A Secure Hierarchical Model for Sensor Network

Malik Tubaishat, Jian Yin, Biswajit Panja, Sanjay Madria
Department of Computer Science, University of Missouri-Rolla, MO 65401, USA
{mma882, jian, bptfc, madrias@umr.edu}

Abstract

In a distributed sensor network, large number of sensors deployed which communicate among themselves to self-organize a wireless ad hoc network. We propose an energy-efficient level-based hierarchical system. We compromise between the energy consumption and shortest path route by utilizing number of neighbors (NBR) of a sensor and its level in the hierarchical clustering. In addition, we design a **S**ecure **R**outing **P**rotocol for **S**ensor **N**etworks (*SRPSN*) to safeguard the data packet passing on the sensor networks under different types of attacks. We build the secure route from the source node to sink node. The sink node is guaranteed to receive correct information using our *SRPSN*. We also propose a group key management scheme, which contains group communication policies, group membership requirements and an algorithm for generating a distributed group key for secure communication.

1. Introduction

Sensor networks are the new paradigm for the future communication. Sensor networks consist of tiny sensor nodes [1, 2, 3, 4] that collaborate among themselves to establish a sensing network and provide access to information anytime, anywhere by collecting, processing, analyzing, and disseminating data. In sensor networks, researchers are focused on improving three main aspects: energy-efficiency, fault-tolerance and secure routing. This concern is due to the fact that sensor nodes are vulnerable to energy depletion [5, 6, 7], intrusions and attacks [5, 8] and node and link failures [1, 3].

Security is one of the most important aspects in ad-hoc sensor networks. Intrusion and attack have become common threats to distributed sensor networks. A wireless sensor network uses radio frequency (RF) channel [5], which is not a secure channel. Attack can be active or passive. Active attacks involve some modification of the data stream or the creation of a false stream. Passive attacks are in the nature of eavesdropping on transmissions.

In this paper, we design an overview of a secure hierarchical model for sensor networks. In the self-organization process [1, 3], we divide the sensor nodes into different levels. The lower-level sensor nodes only sense and disseminate data, whereas the higher-level sensors find the shortest path to the sink node and aggregate data in addition to forwarding it. Since communication over radio is the most energy-consuming function performed by sensor devices, we need to minimize communications overhead [6]. By using hierarchical architecture we can decrease the number of messages, which lowers the communication overhead.

We propose an energy-efficient level-based hierarchical routing protocol by enabling a sensor node to choose its next neighbor hop according to its children's level and number of neighbors it has. Hence, only this chosen neighbor broadcasts the message. At the end, the shortest reliable energy-efficient route is chosen.

We next propose an extended hierarchical routing protocol to make it secure. We do not use source routing since it contains large size routing packets. In source routing, the identities of the traversed intermediate nodes are accumulated in the route request packet [9]. This is not feasible in sensor networks since usually there are thousands of sensors in the networks [6]. Our routing packet size is very small and we do not aggregate the intermediate nodes' data. We only use the routing table in the intermediate nodes to create the route. The cluster head aggregates the data, then sends the data to sink node along the route. Thus, it can save the energy involved in communication with large messages. Our secure routing protocol guarantees that the secure route reaches the sink node even if malicious nodes exist in the path. In addition, it also guarantees that the data is originated from authenticated source node and is not tampered.

In addition, we propose a group key management scheme for a hierarchical sensor network. In this scheme, every sensor node in a group contributes its partial key for computing the group key. The partial key can be a unique random number. Two entities are important; they are initiator and the leader. The key computation starts by the contribution of partial key from the initiator node. Every node contributes its partial key. One of the sensor nodes will act as a leader. It will compute the final group key using all the partial keys. There is no fixed rule for selecting the group initiator or leader. We have considered two important concepts for group key management; they are 1) Key trees 2) Diffie-Hellman [10] key exchange protocol. Key tree is used for updating group keys, and Diffie-Hellman protocol is used for computing group key.

The rest of the paper is organized as follows. A hierarchical architecture is given in Section 2. In Section 3, we present a secure routing protocol. In Section 4, we develop a key management protocol. We conclude the paper in Section 5.

2. Hierarchical Architecture

We build an energy-efficient hierarchical routing protocol, where sensors have different levels of responsibilities. At higher-levels, sensors take decisions supported by lower-level sensors in the hierarchy. We will discuss the level approach in the next section.

2.1 Characteristics

We consider an environment where thousands of sensor nodes are embedded in the targeted area. These sensors communicate and collaborate among themselves in an ad hoc manner to build a self-organizing sensor network. These sensors can provide access to information anytime, anywhere by collecting, processing, analyzing and disseminating data [1, 3]. We assume that sensors are not mobile; every broadcast is heard by neighbors within acceptable radius, and sensors are embedded with timer and GPS.

The sensor network in Figure 1 shows a one-to-many relationship between the sink node (a powerful node that creates and sends queries, and gathers all the information of the system) and the sources (objects to be sensed). The sensors act as an interface between the sink and the sources. These sensors collaborate among themselves to either disseminate data from the sink to targeted location, where sources are expected to be, or retrieve information from sensors near the source to send to the sink node.

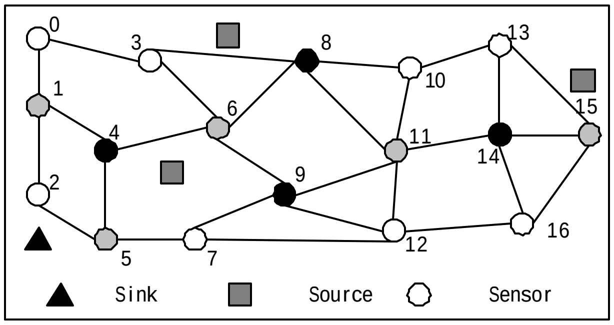

Figure 1 Level-Based Hierarchical Routing Protocol

2.2 Self-Organizing Sensors

Here, we discuss how sensors communicate among themselves to create hierarchical levels of communication. Sensors in the field form clusters to reduce the consumption of the energy in the network [7, 11]. Later these clusters merge to form hierarchical clusters [2].

Neighborhood Formation

First task in building a hierarchical routing is broadcasting sensors' IDs on a specific day and time set before embedding themselves into the targeted area. After broadcasting its ID, a sensor listens to its neighbors, adds their IDs in its routing table, and calculates the number of messages it receives to find the number of neighbors (NBR) it can reach. Later these connected neighbors build their own group or cluster. To determine the cluster head, sensors broadcast their IDs and NBRs. Every sensor keeps a list of all its neighbors' NBRs. A sensor becomes a cluster head if it has the highest NBR. That is, the node that is connected to the highest number of nodes becomes the cluster head. We choose this approach because cluster heads receive more messages than other nodes. In general, cluster heads aggregate, filter, and disseminate data. The cluster head with its connected neighbors form the cluster or the group. We call the cluster head's neighbors its children. Hence, the cluster head and its children have parent-child relationship in the tree-based network.

For a node to join a neighborhood it needs two broadcasts. Energy is consumed when broadcasting the sensor's ID and then when broadcasting again its ID and its NBR. The energy consumed for sensor *i* is:

$$Eng(s_i) = Eng\ [Brdcst(id)] + Eng\ [Brdcst\ (id, NBR)]$$

For a node to become a cluster head at a higher-level it needs to consume more energy by broadcasting its level.

$$Eng\ (s_i) = Eng\ [Brdcst\ (id)] + Eng\ [Brdcst\ (id, NBR)] + Eng\ [Brdcst\ (id, level)]$$

Level-Based Routing Protocol

In our hierarchical routing algorithm, the lower-level sensors only disseminate data, whereas higher-level sensors aggregate and forward the data. In addition, cluster heads determine the appropriate child (i.e., next hop) to forward the packet. We will explain this later. Sensors are initiated at level 0 when embedded in the network. The incremental level depends on a sensor's reliability and its energy consumption. Although, sensors connected to more neighbors tend to deplete energy faster than a sensor with fewer neighbors, these sensors are more fault-tolerant because they have more connected links. On the other hand, the higher-level nodes can degrade their level and pass the responsibility to one of its neighbors when its energy reaches a threshold. When a sensor finds its neighbors it upgrades itself to level 1 and then to level 2 if it becomes a cluster head. Our hierarchical clustering can be extended to N levels. A sensor connected to two or more cluster heads upgrades itself to level 3 (we call this node the *root*). Level 3 node merges two or more groups together. The *root* receives only filtered and accurate data from level 2 nodes (i.e., the cluster heads). The existence of the hierarchical clustering not only robust the sensor networks, but also, it overcomes many problems that face the cluster heads. N level hierarchical clustering advantages over the 2-level (i.e., using cluster heads only) are as follows:

1) *Root* nodes receive information from a wider radius than the cluster heads. Hence, increasing the circle of knowledge.
2) Information maintained by *root* nodes is more accurate and meaningful than in the cluster heads. This is because this information is processed and filtered by the cluster heads before they forward it to their parents (i.e., *root* nodes).

3) *Root* nodes are less congested than the cluster heads because they receive less number of messages.
4) Power efficiency at the higher-level nodes exceeds that in the cluster heads because they receive fewer messages than the cluster heads.

As we mentioned earlier, sensors at higher-levels have different degrees of responsibilities. Hence, when a sensor becomes a cluster head it activates its GPS to locate its position. Cluster head then broadcasts its *id*, *level*, and *pos*. This step allows its children (i.e., neighbors) to know their locations. Sensors in one group are physically close to each other. Hence, the locations of sensors in one group are not noticeably diverse. For the aforementioned reason and to save sensor's power not all sensors activate their GPS. Only sensors at higher-levels activate their GPS. In general, only one sensor activates its GPS in a group.

When receiving a data packet, a sensor decides if it needs to forward, aggregate, or drop the packet depending on number of criteria. The sensor drops the data packet if it has the same data or query in its cache, or if the position of the node getting farther from the destination. If the received data is new then it caches the data and broadcasts it to its neighbors. Because higher-level sensors have usually more neighbors, they receive more messages from their neighbors. Hence, these sensors need to aggregate and filter the data they receive from different neighbors. Aggregating data helps in reducing the number of messages transferred.

Routing Path Behavior

Our hierarchical routing is *level-oriented*. Every node stores its neighbors' *id*, *NBR*, and *level*. Hence, when a sensor receives a packet it checks the sender's level. If it has the same level as itself it drops the packet, otherwise it forwards it. Because no two low-level routing is allowed, this strategy ensures a shortest path. As cluster heads activate their GPS, they can decide the appropriate child to forward the packet depending on the destination of the sink or targeted source. Figure 1 depicts a level-based hierarchical routing sensor network. The numbers shown beside the sensors are their unique ids. Darker the sensor is the higher is its level and its importance. Sensors start with level 0 when embedded into the targeted location and start to increase their levels according to their importance among neighbors. Next, we discuss the routing path of the data from the sink to the sensors and vice versa.

1) Flooding Queries from Sink to Sensors

For simplicity of the explanation, we consider one sink node in our network architecture. The sink node creates the queries and broadcasts them to the sensor nodes. In case the application is designed to detect mobile sources, the queries should be flooded into all the sensor nodes in the network. This is because the sink does not know the location of the mobile source. On the other hand, if the object to be sensed is fixed and the sink knows the source's location, no need to flood the network with the query. In the later case, the query is broadcasted to some of the sensors.

2) Disseminating Data from Sensors to Sink

To explain how the data is disseminated from the sensors to the sink, we give the following example. The sink node broadcasts a query to detect any moving object within the network field. After receiving the query all the sensors become ready to detect a moving object within their radiuses. In Figure 1, sensor 15 detects a moving object within its radius, so it broadcasts what it had seen to its neighbors.

In our routing protocol, we choose to broadcast to the node with highest NBR in a group. This is because nodes with higher NBRs are connected to more nodes and hence, the possibility that they can reach their destination much faster.

Table 1 shows the routing table of sensor 15. Node 15 broadcasts its message to its neighbors with an emphasis that only one of its neighbors rebroadcast the message. In this case, it is node 14 as it has the highest NBR and a different level than the sender (i.e., sensor 15). Node 14 then broadcasts to node 11, etc.

ID			**LEVEL**			**NBR**		
15			2			3		
13	14	16	1	3	1	3	4	3

Table 1 Routing Table for Sensor 15

In case a node has neighbors at same level and same NBR it chooses all of them. In Figure 1, node 11 chooses node 8 and 9 but not 14 because it is the sender.

Criteria for choosing the next hop

Assigning a level to each sensor makes it easier for the node to choose its next hop. A sensor selects its next broadcast neighbor destination by comparing the attributes of its neighbors from its routing table (see Table 1). Following are these attributes:

- *GPS* – location attribute has the highest priority. If a level 3 node has two cluster head children then it chooses one that is nearest to the destination.
- *Level* – the level difference helps the sensors to determine the next hop to broadcast the data packet.
- *NBR* – a sensor chooses a neighbor with the highest NBR.

The above three attributes limit the data dissemination routes to the minimum instead of flooding the network and causing data congestion and energy depletion.

3. Secure Routing Protocol

In the Secure Routing Protocol for Sensor Networks (*SRPSN*) Proposed here, every node has a unique *ID* (Identity). Source node is a normal sensor node with resource constraints. Sink node is a super node, which has more power and memory. Sink node stores a table containing (*ID*, *Key*) pairs for all sensor nodes. After self-organization, sink node knows the topology of the sensor network. It sends the encrypted cluster group key to sensor nodes using the shared key between the sensor node and sink node.

Using *SRPSN*, we guarantee that a packet reaches the sink node even if malicious nodes exist. It guarantees that the message is originated from the authenticated source node and is not tampered on the route. Our *SRPSN* has the following three features:

1) We use the cache to store the routing table. We do not use source routing containing large size routing packets [9]. It is not feasible in sensor networks since usually there are thousands of sensors in sensor networks [6].

2) Our protocol uses the hierarchical architecture. The cluster head aggregates the data, then sends the data to sink node along the route. In this way, we can greatly decrease the number of messages to communicate, therefore can lower the communication overhead.

3) We only use high efficient symmetric cryptographic operations to secure messages. We do not use asymmetric operations (such as digital signatures) for cryptography and authentication since it needs large computation and communication overhead [6, 12].

3.1. Secure Route Discovery

In our route discovery, source node initiates the route discovery and sends route request (*RREQ*) to sink node. When sink node receives *RREQ*, it creates route reply (*RREP*) to source node. After route discovery, every node on the route has created the routing table, which stores only previous and next hops on the route.

Secure Route Request (RREQ)

Source node initiates the route discovery by broadcasting *RREQ* to its neighbors. *RREQ* includes the *IDs* of source node and sink node, ID_{RREQ} (a random number), encrypted nonce (the nonce is a random number), and *MAC* (Message Authentication Code) [13]. The *MAC* is generated by a keyed hash algorithm [13]. The inputs are the *IDs* of the source node and sink node, ID_{RREQ}, encrypted nonce and the key of source node. *RREQ* is constructed as follows:

$$\left\{\begin{matrix} ID_{source}, ID_{\sin k}, ID_{RREQ}, E_{key}(nonce), \\ MAC(ID_{source}, ID_{\sin k}, ID_{RREQ}, E_{key}(nonce), Key) \end{matrix}\right\}$$

When the intermediate node receives the *RREQ*, it creates the routing table with the *ID* of the previous two hops. If it receives *RREQ* directly from source node, it adds its *ID* on *RREQ*. If it receives the *RREQ* from other nodes, it replaces ID_{this}, ID_{pre} embedded in *RREQ* with the *IDs* of its previous node and its own *ID*. Finally, it broadcasts the updated *RREQ*. The updated *RREQ* is constructed as follows:

$$\left\{\begin{matrix} ID_{this}, ID_{pre}, ID_{source}, ID_{\sin k}, ID_{RREQ}, E_{key}(nonce), \\ MAC(ID_{source}, ID_{\sin k}, ID_{RREQ}, E_{key}(nonce), Key) \end{matrix}\right\}$$

When sink node receives the *RREQ*, it checks the (*ID*, *Key*) pair table to get the key of source node. Then, it uses the key to calculate *MAC* and verify *MAC*. Sink node only accepts *RREQ*, which is the first to reach sink node with valid *MAC* for the same source node and same ID_{RREQ}. Sink node drops other *RREQs* with same ID_{RREQ}.

Secure Route Reply (RREP)

When sink node accepts *RREQ*, it constructs *RREP*. The *RREP* is composed of *IDs* of source node, sink node, current node, and predecessor node, ID_{RREQ}, and *MAC*. The *MAC* is calculated using *IDs* of source node and sink node, ID_{RREQ}, nonce, and the key of source node. Then sink node broadcasts *RREP*. *RREP* is constructed as follows:

$$\left\{\begin{matrix} ID_{this}, ID_{pre}, ID_{source}, ID_{\sin k}, ID_{RREQ}, \\ MAC(ID_{source}, ID_{\sin k}, ID_{RREQ}, nonce, Key) \end{matrix}\right\}$$

The intermediate node, which receives the *RREP*, checks the *IDs* embedded in the *RREP*. If the *ID* of previous node embedded in the *RREP* is the *ID* of current node, it updates ID_{this}, ID_{pre} embedded in the *RREP* with its current *ID* and the *ID* of its previous node. Then, it broadcasts the updated *RREP*. Otherwise, it drops it. It also updates its routing table to add *ID* of next hop towards sink node.

Source node receives the *RREP* and verifies *MAC* to make sure that it's from sink node. If the *RREP* has not been tampered, source node inserts the *ID* of the next hop on the route to its routing table.

Secure Route maintenance

If one sensor node wants to send data to sink node and there is no route in its routing table to sink node, it initiates the route discovery to sink node. If source node gets the error message after it sends data or routing packet, it triggers the route discovery.

3.2. Secure Data Forwarding

In the hierarchical network architecture, the sensor node sends the data to the cluster head. The cluster head aggregates the data and sends the information to sink node. In the cluster communication, we secure the messages using the group key. Among the clusters' communication, we use the preloaded key to secure the information. We use these two mechanisms to send data to the sink node securely.

3.2.1 Secure Data Forwarding in the Cluster

If a sensor node sends the data to the cluster head, it constructs the data packet as follows:

$$\{[ID, E_{GK}(data)], MAC[ID, E_{GK}(data), GK]\}$$

Here, *ID* is the *ID* of the cluster head, *GK* is the group key of the cluster.

The sensor node broadcasts the data packet. Any node receives the packet, which checks the *ID* embedded. If the *ID* embedded in the packet matches the *ID* it holds, it verifies the authentication and integrity of the data packet through *MAC*. Otherwise, the packet is dropped by the node.

3.2.2 Secure Data Forwarding among the Clusters

The source cluster head checks its routing table. If there is a route to sink node, it constructs the following packet for data dissemination:

$$\begin{Bmatrix} ID_{this}, ID_{next}, [ID_{source}, Q_{ID}, E_{key}(data)], \\ MAC[ID_{source}, Q_{ID}, E_{key}(data), key] \end{Bmatrix}$$

Where ID_{this} is the *ID* of the current node that broadcasts the message, and ID_{next} is the *ID* of next hop in the current node's routing table, ID_{source} is the *ID* of source node, Q_{ID} is a random number, *key* is preloaded key of the source node, and *MAC* is generated by a keyed hash algorithm [13].

The intermediate node receives the packet, and checks the *ID* embedded. If the *ID* embedded in the packet matches the *ID* it holds, it updates the *ID* of the next hop embedded in the packet and broadcasts it. Otherwise, the packet is dropped by the node.

If the source node can not get the packet again that the next hop rebroadcasts, it triggers a new route discovery to the sink node. After the new route is created, the source node broadcasts the data. If the intermediate node can't get the packet broadcasted by the next hop within a certain time, it reports the error message to source node.

After sink node receives the packet, it checks the *ID* of source node and checks the {*ID, Key*) pair table to get the key of source node. Then, it verifies the authentication and integrity of the packet through *MAC*. If the authentication and integrity is guaranteed, sink node gets the correct query result from the source node.

3.3 Security Analysis

If the intermediate node is a malicious node, it can perform the following three actions: broadcast, drop or modify.

3.3.1 Intermediate Node Broadcasts Messages

In a route discovery, the malicious node may have two choices to attack the *RREQ* process. Case 1, it updates *RREQ* packet by inserting wrong *ID* of current node; Case 2, it creates the routing table with wrong information.

Case 1: The next hop records the wrong *ID* of the previous node in its routing table. When the *RREP* packet reaches this hop, *RREP* is updated using the tampered *ID* of the previous node. Other nodes can not accept this node since no *ID* matches the tampered *ID*. The next hop adds the malicious node into the malicious node table since it cannot get acknowledgement. In this case, source node cannot receive the *RREP* packet, and it triggers another route discovery.

Case 2: When *RREP* packet reaches the malicious node, the node broadcasts it with a wrong *ID* of previous node since it has incorrect information in the routing table. The next hop gets the *RREP* from the malicious node, and checks its own routing table. It can detect the tampered *RREP* broadcasted by the malicious node since its routing table stores two previous hops. The next hop blocks the malicious node. Since source node can not receive the *RREP* in this case, it triggers another route discovery.

In *RREP* process, a malicious node broadcasts a tampered *ID* of previous node or current node. This is the same as case 2 *RREQ* process. The next hop of the malicious node can detect it.

3.3.2 Intermediate Node Drops Messages

In a route discovery, if a malicious node drops the *RREQ* packets, it just blocks itself from routing. If a malicious node drops the *RREP* packet, the next hop can detect it since it cannot receive the packet again. Source node cannot get the *RREP* packet, and it triggers another route discovery.

In a data forwarding process, if a malicious node drops the data packet, the former node can detect it since it can not get acknowledgement from the next hop. The former node forwards the error message to source node.

3.3.3 Intermediate Node Modifies Messages

In a route discovery, if a malicious node modifies the *RREQ* core content, such as ID_{source}, ID_{sink}, ID_{RREQ}, or $E_{key}(nonce)$, sink node can detect it from verifying *MAC*. If a malicious node modifies the *RREP* core content, source node can detect it through *MAC*. In data forwarding, we have the same solution as modified *RREQ* core content.

4. Key Management

In this section, we design a key computation technique for hierarchical sensor networks. It is a two levels group key management scheme. One group key is computed for a group of sensors, and the other group key is for a group of cluster heads. Each group has a different group key for encryption and decryption of messages. The key is computed by considering one of the sensor nodes as the initiator and one as the leader. The initiator starts the computation of the group key by providing its partial key, and the leader computes the final group key by using all the partial keys. Every sensor node contributes its partial key for computation of the group key. Likewise, another group key is computed for cluster heads to secure communication among cluster heads.

4.1 Hierarchical Key Management Model

There are two types of sensor groups in our tree-structure hierarchical sensor network. One is a group of general sensor nodes led by a cluster head, while other is a group of cluster heads with one cluster head as head of that group. Two different types of nodes have been used here. These nodes are cluster heads, and general sensor nodes. Under each cluster head there is a group of general sensor nodes. The cluster heads are responsible for collecting data on behalf of a group and communicate with other cluster heads.

We are assuming that each group of sensor nodes has one cluster head and it processes more data than general sensor nodes. A cluster head can communicates with cluster heads of other groups. It is self-reconfigurable sensor network [2] and able to handle failure of nodes and can reconfigure the architecture according to the requirement for sensing coverage. It is a multi-hop network.

For example, consider a military sensor hierarchical architecture where there is a group of sensor nodes say A collecting data from geographical area say A. There is another group of sensor nodes say B in region B. Sensors of group B can get information from B, but not from A. The cluster head in-charge of group A and B will be able to collect information from both the groups, and broadcasts decision to both the regions.

4.2 Group Key Computation

We have modified the multi-party Diffie-Hellman protocol [10] to accommodate it in sensor hierarchical network architecture. To do so, one of the group members is chosen as initiator, and another member as leader. The leaf nodes work as initiator and cluster head as leader. Starting from the initiator sensor node, every sensor node contributes its partial key for computing the group key. Partial key can be a random number generated locally. The leader node accumulates all the partial keys for computation of the group key. This is a bottom up approach, as partial keys are accumulated fro m leaf nodes to the parent nodes.

4.2.1 Group Key Computation without Blind Factor

For group key computation without using blinding factor [5] we use the following approach. Since the leaf nodes work as initiator, they first broadcast their partial keys. The parent sensor nodes of the leaf nodes get the partial keys and add their contribution of partial keys. As it is a bottom up approach, the nodes above leaf nodes broadcast their partial keys along with the leaf levels' partial keys. The nodes above that level get the partial keys and add their partial keys. Finally, the cluster head will have all the partial keys, and it will calculate the group key using its partial key contribution. After that, the cluster head broadcasts the group key.

A symmetric key is used to encrypt and decrypt the partial keys. This is to make sure that unauthentic nodes cannot decrypt the partial keys. The identification of the nodes is used with the encrypted partial keys. The sensor nodes check the identification before decrypting the partial keys, as the parent nodes need the partial keys of their children and they do not need the partial keys of other sensor nodes. The other group members cannot compute the group key because they cannot get the partial key of the cluster head since the cluster head do not broadcast its partial key.

In Figure 2, the leaf nodes are M_1, M_2, …, M_9. M_1, the initiator sensor, computes the partial key g^{S1} and broadcasts it. The parent node M10 of M_1 gets the partial keys from its children. Here, g is a generator of the multiplicative group Z_P* (i.e. the set {1, 2… p-1}, p is the prime) [10] and S1 is a randomly chosen secret number for member M_1. Likewise, the member M_2 computes g^{S2} and broadcasts it, and the parent M_{10} gets the partial key. In this way, the member M_{10} receives g^{S1S2S3}, and raises power by S_{10} to get the intermediate key (IK). Here g^{S10} is the partial key contribution of M_{10}.

Next, we discuss two types of group keys; the intra-cluster and the inter-cluster. The intra-cluster group key is used by group of sensor nodes led by a cluster head, and inter-cluster group key is used by a group of cluster heads.

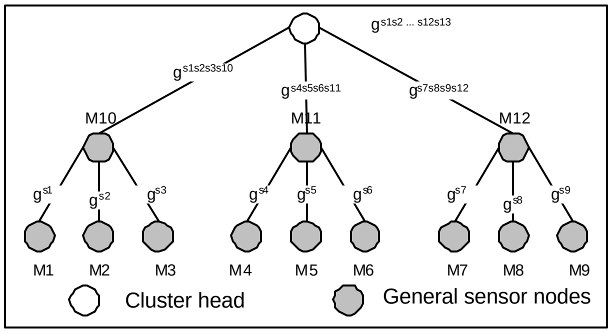

Figure 2 Intra-Cluster Key Computations

The intermediate keys in M_{10}, M_{11}, and M_{12} are $IK1 = g^{S1\ S2\ S3\ S10}$, $IK2 = g^{S4\ S5\ S6\ S11}$, *and* $IK3 = g^{S7\ S8\ S9\ S12}$ respectively. The intermediate keys are encrypted using a symmetric key. All the sensor nodes in a group have the symmetric key. The cluster head calculates the group key K, using IK1, IK2, and IK3 and its contribution g^{s13}.

$K = g^{S1\ S2\ S3\ S10\ S4\ S5\ S6\ S11\ S7\ S8\ S9\ S12s13}$ *[Intra-cluster group key]*

Then it encrypts the group key using the symmetric key. The authentic nodes, which have the symmetric key, can decrypt the group key. The cluster head broadcasts the group key to its group, so that every sensor node in that group gets the group key. This group key is called intra-cluster group key, and is used for encryption/decryption inside the group of sensor nodes.

For inter-cluster encryption/decryption, a different group key is computed. The inter-cluster group key is unknown to the general sensor nodes. Figure 3 shows the computation of inter-cluster group key. The intermediate key in C_7, C_8, C_9 are $IK1_{intra} = g^{C1C7}$, $IK2_{intra} = g^{C2C3C8}$, $IK3_{intra} = g^{C4C5C6C9}$

The head of the cluster heads (HCH) computes the inter-cluster group key C using intermediate keys $IK1_{inter}$, $IK2_{inter}$, $IK3_{inter}$, and its contribution g^{c10}

$C = g^{C1C7\ C2C3C8\ C4C5C6C9c10}$ *[Inter-cluster group key]*

The HCH broadcasts the intra-cluster group key to the cluster heads. The cluster heads use this group key for encryption/decryption of messages among the cluster heads.

4.2.2 Group Key Computation Using Blind Factor

We can compute the group key using blinding factor [8]. The advantage using blinding factor is that an attacker will not be able to get the group key when cluster head broadcasts the group key. In Figure 2, we show that intermediate keys are $IK1 = g^{S1\ S2\ S3\ S10}$, $IK2 = g^{S4\ S5\ S6\ S11}$ and $IK3 = g^{S7\ S8\ S9\ S12}$. After computation of IK1, IK2, IK3

the parent nodes M_{10}, M_{11}, M_{12} broadcast the intermediate keys. The children of M_{10}, M_{11}, M_{12} are interested in those keys, as they need to remove their contribution from the IK. Then they insert randomly chosen blinding factor B. The keys after inserting blinding factor are as follows.
$IKB1 = g^{\mathbf{B}1\ S2\ S3\ S10}$, $IKB2 = g^{S1\ B2\ S3\ S10}$, *and* $IKB9 = g^{S7\ S8\ B9\ S12}$. The cluster head gets the broadcasted keys IKB1,…, IKB9. The cluster head computes the group key K, using IKB1…IKB9 and its contribution g^{s13}.
$K = g^{B1\ S2\ S3\ S10\ S4\ S5\ S6\ S11\ S7\ S8\ S9\ S12 s13}$

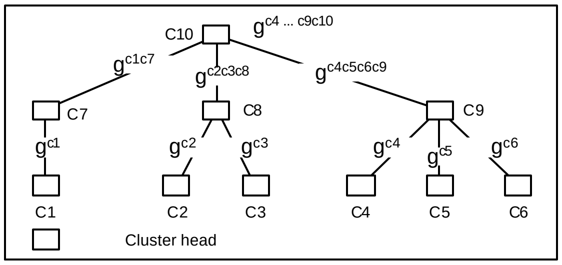

Figure 3 Inter-Cluster Key Computations

After the group key computation, the cluster head broadcasts the group key with a blind factor. Now the authentic sensor node can recognize its blind factor. Each member unblinds [5] its blinding factor that it receives from cluster head. It reinserts its original contribution S_i (i = 1...n) for getting the group key. Same method is used to compute the inter-cluster group key. A symmetric key is used for encryption and decryption of partial keys. Cluster head uses the same symmetric key for encryption of the group key.

5. Conclusion

In this paper, we discussed a secure hierarchical sensor network model. We propose a new routing protocol algorithm that depends on the number of neighbors and their levels to disseminate the queries and data. The level-based hierarchical routing protocol compromises between shortest path and energy consumption. We propose a secure routing protocol in sensor networks (*SRPSN*). *SRPSN* guarantees that the sink node gets the correct query results from the sensor network. In our secure routing protocol, we use high efficient symmetric key and use hierarchical architecture, which greatly lowers the computation and communication overhead. In addition, we propose a group key management scheme. In this scheme, every sensor node contributes its partial key for computing the group key.

References

[1] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, pages 56-67, Boston, MA, Aug. 2000. ACM Press.

[2] L. Subramanian and R. H. Katz, An Architecture for Building Self-Configurable Systems, *IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC'00)*, Boston, 2000.

[3] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks, *In Proceedings of the Fifth Annual International Conference on MobiCOM,* August 1999, Seattle, Washington.

[4] M. Tubaishat and S. Madria. Sensor Networks: An Overview, *IEEE Potentials,* Vol. 22, No. 2, April/May 2003.

[5] D. W. Carman, P. S. Kruus, and B. J. Matt. Constraints And Approaches For Distributed Sensor Network Security. *NAI Labs Technical Report #00-010*, September 2000.

[6] R. Szewczyk, V. Wen, D. Culler, and D. Tygar. SPINS: Security Protocols for Sensor Networks, *Wireless Networks Journal (WINE)*, September 2002.

[7] J. Kulik, W. R. Heinzelman, and H. Balakrishnan, Adaptive Protocols for Information Dissemination in Wireless Sensor Networks, *Proc. 5th ACM/IEEE MobiCom Conference (MobiCom '99)*, Seattle, WA, August, 1999.

[8] J. Kong, P. Zerfos, H Luo, S Lu, and L. Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks. *IEEE Ninth International Conference on Network Protocols (ICNP'01)*, 2001.

[9] P. Papadimitratos and Z. J. Haas. Secure Routing for Mobile Ad Hoc Networks. *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, San Antonio, TX, January 27-31, 2002.

[10] W. Diffie and M. E. Hellman. Privacy and Authentication: An Introduction to Cryptography. *Proceedings of the IEEE*, 67(3):397–427, March 1979.

[11] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, An Application-Specific Protocol Architecture for Wireless Microsensor Networks, *IEEE Transactions on Wireless Communications*, Vol. 1, No. 4, October 2002, pp. 660-670.

[12] Y. Hu, D. B. Johnson, and A. Perrig. SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks. *4th IEEE Workshop on Mobile Computing Systems and Applications*, WMCSA'02, New York, 2002.

[13] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication, Internet RFC 2104, February 1997.