

Resilient Cluster Leader Election for Wireless Sensor Networks

Qi Dong

Department of Computer Science and Engineering
The University of Texas at Arlington
Email: qi.dong@mavs.uta.edu

Donggang Liu

Department of Computer Science and Engineering
The University of Texas at Arlington
Email: dliu@uta.edu

Abstract—Sensor nodes are often organized into *clusters* for efficiency and scalability purposes. Every sensor cluster is managed by a *cluster leader* during the network operation such as routing and data aggregation. Since managing a cluster consumes substantial energy, the cluster leader needs to be re-elected from time to time for load balancing. In hostile environments, it is critical to ensure the security of such leader election. This paper proposes an efficient, resilient, and fully distributed leader election protocol for sensor networks. It only uses efficient symmetric key operations and guarantees that (i) benign cluster members will elect the same leader as long as they are well-connected, and (ii) attackers cannot impact the leader election process to increase or decrease the chance of a benign member being elected as a cluster leader. In addition, the proposed method can quickly recover from message loss or malicious attacks. The evaluation results also demonstrate the efficiency and effectiveness of this approach.

I. INTRODUCTION

In a densely-deployed sensor network, sensors are usually organized into clusters for efficiency and scalability purposes. Every cluster contains a number of sensors that are physically close to each other and is managed by a *cluster leader* to facilitate the network operation such as in-network aggregation and routing. For instance, in data aggregation, a cluster leader can serve as the aggregator that collects the sensing results from other cluster members, computes the aggregation result, and reports the result to the base station.

Since managing a cluster consumes substantial energy, the cluster leaders need to be re-elected from time to time for load balancing. The *leader election problem* is to ensure that each cluster will have a suitable sensor node be selected as the cluster leader whenever needed. A common *election metric* to determine the new leader of a given cluster is the remaining energy on sensor nodes. For the sake of presentation, we call the values (on sensor nodes) used for election purpose as the *election values*. In the above case, the election values will be the remaining energy reported by sensor nodes.

A number of secure clustering protocols have been proposed recently for organizing clusters in sensor networks [1], [2]. These protocols significantly improve the security of clustering in the presence of malicious attacks. However, they only focus on the formation of clusters and do not consider the security of cluster leader election. Without proper protection, an attacker can easily subvert the intended purpose of clustering. For example, if the protocol selects the node with the most remaining energy as the new cluster leader, an attacker can continuously hijack the leadership by always claiming more remaining energy than anyone else in the cluster.

The security of leader election has been considered in several recent studies [3], [4], [5], [6], [7]. However, as we will show later, these protocols are either communication intensive, sensitive to message loss/delay, or vulnerable to simple DoS attacks. As a result, the focus of this paper is to *develop a practical cluster leader election protocol that is more efficient, resilient, and effective than previous techniques*.

However, designing an efficient and resilient protocol for leader election is quite challenging. First, sensor nodes are very limited in computation power, storage space, communication bandwidth, and energy supply. These resource constraints require that the overhead of any sensor operation be as low as possible. Hence, it is undesirable to apply those well-studied but expensive mechanisms. Second, since the wireless channel is essentially a broadcast channel, an attacker can easily eavesdrop, corrupt, forge, or replay any message in such communication channel. This will lead to many attacks such as DoS attacks [8], Sybil attacks [9], and Wormhole attacks [10]. Third, an adversary can easily capture a sensor node and quickly learn all the secrets inside [11]. This makes it possible for the attacker to launch node replication attacks by creating many duplicated nodes in the network to impact the network at a large scale [12].

In this paper, we propose an efficient and resilient leader election protocol for sensor network clustering in hostile environments. The security of the proposed protocol is achieved by (i) making it infeasible to forge the election values (i.e., the remaining energy) of benign sensor nodes and (ii) making the leader election protocol resistant to the malicious election values supplied by compromised sensor nodes. The proposed scheme has several nice properties. First, it guarantees that the benign cluster members in a given cluster will elect the same cluster leader as long as they are well-connected. Second, an attacker cannot impact the leader election protocol to increase or decrease the chance of a benign sensor node being elected as the cluster leader. Third, it can quickly recover from failure due to lossy channels or malicious attacks once the cluster members become well-connected again. This important property makes it very difficult for adversaries to disrupt the leader election protocol by jamming the communication channel for only a short period of time. A long-term jamming attack is very expensive and can be easily detected to physically locate the adversary. Finally, the proposed scheme is fully distributed. This addresses the problem of the single point of failure.

The rest of the paper is organized as follows. The next section reviews existing studies and discusses their limitations.

Section III gives the adversary model and the system assumptions. Section IV presents our resilient leader election protocol with detailed analysis. Section V discusses some implementation issues and evaluation results. Section VI concludes this paper and points out some future directions.

II. RELATED WORK

This section briefly reviews existing approaches that are developed for sensor clustering in hostile situations [1], [2], [3], [7], [4], [5], [6]. We found that these protocols either ignore or do not effectively solve the security of leader election.

The techniques in [1], [2] focus on the security issues during the formation of initial clusters. They do not consider the security of cluster leader election. In this paper, we focus on how to protect the cluster leader election in the presence of malicious attacks. The proposed technique is complementary to the above two secure cluster formation schemes.

The protocols developed in [3], [7], [4], [5], [6] do consider the security of cluster leader election. However, they have a number of limitations. The protocols in [3], [5], [4] simply ask the base station to make a centralized decision and announce the new leaders in the network. Such centralized approaches incur substantial communication and computation costs. Hence, they do not work well in resource-constrained sensor networks. In addition, a centralized protocol is often vulnerable to the single point of failure.

The protocols developed in [7] use digital signatures to guarantee the integrity of the election values at sensor nodes. This works well in dealing with external attackers who do not compromise sensor nodes. However, a malicious insider can always claim to be the most suitable cluster leader (i.e., having the most remaining energy) and therefore win the competition. On the other hand, although sensor nodes are able to do a few signature generations and verifications, it is still computationally expensive to do so. Attackers can simply spam sensor nodes with fake signatures and force them to do a large number of public key operations, eventually exhausting their valuable energy.

Three election protocols are proposed in [6]. They work in a decentralized way and only use lightweight cryptographic algorithms. However, these protocols are vulnerable to many attacks. For example, each node needs to send out its election value as the input of the election process. If the attacker refuses to send/forward these values or injects conflict values, it is possible that different cluster members elect different leaders or no leader is elected at all. Our technique can handle these kinds of problems very well. As we will show, it is much more difficult and expensive for the adversary to achieve the same goal.

III. SYSTEM MODELS AND DESIGN GOALS

In this section, we will discuss the system model, the adversary model, and our design goals.

A. The System Model

This paper considers a sensor network that consists of a large number of resource-constrained sensor nodes. Each sensor node has a unique ID and is randomly scattered to

monitor the activities in the field and report its observations for various purposes.

In this paper, we assume that all cluster members have the same view of the initial members in the cluster. Since this information can be obtained in the cluster formation phase, we will not discuss its detail here. Therefore, we assume that once a cluster is formatted, all cluster members will have the same sorted list of the cluster member IDs, i.e., L_{init} .

In each cluster, a new cluster leader needs to be elected from time to time to coordinate the cluster members and handle the network operations. The leader election protocol works in rounds with one new leader being elected for each round. How to decide the election round (time and duration) is out of the scope of this paper since it is most likely application-dependent. Therefore, our discussion in this paper focuses on a particular round of leader election. In the end of each round, all cluster members should elect the same cluster leader.

B. The Adversary Model

An attacker can launch a wide range of attacks against sensor networks. For example, he can jam the wireless channel or shield the radio signal to sensor nodes, causing a denial of service. Jamming or shielding for a very long period of time is quite expensive and can be easily detected. We thus assume that an adversary launches “stealthy” attacks that only disable the communication for a short period of time to interrupt the delivery of some critical messages. We assume that the attacker can eavesdrop, modify, forge, replay, or block any message. We also assume that the attacker can compromise sensor nodes and learn all the secrets inside. The compromised nodes may also run malicious programs injected by the attacker.

An adversary with the above capability can disrupt the cluster leader election process in many ways if the leader election protocol is not protected very well. First, he can make an arbitrary node elected as the cluster head. For example, the attacker may fool benign cluster members into electing a malicious node as the leader so that he can take control of the whole cluster. As another example, the attacker may fool the cluster members into always electing a particular benign node as the cluster leader, eventually exhausting the energy of the victim node. Second, the adversary can also make no eligible nodes elected at all or more than one cluster leaders elected at a time. If the adversary succeeds, the whole cluster will function incorrectly during the network operation.

C. The Design Goals

The objective of this paper is to provide security for the election of cluster leaders in sensor networks. We identify the following security goals for a cluster leader election protocol. These security goals are quite vague at this moment. However, we will clarify their meanings in our later analysis.

- *Security Goal 1:* Unauthorized sensor nodes cannot join the cluster leader election. That is, only those legitimate members of a cluster can participate in the leader election of this cluster.
- *Security Goal 2:* The attacker cannot arbitrarily increase or decrease the chance of any benign node being elected as a cluster leader. More specifically, the attacker cannot control who will be elected as the cluster leader at any round of leader election.

- *Security Goal 3:* As long as benign cluster members are well-connected in a particular round of leader election, they will always elect the same cluster leader for this round. This means that the result of a given round of leader election is only affected by the network connectivity during that round.

A cluster may include a few compromised sensor nodes. It is certainly possible that a malicious sensor node is elected as the cluster leader at some point. Indeed, it is infeasible to prevent this from happening when this malicious node always acts like a normal sensor node. To mitigate such impact, one option is to ensure that every cluster member has the equal opportunity of being elected. Our protocol is designed to reinforce such fairness between cluster members. On the other hand, the compromised sensor node may start behaving maliciously once it gets elected as the new cluster leader. Detecting such misbehavior is certainly important for the security of sensor networks. However, this is beyond the scope of this paper, and we consider it complementary to our approach. Nevertheless, even without such detection mechanism, our protocol can still tolerate compromised nodes in the sense that any malicious node can only serve as the cluster leader for a given period of time. The leadership will be likely shifted to a benign node in the next round of election.

IV. RESILIENT CLUSTER LEADER ELECTION

In this section we will first briefly review the security primitives used in the proposed approach and then describe its technical detail and analyze its performance.

A. Preliminaries

Our protocol takes advantage of the following two security tools, the *one-way key chain* and the *Blundo's pairwise key pre-distribution scheme*.

One-way key chain: An one-way key chain $\{K_0, K_1, K_2, \dots, K_R\}$ is generated by iteratively performing the one-way function $H(\cdot)$ on the last key K_R in the chain. This key chain has the following property: given any K_j , all former keys can be derived by computing $K_i = H^{j-i}(K_j)$, $0 \leq i < j$, while none of the later keys can be computed due to the one-wayness of function H . Therefore, with the knowledge of $K_0 = H(K_1)$ (called the *key chain commitment*), anybody can verify the authenticity of any later key by only performing hash operations. In our protocol, the one-way key chain will be used to protect the election values reported by cluster members.

Blundo's key pre-distribution: This approach is used as an identity-based pairwise key establishment protocol since it can bind the node IDs with the keying materials. In fact, a successful pairwise key establishment between two nodes also proves their IDs to each other. This scheme works as follows. Before deployment, a central server generates a t -degree symmetric polynomial $f(\cdot, \cdot)$, which has the property of $f(i, j) = f(j, i)$ for any i and j . Every sensor node i is assigned with the polynomial "share" $f(i, \cdot)$. To establish a pairwise key between nodes i and j , node i evaluates $f(i, \cdot)$ at point j , while node j evaluates $f(j, \cdot)$ at point i . Since $f(i, j)$ equals $f(j, i)$, these two sensor nodes can use this value as the pairwise key between them. This scheme is proved to be t -collusion resistant, i.e., it can tolerate up to t compromised nodes [13]. Obviously, a success in key

establishment also indicates that the other party does know the corresponding polynomial share, which further proves the ID. We do not consider the case where a large number of nodes are compromised since most applications will fail anyway in such situation. Therefore, we always assume no more than t compromised nodes in the network.

B. Protocol Description

Similar to most existing cluster leader election protocols, we use the remaining energy on sensor nodes as the metric to determine the cluster leader. However, our approach is also different from them in that we do not use the one with the most remaining energy as the new cluster leader. Instead, the role of cluster leader will be rotated among all cluster members that are qualified for being elected as the cluster leader, i.e., those nodes whose energy is greater than a pre-determined system threshold E_{th} , which is the minimum energy required for serving as the cluster leader. This will make it very difficult for any malicious insider to hijack the role of cluster leader frequently by forging the election value. Our approach includes three steps, *initialization*, *anti-spoofing announcement*, and *distributed decision making*.

Initialization: In this step, every sensor node will be given a unique ID, two one-way key chains, and the keying materials for pairwise key establishment. The commitments of these two key chains determine the node ID, which further determines the keying materials this node will get from the server. Thus, a successful key establishment between two nodes also authenticates the node IDs to each other, which further proves the key chain commitments to each other. The detailed protocol works as follows.

Before deployment, every sensor node i (we will discuss how to do the ID assignment later) will be assigned with two random one-way key chains, a *YES chain* and a *NO chain*. The NO chain is used by this node to tell other nodes that it has no energy left for serving as the cluster leader. Therefore, it only includes a key chain commitment $N_{i,0}$ and a NO key $N_{i,1}$. The YES chain is used to inform other nodes that it has sufficient energy left for serving as the new cluster leader. Therefore, it includes a key chain commitment $Y_{i,0}$ and a number of YES keys $\{Y_{i,1}, Y_{i,2}, \dots, Y_{i,R}\}$, where R is the maximum number of rounds for leader election. The ID i of this node is simply the hash of the two key chain commitments, i.e., $i = H(Y_{i,0} || N_{i,0})$.

After determining the node ID i , we will use the Blundo's pairwise key pre-distribution protocol for key pre-distribution. Specifically, the central server will maintain a symmetric bivariate polynomial $f(\cdot, \cdot)$ and pre-distribute a polynomial share $f(i, \cdot)$ to every node i . Thus, the ID, the commitments, and the keying materials are tied together. Clearly, as long as two nodes can establish a pairwise key, they can immediately authenticate each other's ID as well as the key chain commitments due to our ID assignment scheme.

After deployment, the sensor nodes will be organized into clusters based on some secure cluster formation protocols such as those in [1], [2]. Once a cluster is formed, all cluster members have the same member ID list L_{init} .

At the beginning of cluster leader election, the cluster members need to exchange their key chain commitments among each other. Due to our ID assignment described previously,

```

Procedure SHUFFLE ( $L, n, C, H$ );
  integer  $n$ ; integer array  $L, C$ ; procedure  $H$ ;
  comment SHUFFLE applies a random permutation
  to the sequence  $L$  (the initial ID list  $L_{init}$ ).  $C[i]$  is
  the YES chain commitment  $Y_{i,0}$  of node  $i$ .  $H$  is
  one-way hash function, “||” is the concatenation
  operation, and “%” is the mod operation.
begin
  integer  $i, j, temp, seed$ ;
   $seed := H(C[1]||C[2]||\dots||C[n])$ ;
  for  $i := n$  step-1 until 2 do
    begin  $j := H(seed||i) \% n$ ;
       $temp := L[i]; L[i] = L[j]; L[j] = temp$ 
    end loop  $i$ 
end SHUFFLE

```

Fig. 1. The shuffle algorithm

these commitments can be easily verified according to the node ID. After collecting all those authenticated chain commitments from other members, every cluster member will shuffle the initial ID list L_{init} using the algorithm shown in Figure 1 and produce the same random permutation L_{candi} , called the *candidate list*. This list is used for determining the leader of a cluster in each round of election. Different from the traditional shuffle algorithm [14], our modified version uses one-way hash function H and the YES chain commitments of all cluster members to generate the random candidate list. Since the ID, the key chain commitments, and the pairwise keys are tied together, it is impossible for the attacker to manipulate the candidate list at the beginning. In other words, all benign nodes in the cluster will have the same view of the candidate list.

Anti-spoofing announcement: In each round of leader election, every cluster member needs to assess its own remaining energy and announce the result to other cluster members. If the residual energy E_i of a cluster member i is greater than the threshold E_{th} , it will disclose the next key in its YES chain to announce its willingness to be a cluster leader. If there is not much energy left for serving as the cluster leader, it will release the (only) key in its NO chain to announce its decision. In each round of leader election, a cluster member will broadcast its announcement (the chained key) α times to tolerate the channel loss. We will discuss the impact of α later.

In the r -th round of leader election, if a cluster member j receives a key in the YES chain of node i , node j can verify it by computing and comparing $H^{r-r'}(Y_{i,r})$ with the key $Y_{i,r'}$ received earlier in the r' -th round of election. If the verification succeeds, node j will update the YES key for node i ; otherwise, it will drop the message. If node j receives the correct NO key $N_{i,1}$ from i , node j will remove this node from the candidate list L_{candi} .

In each round of election, every sensor node in the candidate list is required to release either a valid YES key or a valid NO key as the announcement. If any node notices that another cluster member releases both the valid (fresh) YES key for the current round and a valid NO key, it can immediately conclude that this node must have been compromised. It can then directly broadcast these two keys to other members to revoke the malicious node. As a result, we assume that a smart attacker won't release two conflicting keys, i.e., the YES key for the current round and the NO key.

Distributed decision making: In every round r , a cluster

member needs to collect the fresh chained key, i.e., either the r -th key in the YES chain or the key in the NO chain, from every node in the candidate list. However, due to the lossy wireless channel and malicious attacks, the keys from some nodes in the candidate list may not reach others during this round of election. In other words, it is possible that the cluster members have different views of the keys for the nodes in the candidate list. Indeed, it is infeasible to completely eliminate this issue since an attacker can interrupt the communication and temporarily partition the network. Every partition will be likely to have its own cluster leader.

Fortunately, even if the adversary disrupts the key exchanging, he can not convince the nodes to accept forged keys. Thus, we can take advantage of this and ensure that (i) the benign cluster members will always elect the same cluster leader as long as they are well-connected, and (ii) the leader election can recover quickly from failures once the network in the local area becomes well-connected again. A group of cluster members is said to be well-connected if the message from anyone of them can reach all others with a very high probability after no more than α times of broadcast. With these in mind, we develop the distributed decision making process as follows.

Consider a particular cluster member i in the r -th round of leader election. When there are missing keys for some nodes in the candidate list, node i will mark these nodes as *inactive*. Once a cluster member stays inactive for β consecutive rounds of cluster leader election, it will be removed permanently from the candidate list. Note that β also implies the maximum number of hash operations that a sensor node needs to perform for verifying a chained key. The purpose of this is to mitigate DoS attacks targeting at forcing sensor nodes to do a significant number of hash operations. Every cluster member makes its election decision purely based on the candidate list it sees for this round. Specifically, *the first active node following the current cluster leader in the candidate list will be elected as the new leader for the cluster*. Table I shows the leader election process in a 5-node cluster when $\beta = 2$.

In addition, we also develop a **simple recovery protocol** to deal with the inconsistency of the decisions made by cluster members. If a member i finds itself to be the new leader based on its candidate list, it will contact every active member to make an announcement of being the new leader with a message including all NO keys and fresh YES keys it has. Such communication is protected by the pairwise key between them. The message will also be used by the contacted node to update its own set of chained keys. In addition, if the contacted node notices that node i has missed some YES keys or NO keys, it will reply a message including all these keys; otherwise, the contacted node will use node i as the new cluster leader and send a confirm message to node i . When node i receives the missing keys, it will update the candidate list and restart the above recovery protocol since the new leader may change based on its updated list; otherwise, node i will keep checking how many “positive” responses from active members. If it receives confirmation messages from more than half of them, it will start to serve as the new leader.

If a cluster member i notices that it is not the new leader based on its candidate list, it will send the new leader (based on its own view) a message that includes all NO keys and fresh

TABLE I
ILLUSTRATION OF THE RESILIENT CLUSTER LEADER ELECTION PROTOCOL IN A 5-NODE CLUSTER ($\beta = 2$).

	Released Information					L_{candi}	Leader Node	Notes
	1	2	3	4	5			
Before Election	$N_{1,0}$ $Y_{1,0}$	$N_{2,0}$ $Y_{2,0}$	$N_{3,0}$ $Y_{3,0}$	$N_{4,0}$ $Y_{4,0}$	$N_{5,0}$ $Y_{5,0}$	5,2,1,4,3		Exchange commitments and generate L_{candi}
Round 1	$Y_{1,1}$	$Y_{2,1}$	$Y_{3,1}$	$Y_{4,1}$	$Y_{5,1}$	5,2,1,4,3	5	Select leader from L_{candi}
Round 2	$N_{1,1}$	$Y_{2,2}$	$Y_{3,2}$	$Y_{4,2}$	$Y_{5,2}$	5,2,4,3	2	1 quits the election
Round 3		$Y_{2,3}$		$Y_{4,3}$		(5),2,4,(3)	4	Mark 3, 5 as <i>inactive</i>
Round 4		$Y_{2,4}$		$Y_{4,4}$	$Y_{5,4}$	5,2,4	5	Remove 3 and restore 5

YES keys it has. This message will also be protected by their pairwise key. The contacted node will also use this message to update its chained keys. If the contacted node notices that node i has missed some YES keys or NO keys, it will reply a message including all these keys; otherwise, the message from i will be considered as a positive response from i if the contacted node considers itself as the new cluster leader and is also waiting for the responses.

C. Security Analysis

In the following, we will discuss how the proposed protocol achieves the three security goals identified in Section III.

Security goal 1: This security goal requires that only legitimate cluster members can participate in the leader election. In other words, we have to ensure that those unauthorized nodes that don't have the cluster membership cannot join the leader election and cannot impersonate any benign cluster member. These two properties are actually achieved by using the Blundo's key pre-distribution protocol [13]. The identity of a sensor node is tied to its cryptographic key. It is thus not possible for an unauthorized node to join the cluster election.

We further explain that the adversary cannot forge the announcements from benign cluster members. This is because the announcements are the keys in YES or NO chains. To forge these keys, the attacker needs to either invert the hash function H or fool benign cluster members into accepting incorrect key chain commitments. However, because of our ID assignment scheme, the ID of a sensor node i is the hash of its two key chain commitments, i.e., $i = H(Y_{i,0} || N_{i,0})$. Therefore, to forge the key chain commitments, the adversary has to forge the ID as well. This requires the knowledge of keying materials related to the forged ID, which is not available to the adversary. Therefore, the adversary has no way to make its forged commitments accepted by any benign cluster member.

Without the proper chained keys, the adversary may try to replay the keys released earlier by the legitimate member. However, this will not impact the proposed scheme. The reason is that our scheme uses either the fresh YES key or the NO key in each round of leader election. More specifically, the index of the released fresh key in any YES chain will always be the same as the index of the current election round unless the corresponding cluster member gives up the leader election because of insufficient energy and releases the NO key. Once a benign cluster member releases its NO key, it will never release any new YES key.

Security goal 2: Note that a compromised cluster member can always behave normally like benign members and get elected as the new cluster leader at some point during the field operation. There are no effective ways to identify those "passive" malicious cluster members since there are no evidences of them being malicious. As a result, the second security goal

focuses on making sure that an adversary cannot significantly impact the chance of any benign member being elected as the new leader. Thus, even if the leader is compromised, it cannot continue to control the cluster. The election protocol should ensure the fairness of cluster members being elected as cluster leaders. In the following, we would like to show how our scheme achieves this when there are compromised cluster members in the cluster. For simplicity, we assume that the benign members are well-connected. We will evaluate the impact of message loss later.

First, the attacker cannot significantly increase or decrease the chance of a benign node being elected as the new cluster leader. The leader election is based on the candidate list L_{candi} , which is initialized at the beginning of the leader election. All cluster members in L_{candi} will serve as the cluster leader in a round-robin manner unless they release their NO keys during the leader election. When the benign cluster members are well-connected, all benign nodes that are eligible for the leader role will be included in the candidate list L_{candi} . As a result, they will be elected as the new leader one by one in order. It is thus not possible for the adversary to impact the chance of a benign member being elected as the new cluster leader unless it can invert the hash function H and release the *undisclosed* NO key of this benign cluster member. However, inverting a hash function is computationally infeasible. In addition, once a benign cluster member declares its energy insufficiency with its NO key and quits the leader election, the attacker has no way to make other benign members believe that this node is still qualified for the leader role. Because of the above reasons, we can also see that it is not possible for adversaries to significantly increase the chance of those malicious members being elected as the new cluster leader.

Second, an attacker can certainly release the NO keys of compromised cluster members to *slightly* increase the chance of other benign cluster members being elected as the leader. However, this is equivalent to the case of compromised members being isolated since they can no longer participate in the leader election once they quit. Hence, we believe that the attacker will not do this. Due to the limited power supply, most benign cluster members may run out of energy and cannot serve as the leader after the network is on for a long time. However, the adversary may let compromised cluster members stay in the leader election all the time and claim to have sufficient energy. Hence, the attacker will have a higher chance of taking the leader role in the later stage of the field operation. Fortunately, due to the energy limitation on sensor nodes, this can be fixed by limiting the number of rounds a given cluster member can serve as the cluster leader. This fix will not impact benign cluster members much but can effectively thwart the impact of compromised members.

Finally, the adversary may want to take over the cluster

leader role at a specific time. For example, he may want to hide some critical events from being detected or reported during his intrusion into the field. However, in our protocol, the adversary cannot arbitrarily manipulate the order of the candidate cluster members (including the malicious ones) being elected as the new cluster leader. The candidate list L_{candi} is generated by the shuffle algorithm using the hash function. The seed is the hash image of the YES chain commitments of all cluster members. Hence, to manipulate the candidate list, the adversary needs to forge key chain commitments. However, no one can forge key chain commitments because the node ID, the commitment of key chains, and the keying materials of a given node are all tied together during the initialization of sensor nodes.

According to the above discussion, we can see that the adversary has very limited control over the leader election process. As a result, we can conclude that the proposed approach achieves the second security goal.

Security goal 3: The attacker may try to disrupt the leader election by making the candidate lists discovered by different cluster members *inconsistent* to each other. For instance, he may provide different election values to different members and fool them into making different decisions. Fortunately, the proposed scheme can defeat such attacks effectively and guarantee that one and only one leader will be elected as long as the cluster members are well-connected.

First, as discussed before, the attacker cannot forge the announcements from benign cluster members unless he can invert the one-way function. Moreover, replaying an earlier announcement message will not help the adversary at all. This is because the freshness of an announcement from a cluster member can be verified easily based on its index in the key chain. As a result, every benign cluster member will be able to get a correct and fresh announcement from every other benign member as long as the members are well-connected.

Second, a malicious cluster member can certainly give announcements including different decisions to different cluster members. That is, he discloses the fresh YES key to some cluster members and the NO key to some other members. However, this can be easily detected by other members. Indeed, seeing the NO key and the fresh YES key from the same cluster member indicates that this cluster member must have already been compromised.

Third, a malicious cluster member may only give its announcement to some selected cluster members. As a result, some members will have updated information from this malicious node, while others will mark it as *inactive*. This will result in inconsistent views of the candidate list in the cluster. Fortunately, this is quite difficult for adversaries to achieve. The reason is that all the announcement messages will be broadcasted in the cluster. Once a benign cluster member receives a new announcement, it can re-broadcast the message to make sure the other cluster members also receive such message. This process is out of the control of the adversary. In addition, we have a simple recovery protocol to tolerate the inconsistency in the cluster. With this protocol, the inconsistency can be easily resolved as long as the cluster members are well-connected.

From the above discussion, we can see that when the cluster members are well-connected, benign cluster members will

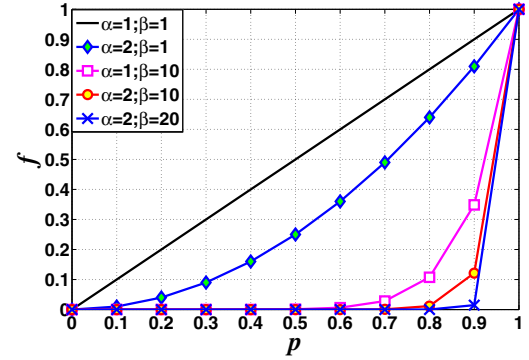


Fig. 2. The probability f that a qualified node is removed from L_{candi} v.s. the probability p that a chained key fails to be delivered.

have the same view of the candidate list. This indicates that they will agree on the same cluster leader at the end of election. Therefore, we conclude that our protocol achieves the third security goal.

Resilience to lossy channels and jamming attacks: Our security analysis has shown that our proposed protocol can work effectively and efficiently as long as the benign members in a given cluster are well-connected. However, it is also interesting to know the resilience of our protocol when the cluster members are not so well-connected all the time. Such investigation can be valuable since in typical sensor networks, we may experience high packet loss rate caused by either highly-lossy channels or intensive jamming attacks.

Fortunately, the proposed scheme can tolerate message loss effectively due to the following reasons. First, the protocol only involves the communication within the same cluster, whose members are physically close to each other. The impact of lossy channels is thus much less than that in those centralized schemes where all messages have to be delivered to a central server via many hops. Second, in each round of election, a cluster member only needs to release one key. As long as the members receive the valid YES key from the *expected leader* (i.e., the first active member following the current leader in the candidate list L_{candi}), they will always make the same decision about the new cluster leader, even if they have different views of L_{candi} . In addition, the result from an earlier round of leader election does not impact a later round of leader election at all since the decision is always made based on the NO keys released so far and the fresh YES keys received in the current round. This means that our approach can effectively tolerate and recover from failures. The adversary has to continuously attack the communication channel (e.g., by jamming the radio) in order to disable our approach, which could be very expensive for most attackers.

Next, we will study in detail how well our protocol handles message loss. According to the protocol, each cluster member will broadcast its chained key α times in each round of election. If the YES keys of a given member are not received in β consecutive rounds of election, it will be removed permanently from the candidates list. Therefore, the attacker cannot forge the YES key and force a receiving node to perform more than β times of hash operations. Although this can mitigate the DoS attack on the verification of chained keys, a benign member that is qualified for being elected as the cluster leader may be removed from the candidate list due to the lossy channel or

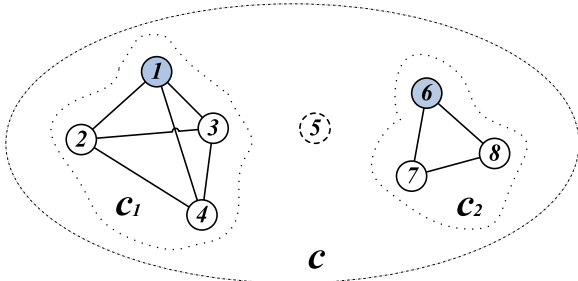


Fig. 3. When the original cluster C is partitioned, each new sub-clusters will elect their own leaders separately.

malicious jamming attacks.

In the following, we will analyze the probability f of a qualified node being removed from L_{candi} . We assume that every member has received the chain commitments of other members and computed the candidate list L_{candi} successfully at the beginning of the leader election. A fresh chained key can be either sent directly from the owner (in one-hop cluster) or relayed by the intermediate nodes (in multi-hop cluster). Let p denote the probability that a cluster member fails to receive a given chained key correctly. A large p indicates either a highly-lossy channel or frequent channel jamming attacks. The probability f can be estimated as $f = p^{\alpha\beta}$, which is shown in Figure 2. We can clearly see that this protocol can tolerate the packet loss effectively. For example, when $p = 0.8$ (a highly-lossy channel), $\alpha = 2$, and $\beta = 10$, f is only 0.0115.

In addition, our simple recovery protocol also helps cluster members to exchange their views of chained keys, and can effectively reduce the inconsistency between benign members. The leader needs to exchange the collected chained keys with other cluster members and collect the confirmation messages from more than half of them. Here, we assume at least half of the cluster members are benign. Otherwise, the whole cluster cannot function correctly anyway. Therefore, if the normal cluster member i chooses the wrong cluster leader j , the view of the candidate list at i will also get corrected if i receives the key either from the actual leader or j .

Certainly, the adversary can launch intensive channel jamming attacks and partition the cluster into several parts. In this case, each part will work like an independent cluster and will elect its own leader. For instance, in Figure 3, the original cluster C is divided into two new sub-clusters C_1 and C_2 . The nodes in each of these two parts are well-connected, but the nodes in C_1 cannot talk to any node in C_2 . Nodes 1,2,3,4 will have the same candidate list where nodes 5,6,7,8 are marked as inactive members; they will elect the same leader among them. The nodes in C_2 will do the same thing.

Fortunately, such network partition caused by malicious jamming attacks will not last long in practice due to the cost of launching the attack and the possibility of the attacker being detected and located. Hence, we assume that it will not last more than β consecutive rounds. In this case, we note that the proposed technique can recover quickly from the network partition once the network in the local area becomes well-connected again. In other words, once the isolated parts can talk to each other again, they can be merged without any additional cost as long as their messages can reach each other. Specifically, in a given round of leader election, if a cluster member i receives a fresh YES key from an inactive member

j in its candidate list, it knows that j becomes active again. This YES key will be propagated among the well-connected cluster members to update the candidate list at other members. In the end, all well-connected cluster members will have the same view about the active members in the candidate list and will thus elect the same cluster leader in the future.

From the above discussion, we can see that it is very difficult for an attacker to disrupt our leader election protocol without launching an intensive channel jamming attacks continuously or compromising a large number of cluster members. Therefore, we can conclude that our leader election protocol can effectively tolerate message loss.

D. Overhead

Before deployment, every sensor node is assigned with an ID i and a t -degree polynomial share $f(i, \cdot)$. It also needs to store two key chains along with their commitments, one contains a single key, and the other consists of R keys, where R denotes the maximum number of rounds for leader election. In a cluster of n members (including the cluster leader), a sensor node will also need to buffer the candidate list of n IDs, $n-1$ pairwise keys, and $2(n-1)$ key chain commitments. These space requirements are usually not a problem since the cluster size is limited.

The computational overhead mainly comes from four parts. First, with the Blundo's key pre-distribution protocol, a sensor node needs to compute a t -degree polynomial to establish a pairwise key and verify the ID of another sensor node. Second, a sensor node has to perform the hash function H one time to verify the key chain commitment and at most β times to verify a chained key. Third, every sensor node has to compute the candidate list using the shuffle algorithm, which needs another $n+1$ times of hash function operations. Finally, every sensor node will need to perform symmetric key operations such as encryption and authentication to protect the communication between sensor nodes. Obviously, all these operations only involve lightweight computation.

The proposed technique only incurs small communication overhead. The Blundo's key pre-distribution technique does not introduce any additional communication overhead at all for two nodes to establish a pairwise key. In the initialization step, each node only needs to disclose its ID and two key chain commitments. In each round of leader election, every cluster member is only required to broadcast a single chained key for α times. In our simple recovery protocol, the chained keys will be exchanged between the cluster members. Different from centralized approaches where sensor nodes exchange messages with the base station that is far away, our protocol only involves the communication between the cluster members that are physically close to each other. Therefore, the communication cost of our protocol is not a big problem.

E. Discussion

In the following, we will discuss some important issues related to the proposed technique. We will focus on the addition of new cluster members, the security of our protocol against typical attacks, and the comparison with the existing secure leader election protocols.

Addition of new cluster members: Our protocol can be easily extended to handle the addition of new cluster members

on the fly. Suppose a new node u meets the requirements of the cluster formation protocol and joins the cluster in the r -th round of leader election. If u has sufficient energy, it can join the election without any problem. Specifically, after establishing the pairwise keys with other cluster members, u only needs to broadcast its two key chain commitments and collect the released NO keys and the fresh YES keys from other cluster members. Once the cluster members (including the new one) have an updated set of commitments for all the cluster members, they can re-execute the shuffle algorithm to generate a new candidate list. Starting from the $(r + 1)$ -th round of leader election, u can participate in the same election process as other members do using the updated candidate list.

Security under typical attacks: In *sybil* attacks [9], an adversary tries to clone sensor nodes with different IDs. Since the keying materials for pairwise key establishment are always tied to the node ID, our approach is free from sybil attacks.

In *wormhole* attacks [10], the attacker creates a wormhole between a pair of nodes that are far away from each other to fool them into establishing an *incorrect* neighbor relation. This attack does not directly affect the leader election process since the cluster membership has been determined before the leader election. However, this attack can impact the initial cluster formation and thus can *indirectly* affect the leader election. Fortunately, several protocols have been developed to mitigate the impact of the wormhole attacks [15], [16], [10], [1]. Any of these techniques can be used to enhance the security of clustering in sensor networks.

In *node replication* attacks [12], the adversary will compromise a sensor node and create many replicas of this node in many different places in the field. Since these replicated nodes have the keying materials taken from the compromised sensor node, they will be accepted by other nodes as a legitimate part of the network and impact the network significantly. The node replication attack does not impact the cluster leader election protocol directly. Instead, it impacts the formation of the initial clusters in the field. Once the clusters are correctly formed, the cluster leader election will function correctly. Hence, we consider techniques to address node replication attacks as a necessary and complementary research topic to our work.

Comparison with other techniques: In the following, we will briefly compare our scheme with other schemes that also considered the security of cluster leader election. We focus on the schemes developed in [7], [3], [4], [5], [6].

First, our proposed protocol is fully distributed and only involves the communications between the sensor nodes that are physically close to each other. Therefore, our protocol avoids the single point failure and can be much more communication efficient than centralized schemes like those in [3], [4], [5]. Second, our protocol is also efficient in terms of computation overhead. It achieves security by only using efficient cryptographic algorithms such as one-way hash function. In contrast, the protocols in [7] uses digital signatures. They involve considerable computation overhead and are vulnerable to DoS attacks on signature verification. Furthermore, our proposed protocol also prevents those compromised nodes from being elected continuously for a long period of time. In addition, our protocol does not require the assumption of no message loss or packet delay as made in [7]. Finally, our protocol ensures that one and only one node will be elected as the cluster head

TABLE II
THE CODE SIZE AND EXECUTION TIME OF EACH COMPONENT

Operations	Running Time (in ms)		Code Size (in bytes)	
	Sender	Receiver	ROM	RAM
ID and CC Distribution	17.27	27.01	2734	S_1^*
Candidate List Randomization	457.49		308	54
Key Release and Verification	0.24	10.19	836	S_2^{**}
Recovery Protocol	Announcement	27.67	26.83	
	Confirmation	9.728	9.785	1766

* $S_1 = 130 + 29(n - 1)$, where n is the number of cluster members.

** $S_2 = R + 4$, where R is the maximum number of rounds for leader election.

as long as the cluster members are well-connected. This is because the election messages (chained keys) are easy to be verified but infeasible to be forged. A cluster member cannot deny its election messages sent before. In contrast, none of the protocols in [6] guarantees such property in the presence of malicious attack.

V. IMPLEMENTATION AND EVALUATION

We have implemented the prototype of our leader election scheme on TinyOS [17] and evaluated it using the TelosB [18] motes. In the experiment, we assume that the cluster size is 50. We use RC5 [19] with 8-byte long keys to implement security primitives such as encryption, decryption, hash, and MAC operations. For the Blundo's pairwise key establishment scheme, we use the implementations provided in *TinyKeyMan* [20] ($t = 8$) for polynomial evaluation.

The prototype contains components to (i) distribute and verify the ID and key chain commitments (CC), (ii) perform the shuffle algorithm to randomize the candidate list, (iii) exchange the YES keys or NO keys between cluster members in each round of election, and (iv) perform the simple recovery protocol to deal with the packet loss.

ID and chain commitments (CC) distribution: In the experiment, each sensor node establishes a pairwise key with another node and verifies the ID and the two key chain commitments of that node through a single message. Specifically, sensor node i first computes the pairwise key $K_{i,j}$ with every other member j , and then encrypts its two key chain commitments and sends that encrypted message to j along with its ID i . Correspondingly, node j will compute $K_{i,j}$ by itself, and decrypt the encrypted chain commitments. j will continue to compute the hash value of those two chain commitments to verify the ID i . If the verification succeeds, j will accept those two key chain commitments.

It costs node i about 17.27 ms to generate an encrypted message of the key chain commitments, and costs node j about 27.01 ms to decrypt the message and verify the key chain commitments. Both i and j need to compute their pairwise key by evaluating the polynomial share. The additional code space needed (in ROM) is 2734 bytes, and the extra data space needed (in RAM) is $130 + 29 \times (n - 1)$ bytes for a cluster of n members. Specifically, every member needs to use 29 bytes of memory to store the ID (2 bytes), the two key chain commitments (2×8 bytes), the pairwise key (8 bytes), and some state information (3 bytes) for every other cluster member. In addition, another 130 bytes is needed for the pre-assigned polynomial share, the data buffers, and some other control-flow variables.

Candidate list randomization: We have implemented the shuffle algorithm in Figure 1. This algorithm is required only

once and used to generate the candidate list for the leader election initialization. The execution time is 457.49 ms. It also costs each node 308 bytes of extra code space in ROM, and 54 bytes of extra data space in RAM.

Key release and verification: In each round of election, every node needs to spend 0.24 ms in broadcasting a new YES or NO key in the chains. It will also spend 10.19 ms in processing each of the released keys from other sensor nodes. The corresponding code requires 836 bytes of extra space in ROM and $R + 4$ bytes of extra space in RAM, where R is the maximum number of leader election rounds. In our experiment, we set $R = 50$.

Simple recovery protocol: This protocol is used for packet loss tolerance. Before serving as the new leader, a sensor node will send all its collected NO keys and fresh YES keys to other sensor nodes for confirmation. Because the RF transceiver of the TelosB motes only supports up to 102 bytes of payload in each packet, those keys will be separated in several messages. In our experiment, each message contains 9 keys and needs 27.67 ms for assembling and computing the MAC. It also costs the receiving node 26.83 ms to verify the MAC and the keys in each message. For a cluster of 50 nodes, all these keys can be transmitted in 6 messages. If all the keys are correct, the receiving node will need 9.728 ms to generate the confirmation message as the reply. Verifying such a confirmation message needs 9.785 ms. The costs of sending and verifying the keys are the same as before. In our experiment, this recovery protocol needs 1766 bytes of extra space in ROM and 16 bytes of extra space in RAM.

Summary: The code size and running time of each component is summarized in Table II. In our experiment, it costs 402 bytes of ROM space and $80 + 8 \times (n - 1)$ bytes of RAM space for a sensor node to set up pairwise keys with all other $n - 1$ sensor nodes using the Blundo's key establishment scheme. It takes 2.22 ms to compute each key using polynomial evaluation. In our implementation, there is 1438 bytes of ROM space allocated to the code for symmetric cryptographic operations. It is also important to notice that our protocol utilizes two basic security services, pairwise key establishment and symmetric cryptographic operations. These two basic services are common for many other security protocols. As a result, our protocol can simply reuse these security modules if they are available in the system, saving significant cost. This means that the actual cost of our protocol can be less than what we see in Table II.

VI. CONCLUSION AND FUTURE WORK

This paper presents an efficient and resilient cluster leader election protocol for sensor networks. Our analysis indicates that the proposed protocol is efficient and effective in dealing with malicious attacks. It is very difficult for an adversary to disrupt the protocol without launching intensive channel jamming attacks for a long time period or compromising a large number of cluster members. This protocol is also resistant to message loss; it can quickly recover from any failure as long as the benign cluster members are well-connected during the time of recovery.

There are still many open problems in secure clustering protocols. In this paper, the new cluster leader is selected based on the remaining energy on cluster members for the load

balancing purpose. We plan to extend our protocol to support other election metrics such as mobility and local network connectivity. We are also interested in detecting and revoking those misbehaving cluster members.

Acknowledgment This work is supported by the Army Research Office (ARO) under grant W911NF-07-1-0227. The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

- [1] D. Liu, "Resilient cluster formation for sensor networks," in *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS)*, 2007, p. 40.
- [2] K. Sun, P. Peng, P. Ning, and C. Wang, "Secure distributed cluster formation in wireless sensor networks," in *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC)*, 2006, pp. 131–140.
- [3] I. Khalil, S. Bagchi, and N. Shroff, "Analysis and evaluation of secos, a protocol for energy efficient and secure communication in sensor networks," *Ad Hoc Networks.*, vol. 5, no. 3, pp. 360–391, 2007.
- [4] M.-Y. Hsieh, Y.-M. Huang, and H.-C. Chao, "Adaptive security design with malicious node detection in cluster-based sensor networks," *Comput. Commun.*, vol. 30, no. 11-12, pp. 2385–2400, 2007.
- [5] L. B. Oliveira, A. Ferreira, M. A. Vilaça, H. C. Wong, M. Bern, R. Dahab, and A. A. F. Loureiro, "Secleach-on the security of clustered sensor networks," *Signal Process.*, vol. 87, no. 12, pp. 2882–2895, 2007.
- [6] M. Sirivianos, D. Westhoff, F. Armknecht, and J. Giro, "Non-manipulable aggregator node election for wireless sensor networks," in *ICST WiOpt*, Jan. 2007.
- [7] S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley, "Leader election algorithms for wireless ad hoc networks," in *Proceedings of DARPA Information Survivability Conference and Exposition*, 2003, pp. 261–272.
- [8] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *IEEE Computer*, vol. 35, no. 10, pp. 54–62, 2002.
- [9] J. Newsome, R. Shi, D. Song, and A. Perrig, "The sybil attack in sensor networks: Analysis and defenses," in *Proceedings of IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Apr 2004.
- [10] Y. Hu, A. Perrig, and D. Johnson, "Packet leashes: A defense against wormhole attacks in wireless ad hoc networks," Department of Computer Science, Rice University, Tech. Rep. DEC-TR-506, 2001.
- [11] C. Hartung, J. Balasalle, and R. Han, "Node compromise in sensor networks: The need for secure systems," U. Colorado at Boulder, Tech. Rep. CU-CS-990-05, Jan. 2005.
- [12] B. Parno, A. Perrig, and V. Gligor, "Distributed detection of node replication attacks in sensor networks," in *Proceedings of IEEE Symposium on Security and Privacy*, May 2005.
- [13] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-secure key distribution for dynamic conferences," in *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, 1993, pp. 471–486.
- [14] R. Durstenfeld, "Algorithm 235: Random permutation," *Commun. ACM*, vol. 7, no. 7, p. 420, 1964.
- [15] D. Liu, P. Ning, and W. Du, "Detecting malicious beacon nodes for secure location discovery in wireless sensor networks," in *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2005, pp. 609–619.
- [16] R. Poovendran and L. Lazos, "A graph theoretic framework for preventing the wormhole attack in wireless ad hoc networks," *Wirel. Netw.*, vol. 13, no. 1, pp. 27–59, 2007.
- [17] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. S. J. Pister, "System architecture directions for networked sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104.
- [18] Crossbow Technology Inc., "Wireless sensor networks," <http://www.xbow.com/Products/productdetails.aspx?sid=156>.
- [19] R. Rivest, "The RC5 encryption algorithm," in *Proceedings of the 1st International Workshop on Fast Software Encryption*, vol. 809, 1994, pp. 86–96.
- [20] D. Liu, R. Li, and P. Ning, "Tinykeyman: Key management for sensor networks," <http://cdl.csc.ncsu.edu/software/TinyKeyMan/>.