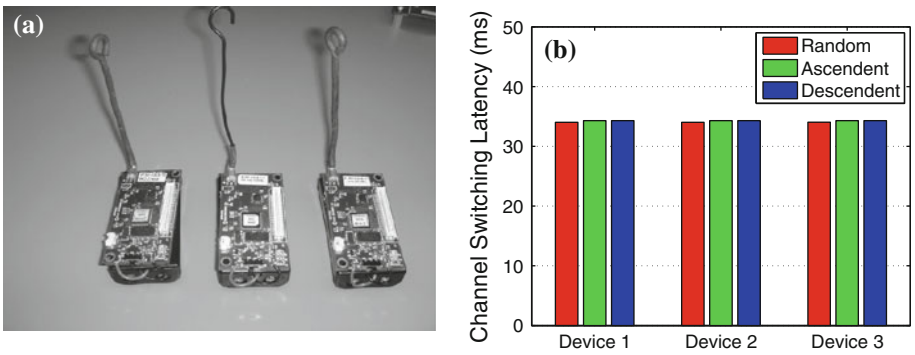

Fig. 5 **a** Three Mica2 motes are used for measuring the channel switching latency. **b** the channel switching latency for the three Mica2 motes

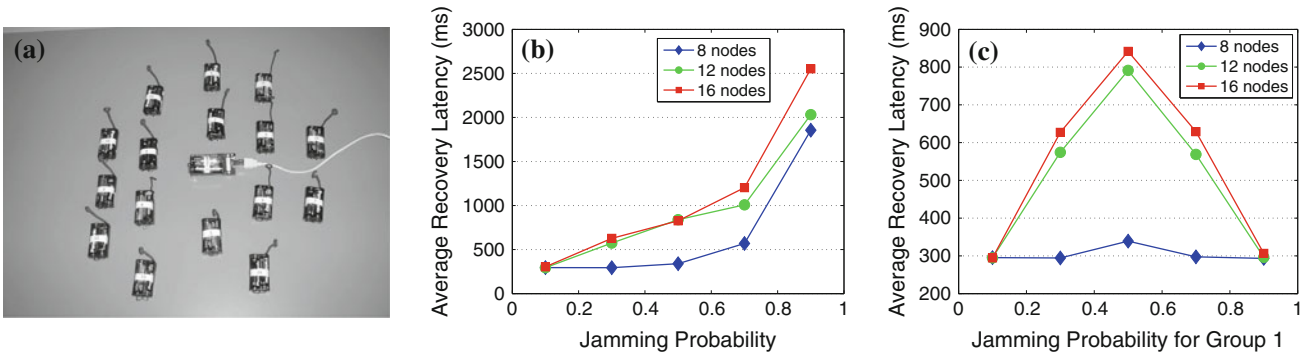

Fig. 6 **a** A network with 16 legitimate nodes and one jammer. **b** The recovery latency of the splitting phase (Phase I and II) for a single group. **c** the recovery latency of the splitting phase (Phase I and II) which is the minimum latency of both groups

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(c) 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 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.

5.3.2 *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 250 ms in Phase II and 70 ms in Phase III. We add 0, 50, 100, 150 and 200 bytes to the jamming packet to construct different jamming durations.

Figure 7(a) 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 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 7(b) 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 $T_s = 1{,}000$ ms and $T_s = 1{,}200$ ms due to the following reason. The splitting phase can be finished between 4 and 5 broadcast rounds. Since the retransmission timeout is 250 ms, the splitting phase should be finished between time 250*4=1,000 to 1,250 ms. If we set T_s smaller than 1,000 ms, the splitting phase may not complete. If we set T_s larger than 1,250 ms, 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

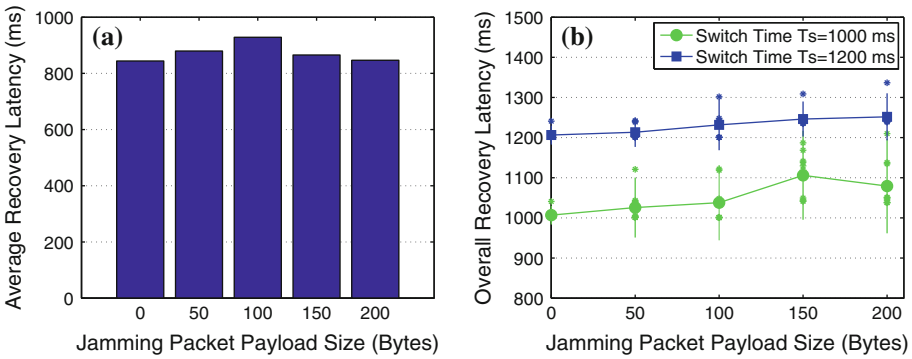

Fig. 7 **a** The recovery latency of the splitting phase (Phase I and II) under different jamming duration (Jamming probability = 0.5, network size = 16 nodes), **b** recovery latency for the split-pairing scheme (including all 3 phases) under different jamming duration (jamming probability = 0.5, network size = 16 nodes)

to be 1,000 and 1,200 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 Fig. 7(b).

For $T_s = 1$, 200 ms, the latency does not change too much compared with the case of $T_s = 1{,}000$ ms. Given Fig. 7(a), 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 $t_s = 1{,}000$ ms. With $T_s = 1{,}000$ ms, the latency increases significantly between 100 and 150 bytes and declines between 150 and 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.

5.4 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.

5.4.1 *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 Fig. 8(a). 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 Fig. 8(b).

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.

5.4.2 *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 Fig. 8(c).

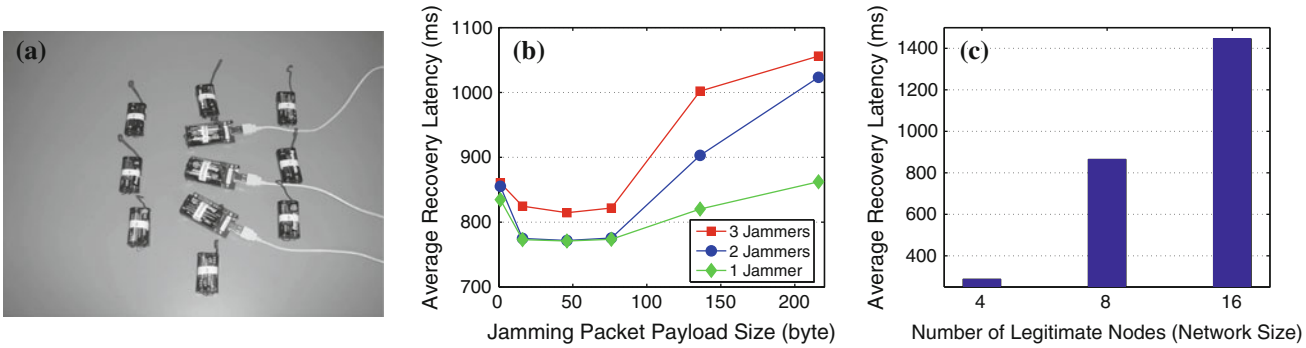

Fig. 8 **a** A network with 8 nodes and 3 jammers. **b** Recovery latency of the tree based scheme under different jamming packet size. **c** The recovery latency under different network size

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 Fig. 8(c), 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 needed and the recovery latency is increased.

5.5 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 7.6 ms. Given the transmission rate of WiFi 54 Mb/s and a key size of 256 bits, more than 1,500 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 [33]. 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 [26]. 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.

6 Related works

Jamming models have been widely studied, classified and evaluated. For example, jammers can be classified in terms of capabilities (broadband or narrowband) or behaviors (constant, deceptive, random, reactive) [37]. Jammers studied in prior works [2, 13, 18, 34, 35, 37] can also be categorized 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. Cross-layer jammers can attack some specific higher layer network protocols such as TCP or UDP to generate infinite retransmissions.

Most physical layer countermeasures rely on the spread spectrum technique. These solutions require that both the sender and receiver share the same key and the same pseudorandom function to generate a hopping or spreading sequence. For example, [19, 25, 26, 31] studied the problem of key establishment without pre-shared secret under jamming. In [26], a node pair establishes a new key by randomly hopping on a large number of channels until they use the same channel. However, this solution does not provide an efficient solution for broadcast since it only deals with one-to-one communication. To fix this problem and support group communication, UDSSS [19] has been proposed for broadcast communication. Basically, the sender sends a message repeatedly and the receivers synchronize the transmission with a sliding-window approach and despread the received message by searching through a set of codes. However, most of the physical layer 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 channel in [12, 28]. Lazos et al. [12] considered an insider attacker 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, Tague et al. [28] 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, group-based schemes [4, 7, 8] have been proposed. Researchers consider a scenario in which receivers in broadcast systems could be compromised and deceived as jammers. The idea is to divide receivers into multiple broadcast groups and different groups use different channels so that a compromised receiver can only jam the communication in the same group. Then, a divide-and-conquer strategy is applied to revoke malicious receivers. However, these schemes require a large number of available channels for broadcast. Otherwise, the compromised nodes could coordinate to jam all channels in a group.

For WSNs, [35] discussed evasion strategies called channel surfing under a narrow-band and intermittent jammer. The basic idea is that the jammed nodes change channels which cannot be predicted by the jammer. However, if the attacker can jam two channels at any time, the channel surfing scheme will not work. To solve the problem, Xu et al. [36] designed a new scheme based on the timing covert channel, which is effective against a broadband and constant/persistent attacker. In their scheme, a timing-based overlay and a coding/decoding scheme are established to convey information. However, this technique can only support low-rate and it is unclear how to extend it to a network with multiple nodes.

In our earlier work Jiang et al. [9], we addressed the insider jamming problem in WSNs. The solution is based on the assumption that only one node can be compromised as an insider jammer. However, in reality, more nodes may be compromised to launch jamming attacks, which have been addressed in this paper.

7 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.

Acknowledgments The authors would like to thank anonymous reviewers for their insightful comments and helpful suggestions. This work was supported in part by Army Research Office under MURI grant W911NF-07-1-0318 and NSF CAREER 0643906.

References

1. Blundo, C., Santis, A. D., Herzberg, A., Kutten, S., Vaccaro, U., & Yung, M. (1993). Perfectly-secure key distribution for dynamic conferences. In *Advances in cryptology, Proceedings of CRYPTO 92*, (pp. 471–486).
2. Brown, T., James, J., & Sethi, A. (2006). Jamming and sensing of encrypted wireless ad hoc networks. In *ACM Mobihoc*.
3. Chan, H., Perrig, A., & Song, D. (2003). Random key predistribution schemes for sensor networks. In *IEEE Security and Privacy Symposim*.
4. Chiang, J. T. & Hu, Y.-C. (2008). Dynamic jamming mitigation for wireless broadcast networks. In *IEEE INFOCOM*.
5. Danev, B. & Capkun, S. (2009). Transient-based identification of wireless sensor nodes. In *ACM/IEEE IPSN*.
6. Datasheet, C. Chipcon cc1000 radio's datasheet. http://www.chipcon.com.
7. Dong, Q. & Liu, D. (2010). Adaptive jamming-resistant broadcast systems with partial channel sharing. In *International conference on distributed computing systems (ICDCS)*.
8. Dong, Q., Liu, D., & Ning, P. (2008). Pre-authentication filters: Providing dos resistance for signature-based broadcast authentication in wireless sensor networks. In *ACM conference on wireless network security (WiSec)*.
9. Jiang, X., Hu, W., Zhu, S., & Cao, G. (2010). Compromise-resilient anti-jamming for wireless sensor networks. In *International conference on information and communications security*.
10. Karlof, C., Sastry, N., & Wagner, D. (2004). Tinysec: A link layer security architecture for wireless sensor networks. In *ACM SenSys*.
11. Kim, Y., Perrig, A., & Tsudik, G. (2004). Tree-based group key agreement. *ACM TISSEC, 7*(1), 60–96.
12. Lazos, L., Liu, S., & Krunz, M. (2009). Mitigating control-channel jamming attacks in multi-channel ad hoc networks. In *ACM WiSec*.
13. Li, M., Koutsopoulos, I., & Poovendran, R. (2007). Optimal jamming attacks and network defense policies in wireless sensor networks. In *IEEE Infocom*.
14. Liu, D., Ning, P., & Li, R. (2003). Establishing pairwise keys in distributed sensor networks. In *ACM CCS*.

15. Navda, V., Bohra, A., Ganguly, S., & Rubenstein, D. (2007). Using channel hopping to increase 802.11 resilience to jamming attacks. In *IEEE Infocom*.
16. Park, T., & Shin, K. G. (2005). Soft tamper-proofing via program integrity verification in wireless sensor networks. In *IEEE transactions on mobile computing*.
17. Perrig, A., Song, D., & Tygar, J. D. (2001). Elk, a new protocol for efficient large-group key distribution. In *IEEE Symposium on Security and Privacy*.
18. Poisel, R. (2004). Modern communications jamming principles and techniques. Artech House Publisher. http://www.amazon.com/Communications-Jamming-Principles-Techniques-Information/dp/158053743X.
19. Popper, C., Strasser, M., & Capkun, S. (2009). Jamming-resistant broadcast communication without shared keys. In *USENIX Security Symposium*.
20. Proakis, J. G. (2000). *Digital communications* (4th ed.). New York: McGraw-Hill.
21. Rodeh, O., Birman, K., & Dolev, D. (2000). Optimized group rekey for group communication systems. In *Network and Distributed System Security Symposium (NDSS)*.
22. Schleher, C. (1999). Electronic warfare in the information age. In *MArtech House*.
23. Seshadri, A., Perrig, A., van Doorn, L., & Khosla, P. (2004). Swatt: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy*.
24. Shaneck, M., Mahadevan, K., Kher, V., & Kim, Y. (2005). Remote software-based attestation for wireless sensors. In *ESAS*.
25. Strasser, M., Popper, C., & Capkun, S. (2009). Efficient uncoordinated fhss anti-jamming communication. In *ACM Mobihoc*.
26. Strasser, M., Popper, C., Capkun, S., & Cagalj, M. (2008). Jamming-resistant key establishment using uncoordinated frequency hopping. In *IEEE Symposium on Security and Privacy* (pp. 64–78).
27. Sun, Y., & Wang, X. (2009). Jammer localization in wireless sensor networks. In *Proceedings of the 5th international conference on wireless communications, networking and mobile computing*.
28. Tague, P., Li, M., & Poovendran, R. (2009). Mitigation of control channel jamming under node capture attacks. *IEEE Transactions on Mobile Computing, 8*(9), 1221–1234.
29. Tinyos homepage. http://webs.cs.berkeley.edu/tos/.
30. Towsley, D., Kurose, J., & Pingali, S. (1997). A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *IEEE Journal on Selected Areas in Communications (JSAC)*.
31. Wang, Q., Xu, P., Ren, K., & Li, X.-Y. (2011). Delay-bounded adaptive ufh-based anti-jamming wireless communication. In *IEEE INFOCOM*.
32. Wong, C., Gouda, M., & Lam, S. S. (1998). Secure group communication using key graphs. In *ACM special interest group on data communication (SIGCOMM)*.
33. Wood, A., Stankovic, J., & Zhou, G. (2007). Deejam: Defeating energy-efficient jamming in ieee 802.15.4-based wireless networks. In *IEEE SECON*.
34. Wood, A. D., Stankovic, J. A., & Son, S. H. (2003). Jam: A jammed-area mapping service for sensor networks. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*.
35. Xu, W., Trappe, W., & Zhang, Y. (2007). Channel surfing: Defending wireless sensor networks from jamming and interference. In *ACM IPSN*.
36. Xu, W., Trappe, W., & Zhang, Y. (2008). Anti-jamming timing channels for wireless networks. In *ACM WiSec*.
37. Xu, W., Trappe, W., Zhang, Y., & Wood, T. (2005). The feasibility of launching and detecting jamming attacks in wireless networks. In *ACM Mobihoc*.
38. Yang, Y., Wang, X., Zhu, S., & Cao, G. (2007). Distributed software-based attestation for node compromise detection in sensor networks. In *IEEE SRDS*.
39. Zhang, W., & Cao, G. (2005). Group rekeying for filtering false data in sensor networks: A predistribution and local collaboration-based approach. In *IEEE INFOCOM*.
40. Zhang, W., Zhu, S., & Cao, G. (2009). Predistribution and local collaboration-based group rekeying for wireless sensor networks. In *Ad hoc networks*.
41. Zhu, S., Setia, S., & Jajodia, S. (2006). Leap+: Efficient security mechanisms for large-scale distributed sensor networks. In *ACM TOSN*.

Author Biographies

Xuan Jiang is a Ph.D. candidate in Computer Science and Engineering at the Pennsylvania State University. He received his B.S. in Electronics Science and Engineering from University of Science and Technology of China in 2005. His research primarily on wireless network security.

Wenhui Hu is a Ph.D. candidate in Computer Science and Engineering at the Pennsylvania State University. He received M.Sc. (Tech) with Distinction, major in Telecommunications Software, from Helsinki University of Technology, Finland, in 2007 and his B.Eng (honored) in computer science and engineering from Southeast University, Nanjing, China, in 2002. Wenhui's research primarily focuses on network applications and network security.

Sencun Zhu is an associate professor at Department of Computer Science and Engineering and College of Information Sciences and Technology, the Pennsylvania State University. He received the Ph.D. degree in Information Technology from George Mason University in 2004. Prior to that, he received the M.S. degree in Signal Processing from University of Science and Technology of China in 1999 and the B.S. degree in Precision Instruments from Tsinghua University in 1996. His research interests include network and systems security with focuses on wireless security, online social network security, and software security.

Guohong Cao received the BS degree from Xian Jiaotong University, China. He received the M.S. degree and Ph.D. degree in computer science from the Ohio State University in 1997 and 1999 respectively. Since then, he has been with the Department of Computer Science and Engineering at the Pennsylvania State University, where he is currently a Professor. His research interests are wireless networks and mobile computing. He has published more than 150 papers in the areas of wireless sensor networks, wireless network security, vehicular ad hoc networks, data access and dissemination, and distributed fault-tolerant computing. His work has been cited over 5000 times and his h-index is 37 based on Google Scholar. He has served on the editorial board of IEEE Transactions on Mobile Computing, IEEE Transactions on Wireless Communications, IEEE Transactions on Vehicular Technology, and has served as program chair, general chair, and program committee member of many conferences. He was a recipient of the NSF CAREER award in 2001. He is a Fellow of the IEEE.