IJCES International Journal of Computer Engineering Science , Volume1 Issue 3, December 2011
ISSN : 2250:3439
https://sites.google.com/site/ijcesjournal
http://www.ijces.com/

# Joint Design of Cluster-Based Hierarchical Networking Architecture and Key Management System for Heterogeneous Wireless Sensor Networks

McKenzie McNeal III, Wei Chen, Sachin Shetty

Tennessee State University, 3500 John A. Merritt Blvd., Nashville, Tennessee 37209
mmcneal01@mytsu.tnstate.edu, {wchen, sshetty}@tnstate.edu

Stanley Aungst

Pennsylvania State University, 1011 Information Science & Technology Building,
University Park, Pennsylvania 16802
sga103@psu.edu

***Abstract.*** *Current communication protocols used for Wireless Sensor Networks (WSNs) have been designed to be energy efficient, low redundancy in sensed data, and long network lifetime. One major issue that must be addressed is the security of data communication. Due to the limited capabilities of sensor nodes, designing security-based communication protocols present a difficult challenge. Since commonly used encryption schemes require high time complexity and large memory, achieving security for WSNs needs unique approaches. In this paper, we consider a heterogeneous wireless sensor network (HWSN), where while most nodes are resource limited a few nodes can have more energy, stronger processing capability and longer communication range and can be used to relax the resource bottleneck. We propose a joint design approach that can best use the benefit that a HWSN brings. We first design a reconfigurable hierarchical networking architecture, where the HWSN is divided by the high-end nodes into regions, the low-end nodes in each region are divided by clusters, and high-end nodes and cluster heads form a communication/relay backbone. We then design a key management system that uses both public and symmetric key cryptography above the hierarchical networking architecture which requires very small number of security keys. The evaluation and simulation results show that by using the proposed networking architecture and key management scheme only a small amount of keys needs to be preloaded before deployment and stored after key setup to achieve secured communication throughout the entire network.*

**Key words:** security, key management, sensor networks, heterogeneity

## 1 Introduction

A wireless sensor network (WSN) of low cost sensor nodes can be densely deployed and used for distributed data gathering, monitoring and surveillance in the applications of wildlife monitoring, military command, distributed robotics, industrial quality control, observation of critical infrastructures, smart buildings, intelligent communications, traffic monitoring, examining human heart rates, etc [1]. Some

research has used WSNs for detecting the release of a poisonous gas [2]. But the information gathered by these applications is not properly protected. Security presents a major challenge in WSN design due to the limited resources and constraints of the sensor nodes. Any attacks that occur on the network could drain the limited resources. A security approach requires a certain amount of resources for implementation, including memory for storing security keys and code space, time complexity for encryption, decryption, and transmission, and energy to power the sensor node. Sensor nodes do not have the capability to support traditional security methods. Some applications may require a WSN to operate unattended or within a hostile environment. This situation exposes the network to physical attacks if the environment is open to adversaries, bad weather, etc. WSNs will also need to be managed remotely, which makes it virtually impossible to detect physical tampering and physical maintenance issues.

In this paper, we consider a heterogeneous wireless sensor network (HWSN), where while most nodes are resource limited a few nodes can have more energy, stronger processing capability and longer communication range and can be used to relax the resource bottleneck experienced in homogeneous WSNs and increase the lifetime of the network by taking on greater responsibilities than a typical resource constrained sensor node [3-5]. Some security methods developed for HWSNs show that heterogeneity helps to provide leverage in security by using high-end sensor nodes (H-nodes) to take on more security responsibilities than low-end sensor nodes (L-nodes) [6-9]. In this paper, we introduce a robust network architecture coupled with a secure communication scheme to provide security for HWSNs. Hierarchical networking architecture is defined by two layers of regions and clusters. The H-nodes divide the deployment area into regions, where each L-node belongs to the region of the closest H-node. In each region, the L-nodes are clustered. Cluster heads and H-nodes form a backbone tree that can be used for data aggregation and relay. This architecture allows for self organization without localization information. A combination of both symmetric and asymmetric keys is used to secure node to node communication. For asymmetric key cryptography we use Elliptic curve cryptography (ECC) that is feasible for sensor nodes, providing a 160-bit key which is securely equivalent to the RSA 1024-bit and thereby provide public key cryptography for WSNs [11-12]. For symmetric key cryptography, L-nodes use preloaded keying materials and neighbor knowledge to dynamically generate a pair-wise key [13-14].

The rest of this paper is organized as follows. Section II discusses related works on security for HWSNs. Section III presents proposed robust network architecture, hierarchical networking architecture self formation/reconfigurations algorithms and discuss the roles of H-nodes and L-nodes in data relay. In section IV, we will discuss the key management scheme and secure routing. Section V shows the simulation results and comparative analysis for key storage. Finally, section VI concludes this paper.

## 2 Related Work

Some research has been done to design security methods for HWSNs. In [6], a hybrid key management scheme called LIGER was proposed. LIGER uses a

probabilistic unbalanced key distribution scheme where a relatively large number of keys are preloaded onto H-nodes than L-nodes. The scheme also has the ability to change from a standalone key-management system called LION to a key distribution center (KDC) based key management system called TIGER in case the sensor network is able to communicate with an existing backbone network.

Lu *et al.* [7] proposed two key distribution schemes, key-pool based and polynomial-pool based schemes, for HWSNs so that H-nodes and L-nodes can established at secure communication. In the key-pool based scheme, if two nodes share a key, they can establish secure communication. In the polynomial-pool based scheme, if two nodes exchange IDs, they can establish a secure link with a key only known by the two communicating nodes. The same polynomial generates a different key for different pair of nodes.

Du *et al.* [8-9] proposed a key distribution scheme that addresses the key exchange issue in homogeneous sensor networks by using both public-key and symmetric key cryptography to establish secure communication between H-nodes and L-nodes. In this scheme, they introduce the c-neighbor concept, where nodes only need to establish a key with communicating neighbors that are in route back to the sink. This helps to save resources of L-nodes so that they are not preloaded with an arbitrary number of keys before deployment. Their performance evaluation show a significant decrease in the amount of keys preloaded before deployment from that of Eschenauer and Gligor key management distribution scheme [15]. Furthermore, the Du-scheme shows better resilience against node compromise by assuming that H-nodes are tamper resistant and that any compromised L-node has minimal effect on the network because it only shares a key with communicating neighbors and not all of its surrounding neighbors.

All of the aforementioned key management system offer beneficial methods to establishing secure communication for HWSNs, but we argue that by establishing a robust hierarchical networking architecture that defines the different roles for H-nodes and L-nodes will offer a measure of security that helps to alleviate some of the challenges of key distribution and offer a foundation for secure communication. Those challenges include designing reliable and available network architecture, forming a high performance network infrastructure through self-organization, preloading and storing the least amount keys necessary to achieve secure communication between each node and leveraging security tasks to maximize network resources.

## 3 Robust Hierarchical Networking Architecture

In a HWSN, we use two types of nodes, H-node and L-node, where the H-node has greater capabilities and more resources than the L-node. The network consists of a large amount of L-nodes with a small amount of H-nodes. We define a cluster-based hierarchical networking architecture (CHNetArch) for the HWSN as follows:

**Definition 1 for HWSN:** A HWSN can be represented by an undirected graph $G = (V,E)$, where $V$ is a set of wireless nodes including $m$ H-nodes and $n$ L-nodes, and $E$ is a set of edges. Given any two nodes $u$ and $v$ in $V$, edge $e = (u,v) \in E$, if and only if $u$ and $v$ are in the communication range with each other (Figure 1). Without loss of generality, we assume that $G$ is a connected graph, $m << n$, and each L-node u is in

the communication range of at least one H-node *v*, i.e., *u* can receive the signal from *v*, though *v* may not be able to receive the signal from *u*, due to the difference of their capabilities.

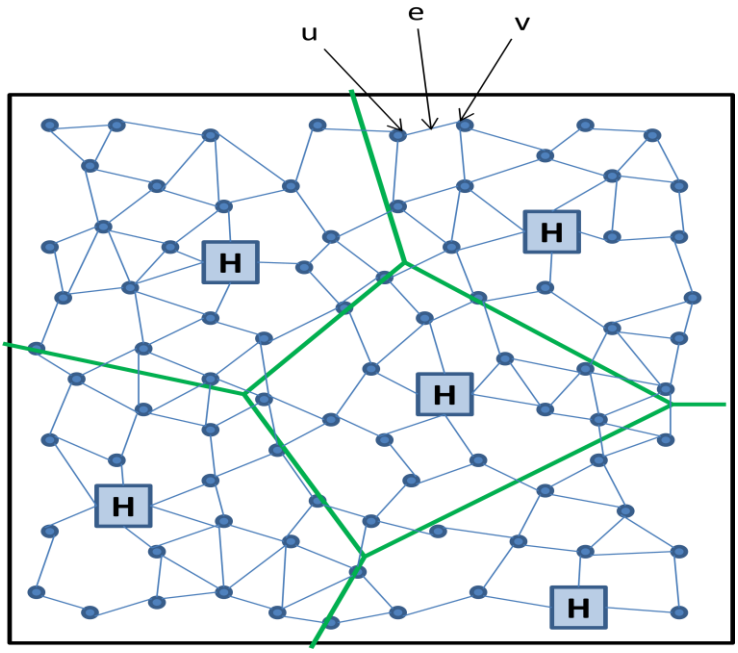


Figure 1 - Graph G with a set of wireless nodes, V, and set of edges, E.

**Definition 2 of *H(G)*:** Given *G*, we define a region division $H(G) = \{G_1, G_2, \ldots, G_m\}$, where *m* is the number of regions. $G_1, G_2, \ldots, G_m$ are sub-graphs of *G* divided by H-nodes $H_1, H_2, \ldots, H_m$. Let $G_i = (V_i, E_i)$. For each node $v \in V$: $v \in V_i$ if and only if *v* is closer to $H_i$ than any other H-node $H_j$ $(j \neq i)$. $G_i$ is the sub-graph induced from $V_i$.
**Definition 3 of *C(G)*: Given *G*,** $C(G) = \{C_1, C_2, \ldots, C_r\}$ is the set of clusters in *G*, where $C_i = (CV_i, CE_i), and \bigcup_{i=1}^{r} CV_i = V$. $C_i$ is the sub-graph of *G* induced from $CV_i$. $C_i$ is a complete graph and there is a node called as cluster head.
**Definition 4 of *BT(G)*:** Given *G*, the backbone tree of G is a spanning tree of the H-nodes and the cluster head nodes from *C(G)*. It is denoted as *BT(G)*.
The structure of *G*, *H(G)*, *C(G)*, and *BT(G)* are defined as follows:

- Data structure for *G*:
  - *u.type* – L-node or H-node
  - *u.id* – identification (ID) of node *u*.
  - *u.energy* – remaining energy of node *u*.
  - *u.nlist* – neighbor list of node *u* in *G*.
- Data structure for *H(G)*:
  - *u.regionhead* – if *u* is an L-node, it has the ID of *u's* regional head.
  - *u.regionlist* – if *u* is an H-node, it has a list of L-nodes in the region.
- Data structure for *C(G)*:
  - *u.status* – node *u* is a cluster member or cluster head.
  - *u.cmlist* – if *u* is a cluster head, it has a list of cluster members.
  - *u.clusterhead* – if *u* is a cluster member, it has the ID of *u's* cluster head.
- Data structure for *BT(G)*:
  - *u.parent* – the ID of the parent node for node *u*.
  - *u.childlist* – the list of child nodes in the BT(G) for node *u*.

There are three algorithms used for self-formation of *H(G)*, *C(G)*, and *BT(G)*; region formation, clustering, and backbone tree formation. There are also three algorithms for self-reconfiguration; node move-in, node move-out, and head rotation. The algorithms are written in rounds where during one round, a node can transmit, receive and process/compute once.

- **Region Formation –** H-nodes divide the sensor field into regions using Voronoi Diagram, where each L-node belongs to the region of the closest H-node. In the region formation algorithm, the L-node selects the H-node which has the strongest signal to be its region head.

  ***Algorithm 1 Regional formation of G***

  **Input:** $G = (V, E)$.

  **Output:** $H(G) = \{G_1, G_2, \ldots, G_m\}$

  Round 1:

  1. Each H-node *w* in *G* broadcasts a declaration message, (*w*, "region declare").
  2. When L-node *u* receives a message (*w*, "region declare"), *u* sets *w* to be *u.region*. If *u* receives more than one message, *u* chooses the sender which has strongest signal strength to be *u.region*.
- **Clustering –** L-nodes find neighbors by broadcasting there IDs and receiving and accepting IDs of neighboring L-nodes in the same region $G_i$. L-nodes are then grouped into completed graphs in each region using the clustering algorithm that assigns one cluster head (*ch*) to the cluster members (*cm*) in the same cluster. If an L-node *u* has chosen a *ch v*, but u is selected to be the *ch* of another node(s), *u* will send a message to remove itself from *v's* cluster and become the *ch* of a new cluster. To assure the head-rotation in the reconfiguration functions, the diameter of the cluster is *d/2*, where *d* is the maximum communication range of L-node *u*.

  ***Algorithm 2 Cluster Formation***

  **Input:** A region $G_i \in H(G)$

  **Output:** $C(G_i)$, a clustering of $G_i$

  ***Algorithm 2.1 – Neighbor Discovery at each node u***

  Round 1:

1. Node *u* broadcasts a message using range *d/4*, (*u*, "Hello neighbor."), and then receives messages from neighbors.
2. If node *u* receives the message, (*v*, "Hello neighbor."), add *v* to *u.nlist*.

***Algorithm 2.2 – Clustering of node u***

Round 1:

1. Node *u* checks *u.nlist* and find neighbor node *v* with smallest ID.
2. Node *u* transmits message to *v*, (*u*, *v*, "Cluster head request.").
3. Node *u* receives message from *v*, (*v*, *u*, "Cluster head request.").

Round 2:

1. If node *u* receives message from *v*, (*u*, *v*, "Cluster head request."), *u* adds *v* to *u.cmlist*.
2. Node *u* transmits acknowledgement message to cluster member *v*, (*v*, *u*, "Cluster head acknowledgement.").
3. Node *u* receives a message from node *v*, (*v*, *u*, "Cluster head acknowledgement.").

Round 3:

1. If cluster member *u* receives a message from node *v*, (*v*, *u*, "Cluster head acknowledgement."), *u* sets *v* as cluster head.
2. If *u* is a cluster member of *v*, and receives a message (*w*, *v*, "Cluster head request.") from neighboring node *w*, then *u* transmits to *v*, (*u*, *v*, "I am cluster head. Remove me as a cluster member.").
3. Node *u* receives messages, (*v*, *u*, "I am cluster head. Remove me as a cluster member.").

Round 4:

1. If cluster head *u* receives message (*v*, *u*, "I am cluster head. Remove me as a cluster member."), and *v* is a cluster member of *u*, *u* removes *v* from *u.cmlist*.
2. If node *u* does not find cluster head in *u.nlist*, then *u* becomes a cluster head.
3. If node *u* is cluster head and has a *u.nlist*, but no *u.cmlist*, then *u* must choose a neighbor node *v* with smallest ID to be its cluster head.

- **Backbone Tree Formation –** The Basestation and H-nodes form a spanning tree *BT(S)* that connects all $BT(G_i)$. *Ch*s and H-nodes form the communication backbone in which the information is sent in bi-direction between the base station and region heads (H-hodes), between the region head and cluster heads in each region, and between the cluster head and cluster members in its cluster through the entire network.

***Algorithm 3.1 – Spanning Tree BT(S) formation***

**Input:** Basestation and H-nodes
**Output:** *BT(S)*

Round 1:

1. Basestation, *B*, broadcasts a message (*B*, "Find children.").

Round 2:

1. If the basestation, *B*, receives message from *h*, (*h*, *B*, "I am a child."), *B* adds *h* to *B.childlist*.

2. *B* sends *h* a confirmation message, (*B*, *h*, "I am your parent"). (This message is only sent from *B* to *h* to ensure that H-nodes out bi-direction communication range do not add *B* as a parent).

Round 3: (Rounds 3 & 4 are repeated for all H-nodes until *BT(S)* is formed)

1. If *h* receives message from *B*, (*B*, *h*, "I am your parent"), then *h* sets *B* to be its parent, *h.parent = B*.
2. If *h* receives messages, (*v*, "Find children") and *h.parent = null*, then *h* chooses *v* with weakest signal within threshold *D* to be its parent, then sends reply.
3. *h* sends reply to *v*, (*h*, *v*, "I am a child").

Round 4:

1. If *h* receives messages, (*v*, *h*, "I am a child"), then *h* adds *v* to *h.childlist*.
2. If *h.parent* ≠ *null*, *h* broadcasts message, (*h*, "Find children"). (H-nodes do not broadcast to find children unless it has a parent.)

***Algorithm 3.2 – Region BT($G_i$) formation***

**Input:** A region $G_i \in H(G)$

**Output:** *BT($G_i$)*

Round 1:

1. H-node, *h*, at region $G_i$ broadcasts a message (*h*, "Find children.").

Round 2:

1. If the H-node, *h*, receives message from *u*, (*u*, *h*, "I am a child."), *h* adds *u* to *h.childlist*.
2. *h* sends *u* a confirmation message, (*h*, *u*, "I am your parent"). (This message is only sent from *h* to *u* to ensure that cluster heads out bi-direction communication range do not add *h* as a parent).

Round 3: (Rounds 3 & 4 are repeated for all cluster head *u* in region until *BT($G_i$)* is formed)

1. If *u* receives message from *h*, (*h*, *u*, "I am your parent"), then sets *h* to be its parent, *u.parent = h*.
2. If *u* receives messages, (*v*, "Find children") and *u.parent = null*, then *u* chooses *v* with weakest signal within threshold *d* to be its parent, then send reply.
3. *u* sends reply to *v*, (*u*, *v*, "I am a child").

Round 4:

1. If *u* receives messages, (*v*, *u*, "I am a child"), then *u* adds *v* to *u.childlist*.
2. If *u.parent* ≠ *null*, *u* broadcasts message, (*u*, "Find children"). (cluster heads do not broadcast to find children unless it has a parent.)

After the network formation is complete, H-nodes and L-nodes have specific roles in the cluster-based hierarchical networking architecture as shown in Figure 2. Those roles are defined as follows:

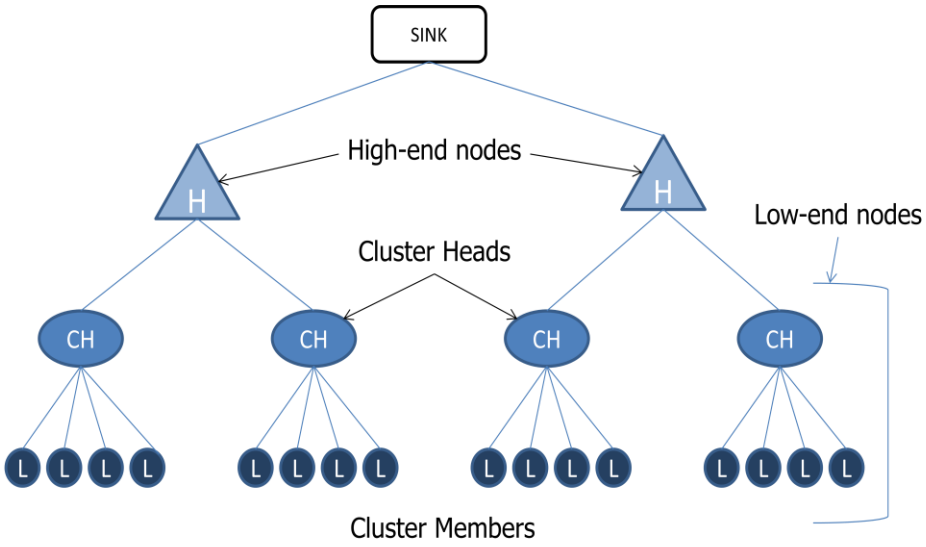


Figure 2 – Cluster-based hierarchical networking architecture (CHNetArch) for HWSNs.

- *Cluster members (cms)* communicate only and directly with their cluster head (*ch*).
- *Cluster heads* communicate with *cm*s in their cluster and with neighboring *ch*s on the backbone tree.
- Information on the backbone tree travels from child to parent (*ch* to *ch*) until it reaches the H-node of their region head.
- H-nodes send information to neighboring H-nodes in route to the sink/basestation.

We assume all collisions are avoided by using MAC protocol CSMA/CA. The network self-reconfiguration is supported by the following functions:

- Node Move-in – New L-node joins the network and becomes a *ch* or *cm* depending on its surrounding neighbors.

  ***Algorithm 4 – Node Move-In for node u in G***

  **Input:** New node *u*, *G*, *H(G)*, *C(G)*, and *BT(G)*

  **Output:** Update *G*, *H(G)*, *C(G)*, and *BT(G)*

  Round 1:

  1. New node *u* broadcast message at range *d/4*, (*u*, "Hello, I want to join").
  2. If cluster head *u* within range, *u* receives message, (*v*, "Hello, I want to join").

  Round 2:

1. If cluster head *u* receives (*v*, "Hello, I want to join"), *u* transmits reply (*u*, *v*, "Hello new cluster member.").
2. If new node *u* receives replies (*v*, *u*, "Hello new cluster member."), *u* chooses *v* with strongest signal and sets *v* to be cluster head.

Round 3:

1. If new node *u* receives reply (*v*, *u*, "Hello new cluster member."), *u* sends a confirmation message, (*u*, *v*, "Cluster member confirmed").
2. If cluster head *u* receives confirmation message, (*v*, *u*, "Cluster member confirmed"), *u* adds *v* to cluster member list, *u.cmlist*.
3. If new node *u* receives no reply, it designates itself as a cluster head, then broadcast a message at range *d*, (*u*, "Hello, I want to join."), to find parent node from backbone tree.
4. If cluster head *u* within range, *u* receives message, (*v*, "Hello, I want to join.")

Round 4:

1. If cluster head *u* receives message, (*v*, "Hello, I want to join."), *u* transmits message, (*u*, *v*, "Hello new child node").
2. If new node *u* receives messages, (*v*, *u*, "Hello new child node"), *u* chooses *v* with strongest signal and sets *v* as its parent, *u.parent* = *v*.

Round 5:

1. If new node *u* receives message, (*v*, *u*, "Hello new child node"), *u* transmits confirmation message, (*u*, *v*, "Parent confirmed").
2. If cluster head *u* receives message, (*v*, *u*, "Parent confirmed"), *u* adds *v* to child node list, *u.childlist*.

- Node Move-out – *ch* or *cm* leaves the network and connectivity is maintained.

***Algorithm 5 – Node Move-Out for u in G***

**Input:** Moving-out node *u*, *G*, *H(G)*, *C(G)*, and *BT(G)*
**Output:** Update *G*, *H(G)*, *C(G)*, and *BT(G)*

Case I

Round 1:

1. If node *u* is pure cluster member, *u* sends message to cluster head *v*, (*u*, *v*, "I'm leaving").
2. If cluster head *u* receives message, (*v*, *u*, "I'm leaving"), *u* removes node v from neighbor list, *u.nlist*, and cluster member list, *u.cmlist*.

Round 2:

1. If cluster head *u* receives message, (*v*, *u*, "I'm leaving"), *u* sends message to each cluster member *v* (including leaving node), (*u*, *v*, "Remove node *w* from neighbor list.").
2. If cluster member *u* receives message, (*v*, *u*, "Remove node *w* from neighbor list."), and *u* = *w*, then *u* leaves cluster; else *u* removes *w* from *u.nlist*.

Case II

Round 1 – 4:

1. If node *u* is cluster head, perform head rotation.

Round 5 – 6:

1. Follow algorithm for Case I.

- Head Rotation – New *ch* is selected when an existing *ch* is low on resources or even compromised.

***Algorithm 6 –Head Rotation for u in C(G)***

**Input:** Cluster head node *u*, *G*, *H(G)*, *C(G)*, and *BT(G)*
**Output:** Update *G*, *H(G)*, *C(G)*, and *BT(G)*

Round 1:

1. Cluster head *u* sends message to request energy remaining from each cluster member *v*, (*u*, *v*, "Energy request").
2. If cluster member *u* receives message from cluster head *v*, (*v*, *u*, "Energy request"), *u* checks remaining energy.

Round 2:

1. Cluster member *u* sends message with remaining energy to cluster head *v*, (*u*, *v*, "Energy remaining").
2. If cluster head *u* receives message from cluster member *v*, (*v*, *u*, "Energy remaining"), *u* chooses cluster member *v* with most remaining energy to be new cluster head.

Round 3:

1. Cluster head *u* sends message to each cluster member *v* that cluster member *w* is the new cluster head, (*u*, *v*, "*w* is the new cluster head").
2. If cluster member *u* receives message, (*v*, *u*, "*w* is the new cluster head") and if *u* = *w*, *u* changes its status *u.status* = *cluster head*, else *u* sets it cluster head to be new cluster head *w*, *u.clusterhead* = *w*.

Round 4:

1. If cluster member *u* receives message, (*v*, *u*, "*w* is the new cluster head"), *u* sends message to old cluster head *v* to confirm new cluster head *w*, (*u*, *v*, "Cluster head *w* confirmed").
2. If old cluster head *u* receives messages (*v*, *u*, "Cluster head *w* confirmed"), *u* changes its status, *u.status*, to cluster member.

## 4 Key Management System

### 4.1 Key Cryptography

We propose a key management system supported by public key and symmetric key cryptography. Both cryptographic methods have their strengths and weaknesses, but when they are used together the weaknesses will be overcome and the security will be provided. We also propose the use of two types of keys for tasks such as data aggregation, which may occur at intermediate nodes during data transmission. Public key cryptography was initially classified as infeasible for WSNs. Recent studies have shown that ECC is feasible for existing sensor node hardware and therefore feasible for WSNs [12]. Our key management system couples ECC with the polynomial-based key distribution scheme. The polynomial-based scheme allows two nodes to generate a pair wise key using a randomly generated symmetric bi-variate *t*-degree polynomial *f(x,y)*, where $f(x,y) = \sum_{i,j=0}^{t} a_{ij} x^i y^j$ over a finite field $F_q$. Each sensor needs to store a *t*-degree polynomial which occupies $(t+1)\log q$ storage space. To establish a pair wise key, both sensor nodes need to evaluate the polynomial at the ID

of the other sensor node. The polynomial-based scheme is secured up to a degree of *t*, where *t* is the number of nodes that needs to be compromised in order for an adversary to know the symmetric key generated between any two nodes [13, 14].

### 4.2 Preloaded Keys and Materials

Before the nodes are deployed, both H-nodes and L-nodes are preloaded with an initial temporary symmetric key $K_G$. Each H-node is preloaded with its ECC public/private key pair. Therefore, each H-node has a total of 3 keys. We propose two cases for preloading L-nodes. Preloading for H-nodes remains the same in both cases. In case 1 each L-node is preloaded with its private key from its ECC public/private key pair for a total of 2 keys. In case 2 each L-node is preloaded with its ECC public/private key pair for a total of 3 keys. Let *M* represent the number of H-nodes and *N* represent the number of L-nodes, then the number of preloaded keys for the entire network is:

$$\text{Case 1: } 3 \times M + 2 \times N . \quad \textbf{(1)}$$

$$\text{Case 2: } 3 \times M + 3 \times N . \quad \textbf{(2)}$$

The L-nodes are also pre-loaded with a randomly generated symmetric bi- polynomial that will used to generate a symmetric key with a neighboring L-node such as the *cm* to *ch* communication.

### 4.3 Stored Keys

After the nodes are deployed, they perform neighbor discovery and clustering processes. The key $K_G$ is used during neighbor discovery and clustering so that information is broadcasted securely. Even though we present 2 cases for preloading keys, key storage is the same for both cases. Once H-nodes have divided the sensor field into regions, L-nodes have been clustered, and the backbone tree has been formed, key exchange can start by using $K_G$ to securely communicate. Key exchange will be discussed in detail in the next section. The number of keys stored depends on the number of clusters throughout the entire network. The following variables help define the amount of keys stored:

- $N_h$ – *number of L-nodes in the region of a H-node*
- $K_h$ – *number of the neighbors of a H-node on backbone tree*
- $N_{ch}$ – *number of the cluster members in a cluster with cluster head ch*
- $K_{ch}$ – *number of the neighbors of a cluster head on the backbone tree*
- $N_c$ – *number of the clusters in the network*

After key exchange each node stores a certain number of keys depending on their role in the network hierarchy:

- H-node – stores ECC public/private key pair, public keys of all L-nodes in its region and the public keys of all its neighboring H-nodes in the backbone tree.

The keys stored for neighboring H-nodes are public keys of parent and children on backbone tree. Let $A_h$ represent the number of total keys stored at one H-node, then:

$$A_h = 2 + K_h + N_h \, . \quad \textbf{(3)}$$

- Cluster Head – stores private key of ECC pair, public key of the regional head, distinct symmetric keys with all *cm*s generated by symmetric polynomial, and distinct key with parent and children on backbone tree. Let $B_h$ represent the number keys stored by a *ch*, then:

$$B_{ch} = 2 + K_{ch} + N_{ch} \, . \quad \textbf{(4)}$$

- Cluster Member – stores private key of ECC pair, the public key of their region head, and a distinct symmetric key with *ch* for a total of 3 keys.

$$C_{cm} = 3 \, . \quad \textbf{(5)}$$

Let $K_{all}$ represent the total number of keys stored in the entire network. By summing equations 3-5 for all H-nodes, cluster heads, and cluster members, then $K_{all}$ is calculated as follows:

$$K_{all} \leq \sum_{all\ h} A_h + \sum_{all\ ch} B_{ch} + \sum_{all\ cm} C_{cm} \, . \quad \textbf{(6)}$$

Let $M$ represent the number of H-nodes and $N$ represent the number of L-nodes, the number of edges in *BT(G)* is $(M + N_c) - 1$. There are no more than 2 keys stored between each edge, meaning that each node stores a key to securely communicate with the neighbor of an edge on *BT(G)*. Therefore the summation of keys stored for $K_h$ and $K_{ch}$ by all H-nodes and *chs* on *BT(G)* is not larger than $2[(M + N_c) - 1]$. Since $N_c$ represents the number clusters as well as *chs* in the network, the number *cms* is $N - N_c$. For keys stored by all H-nodes to securely communicate with L-nodes in its region, $N_h = N$. For keys stored by all *chs* to securely communicate with *cms*, $N_{ch} = N - N_c$. If we rewrite equation 6 in terms of $M$, $N$, and $N_c$, then we get equation 7 for $K_{all}$.

$$K_{all} \leq 2M + 2[(M + N_c) - 1] + 2N_c + N + 4(N - N_c) \, . \quad \textbf{(7)}$$

Therefore, we get,

$$K_{all} \leq 4M + 5N - 2 \leq 4M + 5N \, . \quad \textbf{(8)}$$

### 4.4 Key Distribution and Set-up

As mentioned in the previous section, once the nodes are deployed, they begin to build the CHNetArch. Since nodes do not know their location, signal strength can be used to determine the proximity of a neighboring node. The following key and message notations are used to discuss this section:

- $K_G$ – a temporary preloaded global symmetric key known by all nodes that is used for encryption and decryption, and discarded and no longer needed once the network architecture is established and key exchange is complete.
- $x_{pb}/x_{pr}$ – public and private key of node *x*.
- $K_{uv}$ – symmetric key shared between node *u* and *v*, where $K_{uv} = K_{vu}$, and created by the symmetric bivariate polynomial.
- Broadcast message – *{plain text, encrypted text}*
  - plain text – *{sender.id, encrypted text}*
  - encrypted text – *{plaint text, Encryption_key(sender.id, sender.regionhead, sender.message)}*
  - message from node x – *{x.id, Encryption_key(x.id, x.regionhead, x.message)}.*
- Unicast message from node x to node y – *{plain text, encrypted text}*
  - plaint text – *{sender.id, receiver.id, encrypted text}*
  - encrypted text – *{plain text, Encryption_key(x.id, y.id, x.regionhead, x.message)}*
  - message from node x to node y – *{(x.id, y.id), Encryption_key(x.id, y.id, x.regionhead, x.message)}*

The IDs are sent in plaintext to allow each node to know who is sending the message and whether it is meant for them to decrypt or forward according the communication protocol. Other materials may be sent with the message if requested by the sender or for authentication purposes. These keys are used in the different phases of building the network architecture as follows.
The IDs for the each message is sent in plaintext to allow each node to know who is sending the message, who is receiver and whether the message should be decrypted or forwarded. The IDs for each message is also sent in the encrypted text to verify the ID of the sender and receiver of the message. The IDs in plain text must be the same as the IDs in encrypted text. Public/private keys may be sent with an encrypted message if requested by sender for authentication purposes.

I. Key Management Protocol during self-formation of CHNetArch – The following steps describes how secure information and security keys are exchanged during self-formation of CHNetArch.
   a. **Regional Formation** – regional heads (*h*) use $K_G$ to broadcast an encrypted message, *{h.id, $K_G$(h.id, $h_{pb}$, h.message)}*, including its ID, separating the deployed area into regions where L-nodes select the H-node with the strongest signal as its regional head. Each L-node that receives that broadcast message decrypts the message using $K_G$. The ID of the sending node is added as the regional head for L-nodes. This broadcast message also includes the public key of the regional node's ECC pair. This allows for any future broadcasts after network setup from regional heads to be decrypted and authenticated.
   b. **Neighbor Discovery** – each L-node (*u*) uses $K_G$ to broadcast an encrypted message, *{u.id, $K_G$(u.id, u.region, u.message)}*, including its ID, to its neighboring L-nodes. Each L-node that receives the message from a neighbor uses $K_G$ to decrypt the message and add the ID of the

neighbor to its neighbor list. The ID of the regional head (*u.regionhead*) is also included with the message so that nodes only add neighbors from the same region.

c. **Clustering** – each L-node (*u*) selects the neighbor node (*v*) with the smallest ID and uses $K_G$ to send an encrypted cluster head request message to *v*, *{(u.id,v.id), $K_G$(u.id, v.id, u.region, u.message)}*. If *v* receives the cluster head request, it decrypts the message using $K_G$, changes its status (*u.status*) to be cluster head and sends an acknowledgement message encrypted with $K_G$, *{(v.id,u.id), $K_G$(v.id, u.id, v.region, v.message)}*. If *u* receives the acknowledgement message, it decrypts the message using $K_G$. If an L-node (*u*) receives a cluster head request when it is already a cluster member of cluster, it uses $K_G$ to send an encrypted message to its cluster head (*v*), *{(u.id,v.id), $K_G$(u.id, v.id, u.region, u.message)}*, requesting to be removed as a cluster member. The ID of the regional head is included the messages to prevent nodes from joining a cluster outside their region. After clustering, all L-nodes are either cluster members or cluster heads. Cluster members discard $K_G$ because it is no longer needed. After network setup cluster members and cluster heads the preloaded polynomial to establish a pairwise key, $K_{uv} = K_{vu}$.

d. **Backbone Tree Formation** – Within each region, starting from the regional head (*p* for parent) as the root, $K_G$ is used to broadcast an encrypted, *{p.id, $K_G$(p.id, p.region, p.message)}*, find children (i.e., the cluster heads within *d* communication range of the regional head). Each cluster head has a field *u.parent* initially set to null. This prevents a cluster head that is already a parent from becoming a child of another parent. If a cluster head (*c* for child) within *d* communication range receives the message and *u.parent* = null, it decrypts the message with $K_G$, sets *u.parent = p.id*, and sends an reply encrypted with $K_G$, *{(c.id, p.id), $K_G$(c.id, p.id, c.region, v.message)}*. If *p* receives a reply from *c*, *p* decrypts the message using $K_G$, then *c* is added to *p*'s child list (*p.childlist*). Once a cluster head has found a parent, it broadcasts an encrypted message using $K_G$, *{p.id, $K_G$(p.id, p.region, p.message)}*, to find its children. This continues until all cluster heads have found a parent in the regional backbone tree.

Once the backbone tree is complete, each regional head needs to obtain the public keys of all L-nodes in its region for the future secure communication.

- Case 1 (corresponding to choice 1 in 3.2.2.1) – regional heads (*h*) broadcast "cluster member list request", *{h.id, $K_G$(h.id, h.region, h.message)}*, encrypted using $K_G$, to the cluster heads (*u*) in its region. When cluster heads receive the request, it decrypts the message using $K_G$, and sends a list of cluster members (*u.cmlist*) to its regional head using $K_G$, *{(u.id, h.id), $K_G$(u.id, h.id, u.region, u.cmlist, u.message)}*. After the regional heads receive *u.cmlist* from all cluster heads within its region, via the regional backbone tree, the message is decrypted using $K_G$. $K_G$ is removed from all

remaining nodes and is no longer needed for future communications. Regional heads then send a message with a list of L-nodes in its region to the base station (*B*) encrypted with the regional head's private ECC key, *(h.id,B.id),*$h_{pri}$*(h.id, h.regionlist, h.message)}*, to request the public keys of each L-node in its region. In this scenario, the base station acts as a key distribution center (KDC), which knows the public/private ECC pair for every node in the network.

- Case 2 (corresponding to choice 2 in 3.2.2.1) – Regional heads broadcast an encrypted "cluster member list and key request" message to cluster heads (*u*) in its region using $K_G$, *{h.id,* $K_G$*(h.id, h.region, h.message)}*. When the cluster heads receive the requests, it is decrypted using $K_G$ and each cluster head requests the public key from each cluster member (*v*) using the pairwise key, $K_{uv}$, it shares with each cluster member, *{u.id, v.id,* $K_{uv}$*(u.id, v.id, u.region, v.message)}*. Each cluster member that receives the request, decrypts the message using $K_{vu}$, and sends a reply including its public key, encrypted with $K_{vu}$, *{v.id, u.id,* $K_{vu}$*(v.id, u.id, v.region,* $v_{pb}$*, v.message)}*. When the cluster head receives replies from all cluster members, it then sends a cluster member list with keys to its regional head via the backbone tree using $K_G$, *{(u.id, h.id),* $K_G$*(u.id, h.id, u.region, u.cmlist, u.keylist, u.message)}*. After a regional head receives messages from all cluster heads in its region, $K_G$ is removed from all remaining nodes. Each L-node also discards its public key as a security precaution.

Even though the key storage is the same for both cases, case 1 requires H-nodes to communicate with a KDC during setup and each time a new node is added to the network. Case 2 preloads more keys, but no communication with a KDC is needed, allowing the network to self-organize during key setup and when a new node joins the network. For any new node joining the network, they are preloaded with the keys according to the cases mention in Section A. The number of keys stored by a new node depends on its role once it joins the network. If any L-node has to leave the network due to depleted resources, node failure or node compromise, all keys shared with that node is removed from its communicating neighbors. If that node is a *ch* and has remaining *cm*s, then head rotation is performed to select a new *ch* among the remaining *cm*s according to Section III. New keys are established as previously shown in each phase and according to preloaded keys for case 1 or case 2.

### 4.5 Secure Routing

Figure 2 shows a routing hierarchy and what type of key is used in communication between each type of node. Secure communication occurs from child to parent as follows:

- H-node to H-node – *h* is child, *w* is parent. Information is encrypted with an H-nodes private key and decrypted by the parent H-node with the public key.
  - *{(h.id, w.id),*$h_{pr}$*(h.id, w.id, h.messege)}*
- *ch* to H-node – *u* is child, *h* is parent. Information is encrypted with a ch's private key and decrypted by H-node with ch's public key.
  - *{(u.id, h.id),*$u_{pr}$*(u.id, h.id, u.messege/u.request)}*

- *cm* to *ch – u* is child, *v* is parent. Information is encrypted and decrpyted with pairwise key between cm and ch.
  - *{(u.id, h.id),$K_{uv}$(u.id, h.id, u.messege/u.request)}*

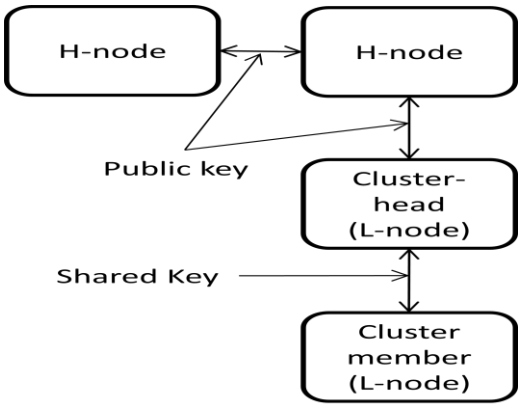


Figure 3 - Secure routing hierarchy for CHNetArch.

## 5 Performance Evaluation

The testing scenario features 1000 L-nodes and 20 H-nodes. The hierarchical network architecture was constructed by computer simulation with the communication range of $d$ = 60m for L-nodes and $D$ = 250 m for H-nodes. When forming clusters, the transmission range of L-nodes are adjusted to a range *d/4* to form the clusters with diameter of d/2, where each cluster is a complete graph, i.e., each node in the cluster is communication range with every other node. The nodes were randomly deployed over a 500m × 500m area. These parameters where chosen to achieve connectivity over the deployment area so that each node would be in communication range with one or more nodes. Substituting the simulation parameters in the equations for the number of preloaded keys in case 1 and case 2, yields 2060 and 3060 keys respectively. Comparing this value to the centralized Du-scheme [8], we reduce the amount of preloaded keys by approximately 90% for case 1 and 86% for case 2. We also can estimate the number of stored keys. As we showed in Section IV (C), the total number of keys is $K_{all} \leq 4M + 5N$, but the precise number of keys $K_{all}$ depends on the number of cluster $N_c$. We conducted 20 simulation runs where

1020 nodes were randomly deployed. The simulation results show that the network stored on an average $is\ K_{all} = 5080$ keys.

## 6 Conclusion

In this paper, we presented a security system for heterogeneous wireless sensor networks that couples robust network architecture with a hybrid key management scheme. The robust network architecture features a hierarchical cluster-based network architecture that defines the role of H-nodes and L-nodes to establish a measure of security through the communication protocol. This network architecture has a direct effect on the key management scheme which uses both ECC and symmetric bi-variate polynomial-based key distribution to provide secure communication via the backbone tree. The network architecture can be self-reconfigured without localization information and it provides an efficient key management scheme for heterogeneous wireless sensor networks. With the storage being a limitation for sensor nodes, only a small amount keys need to preloaded and stored over the entire network.

**Acknowledgements.** This research is supported by the Defense Threat Reduction Agency (DTRA)

## References

1. Kaplantzis, S., Mani, N., Palaniswanmi, M., and Egan, G., "Security models for wireless sensor networks," *Conversion Report, Monash University,* 20th, March 2006.
2. Long, K.J., Haupt, S., Young, G.S., Rodriguez, L.M., McNeal III, M., "Source term estimation using genetic algorithm and scipuff," *7th Conference on Artificial Intelligence and its Applications to the Environmental Sciences*, January 2009.
3. Mhatre, V. and Rosenberg, C., "Homogeneous vs heterogeneous clustered sensor networks: A comparative study," *2004 International Conference on Communications*, Vol. 6: pages 3646-3651, June 2004.
4. Al-Fares, M.S., Sun, Z., and Cruickshank, H., "A hierarchical routing protocol for survivability in wireless sensor network (WSN)," *Proceedings of the International MultiConference of Engineers and Computer Scientists IEEE 2009*, Vol I. March 18-20, 2009.
5. Khan, Z.H., Catalot, D.G., and Thiriet, J.M., "Hierarchical wireless network architecture for distributed applications," *2009 Fifth International Conference on Wireless and Mobile Communications 2009 IEEE*, 2009.
6. Traynor, P., Kumar, R., Choi, H., Cao, G., Zhu, S., and La Porta, T., "Efficient hybrid security mechanisms for heterogeneous sensor networks," *IEEE Transactions on mobile computing*, Vol. 6, June 2007.
7. Lu, K., Yi Qian, Guizani, M., and Chen, H., "A framework for a distributed key management scheme in heterogeneous wireless sensor networks," In *IEEE Transactions on Wireless Communications,* Vol. 7, February 2008.

8. Du, X., Xiao, Y., Ci, S., Guizani, M., and Chen, H., "A routing-driven key management scheme for heterogeneous sensor networks," In *The ICC 2007 Proceedings*, pp. 3407-3412. IEEE Communications Society, 2007.
9. Du, X., Guizani, M., Xiao, Y., and Chen, H., "A routing-driven elliptic curve cryptography based key management scheme for heterogeneous sensor networks," *IEEE Transactions on Wireless Communications*, Vol. 8, No. 3, March 2009.
10. He, T., Huang, C., Blum, B., Stankovic, J., and Abdelzaher, T., "Range-free localization schemes for large scale sensor networks," *MobiCom '03 Proceedings of the 9th Annual International Conference of Mobile computing and networking*, 2003.
11. Gura, N., Patel, A., Wander, A., Eberle, H., and Shantz, S.C., "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," *Proceedings of the 6th International Workshop on Cryptographic Hardware and Embedded Systems*, Boston, MA, August 2004.
12. Liu, A. and Ning, P., "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," *7th International Conference on Information Processing in Sensor Networks (IPSN 2008)*, April 2008.
13. Schmidt, S., Krahn, H., Fischer, S., and Wätjen, D., "A security architecture for mobile wireless sensor networks," *ESAS 2004*, Springer-Verlag Berlin Heidelbierg, 2005.
14. Liu, D., Ning, P., and Li, R., "Establishing pair-wise keys in distributed sensor networks," *ACM Transaction Information Systems Security*, 8(1):41-77, 2005.
15. Eschenauer, L. and Gligor, V.D., "A key management scheme for distributed sensor networks," *Proceedings of the 9th ACM CCS*, November 2002.
16. Chen, W., Miao, H., and Hong, L., "Cross-layer Design for Cooperative Wireless Sensor Networks with Multiple Optimizations," *International Journal of Networking and Computing*, (publication date).
17. Uchida, J., Muzahidul Islam, A.K.M., Katayama, Y., Chen, W., and Koichi, W., "Construction and maintenance of a novel cluster-based architecture for ad hoc sensor networks," *Ad Hoc & Sensor Wireless Networks* Vol. 00, pp. 1-31, 2008.
18. Walters, J.P., Liang, Z., Shi, W., and Chaudhary, V., "Wireless sensor network security: A survey," In Security in Distributed, Grid, and Pervasive Computing Chapter 17, Auerbach Publications, CRC Press 2006.