Secure Leader Election in Wireless Ad Hoc Networks

Sudarshan Vasudevan[1], Brian DeCleene[2], Jim Kurose[1], Don Towsley[1]

[1]Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003
{svasu,kurose,towsley}@cs.umass.edu

[2]TASC, Inc.
55 Walkers Brook Drive
Reading, MA 01867
btdecleene@tasc.com

UMass Computer Science Technical Report 01-50

Abstract

The problem of secure leader election in wireless ad hoc networks is considered and two cheat-proof election algorithms are proposed: *Secure Extrema Finding Algorithm (SEFA)* and *Secure Preference-based Leader Election Algorithm(SPLEA)*. SEFA assumes that all elector-nodes share a single common evaluation function that returns the same value at any elector-node when applied to a given candidate-node. When elector-nodes can have different preferences for a candidate-node, the scenario becomes more complicated. Our *Secure Preference-based Leader Election Algorithm (SPLEA)* deals with this case. Here, individual utility functions at each elector-node determine an elector-node's preference for a given candidate-node. We formally specify and prove correctness and security properties of SEFA and SPLEA using temporal logic. Their message complexity and signature/verification costs are evaluated.

Keywords : wireless ad hoc networks, secure leader election, network protocols, formal specification and verification.

1 Introduction

Leader election algorithms find many applications in both wired and wireless distributed systems. In group communication protocols, for example, a new coordinator must be elected when a group coordinator crashes or departs the system. The problem of leader election in distributed systems has been well-studied and there is a large body of literature for this problem; a good survey can be found in [1]. Most of these works describe *extrema finding* algorithms that elect a node with the maximum identifier from among a set of candidate-nodes.

Recently, there has been considerable interest in using leader-election algorithms in wireless environments for key distribution [3], routing coordination [8], sensor coordination [13], and general control [6, 5]. Here, user mobility may result in frequent leader election, making the process a critical component of system operation. Security is also of particular concern in any wireless environment. Existing leader-election algorithms implicitly assume complete trust among the nodes participating in the leader election process, and consequently are vulnerable to a variety of attacks.

In this paper, we thus consider the problem of secure leader election in ad hoc networks. We present two algorithms that use a round-based hierarchy-building approach towards leader election:

- Our **Secure Extrema Finding Algorithm (SEFA)** belongs to the category of *extrema finding* algorithms. SEFA assumes that all elector-nodes share a single, common evaluation function that returns the same "value" at any elector-node when applied to a given candidate-node. A candidate-node's identifier (name), battery life, or certified level-of-trust within the system are examples of values for which all nodes might well hold such a common view of a candidate node.

- The leader-election problem is more complicated when nodes can have different preferences for a leader. For example, if topological distance is taken as a metric for determining a node's preference among candidate-nodes, each elector-node might well have a different view of the "value" of each candidate. Our **Secure Preference-based Leader Election Algorithm (SPLEA)** deals with this case. Here, individual utility functions at each elector-node determine the elector-node's preference for a given candidate-node, and (loosely speaking) SPLEA serves to aggregate individual elector-node preferences into a single, system-wide choice of a leader.

We formally specify these two algorithms and prove their correctness and security properties, using linear time temporal logic as the formal tool for this purpose. We also analyze their message complexity and the number of digital signature and verification operations required.

The remainder of the paper is organized as follows. In Section 2 we discuss related work. Section 3 describes our assumptions and algorithms and messaging in detail. Section 4 formally describes the various properties we prove. In Section 5 we present an analysis of our algorithms. Section 6 discusses our conclusions and agenda for future work. Finally the appendix includes pseudo-code for SPLEA and detailed proofs of its correctness and security properties.

2 Related Work

Many algorithms for leader election, clustering, and hierarchy construction have been proposed. Leader election algorithms for mobile ad hoc networks have been proposed in [5, 6]. In [5], an algorithm is proposed to elect a leader for each connected component of an ad hoc network. This algorithm is based on TORA [14] (Temporally Ordered Routing Algorithm), which in turn is based on the Gafni-Bertsekas algorithm for constructing a destination-oriented directed acyclic graph[15]. The algorithm uses the mechanism in TORA to detect partitions, with the node detecting a partition being elected as a leader. The leader election algorithms proposed in [6] are classified into two categories, *Non-Compulsory* protocols (which do not affect the motion of nodes) and *Compulsory* protocols (which determine the motion of some or all of the nodes). The *Non-Compulsory* protocols require mobile nodes to move in order to meet and exchange information. If nodes do not meet , the protocol may not elect a unique leader. The mobile nodes are assumed to move in such a way that the number of elector-nodes that participate in protocol execution decreases with time. When there is only one participant left, that node becomes the *leader*. In order to elect a unique leader (with a high probability of success), the *Compulsory* protocols require nodes to perform a random walk on the graph.

The multicast operation of the Ad Hoc On-Demand Distance Vector Routing Algorithm [8] elects a leader for its multicast group. Here, the first member to join the group becomes the leader. When the group partitions, a leader is elected for each group; when partitions merge, the protocol ensures that only one of the group leaders takes leadership of the reconnected partitions.

The top-down hierarchy construction scheme in [10] elects a leader by a simple bully algorithm, where each member broadcasts advertisements to all other members. The Multicast Archie Server Hierarchy (MASH) [9] organizes nodes into a two-level hierarchy of parents and children. Certain nodes choose to become parents and invite other nodes to become their children. A node chooses the nearest parent that is willing to accept it as a child.

The clustering/hierarchy construction proposals in [7, 11, 12, 13] use a bottom-up approach that we have adapted for SEFA and SPLEA. The algorithm in [7] constructs a hierarchy in a sensor network of low power devices by promoting nodes with higher energy up the hierarchy. A Landmark hierarchy is constructed in a bottom-up fashion in [11] for routing purposes. In [12], a structure called an *AC Hierarchy* is constructed that logically organizes processors into various levels. In [13], sensors organize themselves into clusters and cluster heads are elected based on localized coordination and control. In addition, cluster heads are randomly rotated, with each node serving as a head only for a fixed duration. In order to minimize energy spent in communicating with the remote base station, cluster members communicate to the base station through their cluster heads.

As we will see, our leader election algorithms use the idea of bottom-up hierarchy construction, and are similar to the clustering proposals in that respect. However, none of the algorithms described above consider the the notion of *secure* leader election. Indeed, it is not clear whether security can easily be built into their protocols. The main contribution of our work is the development of secure leader-election algorithms, the proof of their correctness and security properties, and the analysis of their overheads.

3 Secure Distributed Leader Election Algorithms

In this section, we describe two algorithms, SEFA and SPLEA, for secure leader election. Each of these algorithms adopts a hierarchical round-based approach in which a leader is elected in a bottom-up fashion. During round one, candidate-nodes compete with their one-hop neighbors. During the second round, the winners from the first round compete with the other round-one winners that are two-hops away. These rounds continue until a single leader is elected.

The fundamental difference between our two algorithms is that SEFA assumes that all elector-nodes share a single, common evaluation function that returns the same “value” at any elector-node when applied to a given candidate-node. SEFA also requires that the parameters used to select the leader in each round remain constant. Thus, the “goodness” of a node is independent of time. SPLEA supports a heterogeneous decision model in which different elector-nodes may have different preferences for a leader. For example, if topological distance is taken as a metric for determining a node’s preference among candidate-nodes, each elector-node might well have a different view of the “value” of each candidate-node. In SPLEA, individual utility functions at each elector-node determine the elector-node’s preference for a given candidate-node, and (loosely speaking) SPLEA serves to aggregate individual elector-node preferences into a single, system-wide choice of a leader.

3.1 Objectives, Constraints, and Assumptions

In developing our secure election algorithms for wireless networks, we must first define the security objectives and related assumptions. Foremost, we assume that malicious nodes have the capacity to “snoop” on a conversation and potentially corrupt messages. However, they cannot delete messages from the network. That is, message loss can be detected. The goal of the proposed algorithms is to efficiently elect a leader without enabling a malicious node to change or disrupt the election process. We make the following assumptions about the nodes and the system architecture:

1. **Unique and Ordered Node IDs**: All nodes have unique identifiers. These are used to construct certificates and establish secure communications. They are also used to identify participants during the election process. Also, node IDs are ordered and therefore useful in breaking ties during election.
2. **Clock Synchronization**: The clocks of all nodes in the network are synchronized. This is used to prevent replay attacks.
3. **Secure Node-to-Node Communications**: Each node *i* has a (public-key,private-key) pair (P_i,K_i) that can be used to securely establish a communication link between any two nodes. The public key is considered shared information and is available to all members of the domain.
4. **Reliable Node-to-Node Communications**: The ad hoc network is modeled as a discrete time system where a node takes one time unit to deliver a packet reliably to its immediate neighbor.
5. **Node Certificates**: Each node has two certificates. The first certificate is a one-time certificate that binds the node's identity with its public key and static characteristics such as its trust-level,processing capabilities etc. This certificate is available from a *Trust Authority* (TA) that is well known to all nodes. The second certificate is shorter-term, and binds dynamic characteristics of a node (e.g., the density of nodes in its region) to the node's identity. This certificate is obtained from a trusted *Group Authority* (GA), as described later. These latter certificates may change from one election to the next, but remain constant (and valid) during a given election.
6. **Reliability and Survivability**: Participating nodes are reliable and do not fail during the election process. Furthermore, no messages are lost or delayed due to buffering or other performance problems at the receiver.
7. **Asymmetric links**: The communication links in the ad hoc network may or may not be bi-directional.
8. **Network Topology**: Although, nodes can be mobile before and after the election, it is assumed that the network topology remains static for the duration of election.

3.2 Secure Extrema Finding Algorithm(SEFA)

SEFA is an *extrema-finding* algorithm in which all nodes use a Common Election Algorithm (CEA) to determine their respective votes. Specifically, we define the CEA as a mapping from m decision factors, $\mathbf{X}_i$, that are associated with node i into the real numbers ($f(\mathbf{X}_i)$, $f : \mathbf{R}^m \rightarrow \mathbf{R}$) such that the larger the CEA value, the more "desirable" the node is as a leader. Decision factors might include node identity, battery life, or level-of-trust. It is important to note that since all elector-nodes use the same CEA, all elector-nodes will compute the same CEA value for a given candidate node. As we will see, SEFA securely elects the node with the maximum CEA value as the leader. We emphasise that the list of decision factors is defined based on the selected algorithm (or class of algorithms in the case of SPLEA) and the number/format of decision factors is chosen accordingly and agreed upon by all nodes prior to the start of election.

The various actions and states taken by an elector node are shown in the finite state machine in Figure 1. Each of the states is described below:

- ***Initialization State***: At the start of the election process, node *i* initializes *Winner*, a boolean variable, to true. This variable reflects *i*'s candidacy status and is true as long as *i* is still a candidate during algorithm execution.

- ***Election State***: The *Election* state corresponds to the beginning of an election. During round k, when in the *Election State*, each node broadcasts an *ELECTION* message to all nodes that are k-hops or less from node i and starts a timer T_e that is proportional to k. This timer interval ensures that node *i* collects *ELECTION* messages from all other candidates whose distance to *i* is *k* or less. Node i's *ELECTION* message contains the values of the decision factors that are needed by the CEA to calculate node i's CEA value. Note that the values for all decision factors used in CEA must be provided by the node in its *ELECTION* message. If a decision factor is not available, its value is taken to be 0. Before sending an *ELECTION* message, the entire message is hashed and the hash is signed using i's private key to protect message integrity. On expiration of T_e, *i* enters state the *Parent Nomination* state.

- ***Parent Nomination State***: On receiving advertisements from other candidate nodes, a node must determine the node that it prefers from among those from whom it has received an *ELECTION* message. This is done via the *TOP* function in Figure 1. This function computes the CEA value of every node from which it received an *ELECTION* message using the transmitted decision factors, and returns a node *w* with maximum CEA value. If *w* is a node different from *i*, *i* accepts *w* as its parent and sends an *OK* message to *i*. If *i* itself is the maximum CEA-value node, then *i* does not have to choose a parent and therefore enters the *Adoption State.* In the event of a tie, a tie-breaking mechanism such as node with maximum identifier, is used. The *OK* message originated by *i* contains a certificate that binds *i*'s identity with that of its parent. This certificate is signed using *i*'s private key. As before, a hash of the message is computed and signed using *i*'s private key. Since *i* is no longer a candidate in the election, it sets its variable *Winner* to false. Even though *i* is no longer a candidate, *i* could have been chosen as the parent of some other node. In order to receive their *OK* messages, *i* enters the *Adoption State.*

- ***Adoption State***: In this state, a node waits for *OK* messages from its children-to-be. All parent-child relationships in the hierarchy are established in this state. Every node *i* maintains a list, I_i, of its immediate children in the hierarchy and adds nodes in this list as it receives *OK* messages from them.

  Node *i* starts a timer T_o on entering this state, waiting for *OK* messages from its potential children. Upon receipt of an *OK* message, *i* adds the originator of the *OK* message to the set of its immediate children, I_i. On expiration of T_o, *i* enters *Parent/Decision State.*

- ***Parent/Decision State***: In this state, *i* sends a *PARENT* message to every node from which it received an *OK* message. It starts a timer T_p for its *PARENT* message to reach its children and also for receiving a *PARENT* message from its parent. Following expiry of T_p, *i* evaluates its candidacy status and decides an appropriate course of action. If the boolean variable *Winner* is still true and if *L* rounds of election are not yet over, it

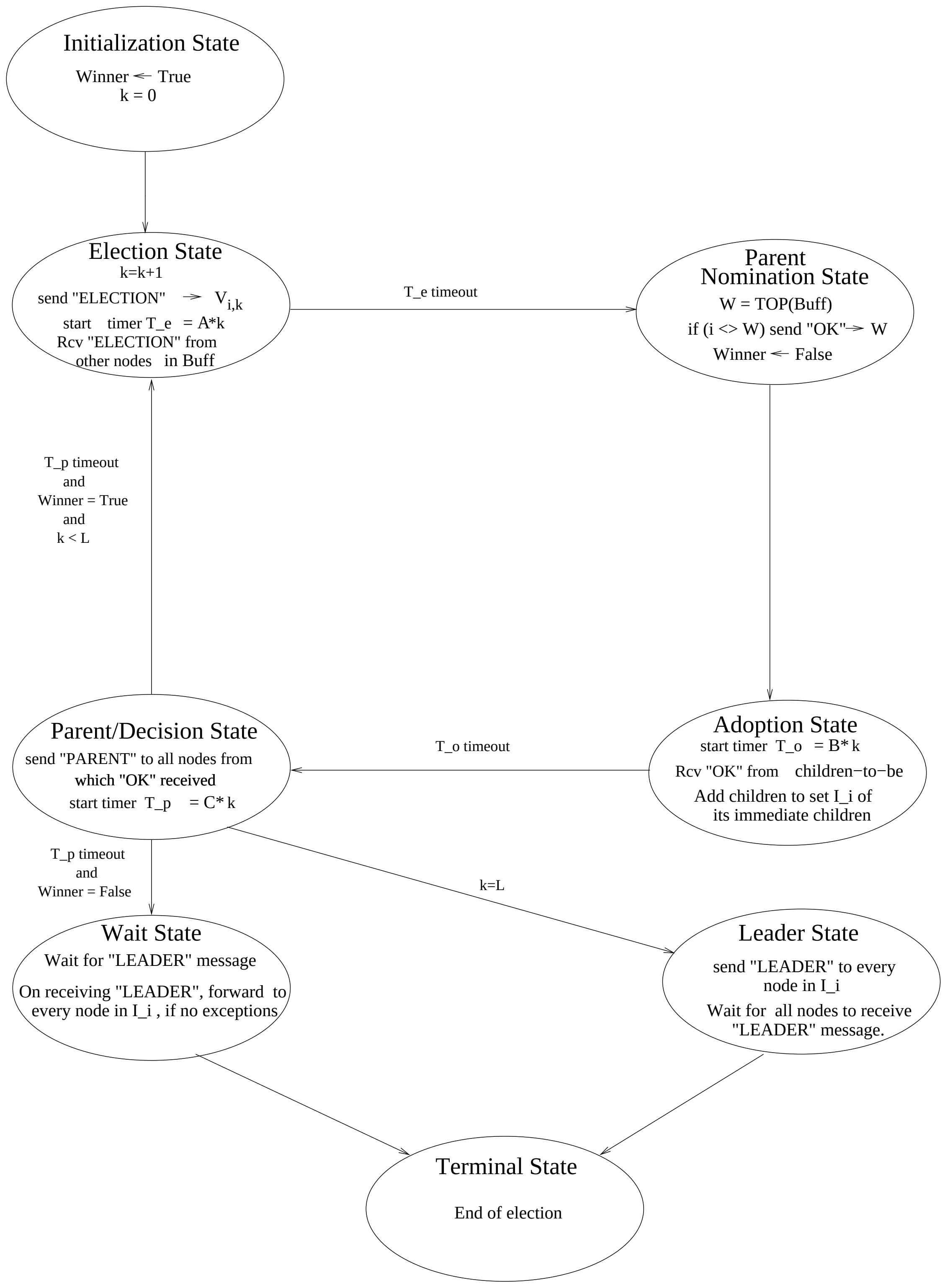
Initialization State
Winner <- True
k = 0
Election State
k=k+1
send "ELECTION" -> V_i,k
start timer T_e = A*k
Rcv "ELECTION" from other nodes in Buff
T_e timeout
Parent Nomination State
W = TOP(Buff)
if (i <> W) send "OK"-> W
Winner <- False
T_p timeout and Winner = True and k < L
Parent/Decision State
send "PARENT" to all nodes from which "OK" received
start timer T_p = C* k
T_o timeout
Adoption State
start timer T_o = B* k
Rcv "OK" from children−to−be
Add children to set I_i of its immediate children
T_p timeout and Winner = False
k=L
Wait State
Wait for "LEADER" message
On receiving "LEADER", forward to every node in I_i , if no exceptions
Leader State
send "LEADER" to every node in I_i
Wait for all nodes to receive "LEADER" message.
Terminal State
End of election

Figure 1: Finite State Machine diagram of SEFA. $V_{i,k}$ denotes the set of all nodes *k* hops away or less from node *i*

means that *i* is still a candidate in the election and so *i* loops back to *Election State* to participate in the next round. If *Winner* has already been set to false, *i* goes to *Wait State*. If *L* rounds are over and the boolean variable *Winner* is still true for *i*, it enters the *Leader State*.

- ***Leader State***: This state marks the end of all the rounds of election. Node *i* first verifies that all nodes with which it competed in round *L* have sent an *OK* message, accepting *i* as their parent. If this is not the case, *i* broadcasts an exception message to ensure that some other node does not maliciously declare itself the leader after the *L*th round.

  Following this, *i* sends a *LEADER* message to all other nodes, declaring itself the leader. This is done either distributing the *LEADER* message down the parent/child hierarchy, or by a broadcast to all nodes. In the former case, *i* sends the *LEADER* message to its immediate children in I_i, when in turn forward the *LEADER* message to their immediate children, and so on, until all nodes in the hierarchy receive the *LEADER* message. If none of the recipients of the *LEADER* message report an exception after a finite time interval, *i* becomes the leader of the group.

- ***Wait State***: In this state, *i* simply waits for a *LEADER* message from the eventual leader. Upon receipt of a *LEADER* message, *i* checks to see if the proclaiming leader is not already its child and only forwards the *LEADER* message to its immediate children if it is not. If the proclaiming leader is a child of *i*, then i broadcasts an exception alerting other members of a malicious attack. After entering the *Wait State*, if *i* does not hear a *LEADER* message for a finite time interval, *i* will broadcast an exception.

To illustrate the operation of SEFA, consider the network shown in Figure 2. Each node has a unique identifier followed by a bracketed number indicating its CEA value, and links are assumed to be bi-directional. During the first round, node C receives decision factors from nodes A, B, and D. Calculating their corresponding CEA values and comparing, node C selects node B as its parent, and send an OK message to B. Similarly, A and B also select B as their parent. The final hierarchy resulting from execution of SEFA on graph of Figure 2 is shown in Figure 3. At the end of round 1, nodes B,E,I and K are eligible to compete in round 2. At the end of round 2, I is the only surviving node.

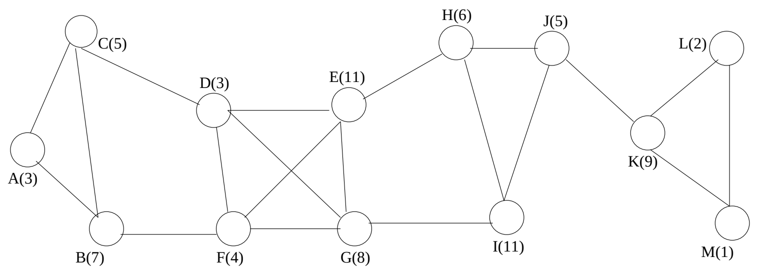

Figure 2: An example graph of an ad hoc network. Circles represent the mobile nodes and the edges between nodes indicate that the two nodes can directly talk to each other.

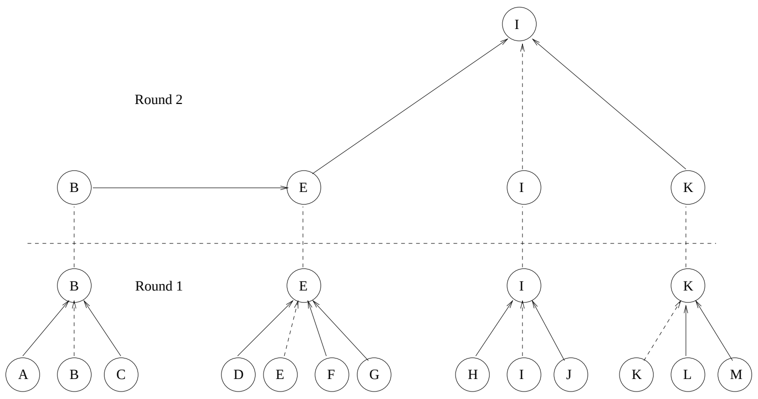

Figure 3: Hierarchy constructed on execution of SEFA

Note that a node *i* may receive *OK* messages from other nodes, but itself have sent an *OK* message to some other node. In this case, *i* sends an *PARENT* message to the nodes that sent an *OK* message to it, but does itself not participate futher in the election, since it has already sent an *OK* message to another node. This situation is illustrated in Figure 3 - node B sends an *OK* messages to node E, while node E itself sends an *OK* message to node I. According to the algorithm, node E will send a *PARENT* message to node B, but will not participate any further in the election.

SEFA elects the node with the maximum CEA value as the leader, a result discussed in Section 4 and proven in Appendix B. Intuitively, at every round high CEA-value nodes are propagated up the hierarchy. It is easy to see that the CEA value of any node is greater than all of its descendants. Since the leader is the root of the hierarchy, its CEA value is the highest of all of its descendants. We also prove that this algorithm prevents malicious nodes, that do not send the required *OK* messages but continue to participate in the election process, from being elected leader.

3.3 Secure Preference-based Leader Election Algorithm (SPLEA)

Recall that SEFA assumes that all elector nodes share a common CEA, and thus view the "desirability" of a candidate node identically. Our second leader election algorithm, SPLEA, allows each elector node to have different view of the "desirability" of a candidate node. We define elector node i's *utility function* for a candidate node, j, as a mapping from the m decision factors of node j to the real numbers ($f_i(\mathbf{X}_j) = u_{j,i}, f : \mathbf{R}^m \rightarrow \mathbf{R}$). We assume, without loss of generality, that the node j with the largest utility value from among a set of candidate nodes is the preferred leader from the perspective of node i.

We note that i's utility function is general in that no assumption is made about the decision factors, $\mathbf{X}_j$, that j passes to i. For example, one of the decision factors that j might pass to i is the number of nodes for which it is currently the parent in the SPLEA algorithm. This is a value that will change as j is promoted up the leader-election hierarchy.

We note that while SEFA elects the leader with the largest CEA value (and in that sense might be termed an "optimal" leader), SPLEA does not elect an optimal leader. Indeed, no global system-wide objective function has been defined and hence the issue of optimality is itself not meaningful. A utility function simply represents the preferences of the individual nodes, and allows elector nodes to vote for a candidate node that they prefer (via their utility function) over other candidate nodes. The voting process serves to aggregate individual elector node preferences into a system-wide decision for the elected leader. The extent to which the elected leader is "good" (from a performance standpoint) or close to optimal (according to some meaningful system-wide objective function) remains an open question for future research.

The finite state machine diagram of SPLEA is shown in Figure 4. The pseudo-code for the algorithm has also been provided in Appendix A. Each state in the state diagram is described below:

- ***Initialization State***: This state is identical to the SEFA Initialization State described earlier.

- ***Election state***: This state is similar to the SEFA Election State described earlier, with the following modification. Each *ELECTION* message in SPLEA also contains the list of descendants of node *i* as of round *k*, maintained by *i* in the form of a list denoted by $D_{i,k}$. The list of descendants is in the form of a set of certificates - one for each of its descendants, thus allowing i to prove to others that it is indeed an ancestor for each of its descendants. The inclusion of the list of i's descendants in the *ELECTION* message allows other nodes to consider this information in their evaluation of the utility of i.

- ***Voting State***: In this state, every node chooses a node other than itself (from the set of candidate nodes from whom it has received an *ELECTION* message) that has the highest utility, and then votes for that node. Informally, the goal of voting is to allow candidates with a large number of votes to proceed on to the next round of the election.

  Upon entering the *Voting State*, *i* executes a function *TOP* that computes the utility of every node (from whom it has received an *ELECTION* message) with respect to *i*, using *i*'s utility function, f_i. If node w has the highest utlity, *i* sends a *VOTE* message indicating *w* as the choice for its parent. This message is to be broadcast by *i* to all members from whom it received an *ELECTION* message, to assure them that it has made a choice for its parent. This rule is enforced in order to prevent nodes from refusing to cast their votes. The *VOTE* message contains a digital signature that binds *i* with its choice. A timer T_v is started to allow other competitor's *VOTE* messages to reach *i*. Following the expiration of this timer, *i* enters the *Announce Votes State.*

- ***Announce Votes State***: In this state, *i* first reports an exception if it has not heard a *VOTE* from any of its competitors, thus forcing nodes which refrained from voting to cast their vote. Node *i* announces the votes it has received from its competitors (i.e., those nodes who chose i as their parent). An *ANNOUNCE* message is broadcast to all members from which *i* received an *ELECTION* message, indicating the votes it has received. The absence of an *ANNOUNCE* message from a competitor indicates that the competitor has received no votes. Again, a timer T_a is started by *i* to collect *ANNOUNCE* messages from other competitors. The expiration of timer T_a causes a transition to the *Parent Nomination State.*

- ***Parent Nomination State***: As in SEFA, each node *i* chooses its parent based on the vote count of each of its competitors, choosing as its parent the node with the maximum number of votes. *i* sends an *OK* message to

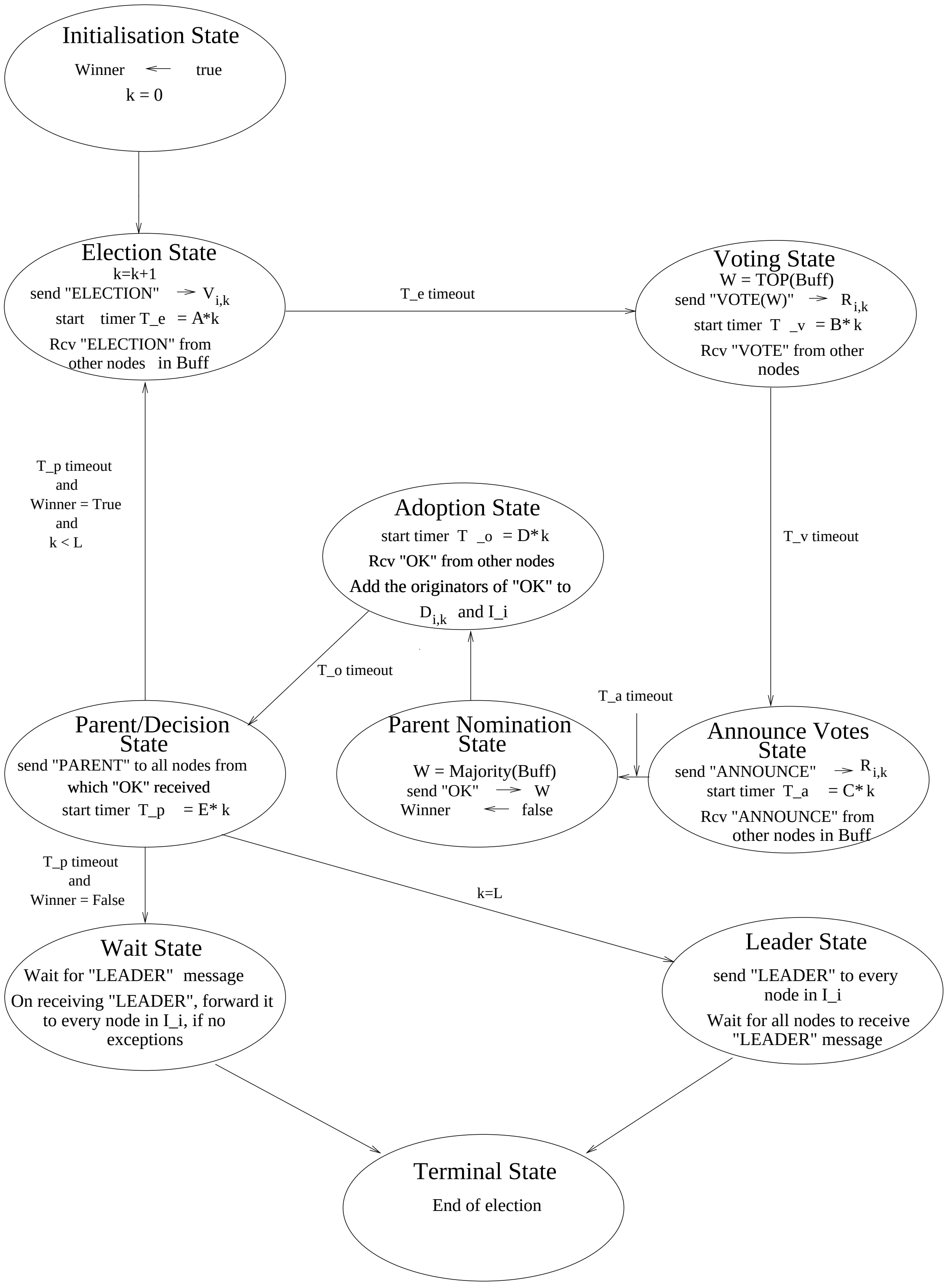

Figure 4: Finite State Machine diagram of SPLEA. $V_{i,k}$ denotes the set of all nodes *k* hops away or less from node *i* and $R_{i,k}$ denotes the set of all nodes from which *i* received *ELECTION* message in round *k*

that node accepting it as its parent and starts a timer T_o. As in the case of the *VOTE* message, the *OK* message is sent to all of its competitors. Upon expiration of T_o, a competitor *j* of node *i* that has received more votes than *i* will report an exception if it does not hear an *OK* from *i*. Once an *OK* is sent, the *Winner* variable is set to false and a transition to *Adoption State* is made.

- ***Adoption State***: This state is identical to the SEFA Adoption state described earlier. The only difference from SEFA is that upon receipt of an *OK* message from node *j*, *i* adds node *j* and all of *j*'s descendants (which it knows from the *ELECTION* message) to the list $D_{i,k}$ of its descendants.
- ***Parent/Decision State***: This state is identical to the SEFA Parent/Decision State described earlier.
- ***Wait State***: This state is identical to the SEFA Wait State described earlier.
- ***Leader State***: This state is identical to the SEFA Leader State described earlier.

An important feature of SPLEA is that it is robust against incomplete lists of descendants in the *ELECTION* message. To illustrate why, let consider the following example. Suppose, that in a certain round *k*, a node *i* chooses *j* as its parent which in turn chooses another node *l* as its parent in the same round. In the following round, *l* only knows about *j* and does not know that *i* is its descendant, since *j* does not forward this information to *l*. This has important security implications, as the node *i*, which is potentially malicious, can now compete with *l* in the next round and get elected as the parent if *l* chooses *i*. With some luck, *i* could survive round *L* as well. This would result in a loop in parent-child relationships. Should this occur, however, *i* will be exposed when its *LEADER* message is received by *j*, which had earlier received an *OK* message from i.

In Appendix B, we formally prove that a node that sends an *OK* message in a round cannot become the leader. This result forms the basis of a cheat-proof leader election.

3.4 Message Security

Secure leader election must ensure that no node can usurp the election by providing false credentials. In this section, we describe the details of the messages formats used in the SEFA and SPLEA algorithms and how they prevent such false results.

Let (P_{TA}, K_{TA}) and (P_{GA}, K_{GA}) be the (public key,private key) pairs of *TA* (Trust Authority) and *GA* (Group Authority) respectively. In general, (P_n, K_n) represents the (private key,public key) pair of node *n* and K_n(*x*) means that data *x* is encrypted using *n*'s private key.

As outlined previously, SEFA requires four messages: *ELECTION*,*OK*, *PARENT* and *LEADER*. SPLEA involves two additional messages, viz. *VOTE* and *ANNOUNCE* and, in addition, modifications to *ELECTION* and *OK* messages have been made.

Election Message

The SEFA *ELECTION* message is used by a node to initiate a new round of the election process. In this message, the node advertises its characteristics to other competitors, and has the following fields:

- Node *i*'s identity
- current time
- round number
- $K_{TA}(i, x_1, x_2, \ldots, x_{m'}, P_i)$
- $K_{GA}(i, x_{m'+1}, x_{m'+2}, \ldots, x_{m''}, \text{current time})$
- $K_{TA}(GA, x_1, x_2, \ldots, x_{m'}, P_{GA})$
- $x_{m''+1}, \ldots, x_m$
- K_i(Hash)

The current time and round number fields indicate the time and the round in which messages are generated. The fourth field in the message is the one-time certificate of *i* that is signed by *TA* using its private key K_{TA}. This certificate binds *i*'s identity with its public key and its static decision factors such as trust level, battery life, node identifier, etc. Note that these characteristics are assumed to be fixed throughout the node's lifetime. Hence it is certified along with the node's public key in the one-time certificate. Since the public key P_{TA} of the *TA* is well known to all members of the domain, every node inside the domain can decrypt the certificate signed by the *TA*. Note that i's one-time TA-signed certificate cannot be tampered with as it is signed using the TA's private key.

The next field is the certificate that *i* has obtained from the *GA*. The *GA* is co-located with the group leader and hands out certificate to every group member when a leader is elected. When the leader departs with the *GA*, every member uses the certificate of the last *GA*. This certificate is shorter-term and binds *i*'s identity with its dynamic characteristics. This certificate can change from one election to next but remains constant during an election. In order to prevent this certificate from being replayed, the *GA* also includes the time of signature in the certificate. Since clocks of nodes are assumed to be synchronized, a recipient of the certificate can determine whether the certificate is recent or old. Since the *GA*'s public key might not be known to the recipient, *i* includes the one time certificate of *GA* signed by the *TA* in the message. This field is followed by the values of the factors in $\mathbf{X}_i$ that have to be taken on trust.

Finally, to protect the message integrity, *i* computes a hash over the entire message using a well-known algorithm such as MD5 [16]. The hash serves two purposes, viz. (i) protects the integrity of the message and (ii) reduces the length of the message to be encrypted. In the absence of the hash, *i* will have to sign every field of the *ELECTION* message with its private key. Encryption/Decryption using a public key/private key can be expensive and depends on the length of the message being encrypted/decrypted. Since a hash is very easy (computationally) to compute and potentially much smaller in length in comparison with the actual message itself, *i* encrypts only the hash with its private key.

The SPLEA *ELECTION* message incorporates descendant information and has the following fields:

- *i*'s identity
- current time

- round number
- K_{TA}(i,$x_1, x_2, \ldots, x_{m'}$,P_i)
- K_{GA}(i,$x_{m'+1}, x_{m'+2}, \ldots, x_{m''}$,current time)
- K_{TA}(GA,$x_1, x_2, \ldots, x_{m'}$,P_{GA})
- $x_{m''+1}, \ldots, x_m$
- For each node *c* in $D_{i,k}$:
  - K_c(*c*,parent(*c*),OK,*d*(*parent(c)*,*c*) round number,current time)
  - K_{TA}(*c*,$x_1, x_2, \ldots, x_{m'}$,P_c)
- K_i(Hash)

Note that $D_{i,k}$ denotes the list of descendants of node *i* as of round *k*. For every descendant in the list $D_{i,k}$, *i* produces a certificate signed by the descendant. This certificate is signed by the issuer in its *OK* message. The public key of the descendant is also included to allow *i*'s competitors verify that *i* is indeed an ancestor of that descendant. The certificate also includes information about a parent's distance to the child, *d*(*p*(*c*),*c*), for a child *c*. Since a node *i* contains the certificate of each node in $D_{i,k}$, *i*'s distance to each of the nodes in $D_{i,k}$ can be computed by *i*'s competitors. This information can be used to evaluate *i*'s regional density i.e. the ratio of number of descendants of *i* to the average distance of *i* to any of its descendants. If a node *i* chooses node *j* as its parent, then *j* adopts *i* and all of its descendants as its own descendants and adds each of those nodes to the list $D_{i,k}$. Thus information about a node's descendants gets aggregated and propagated up the hierarchy as rounds go by.

OK Message

An *OK* message is used by SEFA to accept the results of a round and to select its parent. The *OK* message has the following fields:

- *i*'s identity
- current time
- round number
- identity of *i*'s parent
- K_{TA}(i,$x_1, x_2, \ldots, x_{m'}$,P_i)
- K_i(*i*,$parent_k$(*i*),round number,current time)
- K_i(Hash)

A node *i* accepts a node as its parent, denoted by $parent_k(i)$, by signing a certificate that binds its identity with that of its parent. To prevent the certificate from being replayed, the time of signing the certificate is recorded. In case exceptions occur at a later time, the round number field provides information about the round in which the signature operation was carried out.

Parent Message

A *PARENT* message is used by both the SEFA and SPLEA algorithms and is issued by a node to its proposed children (i.e. those that issued the *OK* message) that they have been assigned as its children. The *PARENT* message has the following fields:

- *i*'s identity
- current time
- round number
- a list of nodes
- $K_{TA}(i,x_1, x_2, \ldots, x_{m'},P_i)$
- K_i(Hash)

Leader Message

After *L* rounds of election are over, the leader (under both SEFA and SPLEA) sends out a *LEADER* message to every node in I_i. The nodes in I_i in turn forward the *LEADER* message to their immediate children and the process repeats until all nodes in the hierarchy receive the *LEADER* message. The *LEADER* message has the following fields:

- *i*'s identity
- current time
- $K_{TA}(i,x_1, x_2, \ldots, x_{m'},P_i)$
- K_i(Hash)

Each recipient of this message checks to see if the proclaiming leader is not already its immediate child and broadcasts an exception if it is. Otherwise, it waits for a finite timer interval to see if any other node reports an exception to the result of the election. If no exceptions are reported, then the originator of the *LEADER* message is accepted by all other members of the group as the new leader.

Vote Message

A *VOTE* message is unique to SPLEA and used by a node to announce its "choice" of a leader other than itself. Each vote is signed by the issuer. The choice of leader is indicated as *j* in the certificate. The *VOTE* message has the following fields :

- *i*'s identity
- current time
- round number
- $K_{TA}(i,x_1, x_2, \ldots, x_{m'},P_i)$
- $K_i(i,j,$VOTE,round number,current time)
- K_i(Hash)

The third argument in the certificate is used to indicate that it is the certificate used in the *VOTE* message and hence different from the certificate used in the *OK* message. This is done in order to prevent a node from using *VOTE* message certificates in an *ELECTION* message in future rounds. The remaining fields are for purposes of message integrity and replay protection, as already discussed.

As with the *VOTE* message, the SPLEA *OK* message carries a certificate that makes the SPLEA *OK* message different from the SEFA *OK* message. The certificate in a node *i*'s *OK* message also includes shortest path distance of parent node *j* to itself, *d*(*j*,*i*). This information will be helpful in computation of regional density of its ancestors in future rounds. The *OK* message in SPLEA is thus:

- *i*'s identity
- current time
- round number
- identity of *i*'s parent
- $K_{TA}(i,x_1, x_2, \ldots, x_{m'},P_i)$
- K_i(*i*,*j*,OK,*d*(*j*,*i*),round number,current time)
- K_i(Hash)

Announce Message

This message is also unique to SPLEA. Node *i* announces the set of votes it has received in the *ANNOUNCE* message. Each vote in the *ANNOUNCE* message is signed by the issuer (in its *VOTE* message) to prevent the announcing node from spoofing the results and making it appear as if it has received more votes than were actually issued.

Let $G_{i,k}(t)$ = $\{x|$ *x* sends a *VOTE* message in round *k* with *i* as its choice for its parent $\}$. Node *i*'s *ANNOUNCE* message has the following fields :

- *i*'s identity
- current time
- For each node c in $G_{i,k}(t)$:
  1. K_c(*c*,*i*,VOTE,round number,current time)
  2. $K_{TA}(c,x_1, x_2, \ldots, x_{m'},P_c)$
- K_i(Hash)

4 Formal Specification and Verification of Properties

In Section 3, we described our secure election algorithms. An important contribution of this paper is to formally specify and verify the correctness and security properties of our algorithms. We use linear time temporal logic as the tool for doing so. The authors in [17] have used this tool in proving properties for their Leader Election algorithms. [19] provides an extensive introduction to the use of temporal logic for communication protocols while [18] also discusses the use of temporal logic.

A temporal formula consists of predicates, boolean operators ($\vee, \wedge, \neg, \Rightarrow, \iff$) and quantification operators ($\forall, \exists$) and temporal operators. We make use of only 2 temporal operators for our proofs, viz. $\Box$ which means 'at every moment in the future' and $\Diamond$ which means at 'some moment in the future'. If φ is any arbitrary formula, then $\Box\varphi$ means φ is true at every moment in the future and $\Diamond\varphi$ means φ will be true at some moment in the future.

Before formally specifying the properties of our algorithm, we introduce some notation. Let L_i be a predicate that evaluates to true if *i* is a leader and ${SEND_i}^k(j, msg)$ be a predicate that evaluates to true if node *i* sends *msg* to node *j* in round *k*.

The properties of our algorithms have been divided into two classes: *correctness* properties and *security* properties. Correctness properties include those properties that are required of a leader election algorithm regardless of security aspects; the security properties specify the security claims that our algorithms satisfy.

The properties (C1-C3 and S1-S2) are proved in Appendix B and have been included for review. The correctness properties C1,C2 and C3 hold for both algorithms. Although while proving them we have used some specific details of SPLEA, the proof procedures will remain exactly the same and will differ only in minor details when applied to SEFA. We leave it to the reader to see how proofs for P1,P2 and P3 can be applied to SEFA as well. The properties C3 and S2 are specific to SEFA.

4.1 Correctness Properties

- *The election algorithms elect a unique leader,* under the assumption that the number of rounds (*L*) in the election is at least as large as the diameter of the network.

  $$C1 : (\exists i :: L_i \Rightarrow (\forall j : i \neq j : \neg L_j)) \tag{1}$$

  The proof shows that at the end of *L* rounds of election, if there is a surviving candidate then every other node is in the failed state i.e., is no longer in contention to become the leader.

- *The algorithms satisfy the liveness property* that eventually there is a node *i* that is elected as the leader. More formally, we prove that :

  $$C2 : \Diamond(\exists i :: L_i) \tag{2}$$

  The liveness property is proved by showing that at the end of *L* rounds of election there is at least one surviving candidate and in conjunction with property C1, we infer that the algorithm indeed terminates electing a leader.

- *SEFA elects a node with the maximum CEA value as the leader.* We show that :

$$C3 : (\forall i :: L_i \Rightarrow (\forall j : i \neq j : u_i > u_j)) \tag{3}$$

This property is proved using a very similar reasoning as for property C2. We show that at the end of *L* rounds, the maximum CEA-value node is still in contention and every other node is in the failed state. Hence the leader is indeed the maximum CEA-value node.

4.2 Security Properties

The algorithms that we propose are for secure election and the biggest challenge in designing such an algorithm is to ensure a cheat-proof election mechanism across a distributed system of nodes. In this section, we state the various security properties of our algorithms. Some of these properties are formally proved, while the others are obvious (though no less important) and we state them for sake of completeness.

- *The algorithms guarantee integrity of every message.* This is facilitated by hashing the message contents and signing the hash using the sender's private key. This allows the recipient of the message to verify the signature by using the sender's public key which is also sent in the message. Hence if a malicious intervening node tries to corrupt a message from some other node, it will be detected by the recipient. Also if the integrity of the message is verified, then the source of the message is also verified. This is because only the source knows its private key and hence the signature must have been generated by the source only and no one else.

- *All messages in the algorithms are replay protected.* In each message, the sender is required to include the time when the message is originated. Since all nodes have synchronized clocks, a recipient of a message can determine whether a message is recent or not.

- *The election algorithms are cheat-proof.* We show that a node *i* that sends an *OK* message in any round *k* can never get elected as the leader. This property means that our algorithms prevent nodes from cheating once they lose in a round.

$$S1 : (\forall i, j, k :: SEND_i{}^k(j, OK) \Rightarrow \Box\neg L_i) \tag{4}$$

This property is proved by showing that if a node that sent an *OK* message in a certain round *k*, but competes in the *L*th round and survives that round, its *LEADER* message will reach all nodes in the hierarchy including its parent in round *k*, which will report an exception.

Moreover, in SPLEA, we make it compulsory for nodes to send an *OK* message when they encounter competitors with higher number of votes. As described in Section 3, every node monitors its competitor closely and reports exceptions as and when a malicious behavior is noticed.

Since there is no enforcement mechanism in SEFA for nodes to send *OK* messages, a malicious node might not send an *OK* message even after encountering a node with higher CEA value. We show that if a node *i* is not the maximum CEA-value node, then *i* cannot get elected as the leader even if *i* competes with higher utility nodes and does not send *OK* messages. This forms the basis of a cheat-proof election in SEFA.

	Message Complexity	Network Bandwidth
Worst Case	$O(Ln)$	$O(L^2n)$
Best Case	$O(n)$	$O(n)$
Fully Balanced Hierarchy	$O(mn)$	$O(m^2Rn)$

Table 1: Complexity analysis of the two algorithms

$$S2 : (\forall i, j, k : i \neq j : COMPETE(i, j, k) \wedge (u_j > u_i) \wedge \neg SEND_i(OK) \Rightarrow \Box\neg L_i) \qquad (5)$$

This property is shown by proving that maximum CEA-value node, say *w*, will be a candidate in round *L* and all nodes have to compete with one another in round *L*. Hence if the malicious node does not send an *OK* message accepting *w* as its parent in round *L*, *w* will report an exception.

5 Complexity Analysis

In this section, we summarize our analysis of the two algorithms in various cases. In addition to message complexity, we also study the network bandwidth consumed by various messages. The network bandwidth is measured as the sum of number of hops over all messages in the entire election. Finally, we study signature and verification costs of our algorithms.

Table 1 shows message complexity and network bandwidth consumed by the messages under various scenarios. The complexity parameters are number of rounds of election (*L*), number of nodes in the hierarchy (*n*) and the average number of neighbors of any node(*R*). Three different cases have been considered.

The worst case occurs when every node is *L* hops away from every other node. We would like to emphasize that the worst case cannot occur and we are using this hypothetical situation only as an upper bound on the various costs of our election algorithms. In this case, every node sends an *ELECTION* message in every round of the election until round *L* - 1 but does not hear any one else's *ELECTION* message. In round *L*, every node hears everyone else's *ELECTION* message. The best case is when every node is one hop away from every other node. Finally, a fully balanced hierarchy is formed when every candidate adopts *m* - 1 children at every round of the election.

SEFA and SPLEA have the same message complexity, and the same network bandwidth requirements. Their message complexity is dominated by the *ELECTION* messages in both algorithms and even though there is additional messaging in SPLEA, they do not increasse the complexity. The constants subsumed within the *O* notation are, however, larger for SPLEA than for SEFA.

The signature and verification costs of the two algorithms are shown in Table 2. The complexity parameters are *h*, the height of a node in the hierarchy, *n*, the number of nodes competing in the election and in case of a fully balanced hierarchy, m, the number of children that every non-leaf node in the hierarchy has. The height is 1 for leaf nodes.

Interestingly, in a one hop network (that corresponds to best case in Table 1) every node must perform $O(n)$ verifications and $O(1)$ signatures. The $O(n)$ verification cost is as a result of competing with every other node. Similarly in the case when every node is *L* hops away from every other node (which we refer to as *L* hop network for

	SEFA		SPLEA	
	Signature Cost	Verification Cost	Signature Cost	Verification Cost
One Hop Network	$O(1)$	$O(n)$	$O(1)$	$O(n)$
L Hop Network	$O(L)$	$O(n)$	$O(L)$	$O(n)$
Fully Balanced Hierarchy	$O(h)$	$O(m.h)$	$O(h)$	$O(m^h)$

Table 2: Signature and Verification costs per node of the two algorithms

conciseness), every node has to perform $O(n)$ verifications and $O(L)$ digital signature operations. The table shows the costs in various scenarios.

For a general fully balanced hierarchy, the higher the node is within the hierarchy, the greater is the cost incurred by it due to signature generation and verification. It can be seen that the highest cost is always incurred by the leader. In the case when $h = log_m(n)$, it can be seen that signature cost is $O(log_m(n))$ and verification cost is $O(n)$.

6 Conclusions and Future Work

In this paper, we have proposed two new algorithms for secure leader election in wireless ad hoc networks. SEFA assumes that all elector-nodes share a single common evaluation algorithm that returns the same value at any elector-node when applied to a given candidate-node. We showed that SEFA elects the leader with the maximum CEA value. The *Secure Preference-based Leader Election Algorithm (SPLEA)* deals with the case that each elector node has an individual utility function that determines its preference for a given candidate-node. We formally specified and proved the correctness and security properties of SEFA and SPLEA using temporal logic. Their message complexity and signature/verification costs was evaluated.

We note that while SEFA elects the leader with the largest CEA value (and in that sense might be termed an “optimal” leader), SPLEA has no well-defined system-wide objective function, and hence the issue of optimality is itself not meaningful. The SPLEA voting process serves to aggregate the individual preferences of elector nodes (as expressed through their individual utility functions) into a system-wide decision for the elected leader. The extent to which the elected leader is “good” (from a performance standpoint) or close to optimal (according to some meaningful system-wide objective function) remains an open question for future research.

Many implementation issues must also be addressed before these algorithms can be used in practice. The algorithms we have described are synchronized and proceed in a lockstep fashion. We plan to investigate the possibility of making the election process asynchronous. Nodes are also assumed to not fail (or leave the network) during the election process. Fault tolerance is another important issue that we will address in our future work.

References

[1] G. Tel. Introduction to Distributed Algorithms. Second Edition, Cambridge University Press.

[2] C. Wong, M. Gouda and S. Lam. Secure Group Communication using Key Graphs. In *Proceedings of ACM SIGCOMM ’98*, September 1998.

[3] B. DeCleene *et al.* Secure Group Communication for Wireless Networks. To appear in *Proceedings of MILCOM 2001, VA*, October 2001.

[4] H. Harney and E. Harder. Logical Key Hierarchy Protocol. Internet draft, draft-harney-sparta-lkhp-sec00.txt, March 1999.

[5] N. Malpani, J. Welch and N. Vaidya. Leader Election Algorithms for Mobile Ad Hoc Networks. *Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Boston, MA*, August 2000.

[6] K. Hatzis, G. Pentaris, P. Spirakis, V. Tampakas and R. Tan. Fundamental Control Algorithms in Mobile Networks. In *Proceedings of 11th Annual ACM Symposium on Parallel Algorithms and Architectures, pages 251-260*, 1999.

[7] D. Estrin,R. Govindan,J. Heidemann and S. Kumar. Next Century Challenges : Scalable Coordination in Sensor Networks. In *Proceedings of ACM MobiComm*,August 1999.

[8] E. Royer and C. Perkins. Multicast Operations of the Ad Hoc On-Demand Distance Vector Routing Protocol. In *Proceedings of Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), pages 207-218*, August 15-20, 1999.

[9] A. Rosenstein, J. Li and S. Tong. MASH : The Multicasting Archie Server Hierarchy. ACM Computer Communication Review, 27(3), July 1997.

[10] D. Thaler and C. Ravishankar. Distributed top-down hierarchy construction. In *Proceedings of the IEEE INFOCOMM*, 1998.

[11] D. Coore, R. Nagpal and R. Weiss. Paradigms for Structure in an Amorphous Computer. Technical Report 1614, Massachussetts Institute of Technology Artificial Intelligence Laboratory, October 1997.

[12] P. Tsuchiya. The Landmark Hierarchy : A new hierarchy for routing in very large networks. In *Proceedings of the ACM SIGCOMM*, 1988.

[13] W. Heinzelman,A. Chandrakasan and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of Hawaiian International Conference on Systems Science*, January 2000.

[14] V. Park and M. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proceedings of IEEE INFOCOM*, April 7-11, 1997.

[15] E. Gafni and D. Bertsekas. Distributed Algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, C-29(1):11-18, 1981.

[16] R. Rivest. The MD5 Message Digest Algorithm. RFC 1320, April 1992.

[17] J. Brunekreef, J. Katoen, R. Koymans and S. Mauw. Design and Analysis of Leader Election Protocols in Broadcast Networks. *Distributed Computing*, vol. 9 no. 4, pages 157-171, 1996.

[18] Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems - Specification. Springer-Verlag, New York,1992.

[19] R. Gotzhein. Temporal logic and its applications - a tutorial. *Computer Networks and ISDN systems*, 24:203-218, 1992.

APPENDIX

A Detailed Algorithm (SPLEA)

A.1 Table of Variables and functions

1. Variables of LEADER_ELECTION Algorithm of Node *i* :

Variable Name	**Meaning**	**Domain of Values**
RoundNumber,k	current round number of election	1,2,. . .,L
Winner	true if *i* is still a candidate	True,False
$T_e, T_v, T_a, T_o, T_p, T_l$	Timer Variables	$\mathbf{R}$
A,B,C,D,E	constants	$\mathbf{R}$
Parent	parent of node *i*	character string
Event	event that caused the interrupt	-
M_MessageQueue	array of messages of type *M*	-
$V_{i,k}$	list of nodes each of which is at a distance of *k* hops or less from *i*	-
$R_{i,k}$	list of nodes from which *i* received *ELECTION* message in round *k*	-
$D_{i,k}$	list of descendants of *i* as of round *k*	-
I_i	list of immediate children of *i*	-

2. Functions of LEADER_ELECTION Algorithm of Node *i* :

Function Name	**Meaning**
StartTimer(T)	sets timer value to *T* and starts timer
ExpireTimer(T)	returns true if timer *T* expires
send(*j*,*m*)	*i* sends message *m* to *j*
recv(*m*)	*i* receives a message *m*
check(*m*)	check the validity and authenticity of message *m* returns 1 if valid else returns 0
Top(ELECTION_MessageQueue)	verify signatures of all the messages in ELECTION_MessageQueue and return a *VOTE* message
Majority(VOTE_MessageQueue)	verify signatures of all the messages in VOTE_MessageQueue and return the parent of node *i*

1

A.2 Pseudo Code for SPLEA

```
LEADER_ELECTION() at node i :
    RoundNumber = 1;
    Winner = True;
    while (RoundNumber $\leq L$) do
        if (Winner = True) then
            k = RoundNumber;
            construct ELECTION packet for the current round;
            send($V_{i,k}$,ELECTION);
            $T_e$ = A.k;
            StartTimer($T_e$);
            while (1) do
                switch(Event)
                    case recv(ELECTION) : add ELECTION message to the ELECTION_MessageQueue
                                          add the originator of ELECTION message to $R_{i,k}$
                    case ExpireTimer($T_e$) : exit out of while loop;
                endswitch
            endwhile
            VOTE = Top(ELECTION_MessageQueue);
            send($R_{i,k}$,$VOTE$);
            $T_v$ = B.$k$;
            StartTimer($T_v$);
            while (1) do
                switch(Event)
                    case recv($VOTE$) : add $VOTE$ message to the VOTE_MessageQueue;
                    case ExpireTimer($T_v$) : exit out of the while loop
                endswitch
            endwhile
             Broadcast an exception if a node $j \in R_{i,k}$ did not send a VOTE message
             send($R_{i,k}$,ANNOUNCE);
            $T_v$ = C.$k$;
            StartTimer($T_a$);
            while (1) do
                switch(Event)
                    case recv(ANNOUNCE) : add ANNOUNCE message to the ANNOUNCE_MessageQueue
                    case ExpireTimer($T_a$) : exit out of while loop;
                endswitch
            endwhile
             Parent = Majority(VOTE_MessageQueue);
             if (Parent $\neq$ $i$) then
                 Winner = False;
                 send($R_{i,k}$,$OK$);
             endif
            $T_o$ = D.$k$;
            StartTimer($T_o$);
            while (1) do
                switch(Event)
                    case recv($OK$) : add $OK$ message to the OK_MessageQueue;
                                      if originator chooses $i$ as its parent then
                                          add the originator of OK message to $I_i$
                                          add the originator of $OK$ message and its descendants to $D_{i,k}$;
                                      endif
                    case ExpireTimer($T_o$) : exit out of the while loop
                endswitch
            endwhile
             Broadcast an exception if a node $j \in R_{i,k}$ with fewer votes than $i$ did not send an OK message.
            if (not empty(OK_MessageQueue)) then
                send(all nodes that accepted $i$ as parent, PARENT);
            endif
            $T_p$ = E.$k$;
            StartTimer($T_p$);
            while (1) do
                switch(Event)
                    case recv(PARENT) : check(PARENT);
```

```
            case ExpireTimer($T_p$) : exit out of the while loop
          endswitch
        endwhile
        RoundNumber = RoundNumber + 1;
      else
        $T_l = \frac{(L(L+1)-k(k+1)).(A+B+C+D+E)}{2}$
        StartTimer($T_l$);
        while (1) do
          switch(Event)
             case recv(LEADER) : exit out of the while loop;
            case ExpireTimer($T_l$) : exit out of while loop;
          endswitch
        endwhile
      endif
  endwhile
  if (Winner == True) then
      send($I_i$,LEADER);
  else
      Broadcast exception if LEADER not received
      if (check(LEADER) == 1)
        forward LEADER to every node in $I_i$
      else broadcast exception;
      endif
  endif
```

B Proofs of Properties

We first introduce some notations and terminology that we will be using while verifying the properties of our algorithm.

1. $d(i,j)$: shortest path distance from *i* to *j*
2. $V_{i,k}$: set of all nodes that are at a distance of *k* or less from node *i*
3. $C_{i,k}$: a predicate that evaluates to true if node *i* is a candidate at the beginning of round *k*.
4. L_i : a predicate that evaluates to true if node *i* is a leader.
5. $p(i)$: parent of node *i* in some round that *i* that participated in.
6. $p_k(i)$: parent of node *i* in round *k*.
7. I_i = $\{j|p_(j) = i\}$ is the set of *i*'s immediate children in the hierarchy.

8. $D_{i,k}$: set of descendants of node *i* at the beginning of round *k* and is defined as follows :

$$D_{i,k} = (\forall j \in V_{i,k-1} s.t. p_{x \in [1,k-1]}(j) = i : \bigcup D_{j,k-1} \tag{6}$$

$$D_{i,2}(t) = \{j | p_1(j) = i\} \tag{7}$$

9. $N^k(i)$: number of votes that node *i* received in round *k*.

10. u_i : CEA-value of node *i*.

11. $S_k = \{i | C_{i,k}\}$. In other words, S_k is the set of surviving candidates at the beginning of round *k*.

12. $DEAD(i)$: a predicate that evaluates to true if node *i* is dead.

13. $SEND_i(j, msg)$: a predicate that evaluates to true if node *i* sends *msg* to node *j*.

14. ${SEND_i}^k(j, msg)$: a predicate that evaluates to true if node *i* sends *msg* to node *j* in round *k*.

15. $RCV_i(j, msg)$: a predicate that evaluates to true if node *i* receives *msg* from node *j*.

16. ${RCV_i}^k(j, msg)$: a predicate that evaluates to true if node *i* receives *msg* from node *j* in round *k*.

17. $COMPETE(i, j, k)$: a predicate that is defined as follows :

$$(\forall i, j, k : i \neq j : C_{i,k} \wedge C_{j,k} \wedge COMPETE(i, j, k) \iff {RCV_i}^k(j, ELECTION) \wedge {RCV_j}^k(i, ELECTION)) \tag{8}$$

18. $TIMEOUT_k(T)$: evaluates to true if timer *T* expires in round *k*.

19. $EXCEPTION(i, k)$: evaluates to true if node *i* broadcasts an exception.

20. *OK*(*i*,*l*) : node *i* chooses node *l* as its parent.

21. *LEADER*(*m*) : *LEADER* message originated by node *m*.

We now formally define some relevant assumptions.

Assumption 5.1

$(\forall i :: C_{i,1} \Rightarrow \Box(\neg DEAD(i)))$

Assumption 5.2

Initially $(\forall i :: C_{i,1})$ holds true.

Assumption 5.3

$(\forall i, j : i \neq j : 0 < d(i, j) \leq L)$

Assumption 5.4

For the sake of simplicity of verification and without compromising on generality we assume that:

In SPLEA : $(\forall i, j : i \neq j : N^k(i) \neq N^k(i))$

In SEFA : $(\forall i, j : i \neq j : u_i \neq u_j)$

Assumption 5.5

V(*t*) is finite.

We restate the properties we are about to prove :

1. C1 : $(\exists i :: L_i \Rightarrow (\forall j : i \neq j : \neg L_j))$
2. C2 : $\Diamond(\exists i :: L_i)$
3. S1 : $(\forall i, j, k : i \neq j : SEND_i{}^k(j, OK) \Rightarrow (\Box \neg L_i))$
4. C3 : For SEFA, $(\forall i :: L_i \Rightarrow (\forall j : i \neq j : u_i > u_j))$
5. S2 : For SEFA, $(\forall i, j, k : i \neq j : COMPETE(i, j, k) \wedge (u_j > u_i) \wedge \neg SEND_i(OK) \Rightarrow \Box \neg L_i)$

Lemma 5.1

$(\forall i, j, k : (i \neq j) \wedge i, j \in S_{k+1} : d(i, j) > k \vee d(j, i) > k)$

Stated in words, this lemma means that for any pair (*i*,*j*) of nodes in S_k , either *d*(*i*,*j*) $>$ *k* or *d*(*j*,*i*) $>$ *k*. This result will be used in proving property C1.

Proof : The proof is by induction on the number of rounds *k*.

Let $Q_k : (\forall i, j, k : (i \neq j) \wedge i, j \in S_{k+1} : d(i, j) > k \vee d(j, i) > k)$

1. **Base Case :** *k* = 1.

   $S_1 = \{i | C_{i,1}\}$

   Since $(\forall i, j : i \neq j : d(i, j) > 0)$, Q_k holds for *k* = 1.

   Assume Q_k holds for *k* = *m*, i.e., $(\forall i, j, m : (i \neq j) \wedge i, j \in S_m : d(i, j) > (m - 1) \vee d(j, i) > (m - 1))$

2. **Inductive step :** We shall now prove the following :

   $Q_{m+1} : (\forall i, j, m : (i \neq j) \wedge i, j \in S_{m+1} : d(i, j) > m \vee d(j, i) > m)$

   From the protocol description of SPLEA we have the following properties :

$$(\forall i, j, k : i \neq j \wedge i, j \in S_k : RCV_i{}^k(j, ELECTION) \iff d(j, i) \leq k) \qquad (9)$$

   A candidate *i* in round *k* receives an *ELECTION* message from another candidate *j* if and only if $d(j, i) \leq k$.

$$(\forall i, j, k, k' : (i \neq j) \wedge (i, j \in S_k) \wedge (k' > k) : COMPETE(i, j, k) \Rightarrow i \notin S_{k'} \vee j \notin S_{k'}) \qquad (10)$$

   The statement (10) indicates that for any pair of nodes (*i*,*j*) for which $COMPETE(i, j, k)$ holds true, at the end of round *k* at least one of the 2 nodes must go to the failed state i.e. at least one of the two nodes can no longer be a candidate at the end of the round.

Restating (8),

$$(\forall i,j,k : i \neq j \wedge i,j \in S_k : COMPETE(i,j,k) \iff (RCV_i{}^k(j, ELECTION) \wedge RCV_j{}^k(i, ELECTION))) \tag{11}$$

A candidate *i* competes with another candidate *j* in round *k* if and only if each receives the other's *ELECTION* message in that round.

The contrapositive of statement (10) can be written as :

$$(\forall i,j,k,k' : (i \neq j) \wedge (k' > k) \wedge (i,j \in S_{k'}) : \neg COMPETE(i,j,k)) \tag{12}$$

Substituting (9) and (11) into (12) we get,

$$(\forall i,j,k,k' : (i \neq j) \wedge (k' < k) \wedge (i,j \in S_k) : \neg(d(j,i) \leq k')) \vee \neg(d(i,j) \leq k')) \tag{13}$$

$$(\forall i,j,k,k' : (i \neq j) \wedge (k' < k) \wedge (i,j) \in S_k : (d(j,i) > k') \vee (d(i,j) > k')) \tag{14}$$

Substituting *k* = *m* + 1 in (14) we get :

$$(\forall i,j,m,k' : (i \neq j) \wedge (k' < m+1) \wedge (i,j \in S_{m+1}) : (d(j,i) > k') \vee (d(i,j) > k')) \tag{15}$$

This proves Lemma 5.1.

We shall now prove the following lemma :

Lemma 5.2

$(\forall i,j : (i \neq j) \wedge (i,j \in S_L) : COMPETE(i,j,L))$

Stated in words, every node *i* that is a candidate at the beginning of round *L* will compete with every other candidate *j* at the beginning of round *L*.

Proof: Lemma 5.1 states that :

$$(\forall i,j,k : (i \neq j) \wedge (i,j \in S_{k+1}) : d(i,j) > k \vee d(j,i) > k) \tag{16}$$

Substituting *k* = *L* - 1 in (16) gives :

$$(\forall i,j : (i \neq j) \wedge (i,j \in S_L) : d(i,j) > L-1 \vee d(j,i) > L-1) \tag{17}$$

Substituting Assumption 5.3 in (17) we get:

$$(\forall i,j : (i \neq j) \wedge (i,j \in S_L) : (d(i,j) = L \wedge d(j,i) \leq L) \vee (d(j,i) = L \wedge d(i,j) \leq L)) \tag{18}$$

Substituting (9) in (18) we get,

$$(\forall i, j : (i \neq j) \wedge (i, j \in S_L) : {RCV_j}^L(i, ELECTION)) \tag{19}$$

Substituting (11) into (19) we get,

$$(\forall i, j : (i \neq j) \wedge (i, j \in S_L) : COMPETE(i, j, L)) \tag{20}$$

This completes the proof of Lemma 5.2 .

Since a node can either be in the candidate state or in the failed state, we have :

$$(\forall j : j \notin S_L : \neg C_{j,L+1}) \tag{21}$$

We can now complete the proof of property C1.
From (10) we get,

$$(\forall i, k :: C_{i,k+1} \Rightarrow (\forall j : (i \neq j) \wedge (COMPETE(i, j, k)) : j \notin S_{k+1})) \tag{22}$$

Substituting (20) in (22) we get,

$$(\exists i :: C_{i,L+1} \Rightarrow (\forall j : (i \neq j) \wedge (j \in S_L) : \neg C_{j,L+1})) \tag{23}$$

This predicate states that if there is a node *i* that is a candidate at the beginning of round *L* + 1 then every other node *j* that is in S_L goes to the failed state.

Predicate (21) states that every node *j* that is not in S_L goes to the failed state. Substituting (21) in (23) we get,

$$(\exists i :: L_i \Rightarrow (\forall j : i \neq j : \neg L_j)) \tag{24}$$

This states that every node except the leader is in the failed state and completes the proof of property C1.

We now prove that our election algorithm SPLEA satisfies the liveness property. More formally :

C2 : $\diamond(\exists i :: L_i)$

Proof : Restating the definition of S_k,

$$S_k = \{i | C_{i,k}\} \tag{25}$$

Stated in words, S_k is the set of surviving candidates at the beginning of round *k*.

Now S_k is a finite set and must contain at least a candidate *j* which received maximum number of votes of all candidates in round *k* - 1.

S_k will contain at least the node *j* such that $(\forall i : (i \neq j) \wedge (i, j \in S_{k-1}) : N^{k-1}(j) > N^{k-1}(i))$, since node *j* could not have sent an *OK* message to any other node. Hence,

$$(\forall k :: |S_k| \geq 1) \tag{26}$$

Substituting *k* = *L* + 1 in (26), we get

$$|S_{L+1}| \geq 1 \tag{27}$$

Statement (27) indicates that there is at least one surviving candidate at the end of round *L*. We have already proved (24) which states that if there exists a surviving candidate at the end of round *L*, then every other node must be in the failed state.

From (27) and (24), we can infer that, $|S_{L+1}| = 1$ and contains the leader. Every round lasts a finite amount of time and *L*th round results in a leader being elected. This proves the liveness property C2.

We shall now proceed to prove the security properties of SPLEA.

S1 : $(\forall i, j, k, l : i \neq l : SEND_i{}^k(j, OK(i, l)) \Rightarrow (\Box \neg L_i))$

Proof : ¿From the protocol description, we have

$$(\forall i, j, k, l : (i \neq j) \wedge (i \neq l) : SEND_i{}^k(j, OK(i, l)) \Rightarrow p_k(i) = l) \tag{28}$$

We shall prove this result by assuming that there is a malicious node *m*, that continues to participate in the election process even though it has sent an *OK* message in an earlier round. Moreover, we also assume that *m* causes a loop in ancestor/descendant relationship. More formally :

$$(\exists m, i, j, k, k', k'' : (i \neq j \neq m) \wedge (k \leq k' < k'') : (p_k(m) = i) \wedge (i \in D_{j,k'}) \wedge (p_{k''}(j) = m)) \tag{29}$$

If *m* happens to survive round *L*, then according to the protocol, before getting elected as the leader it will have to send a *LEADER* message to all its immediate children in I_m.

$$(\exists m, i, j, k, k', k'' : (i \neq j \neq m) \wedge (k \leq k' < k'') : p_k(m) = i \wedge i \in D_{j,k'} \wedge p_{k''}(j) = m \wedge C_{m,L+1} \Rightarrow$$
$$(\forall l : (l \neq m) \wedge (l \in I_m) : SEND_m(l, LEADER(m))) \tag{30}$$

The protocol description also states that every node upon receiving a *LEADER* message checks to see if the originator of *LEADER* message is not already its immediate child and forwards the *LEADER* message to its immediate children only if it is not the case.

$$(\forall i, j, k, m : i \neq j \neq m : (p_k(i) = j) \wedge RCV_i(j, LEADER(m)) \wedge \neg(p(m) = i) \Rightarrow$$
$$(\forall l : (l \neq i) \wedge (l \in I_i) : SEND_i(l, LEADER(m))) \tag{31}$$

We have assumed in our network model in Section 3.1 that packets are delivered reliably to their destination.

$$(\forall i, j, msg : i \neq j : SEND_i(j, msg) \iff RCV_j(i, msg)) \tag{32}$$

Substituting (32) in (31) we get,

$$(\forall i, j, k, m : i \neq j \neq m : p_k(i) = j \wedge RCV_i(j, LEADER(m)) \wedge (\neg(p(m) = i)) \Rightarrow$$
$$(\forall l : (l \neq i) \wedge (l \in I_i) : RCV_l(i, LEADER(m)))) \tag{33}$$

Recursive application of (33) yields,

$$(\forall i, j, k, m : i \neq j \neq m : p_k(i) = j \wedge RCV_i(j, LEADER(m)) \wedge (\neg(p(m) = i)) \Rightarrow$$
$$(\forall l, k' : (l \neq i) \wedge (l \in D_{i,k'}) \wedge (k' \leq k) : RCV_l(p(l), LEADER(m)))) \tag{34}$$

Substituting (34) in (30)

$$(\exists m, i, j, k, k' : (i \neq j \neq m) \wedge (k \leq k' < k'') : p_k(m) = i \wedge i \in D_{j,k'} \wedge p_{k''}(j) = m \wedge$$
$$C_{m,L+1} \Rightarrow RCV_i(p(i), LEADER(m))) \tag{35}$$

$$(\exists m, i, j, k, k' : (i \neq j \neq m) \wedge (k \leq k' < k'') : p_k(m) = i \wedge i \in D_{j,k'} \wedge p_{k''}(j) = m \wedge$$
$$C_{m,L+1} \Rightarrow EXCEPTION(i))) \tag{36}$$

This statement indicates that a node *i* upon seeing a *LEADER* message from one of its immediate children will immediately broadcast an exception and will thus prevent that node from taking over as the leader. This shows that a node that has already sent an *OK* message will be unsuccessful in becoming the leader which proves Property S1.

$$(\forall i, k \exists j : (i \neq j) \wedge (i, j \in S_{k-1}) : TIMEOUT_k(T_v) \wedge COMPETE(i, j, k) \wedge \neg RCV_j(i, VOTE)) \Rightarrow$$
$$EXCEPTION(j, k)) \tag{37}$$

This property means that every node that competes with at least one other node in a round will be forced to send the *VOTE* message in that round.

$$(\forall i, k \exists j : (i \neq j) \wedge (i, j \in S_{k-1}) : TIMEOUT_k(T_o) \wedge COMPETE(i, j, k) \wedge N^k(j) > N^k(i)) \Rightarrow$$
$$EXCEPTION(j, k)) \tag{38}$$

This property ensures that for any pair (*i,j*) of nodes that compete with each other in a round, if one of the nodes finds that it has more votes than the other, then it will make sure that the node with fewer votes sends an *OK* message.

Properties S1, (37) and (38) are the core security properties of SPLEA.

We now prove property C3 associated with SEFA. The proof of property C3 is fairly simple.

$$S_k = \{i | C_{i,k}\} \tag{39}$$

S_k represents the set of surviving nodes at the beginning of round *k* i.e. nodes which have not sent an *OK* message to any other node until the end of round *k* - 1. This set is finite and therefore must contain the node, say *w*, with maximum leadership-value at the end of every round. This is because *w* could not have sent an *OK* message to a node with a lower leadership-value. Thus $w \in S_{L+1}$ must hold true.

The above observations can be stated more formally as :

$$(\exists w : w \in S_{L+1} : (\forall i : (i \neq w) : u_w > u_i)) \tag{40}$$

Hence,

$$|S_{L+1}| \geq 1 \tag{41}$$

Statement (41) tells that S_{L+1} contains at least the node with maximum leadership-value. We have already proved (24) which states that if there exists a surviving candidate at the end of round *L*, then every other node must be in the failed state.

From (41) and (24), we can infer that, $|S_{L+1}| = 1$ and contains the node *w* with maximum leadership-value.

Thus SEFA elects a node with maximum leadership-value as the leader. This proves property C3.

We turn our attention to proving an important security property offered by SEFA, property S2. This property can be proved using previously proved results.

(40) states that

$$(\exists w : w \in S_L : (\forall i : (i \neq w) : u_w > u_i)) \tag{42}$$

Since *w* is the maximum utility node, *w* could not have sent an *OK* message to any other node. Lemma 5.2 states that all candidates in round *L* compete with each other.

Substituting Lemma 5.2 in (43), we get

$$(\exists w : w \in S_L : (\forall i : (i \in S_L) \wedge (i \neq w) : (u_w > u_i) \wedge COMPETE(i, w, L))) \tag{43}$$

From the protocol description, all candidates in round *L*, which follow the protocol, will elect *w* as their parent and will send an *OK* message to *w*. This is because every node will hear *w*'s *ELECTION* message and *w* is the maximum CEA value node.

$$(\exists w : w \in S_L : (\forall i : (i \in S_L) \wedge (i \neq w) : (u_w > u_i) \wedge p_L(i) = w)) \tag{44}$$

But if a malicious node *m*, participates in round *L* and does not send an *OK* message to *w*, then *w* will report an exception.

$$(\exists m : m \neq w : COMPETE(m, w, L) \wedge \neg SEND_m{}^L(w, OK(w, L)) \Rightarrow EXCEPTION(w, L)) \tag{45}$$

This shows that only the maximum CEA value node can become the leader and no one else. Thus a malicious node could have gotten away by not sending an *OK* message until round *L*, but it will be forced to do so in the final round. This proves property S2.