POST-DEPLOYMENT KEY MANAGEMENT IN HETEROGENEOUS WIRELESS SENSOR NETWORKS

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

by

Paul Edward Loree

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

June 2010

Fargo, North Dakota

North Dakota State University
Graduate School

Title

POST-DEPLOYMENT KEY MANAGEMENT IN

HETEROGENEOUS WIRELESS SENSOR NETWORKS

By

PAUL EDWARD LOREE

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

ABSTRACT

Loree, Paul Edward, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, June 2010. Post-Deployment Key Management in Heterogeneous Wireless Sensor Networks. Major Professor: Dr. Kendall Nygard.

Many wireless sensor network applications require secure communication between nodes in the network. However, establishing pair-wise keys between nodes to provide security is challenging due to the limited resources in sensor nodes and the hostile environments in which they are deployed. Many key establishment schemes have been previously proposed for wireless sensor networks. However, most of these schemes were designed to work in a homogeneous network environment in which the nodes all have similar capabilities. Our work establishes that better performance can be achieved in a heterogeneous sensor network environment. We present a key management scheme for establishing pair-wise keys after deployment in a heterogeneous wireless sensor network. We take advantage of the more powerful nodes present in a heterogeneous network to reduce the communication overhead and ultimately the power consumption necessary to perform these services to the network. Additionally, by taking advantage of these nodes we are able to increase the overall network connectivity and resiliency against node capture attacks.

ACKNOWLEDGMENTS

I would first like to thank my parents, my brothers and my sister for supporting me while I pursued this master's degree. I would also like to thank Larry Atwood, Scott Kast and Dr. Selmer Moen at Minot State University who encouraged me to pursue a graduate degree. I would also like to thank Dr. Kendall Nygard and Dr. Xiaojiang Du for their support and advice while completing my research. Finally, I would like to thank the North Dakota EPSCoR for funding my research in 2008-2010.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Several wireless sensor network applications require being able to securely transmit messages between nodes in the network to ensure the confidentiality and integrity of the data being transmitted throughout the network has not been compromised by an attacker. Wireless sensor networks may be deployed to monitor an area in dangerous or secure environments. For instance, sensors may be deployed in battlefields to monitor troop movements of an enemy or to monitor building infrastructure intrusions. For many applications however it is not possible to physically place each node in specified locations over the area to be covered. In many cases when it is not possible to place individual nodes, thousands to tens of thousands of nodes may be randomly deployed over an area by dropping them from aircraft or some other method which scatters the nodes over the area. However, this creates a unique problem when attempting to establish symmetric keys between neighboring nodes since deployment location is unpredictable and preloading symmetric keys on each node may not allow nodes to communicate securely if they do not share a key in common.

A couple naïve approaches would be to preload each node with a single globally used pair-wise key or a unique pair-wise key for every other node being deployed in the network to ensure each node is able to securely communicate between any potential neighboring nodes after deployment. However,in both of these approaches if any one node is compromised by an attacker, the entire network becomes compromised since the attacker would be able to determine all of the keys being used to encrypt the data. The main benefit of the first approach is that it requires very little memory to store the key needed to secure data transmission between nodes in the network and is highly scalable for the addition of new nodes in the network after deployment. The second approach also allows for perfect connectivity between nodes in the network after deployment but this approach does not scale well with very large networks since preloading all of the keys onto each node may not be possible to do because of the severe memory limitations of wireless sensor nodes. Furthermore, this approach does not easily allow for new nodes to be introduced to the network after deployment.

Wireless sensor nodes are severely limited by available resources which must be taken into consideration when developing any security mechanism into the network. For instance two popular wireless sensor nodes are the MICA2 and MICAz Motes. These devices use 8 bit, 16 MHz processors and have a total of 4K bytes of configurable EEPROM and 128K bytes of programmable memory [1], [2]. Due to the limitations of memory space to store programming code and the necessary keys to ensure security along with the limitations in processing power, existing public-key encryption schemes available for standard networks are not feasible options. Additionally, these devices are designed to run on two standard AA batteries, which may not be replaceable after deploying the nodes and thus the battery power available in each node must be preserved as much as possible in order to ensure the network continues to operate for as long as needed by the application. This presents another constraint when developing a security mechanism into the network since wireless communication and computational processes are expensive to battery power and must be limited as much as possible so each node is still able to perform its necessary functions after the deployment of the network [3]. As such security algorithms developed should attempt to be as efficient as possible and attempt to limit the number and size of necessary messages being exchanged between neighboring nodes and limit the number of computations necessary to perform the algorithms.

1.1. Network Model

In this work we consider two classes of wireless nodes deployed in the wireless sensor network (WSN) to form a heterogeneous sensor network (HSN). Low powered nodes, L-sensors, such as the MICA2 or MICAz motes and much more powerful nodes, H-sensors, such as the Imote2 [2]. For instance, the Imote2 uses a 32 bit processor which can be dynamically configured for either low powered use at 13 MHz up to high powered use of 416 MHz operational speed and contains 32M bytes of flash memory and SDRAM memory for code and storage. These more powerful H-sensors are comparable to PDAs in terms of computational power and have a much longer transmission range than the L-sensors present in the network. Most of the existing work which has been done in the past for key distribution in WSN has been focused primarily on homogeneous networks consisting of only L-sensors. Since the overall cost of a large scale WSN is typically considered we focus on heterogeneous wireless sensor networks (HSN) consisting of a large number of L-sensors and significantly fewer H-sensors which will be used as cluster heads in the network.

We take advantage of these more powerful nodes present in the HSN to provide pair-wise key distribution to the lower powered L-sensors after deployment of the nodes without the need of deployment knowledge. Furthermore, since H-sensors have a much larger transmission range they are able to transmit messages directly to L-sensors within their cluster in a single hop which reduces the number of messages being transmitted by the network. Additionally since L-sensors must be deployed as economically as possible in large numbers we assume that L-sensors are not tamper resistant. Therefore they are not protected from being compromised physically by an attacker and as such may release all information they contain. We also do not assume H-sensors contain tamper resistant hardware and may also release any information they contain to a physical attacker. However, tamper resistant sensors are available which could be used in the network to protect from these types of attackers.

1.2. Work Overview

The purpose of this study is to develop a set of algorithms to provide efficient generation and distribution of pair-wise keying information to a heterogeneous wireless sensor network after deployment has occurred. This keying information can then be used to create pair-wise keys needed for symmetric encryption of data being transmitted between neighboring sensors. To achieve this we have several objectives and tasks which need to be completed.

1.2.1. Objective One

The first objective is to design new algorithms to be used for generating the keying information necessary to create the pair-wise keys in a heterogeneous sensor network environment. This requires several tasks to be completed. The first task is to design an algorithm for generating keying information by the more powerful H-sensors being deployed in the network which takes advantage of the extra resources present. This algorithm can be found in Subsection 2.6.1. The second task is to design an algorithm used by the lower powered L-sensors which allows the advantages of the H-sensor nodes to be utilized. The third task is to ensure that a backup method of generating keying information for the network exists when the higher powered H-sensors are not available to do so.

Details of this algorithm can be found in Subsection 2.6.2. Lastly we must find an efficient algorithm for distributing this keying information to the sensors in the network so that pair-wise keys between neighbors can be established. This algorithm is described in Subsection 2.6.3.

1.2.2. Objective Two

The second objective of this study is to analyze our method and compare it against existing methods of key management designed for wireless sensor networks to ensure improved performance. Several tasks must be completed to do this. The first task is to analyze the communication overhead of our scheme and compare it to existing schemes for WSNs. These comparisons can be found in Subsection 2.7.1. Our second and third tasks are to analyze the storage overhead and computational overhead of our scheme in comparison with existing schemes. These results can be found in Subsection 2.7.2. The last task required is to investigate the resiliency of our scheme against node capture attacks. Subsection 2.7.3 describes these results.

1.2.3. Objective Three

Our third objective is to simulate our scheme in order to evaluate network connectivity in comparison with existing schemes for WSNs. The first task needed for this is to develop simulation software for our scheme and a few of the existing schemes. The second task is to experiment using numerous simulations of the schemes under different scenarios. Details of regarding the simulation software design and setup can be found in Subsection 2.7.4. The last task is to compare the simulation results of our scheme with the existing schemes to show the network's performance improvements with our methods. Section 2.7.5 contains the results of our simulations.

1.3. Thesis Overview

In this work we present a key generation and distribution scheme for heterogeneous sensor networks which take advantage of the additional capabilities of H-sensors present in the network. Additionally, our scheme does not require *a priori* knowledge of sensor deployment locations and allows for sensor nodes to be randomly distributed over the deployment area. Chapter 2 discusses our three algorithms for generating pair-wise keys depending on if an H-sensor or an elected L-sensor to be used by nodes in the network and the algorithm needed for distributing the keys to the nodes in each cluster. We then examine our scheme through simulation in comparison to existing schemes which have been previously proposed. Finally we conclude our work in Chapter 3.

CHAPTER 2. POST-DEPLOYMENT KEY MANAGEMENT

2.1. Introduction

In this chapter we describe our post-deployment key management scheme for heterogeneous sensor networks. We present three algorithms in our scheme to provide key generation and distribution to nodes in the network. In our work we take advantage of H-sensors present in heterogeneous sensor networks to provide keying information to L-sensors. To take advantage of these nodes, we present an algorithm in Subsection 2.6.1 designed to be used on these H-sensors for generating keying information. The Subsection 2.6.2 we present an algorithm to be used on L-sensors which can allow these nodes to generate keying information and distribute this information to their neighboring nodes only when needed. In Subsection 2.6.3 we present a third algorithm which describes the process for distributing keying information once this information has been created.

We follow up in Section 2.7 with analysis of our scheme against previously proposed schemes which have been designed for wireless sensor networks. We then present our results from simulations in Subsection 2.7.5 performed on our scheme and compare these results with a predistribution scheme and a post-deployment scheme which have been previously proposed.

2.2. Related Work

Many schemes have been proposed in the past for securing the wireless transmission of data in WSN. However, most of these works focus only on homogeneous sensor networks which limit the capabilities of the network. Eschenauer and Gligor proposed a scheme (E-G) in [4] which preloads a set of pair-wise keys onto each node being deployed by randomly selecting a quantity of keys from a larger pool of keys that have been generated prior to deployment. After the nodes have been deployed in the area and they have completed their bootstrapping process, nodes in the network are able to establish pair-wise keys with their neighboring nodes if their neighbors have also selected one or more keys in common from the pool of keys. However, this approach relies on the probability of two neighboring nodes having selected at least one of the keys from the pool in common so either a large selection of keys must be preloaded on each node or the key pool size must be kept small to ensure a high probability of connectivity.

Eschenauer and Gligor's scheme was improved upon in [5]. Chan et al. proposed a scheme which increases the security of the network by requiring $q > 1$ keys must be shared between neighboring nodes in order to create a secure link between nodes. This requirement of requiring more keys in common increases the security of the network. This also in turn requires more nodes which must be compromised by an attacker in order to compromise the network. However, by increasing the number of keys which must be shared between nodes it decreases the probability of any two neighboring nodes sharing q keys in common.

Zhu et al. presented a key management scheme in [6] where they defined a minimum time, T_{min}, where the network is considered secure after deployment. Any processing necessary for booting the network and establishing secure communication links between nodes must take place in less than or equal to this time limit, $T_{est} \leq T_{min}$. Their approach preloads a global symmetric key to use during network discovery while bootstrapping the nodes and this key is assumed to remain secure during T_{min} while pair-wise keys can be established between neighboring nodes.

Two independently proposed predistribution schemes were presented in [7] and [8]. These two schemes are similar in their approach by generating a pool of key spaces and preloading each node with randomly selected keying shares from the key spaces pool, where one keying share belongs to one of the key spaces from the pool. The difference in their approaches lies on the key space model they use which are presented in [9] and [10], respectively. Two nodes in the network are then able to establish a pair-wise key after being deployed if they contain keying shares from the same key space. These key space models presented in [9] and [10] are further discussed in Section 2.3 below.

Group-based key predistribution schemes have also been proposed in [11], [12], [13] and [14]. In [11], Du et al. proposed grouping sensor nodes prior to deployment which would be dropped into the coverage area in groups. In this scheme a large pool of keys is generated and further partitioned into groups of keys with each group of sensor nodes being assigned one of the subsets from the larger key pool. When groups being deployed will be neighboring on another, the subsets of keys associated with those groups will contain overlap of keys to be used in order to increase the probability of connectivity between neighboring groups of nodes. Nodes are then preloaded with randomly selected keys from the subset assigned to them and deployed in their geographical regions accordingly. In [12], Yu and Guan proposed a similar scheme however they divided the deployment area into regular hexagon regions and each group of nodes preloaded with a selection of pair-wise keys was deployed into a specific hexagonal region. Liu et al. proposed group-based scheme in [13] which creates groups based on IDs which have been assigned to each sensor prior to deployment. In their scheme each node belongs to two groups, a disjoint deployment group and a disjoint cross group. Nodes belonging to the same group can establish a secure communication link between themselves based on key predistribution schemes found in key based schemes [4], [5], or [15] or key space schemes in [9] or [10]. A similar approach was proposed in [14]. In this scheme Zhao et al. defined different deployment groups and cross groups which result in nodes belonging to more than one of each type of group. The major drawback to all of these group-based approaches is the requirement of deployment knowledge for sensor placement before the sensors have been deployed in the network. In our work we eliminate the need for prior deployment knowledge.

In [16], Liu et al. proposed an *in situ* scheme for WSNs which does not require deployment knowledge and removes the probabilities of any two neighboring nodes sharing a common key after deployment by generating keying shares during the bootstrapping phase of deployment. Their scheme, called SBK, uses elected service nodes to generate keying shares from either of the previously mentioned key space models in [9] and [10]. After keying shares have been generated by these probability-based elected service nodes, the service nodes distribute the keying information to worker nodes within a maximum hop distance from the service node. Since this scheme was designed for homogeneous WSNs one of the drawbacks to this scheme is the computation and communication overhead required by the elected service nodes. These nodes are considered expendable in the network as they are likely to die shortly after the bootstrapping phase from battery depletion.

Our work extends on the SBK scheme in [16] by introducing a HSN network model. With the inclusion of H-sensors in the network we develop algorithms to put much of the workload of generating and distributing keying shares based on the key space models presented in [9] and [10] which have been redesigned for use in WSNs in [7] and [8], respectively. Additionally, preliminary work discussed here was presented in [17].

2.3. Key Space Models

Two key space models which were presented in [9] and [10] will work for our scheme for generating and distributing keying information to allow nodes in the network to establish pair-wise keys with their neighbors. In [10] a polynomial based key space was proposed. This model uses a bivariate symmetric n-degree polynomial such that

$$f(x,y) = f(y,x) = \sum_{i=0}^{n} a_i x^i y^i \tag{1}$$

over a finite field F_q, where q is a large prime number suitable for cryptographic keys. A pair-wise key can be established between two nodes in a network which share the same coefficients a_i, the key space, by exchanging their IDs and computing the formula found in Equation 1. For example, assume sensor nodes with IDs u and v. By exchanging their IDs and inputting their ID along with their neighbors ID into the formula, a pair-wise key can be established by nodes u and v since $f(u,v) = f(v,u)$. Thus the key $k_{uv} = k_{vu}$.

The second key space model presented in [9] is similar to the polynomial based model but is a matrix based approach. In this model, a symmetric $(n+1) \times (n+1)$ public matrix G and a symmetric $(n+1) \times (n+1)$ private matrix D are with D over a finite field $GF(q)$ and q is a large enough prime number suitable for cryptographic keys. These two matrices are used to create a matrix A, shown in Equation 2.

$$A = (D \cdot G)^T \tag{2}$$

Sensors are given the information needed to establish a pair-wise key by giving a sensor, for instance the node with ID u, the uth row of A and the uth column from G. Neighboring nodes are able to establish a pair-wise key by exchanging their columns from G. For example, assume neighboring nodes have the IDs u and v. Sensor u and v exchange their columns G and compute their pair-wise key $k_{uv} = k_{vu}$ with Equation 3 below:

$$K = A \cdot G \tag{3}$$

since if D is symmetric than K is also symmetric.

Furthermore, with a properly designed G matrix, message transmission lengths can be reduced between nodes by only requiring a seed value of G to be transmitted to the neighboring node (e.g. using a Vandermonde matrix [7]). However, this does increase the computational needs of the receiving sensor since the column must be generated after receiving the seed.

A unique property for both of these models is that each is n-collusion resistant. This means that as long as less than n nodes which share the key space are compromised then the entire key space remains perfectly secure as shown in [9] and [10]. Furthermore it has been shown in [16] that the storage requirements for these key spaces are close to $(n + 1) \cdot \log q$ for the polynomial based model and $(n + 2) \cdot \log q$ for the matrix based model as long as the matrix has been properly designed which allows only a seed value to be used for G. However, this requires an additional $(n - 1)$ modular multiplications to generate the column for G before computing the pair-wise key.

Due to the mathematical similarities between these models, either key space model can be used for generating key spaces in our work. We take advantage of the H-sensors present in our network to generate these key spaces after deployment and distribute keying information to nodes within their transmission range except in the rare occurrences where an H-sensor may not provide adequate coverage in areas of the network to cover all possible L-sensors being deployed. In instances such as this we use a similar election method as [16] to choose L-sensors to become cluster heads in order to generate and distribute the necessary keying information in their area. It should be noted however that when using a polynomial based model only the IDs of the nodes in the network must be exchanged while if the matrix key space model is used either a an entire column of G or a seed of the column must be exchanged.

2.4. Rabin's Cryptosystem

In this work we assume two network models, one that is assumed to remain secure long enough for bootstrapping and key establishment to take place and one which is not. In the first model we assume the network will remain secure during some time limit as proposed in [6]. In the latter model however we employ Rabin's Cryptosystem [17] which provides a computationally asymmetric encryption to be used temporarily for exchanging keying information and establishing pair-wise keys. Rabin's cryptosystem is a primitive asymmetric encryption algorithm which uses a public key n and a private key (p, q). When using Rabin's cryptosystem each node in the network would be preloaded with randomly selected large primes necessary to create the public key $n = p \times q$. However, this is only used by the cluster heads in the network to create the public key and after bootstrapping has completed this information can be deleted from memory. Additionally each node must be preloaded with a predefined padding sequence, B, to be appended to the messages in order for Rabin's cryptosystem to function properly.

Encrypting a message with n is computationally cheap for a node and requires one modular squaring operation. However, decrypting the message is computationally similar as decrypting a message using RSA encryption. When the wireless channel is not assumed secure, cluster heads compute n and broadcast a message containing their ID and the value of n to all nodes within transmission range. The nodes receiving n then send their preloaded randomly generated symmetric key, k, to the cluster head be used for further transmissions during the key distribution process. This key is encrypted using n and is denoted by $E_n(k||B) = (k||B)^2 \; mod \; n$. This information is then sent back to the cluster head, along with its ID, which uses the large primes p and q to decrypt the received message. Decryption is denoted by $D_{p,q}(E_n(M||B))$. This message, M, contains the symmetric key, k, to use along with the node ID associated with the key.

2.5. Assumptions

In our work we consider a heterogeneous wireless sensor network and assume that no knowledge of node placement is known prior to deployment. For example, nodes may be randomly deployed throughout the area wishing to be covered by an aircraft drop. Our network is comprised of a large number of tiny, low cost, low powered, low-end L-sensors and significantly fewer numbers of high-end, more powerful H-sensors. H-sensors have much higher computational capacity, longer battery life, a longer transmission range and a larger memory space available to them. We also assume that each node has been loosely time synchronized prior to deployment. Loose time synchronization can be obtained through procedures described in [19] and [20]. Nodes are preloaded with parameters described in Table 1 below. We assume that H-sensors will be able to continue normal function after distributing keying information in their clusters but that L-sensors may die shortly after bootstrapping is complete if they were elected as a low powered cluster head (LPCH). Additionally we assume two network models, one which is assumed secure during the maximum amount of time required for the bootstrapping period and one which is not. We assume an appropriate MAC protocol is in place to avoid communication collisions over the wireless channel. Finally we assume a multi-hop routing protocol is in place for transmitting messages from L-sensors back to H-sensors since their transmission range is much less than H-sensors.

2.6. Post-Deployment Key Management

We present our key management scheme for HSNs in this section which was published in [17]. Each node in the network requires two algorithms to be preloaded onto them prior to deployment. More specifically, we use a separate key generation algorithm for H-sensors and L-sensors, which is loaded onto the sensor depending on the type of sensor, and a key distribution algorithm which is loaded onto each sensor prior to deployment. Due to the extended transmission range of H-sensors which allows for better coverage in a deployment area and the fact that they are used primarily as key management nodes in our scheme we limit messages to one-hop transmissions and do not transmit messages further than a node's transmission range. Table 1 shown below details the various parameters which also must be preloaded onto each node prior to network deployment. Each node is loaded with a maximum cluster size depending on its sensor classification, $\lambda_1$which specifies the cluster size to use for H-sensors and λ_2 defines the maximum cluster size for L-sensors. Since H-sensors have more computational power, memory space and transmission range available to them, λ_1 is usually much larger than λ_2. Furthermore, by setting these values to the degree of the polynomial or dimensions of the matrices used we can ensure each key space remains n-collusion secure as described in [9] and [10]. We assume this is the case in our study as it prevents node capture attacks by ensuring uncompromised nodes in the network are still able to communicate securely.

In addition to the maximum cluster size specifications each node contains a unique ID and is loaded with a time limit t_w. This time limit is the maximum bootstrapping time allowed for the network necessary for generating and distributing keying information. If the network is not assumed to remain secure during t_w, each node must also be preloaded with large primes p and q, a randomly generated symmetric key k, and a predefined pattern B which are needed for Rabin's cryptosystem. Additionally, each node must also be loaded with a large prime used for generating key spaces over a finite field, r and L-sensors contain a probability value for election purposes defined as P.

Table 1. Preloaded Parameters

λ_1	Maximum number of nodes in H-sensor clusters (H-sensors only)
λ_2	Maximum number of nodes in L-sensor clusters (L-sensors only and $\lambda_2 \ll \lambda_1$)
n	Degree of polynomial or matrix dimensions for generating key spaces
t_w	Maximum bootstrapping wait time
ID	Unique sensor ID
r	Large prime for generating key space over finite field
P	Probability value of an L-sensor to be elected a cluster head
p, q	Large primes needed for Rabin's cryptosystem (optional)
B	Predefined padding for Rabin's cryptosystem (optional)
k	Randomly generated symmetric key to use in Rabin's cryptosystem (optional)

2.6.1. H-sensor Algorithm

This section describes our H-sensor algorithm. Each H-sensor deployed in the network works as a cluster head and serves as many nodes defined by $\lambda_1$within transmission range. Figure 1 below details the steps of this algorithm. When a H-sensor is in its bootstrapping phase it calls up the H-sensor algorithm and generates a key space using one of the two models over a finite field r as shown on line 2. When using the polynomial based model the key generation involves randomly generating n coefficients, a_i over the finite field r. These coefficients will be distributed to nodes in the network which will be used in by the formula previously described to generate pair-wise keys with their neighbors. If the matrix based model is used the cluster heads will generate $(n+1)^2$ elements to be used for the public matrix G and the private matrix D. However, if the matrix G is constructed properly, such as using a Vondermonde matrix, then only a seed will need to be generated. Each node in the network will then need to reconstruct their elements from G after receiving this seed.

Once the H-sensor has completed generating a key space it stores one of the key shares for itself which will be used to create pair-wise keys with its neighboring nodes and sets a wait time before broadcasting to nodes within transmission range that it has keying information available to be distributed, on line 3. The wait time is a randomly selected time between zero and half the total time limit, $0 \leq w \leq t_w/2$, specified in the parameters.

By dividing the total time for allowed for booting up the network in half and randomly selecting a waiting time period serves two different purposes, the first purpose by dividing the time limit for bootstrapping is to allow sufficient time so that L-sensors will not start their election processes until a time period greater than $t_w/2$ has elapsed allowing for L-sensors to attempt to join clusters formed around H-sensors first. The second part, which is randomly selecting a wait period, is also to attempt to avoid any message collisions between neighboring H-sensors which may have been deployed within transmission range of each other.

Once this value is set the algorithm starts a loop on line 4 which first listens for any neighboring H-sensors broadcasts. If it hears a broadcast from another H-sensor during this loop it will request keying information from that cluster head to form a secure cluster head to cluster head connection. This allows for H-sensors to be able to communicate directly with each other securely and can reduce the number of messages needing to be sent between L-sensors for services such as data aggregation back to a base station. Once this wait time has elapsed on line 10 the H-sensor initiates the Key Distribution Algorithm, as shown in Figure 3. The Key Distribution Algorithm will be discussed in Subsection 2.6.3 which details the broadcasting method to announce that a key space has been generated and to distribute keying information to nodes within transmission range. Once this process has completed H-sensors then use their keying information to setup pair-wise keys with any other H-sensors and L-sensors within their transmission range to establish a secure connection with each.

```
H-sensor Algorithm
1: function HSensor ($\lambda_1$, $t_w$)
2:      keys ← genKeySpace() > construct key space for $\lambda_1$ nodes, store one for self
3:      w ← random(0 < $t_w$ / 2) > random wait time
4:      while w > 0 do
5:          listen for broadcasts from neighboring H-sensor cluster heads
6:          if BROADCAST heard
7:              RequestKeyInfo > get keying info (cluster head to cluster head
                communication)
8:          end if
9:          elapse(w)
10:     end while
11:     KeyDistro()
12: end function
```
Figure 1. Algorithm for H-sensors

2.6.2. L-sensor Algorithm

In this subsection, we present our L-sensor algorithm which is used for receiving keying information from cluster heads and if necessary elect themselves as a LPCH to provide keying information to nearby nodes in the network. Figure 2 below shows our algorithm we use for L-sensors in the network. When this algorithm starts the first thing the L-sensor does is sets its wait time to half of the total bootstrapping time so that it may listen for broadcasting cluster heads. The algorithm then sets a Boolean variable, *elected*, to false on line 3 which is used to determine if the node is eligible to be elected as a LPCH later on in the algorithm. Once these variables have been set the algorithm starts a loop to listen for broadcasting H-sensor cluster heads announcing they have keying information available to be distributed. The L-sensor would request keying information from all H-sensors it hears giving this announcement. This is to allow for nodes residing on boundaries of clusters to be able to join all of the clusters and establish keys with nodes residing further into the cluster which may not have heard the other H-sensors broadcasts. These nodes then provide communication links between clusters if messages need to be routed through L-sensors. If a L-sensor received keying information from a H-sensor it sets its probability value, $P = 0$, to remove the possibility of being elected a LPCH. This loop continues until the time limit w has elapsed.

Once this wait time is over, one of two things can occur. If a node has successfully received keying information the value of P will equal zero and the rest of the algorithm is skipped and the function ends. However, if a node did not receive keying information from an H-sensor the node starts the election process. The first step which occurs, on line 13, is to reset the variable w to a random time period less than the remaining time of the bootstrapping phase. It will then continue through a loop on line 14 which attempts to elect the sensor as an LPCH using the preloaded probability P as the probability to be elected. If the node is elected it will generate a key space with the same process an H-sensor has generated its key space with the exception that it may use a lower value for the number of nodes it will provide keying information to. If the node has not been elected a LPCH the lines 18 through 20 can be added to the algorithm to increase the chance of being elected the next time through the loop. This helps speed up the overall network setup time but is not necessary for the function of the algorithm.

Once the key space has been generated the node will first listen for any other LPCHs which may have elected themselves earlier then themselves and are already broadcasting they have keying information available as shown on line 21. If another node has already elected itself then the node which had generated its key space after the announcing LPCH deletes its key space information on line 24 and sets its Boolean allowing it to be elected to false.

The L-sensor will then request keying information from the announcing LPCH on line 27. However, if another LPCH is not heard this node will continue to remain an LPCH and start the key distribution algorithm to announce that it has keying information available for nodes within its transmission range as shown on line 30. Once the key distribution algorithm has completed the node finishes this function on line 35. After this algorithm has completed elected LPCHs are not assumed to have much remaining battery power left and as such do not contain their own keying information to setup pair-wise keys with their neighboring nodes. However, if they have not been elected the L-sensors remaining will initiate steps to find their local neighboring nodes and establish pair-wise keys with them by either exchanging their IDs or matrix information as previously described in Section 2.3.

2.6.3. Key Distribution Algorithm

This subsection details the key distribution algorithm used by our HSN when a cluster head (either H-sensor or LPCH) has finished generating a key space and needs to announce they have keying information available. Figure 3 details the steps of this process below.

```
L-sensor Algorithm
1:  function LSensor ($\lambda_2$, $t_w$, $P$)
2:      w ← $t_w$/ 2
3:      elected ← false
4:      while w > 0 do
5:          listen for broadcasts from cluster heads
6:          if BROADCAST heard
7:              RequestKeyInfo > Tx to request keying info (l-sensor to cluster head
                message)
8:              $P$ ← 0 > remove possibility of becoming LPCH
9:          end if
10:         elapse(w)
11:     end while
12:     if $P$ > 0
13:         w ← random($t_w$ / 2)
14:         while w > 0 do
15:             elected ← BecomeLPCH(P) > w/ probability P
16:             if elected then
17:                 keys ← genKeySpace() > construct key space for $\lambda_2$ nodes
18:             else
19:                 increase probability chance (optional)
20:             end if
21:             listen for neighboring node BROADCAST
22:             if BROADCAST heard
23:                 if elected
24:                     delete key space generated
25:                     elected ← false
26:                 end if
27:                 RequestKeyInfo > get keying info (cluster head to cluster
                    head communication)
28:             end if
29:             if elected
30:                 KeyDistro()
31:             end if
32:             elapse(w)
33:         end while
34:     end if
35: end function
```
Figure 2. Algorithm for L-sensors

The first step of this algorithm is to broadcast the cluster head *ID* to all nodes within its transmission range. Once this has been completed the cluster head will start to accept keying information requests from nodes within range which wish to join the cluster until all keying information has been distributed. This is dependent on the number of nodes each cluster is allowed to contain as shown on line 4. If Rabin's cryptosystem is used during this phase the broadcast message contains the cluster heads *ID* and the value for n to use for encrypting the information request reply message. Nodes within range request keying information by sending a unicast message back to the cluster head containing their *ID* and if Rabin's cryptosystem is used they also include the value k which has been encrypted by n as described previously in Section 2.4.

Key Distribution Algorithm

```
1:function KeyDistro($\lambda_1||\lambda_2$, ID, keys)
2:    BROADCAST(id) > broadcast to all nodes within range
3:     while keyed < $\lambda_1 - 1||\lambda_2$, do
4:            if received(RequestKeyInfo(ID))
5:                  KeyingInfo ← an unused key share
6:                  keyed ← keyed ∪ {KeyingInfo}
7:                  send(ID, $k_0$(KeyingInfo))
8:            end if
9:      end while
10:end function
```

Figure 3. Distributes keying information to nodes requesting to be added to the cluster

When a request is received from another node the cluster head selects an unused set of keying information to send to the requester from the key space it generated as shown on line 5. This keying information is added to a used keying information list on line 6 and the keying information is sent back to the requesting node using a unicast message which may be encrypted using k as shown on line 7. Once all keying information has been distributed the function ends and the cluster head may be allowed to delete any keying algorithms and variables in order to recover valuable memory space.

2.7. Performance Evaluation

In this section we evaluate our scheme and analyze the communication overhead, storage and computational overhead and the resilience to node capture attacks. We follow this up with evaluating our scheme through simulations.

2.7.1. Communication Overhead

This subsection analyzes the communication overhead needed by our scheme and compares this overhead with the SBK scheme proposed in [16]. Figure 4 below describes the message exchanges between a cluster head and nodes being added to the cluster by requesting keying information as described in Subsection 2.6.3.

Message Exchange Process

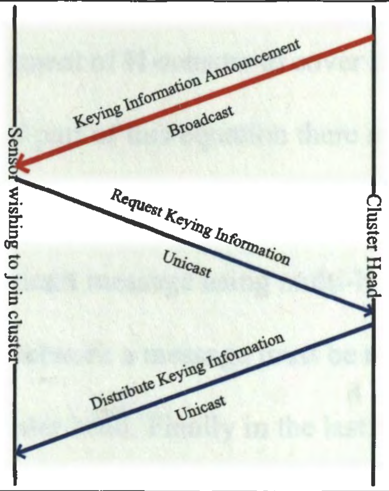
Keying Information Announcement
Broadcast
Request Keying Information
Unicast
Distribute Keying Information
Unicast
Sensor wishing to join cluster
Cluster Head

Figure 4. Messages exchanged between cluster heads and cluster sensors

As shown in Figure 4 the cluster head uses a broadcast message to announce keying information is available. Once this has been heard by sensors wishing to join clusters they request keying information from the cluster head using a unicast message to the cluster head. Keying information is distributed to the sensors joining the cluster using another unicast message sent from the cluster head back to the sensor joining the cluster.

Maximum total network wide communication overhead can be computed using the following equation:

$$H(B + \lambda_1) + L(B + \lambda_2) + \bar{h}H\lambda_1 + d \tag{4}$$

In this equation, H is the number of H-sensors being deployed in the network which have distributed keying information to nodes within their transmission range and have joined their cluster. B is the total number of broadcasts which have been performed by each cluster head to announce they have keying information available while λ_1 is the total number of nodes allowed into the cluster which a unicast messages is sent to distribute keying information to the nodes requesting keying information. Similarly, the total communication for L-sensors can be computed using the second part of this equation. However, with adequate deployment of H-sensors to cover the area sufficiently we assume L will be very small. In the third part of this equation there may be a total of $H\lambda_1$ messages received by the H-sensor cluster heads by nodes requesting information. Additionally since L-sensors must reply using a unicast message using multi-hop routing $\bar{h}$ represents the average number of hops in the network a message must be routed in order for the request to be received by the H-sensor cluster head. Finally in the last term of this equation d represents the average node degree (i.e. average number of neighboring nodes) in the network to represent the number of messages needed to exchange information in order to establish pair-wise keys with their neighboring nodes.

The communication overhead for SBK [16] can be determined similarly and is shown below in Equation 5.

$$s(B + \lambda_2) + \bar{h}s\lambda_2 + d \tag{5}$$

In this equation, s represents the total number of service nodes elected by the SBK scheme. Service nodes provide similar functions as cluster heads in our scheme. The remaining variables remain the same as in our scheme.

The communication overhead is now compared between our scheme and SBK scheme in [16]. Due to the longer transmission range of a H-sensor our scheme produces significantly less broadcast messages than the SBK scheme. For example, assume 10,000 sensors have been deployed uniformly over an area of 1,000 by 1,000 meters. Suppose L-sensors with a transmission range of 20 meters and the H-sensors have a transmission range of five times that of the L-sensors or 120 meters. In our scheme, 23 H-sensors would need to be deployed to distribute keying information in this area. However, in the SBK scheme, which is a scheme designed for homogeneous WSNs, 796 L-sensors would need to be elected to distribute keying information for the same area. This shows that the value for H is much smaller in Equation 4 than the value for s shown in Equation 5. Thus our scheme uses significantly less broadcast messages than the SBK scheme in terms of total communication overhead.

2.7.2. Storage and Computational Overhead

In this subsection we investigate the storage and computational overhead required by our scheme. We also compare our scheme with three other previously proposed schemes, SBK [16], LEAP [6], and the E-G scheme [4]. In our scheme each node may receive a maximum of one piece of keying information from each cluster it joins. Furthermore each H-sensor, H, may distribute at most $\lambda_1 - 1$ pieces of keying information to nodes within transmission range and each LPCH, L, may distribute λ_2 pieces of keying information to nodes within range. Based on the storage requirements described previously in [16] the total storage upper bound for each node can be found where H is the total H-sensors in the network and L is the total LPCHs elected. In the equations below, Equation 6 shows the upper bound memory requirements for a each node when using the polynomial based key space and Equation 7 shows the upper bound requirements when the matrix based key space model is used.

$$H(n+1)\log r + L(n+1)\log r \tag{6}$$

$$H(n+2)\log r + L(n+2)\log r \tag{7}$$

In these equations n denotes the degree of the polynomial or the dimensions of the matrix being used and r is the large prime preloaded onto the nodes to define the finite field to be used when generating key spaces. We can compute the total upper bound storage required by summing the storage space needed for each node in the network using the Equations 6 and 7 above. We expect the number of LPCH created after deployment to be fairly minimal since most L-sensors should be within range of H-sensor clusters. Furthermore, due to the additional computational power of H-sensors, H-sensors can create larger clusters than L-sensors which also increases the resiliency to node capture attack which will be discussed in Subsection 2.7.3.

Below in Table 2 we compare the number of pair-wise keys for three existing schemes previously listed: the SBK scheme [16], the LEAP scheme [6], and the E-G scheme [4]. Since our scheme and the SBK scheme achieve perfect connectivity between neighboring nodes the total number of pair-wise keys shared stored in each node is the average node density of the network, d. The LEAP scheme requires three additional keys to be stored as shown in Table 2. In the E-G scheme, S is the number of keys preloaded onto each sensor prior to deployment.

Table 2. Keys Stored

Our Scheme	SBK	E-G	LEAP
d	d	s	$d+3$

Below in Table 3 we look at an example for pair-wise keys needing to be stored for 90% or greater connectivity. SBK, LEAP and our scheme are all able to achieve 100% connectivity between nodes and their neighbors. We define connectivity as the ability to establish pair-wise keys between two neighboring nodes. In this example for an average node density of 20 nodes SBK and our scheme would only require 20 keys to be stored to achieve connectivity, while LEAP requires an additional three keys and E-G requires the most keys needing to be preloaded at 150 keys. Furthermore, E-G scheme achieves approximately 90% connectivity and requires a substantial amount of memory to store these keys. From this example we show that we are able to achieve at least as efficient key storage as SBK and slightly better storage than LEAP but significant storage improvements from the E-G scheme.

Table 3. Keys Stored with $\geq$ 90% Connectivity

Scheme	Ours	SBK	E-G	LEAP
Keys stored	20	20	150	23
Keys established	100%	100%	90%	100%

We now take a look at computational overhead. As discussed in [16] it requires $n+1$ modular multiplications for either key space models to establish pair-wise keys. When a seed for columns in G are used however each sensor receiving information would need to rebuild the column which would require an additional $n-1$ modular multiplications for each cluster it joins. This is the same amount of computations as required in the SBK scheme [16].

2.7.3. Resilience

In this subsection we discuss the resiliency for our scheme in regards to node capture attacks and the disclosure of pair-wise keys to the attacker. As previously mentioned in Section 2.3, these key space models have a property known as n-collusion resistance and shown in [9], [10], and [16]. This means that n or more nodes in each cluster must be compromised before the entire key space is compromised and all of the pair-wise keys generated by it can be calculated. Thus for better resilience against an attacker, when the value for n is equal to or greater than the cluster size used in our scheme the key space remains perfectly secure even if the entire cluster is compromised by a physical attacker. This means that if one or more nodes in a cluster have not been compromised then any pair-wise keys they contain cannot be determined by the information gathered from the compromised nodes.

Assume that we have an n-degree polynomial which has been used for generating a polynomial based key space, and we set our cluster size to N nodes. Let us also assume $N > n$. Then when compromised nodes is greater than or equal to $n + 1$, the coefficients generated for the polynomial can be calculated by the attacker and the pair-wise keys which remain in the $N - (n + 1)$ uncompromised nodes in the network can be computed by using the uncompromised node IDs by solving for a solution to the set of polynomials which have been generated by the key space. Thus one can achieve perfect resilience against node capture attack by setting n to be greater than or equal to the cluster size specified in our scheme. Similarly this can also be shown for the matrix based key space model.

2.7.4. Simulation Design and Setup

In this subsection we discuss the simulation software we designed to simulate our scheme, the SBK scheme [16], and the E-G scheme [4]. Our simulation software applications were written using the C# programming language with Visual Studio 2008 and .NET 3.5. Each of these applications are command line console applications and simulations were performed on machines running Ubuntu 8.04 LTS using Mono 2 for .NET applications. We designed three console applications which take the various parameters needed to simulate each unmodified scheme. Additionally a fourth simulator was written to simulate a modified version of SBK with the addition of H-sensors into the network running the SBK scheme which does not take advantage of these types of sensors. For our simulation purposes we used the polynomial based key space model for both the SBK scheme and our scheme.

Each simulator creates a user defined Cartesian plane grid space to be used for deploying sensor nodes for the network. The user specifies number of nodes to deploy in this grid space which are then deployed by randomly generating (x, y) coordinates for each node. This type of deployment method would be similar to an aircraft drop which does not consider deployment knowledge exists prior to distributing sensor nodes into a geographical region. Each scheme also requires input to specify the number of sensors to deploy. For the unmodified SBK and E-G scheme this is only for L-sensors while our scheme and the SBK with H-sensor simulator requires both the number of L-sensors and the number of H-sensors to be deployed. For our simulations we specified 10,000 L-sensors to be deployed in the region to simulate a dense WSN. In the SBK scheme which additionally deployed H-sensors we specified 150 H-sensors to be deployed in addition to the 10,000 L-sensors. We specified 150 H-sensors for our modified version of SBK since 150 H-sensors provides near perfect network connectivity in our scheme based on simulation results with the least number of H-sensors needing to be deployed.

In each of the simulated schemes, the transmission range needs to be specified for L-sensors and H-sensors. In our study we specified our transmission range for L-sensors to be 20 and H-sensors with a transmission range of 120. In the SBK scheme we specify the degree of the polynomial to use to be 100 in both of the two simulators for this scheme. Furthermore, we specify the degree of the polynomial for H-sensors in our scheme to 500 and the degree used by the L-sensors to 100. This larger degree polynomial is used to create larger clusters than clusters formed by L-sensors and provide keying information to more nodes within the much larger transmission range of the H-sensors. Lastly, for our scheme and the SBK scheme we set the probability of an L-sensor to be randomly elected to provide keying information to 10% in each test.

We varied one of the parameters for each scheme which would increase or decrease the network connectivity in order to compare these schemes to ours. For the E-G scheme we varied the number of pair-wise keys preloaded onto each of the distributed nodes in the network out of a key pool size of 10,000 keys. For both the unmodified and modified simulators for the SBK scheme we varied the time to live (TTL) for messages being sent. By increasing the TTL the nodes in the SBK scheme are able to extend the range messages are able to travel outside of transmission range of the L-sensors. These messages are rebroadcast from their origin while decreasing the TTL until the TTL is equal to zero.

For our scheme we varied the number of H-sensors being deployed in the area. By increasing the number of H-sensors we deploy we can increase the overall coverage in the network. We can then use these results for comparison with SBK since by eliminating H-sensors from our scheme we are using a method similar to SBK but we are limited to one-hop transmissions and are not able to rebroadcast the messages from cluster heads. Table 4 below shows the parameters used for each scheme simulation run for the E-G scheme. Table 5 lists the parameters used by the SBK and SBK with H-sensor schemes. In Table 6 we list the parameters used for our scheme during simulation. Each simulation test was run forty times. The results were then compiled by computing the mean, standard deviations and confidence scores for the forty simulation test runs.

Table 4. E-G Simulation Parameters

Test #	1	2	3	4	5	6	7	8	9	10	11	12
Keys stored	25	50	75	100	125	150	175	200	225	250	275	300

Table 5. SBK Simulation Parameters (both versions)

Test #	1	2	3	4	5	6
TTL (hops)	1	2	3	4	5	6

Table 6. PDKM Simulation Parameters

Test #	1	2	3	4	5	6	7	8	9	10	11	12
H-sensors	25	50	75	100	125	150	175	200	225	250	275	300

2.7.4.1. E-G Simulator Process

When the E-G scheme simulator starts the first step which occurs is the simulator randomly generates the number of keys to use for the key pool. Once these keys have been generated the simulator creates sensor objects and randomly selects the number of keys to store in a list on each sensor from the key pool. Once these keys have been selected by each node the sensor objects are randomly given their grid locations by randomly selecting the (x, y) coordinates within the specified grid size. After all nodes have been given coordinates within the grid space one-hop neighbors are determined for each node and stored in a list of neighboring nodes in each sensor object. Neighboring nodes are found by finding the Euclidean distances between the target sensor object and all other sensor nodes created by the simulator. After all of the one-hop neighbors are found for a target sensor object and stored into a list of IDs the simulator then checks the lists of keys stored on the target sensor and its neighbors for matching pair-wise keys from their selected keys. If a target node and its neighbor share at least one key in common these nodes are marked as a secured connection and their IDs are stored by each other in a separate key value pair list containing the ID of their neighbor and the first key found which they share in common. Finally once all nodes deployed in the network have determined their neighboring nodes which share keys in common with them the network connectivity is calculated based on the number of network wide possible links found between sensor objects and their one-hop neighbors and the number of links between nodes which contain a key in common.

2.7.4.2. SBK Simulator Process

The first step of the SBK simulator creates sensor objects containing the necessary parameters needed by the algorithm and then randomly selects grid coordinates for their deployment locations. Once all nodes have been deployed in the simulator, the next step of the simulator randomly chooses sensor objects to attempt to elect themselves as service nodes. If a node fails to elect itself another randomly chosen sensor object is selected and allowed to run the SBK algorithm to attempt to elect itself. This random selection of sensor objects continues until a node has elected itself as a service node. Once a node has been elected this sensor object runs the key generation algorithm by randomly choosing λ_2 coefficients to use for the polynomial. Furthermore, its ID is stored into a list by the simulator of elected service nodes which is used to determine the number of nodes elected during the test. A separate list is used for elected H-sensors in the modified version of the

SBK scheme simulator. After λ_2 coefficients have been randomly generated the sensor object begins to locate its m-hop neighbors using a recursive function in the simulator until all m-hop neighbors are located based on their Euclidean distances. These steps simulate the process of a node in the network electing itself and broadcasting it has keying information available to distribute before another nearby neighbor has done so. Once these neighbors have been determined the simulator calculates the product of the coefficients with the IDs of the neighboring nodes over the finite field specified by r and stores these products as a list on each of the neighboring nodes until at most n nodes have keying information. This keying information is stored in a key value pair list containing the ID of the service node and the list of products from the coefficients. This process of randomly selecting a node in the network and giving it a chance to elect itself continues until all nodes either have been elected or contain keying information from a service node.

Once distribution of keying information has completed the simulator finds all one-hop neighbors for each node which has not been elected based on their Euclidean distance. After each node has located its one-hop neighbors the simulator checks the key value pair containing the keying information for matching service node IDs between a target node and its one-hop neighbors to see if they belong to the same key space. If a neighboring node and the target node share keying information from the same service node a pair-wise key is calculated over the finite field and stored in a key value pair list along with the neighbors ID. After all nodes have calculated pair-wise keys with neighbors which contain the same key space the network connectivity is calculated using the same process as the E-G simulator.

2.7.4.3. PDKM Simulator Process

In the PDKM simulator the first step of the simulator creates sensor objects containing the specified parameters needed. Once this step has completed each node is given randomly generated grid coordinates for deployment. After all nodes have been deployed into the grid space the simulator first initializes each of the H-sensors deployed and allows the H-sensor objects to generate the random coefficients of the polynomial for keying information. Once the H-sensor objects have generated keying information they locate their one-hop neighbors based on the Euclidean distance between itself and the remaining nodes in the network. Nodes which have a distance less than or equal to the transmission range of the H-sensors are then given a list of keying information by taking the product of the neighboring node's ID and the randomly generated coefficients in the key space until at most $\lambda_1 - 1$ neighboring nodes are given keying information. We store one piece of keying information generated by the H-sensor in a key value pair list so the H-sensor is able to establish pair-wise keys with its neighbors within transmission range after bootstrapping. By generating and distributing keying information on the H-sensors first we simulate the time period between time zero and half of the maximum time, or $t_0 \leq w \leq t_w/2$ allowed for bootstrapping the network.

After each of the H-sensors have generated and distributed their keying information the remaining L-sensors which do not contain keying information are randomly selected and given a chance to elect themselves as an LPCH. If a node is successful electing itself it generates keying information for an λ_2-degree polynomial and distributes the keying information to its one-hop neighbors in the same manner as the H-sensors with the exception that the LPCH does not store keying information for itself. This simulates that this node will die out shortly after bootstrapping and it will not attempt to establish pair-wise keys with its neighboring nodes. Additionally the elected L-sensor's ID is stored in a list which keeps a record of the number of LPCHs in the network.

Once every node in the network either contains keying information or has been elected one-hop neighbors establish pair-wise keys with each other if they contain keying information from the same cluster head by computing the key using their neighbors ID. After each node has completed creating keys network connectivity is calculated by the simulator and the number of elected L-sensors is recorded from the test.

2.7.5. Simulation Results

In this subsection we review the results from our simulations. Each scheme was simulated forty times for each test listed in Tables 4, 5 and 6. We first examine the network connectivity between our scheme and the E-G and SBK schemes. We define network connectivity as the average percentage of nodes which shares a pair-wise key with their neighboring nodes allowing for secure communication to occur. Figure 5 below shows the results from the E-G scheme simulation tests and the standard deviations from the results. Figure 6 shows the results from the SBK simulation for network connectivity and the results standard deviations for the simulations with and without H-sensors included. In Figure 7 we present the results from the simulation of our scheme on a HSN. These are the means of the results from 40 simulation runs in each test.

2.7.5.1. E-G Network Connectivity

In the results shown in Figure 5 we see that approximately 90% connectivity is achieved when 150 keys are selected for each node out of a pool of 10,000 keys. However, in order to achieve near perfect network connectivity it requires almost 250 keys to be preloaded onto each node. Due the memory limitations of L-sensors however this would be infeasible to do in a large network. For instance, for the network to support 90% connectivity using 128-bit keys each sensor would require 2.344K bytes of storage. In order to have near perfect connectivity this scheme requires 250 keys to be preloaded. This would require 3.906K bytes of memory storage in order to store all of the necessary keys on each sensor. The standard deviation for these means are shown below in Figure 6.

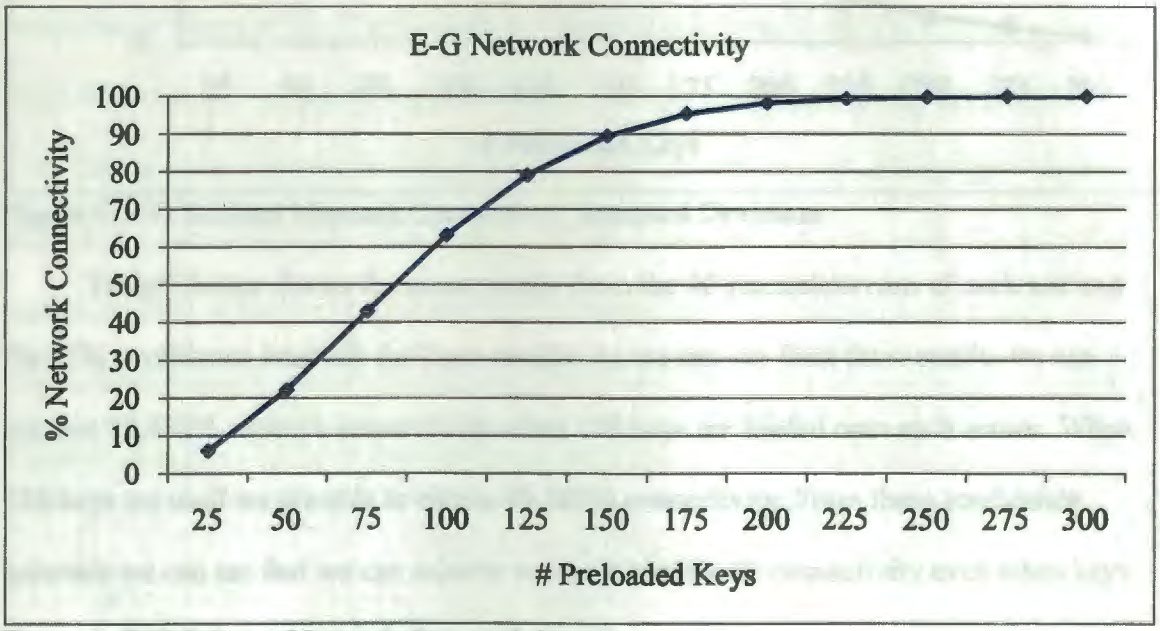

Figure 5. E-G Scheme Network Connectivity

The benefit of using such a scheme however is its simplicity. This scheme does not require complex algorithms to be loaded onto the sensors to create the keys. Since this scheme does not require complex algorithms to run after the nodes are deployed the network can be established using very little power due to low computational overhead. Furthermore, communication overhead can be reduced by having each node use broadcast messages in its area to announce which keys it contains instead of unicast messages to each neighbor.

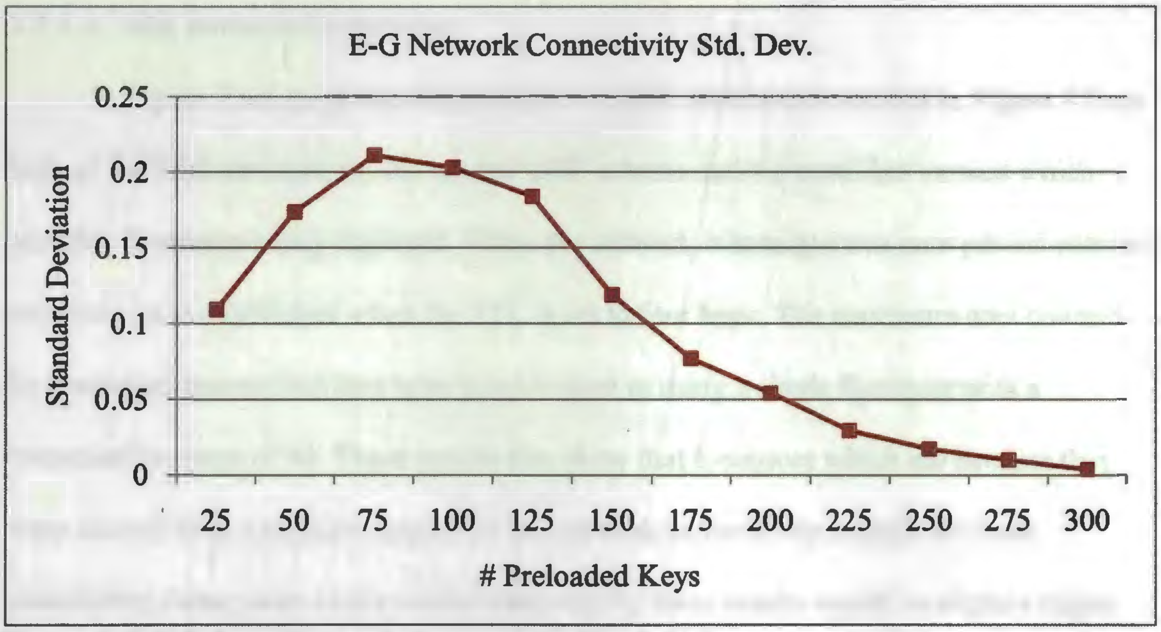

Figure 6. E-G Scheme Network Connectivity Standard Deviation

Table 7 below shows the mean results from the 40 simulation runs of each test and the 95% confidence intervals for these results. As we can see from these results we can achieve 89.458% network connectivity when 150 keys are loaded onto each sensor. When 250 keys are used we are able to obtain 99.805% connectivity. From these confidence intervals we can see that we can achieve acceptable network connectivity even when keys are randomly selected from a very large pool of keys.

Table 7. E-G Scheme Network Connectivity Means and 95% Confidence Intervals

Keys Stored	Mean	Lower	Upper
25	6.02	5.98	6.05
50	22.11	22.06	22.17
75	42.98	42.91	43.04
100	63.18	63.11	63.24
125	79.00	78.94	79.06
150	89.46	89.42	89.50
175	95.34	95.31	95.36
200	98.17	98.15	98.18
225	99.36	99.35	99.37
250	99.81	99.80	99.81
275	99.95	99.95	99.95
300	99.99	99.99	99.99

2.7.5.2. SBK Network Connectivity

In Figure 7 we show the mean results and their standard deviations in Figure 8 from both of the SBK simulations; the normal SBK scheme and the modified version which includes H-sensors being deployed. When the network is homogenous near perfect network connectivity is established when the TTL is set to four hops. The maximum area covered by a message transmitted four hops is equivalent to using a single H-sensor with a transmission range of 80. These results also show that L-sensors which die because they were elected have a negative impact on the network connectivity overall. Without considering these nodes in the overall connectivity these results would be slightly higher than shown here. However, when a node is elected and allowed to die prematurely in the network a hole is formed in the center of the cluster where the service node was located. This creates breaks in communication links between other nodes since messages must be transmitted around this area since no L-sensors are present in these locations anymore.

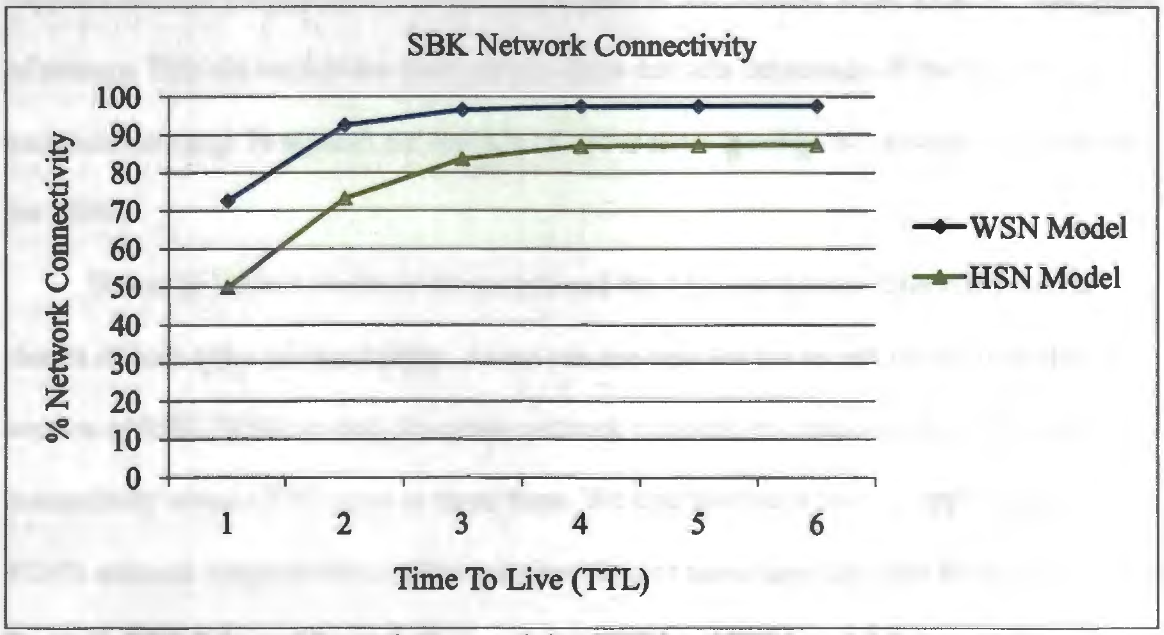

Figure 7. SBK Scheme Network Connectivity (WSN and HSN models)

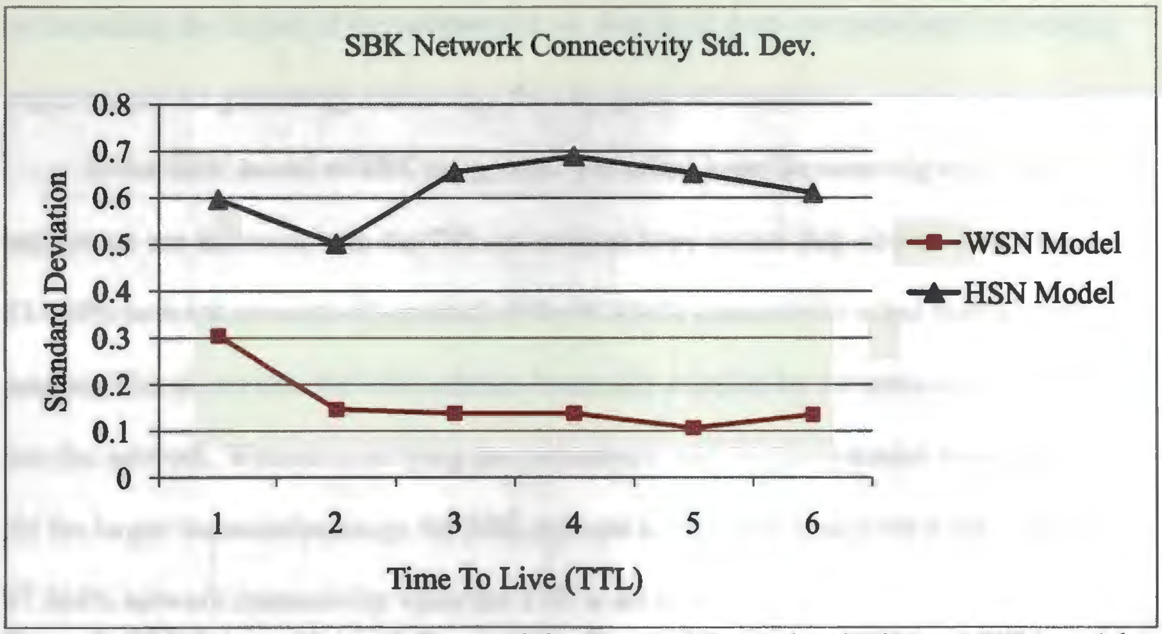

Figure 8. SBK Scheme Network Connectivity Standard Deviation (WSN and HSN models)

By adding H-sensors to the network and not adjusting the algorithm to take advantage of the longer transmission ranges, the overall network is severely crippled. This is due to many more L-sensors being elected compared to the number of H-sensors. This happens because the probability of electing a node in the network is the same for both types of sensors. This shows that the SBK scheme does not take advantage of the longer transmission range H-sensors are capable of and is not a feasible key management scheme for HSNs.

Below in Table 8 we show the means and the 95% confidence intervals from the results of both types of simulations. As we can see here for the results of the unmodified version of SBK (WSN model), the mean network connectivity achieves 96.456% network connectivity when a TTL is set to three hops. We also see that a limit of approximately 97.4% network connectivity is achieved when four or more hops are used for the TTL. This could be potentially increased by increasing the degree of the polynomial used. However, by increasing the degree of the polynomial we also incur extra computational and storage requirements for generating and storing the key space information.

In the HSN model of SBK being deployed which uses the same algorithm as the L-sensors we see that even with the TTL set to three hops we are only able to achieve 83.605% network connectivity instead of the 96.456% connectivity when they are not present. This shows that the SBK scheme is actually crippled by the inclusion of H-sensors into the network. Without modifying the parameters used by the H-sensor to compensate for the larger transmission range the SBK scheme is only able to achieve a maximum of 87.264% network connectivity when the TTL is set to six.

There are two possible modifications to the parameters which could increase the network connectivity in this case. The first is to increase the probability of election on the H-sensors to be much higher than that loaded onto the L-sensors. This would allow for the H-sensors to have a higher chance of being elected first. A second modification would need to be made to the parameters loaded onto the H-sensors to compensate for their higher probability of being elected. Since they are able to cover a larger area which would include more L-sensors within transmission range the degree of the polynomial used must also be increased. If this was not increased the network would still be crippled since the elected H-sensors would not be able to provide keying information to all of the L-sensors within its transmission range. This is due to the fact that the cluster formed by the service nodes is at most as large as the degree of the polynomial being used while still remaining perfectly secure against node capture attacks.

If an elected H-sensor is not able to provide keying information to all of the sensors within range of the multi-hop broadcast transmission then nodes which are not able to receive keying information will not be able to establish secure communication with their neighbors. This is due to the fact that the SBK scheme does not allow nodes which hear a broadcast message announcing keying information to re-elect themselves if they are not able to receive keying information from a broadcasting node. When this occurs these nodes are effectively useless to the operation of the entire network as they are not able to establish pair-wise keys with their neighbors.

Table 8. SBK Scheme Network Connectivity Means and 95% Confidence Intervals

TTL	WSN Mean	WSN Lower	WSN Upper	HSN Mean	HSN Lower	HSN Upper
1	72.69	72.59	72.78	49.95	49.76	50.13
2	92.43	92.39	92.48	73.27	73.11	73.42
3	96.46	96.41	96.50	83.61	83.40	83.81
4	97.42	97.38	97.46	87.02	86.80	87.23
5	97.48	97.44	97.51	87.19	86.99	87.39
6	97.46	97.42	97.50	87.26	87.08	87.45

2.7.5.3. PDKM Scheme Network Connectivity

In Figure 9 below we show the mean results from 40 runs of each test of our scheme on a HSN. We always achieved higher than 90% connectivity or better with near perfect connectivity occurring when 150 H-sensors are deployed with the 10,000 L-sensors in the network. For instance when as few as 25 H-sensors are randomly deployed in the network we are able to achieve a mean of 91.043% network connectivity. Additionally when only 100 H-sensors or more are deployed in the network our scheme achieves over 99% network connectivity. In comparison with the previous schemes we show we have a much higher probability of having a fully connected network by taking advantage of these nodes to manage key generation and distribution. Figure 10 below shows the standard deviations for these results.

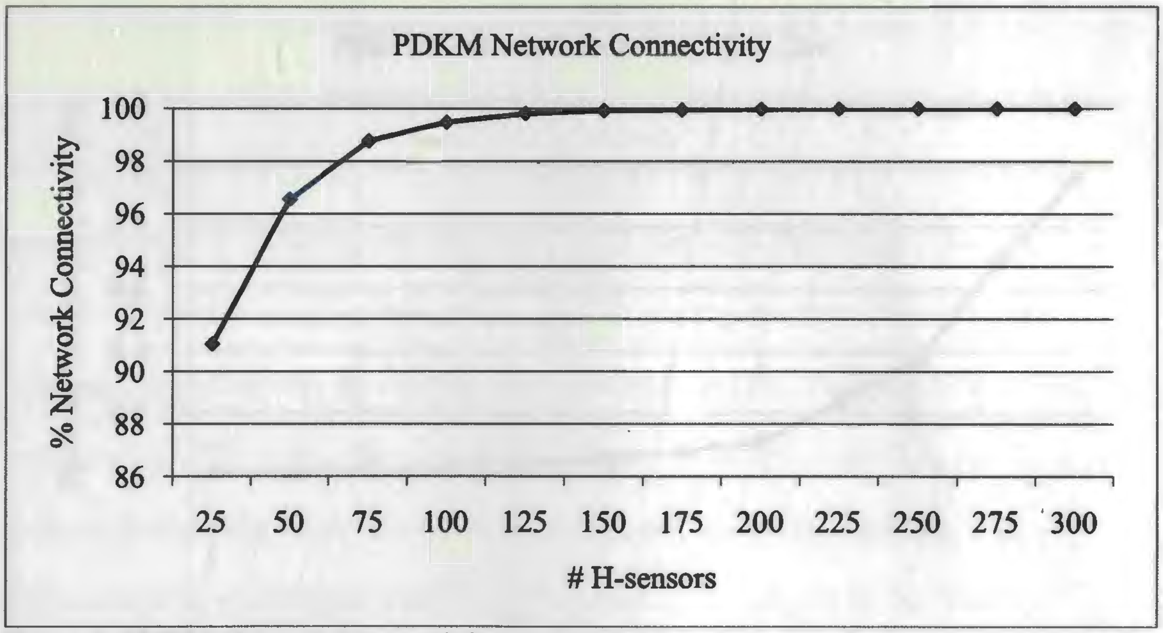

Figure 9. PDKM Network Connectivity

Table 9 below shows the means from the results of our scheme in simulation along with the 95% confidence intervals. As we can see when 150 H-sensors are deployed in a network with 10,000 L-sensors our scheme achieves 99.9% network connectivity. Furthermore, in comparison with the SBK scheme we can see that with as few as 75 H-sensors being deployed we are able to achieve higher network connectivity than the SBK scheme in all of the tests ran with a mean of 98.774% network connectivity. This increased network connectivity in our scheme is due to the fact that H-sensors in the network do not rely on any probability of being elected. Additionally due to the increased capabilities of the H-sensors they are configured to form much larger clusters than the L-sensors being deployed. This increased cluster size also increases the resiliency of the network against node capture attacks since many more nodes would need to be compromised.

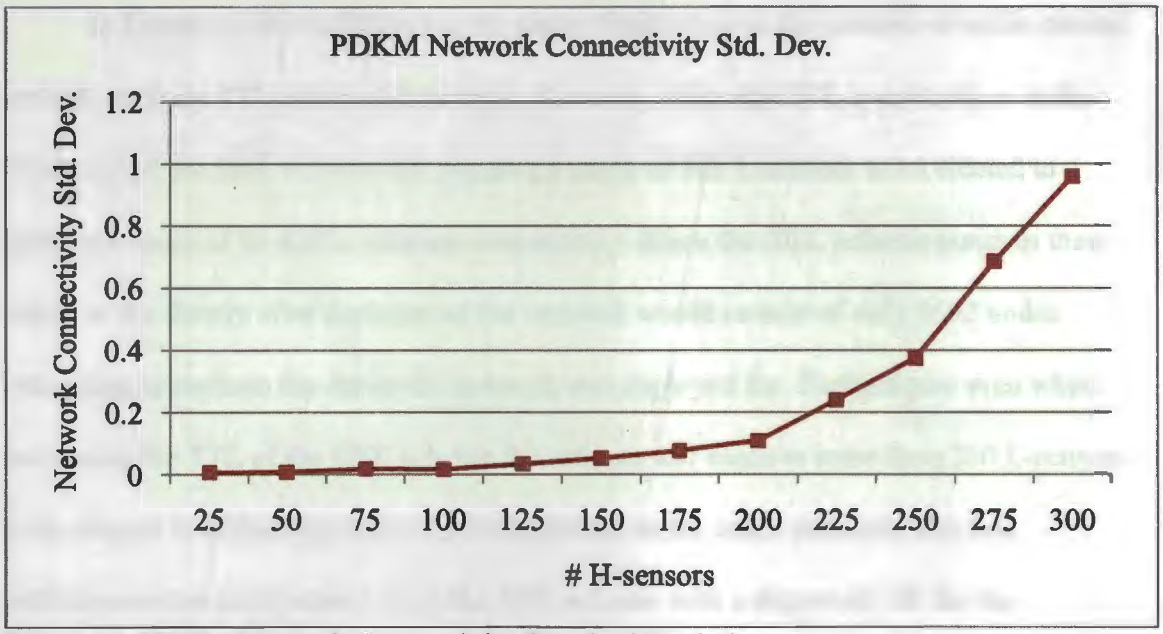

Figure 10. PDKM Network Connectivity Standard Deviation

Table 9. PDKM Network Connectivity Means and 95% Confidence Intervals

H-Sensors	Mean	Lower	Upper
25	91.04	90.75	91.34
50	96.56	96.35	96.78
75	98.77	98.66	98.89
100	99.49	99.41	99.56
125	99.81	99.78	99.85
150	99.92	99.90	99.95
175	99.95	99.94	99.97
200	99.98	99.97	99.99
225	99.99	99.99	100.00
250	99.99	99.99	100.00
275	100.00	100.00	100.00
300	100.00	100.00	100.00

2.7.5.4. SBK Scheme Elected Sensors

Next we take a look at the number of L-sensors elected in our scheme versus the SBK schemes and also the number of H-sensors elected when adding H-sensors to the SBK scheme. Figure 11 shows the results from the SBK scheme for elected nodes for both simulation types. The standard deviations for these results is shown in Figure 12.

In Figure 11 shown below we see a significant drop in the number of nodes elected in SBK until the TTL is set to three hops. However, when the TTL is set to three in the WSN model the SBK scheme still requires a mean of 308 L-sensors to be elected to achieve a mean of 96.456% network connectivity. Since the SBK scheme assumes these nodes to die shortly after deployment the network would consist of only 9692 nodes remaining to perform the duties the network was deployed for. Furthermore even when increasing the TTL of the SBK scheme the scheme still requires more than 200 L-sensors to be elected in order to provide key management to the entire network. The best performance we could expect from the SBK scheme with a degree of 100 for the polynomial is when the TTL is set to five or higher. When the TTL is set to five the SBK scheme needs to only elect 218 nodes in the network to provide key management. In comparison with our scheme we can achieve better network connectivity by including only 75 H-sensors into the network and near perfect connectivity when 150 H-sensors are deployed.

Since there is a much denser population of L-sensors being deployed compared to H-sensors in the SBK scheme H-sensors do not assist in key management but actually cripples the overall network connectivity. This is because the probability of one of the L-sensors successfully electing itself is approximately 98.5% more likely than the chance that one of the H-sensors is elected with the addition of 150 H-sensors in the network. As these results show compared to network connectivity the increased TTL allows for fewer L-sensors needing to be elected in both network models. However, even when the TTL is configured for six hops it still requires a mean of 357 nodes to be elected when H-sensors were present in the network.

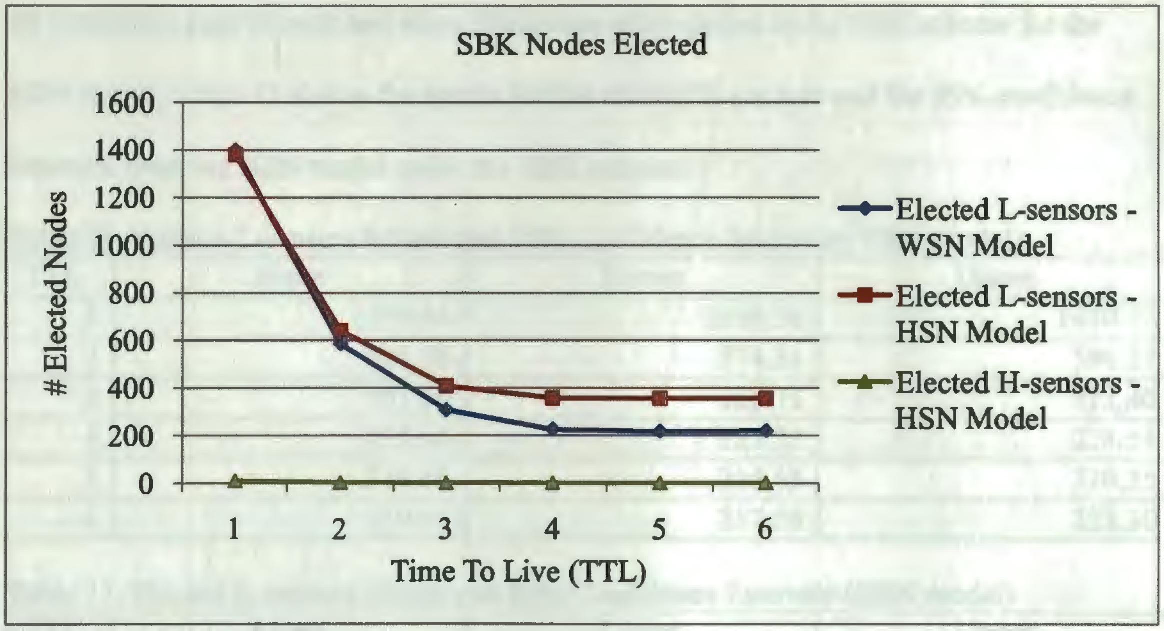

Figure 11. SBK Nodes Elected

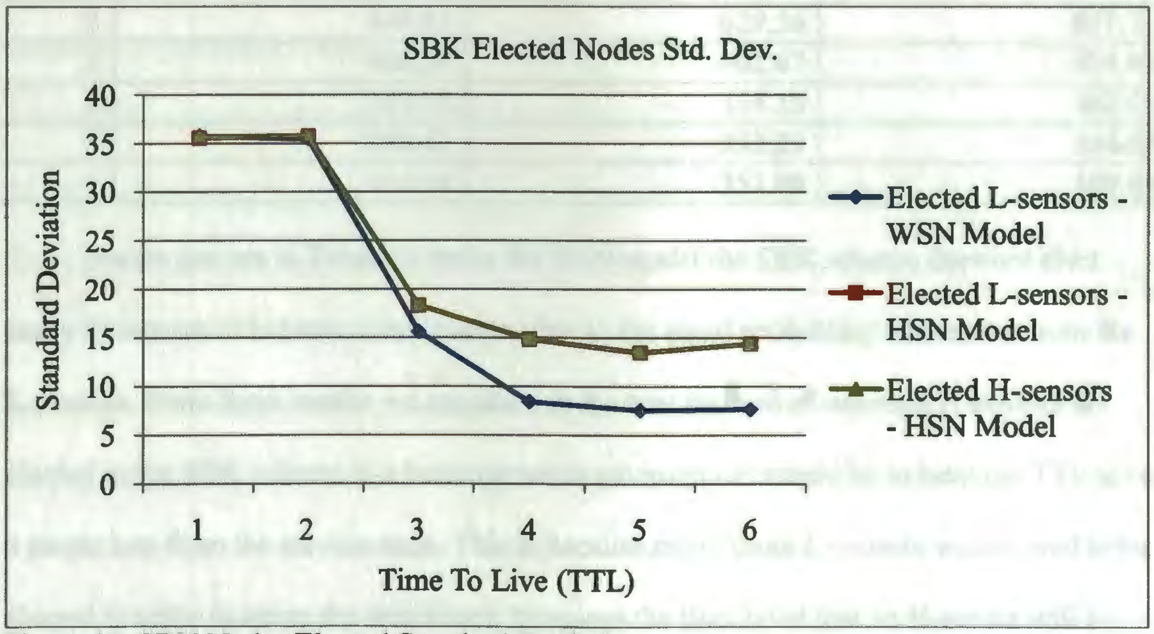

Figure 12. SBK Nodes Elected Standard Deviation

Table 10 shows the means of the elected L-sensors in the WSN model, which does not contain any H-sensors, for each test of the SBK scheme under simulation. This table also shows the 95% confidence intervals from each of the 40 simulation runs of each test. Table 11 shows the means of the elected L-sensors and 95% confidence intervals from the

40 simulation runs of each test when H-sensors are included in the SBK scheme for the HSN model. Table 12 shows the means for the elected H-sensors and the 95% confidence intervals from the HSN model under the SBK scheme.

Table 10. Elected L-sensors Means and 95% Confidence Intervals (WSN model)

TTL	Mean	Lower	Upper
1	1399.64	1388.56	1410.72
2	585.29	574.31	596.27
3	307.56	302.71	312.40
4	225.94	223.32	228.55
5	218.00	215.65	220.35
6	219.93	217.55	222.30

Table 11. Elected L-sensors Means and 95% Confidence Intervals (HSN model)

TTL	Mean	Lower	Upper
1	1379.30	1368.29	1390.31
2	640.63	629.56	651.71
3	408.34	402.67	414.01
4	357.93	353.33	362.53
5	356.47	352.29	360.65
6	356.44	351.98	360.89

As we can see in Table 12 under the HSN model the SBK scheme does not elect many H-sensors to become service nodes due to the equal probability of election from the L-sensors. From these results we can see that the best method of ensuring H-sensors are elected in the SBK scheme in a heterogeneous environment would be to keep the TTL to be a single hop from the service node. This is because many more L-sensors would need to be elected in order to cover the area which increases the likelihood that an H-sensor will be elected instead. As stated previously two parameter changes for H-sensors being deployed in the SBK scheme can be done to increase the number of H-sensors being elected. The first would be to set a higher probability of election on the H-sensors while also increasing the degree of the polynomial used by the H-sensors. This would allow for a larger cluster to be formed around more H-sensors in the network instead of the L-sensors being deployed.

Table 12. Elected H-sensors Means and 95% Confidence Intervals (HSN model)

TTL	Mean	Lower	Upper
1	10.80	0.00	21.86
2	2.69	0.00	13.79
3	1.12	0.00	6.80
4	1.00	0.00	5.62
5	1.33	0.00	5.52
6	1.01	0.00	5.47

2.7.5.5. PDKM Scheme Elected L-sensors

In Figure 13 we show the mean results of elected L-sensors from the simulation tests of our scheme. The standard deviations for these results is shown in Figure 14. We can see that many fewer L-sensors need to be elected even when only 25 H-sensors have been included when using our algorithms. This decreased number of elected nodes allows for better overall network connectivity since the network continues to stay dense in areas where SBK service nodes would have perished after bootstrapping has been completed. When we use our scheme in an HSN 543 L-sensors are elected when only 25 H-sensors are deployed compared to approximately 1400 needing to be elected when considering a one hop TTL in the WSN model of SBK and 1380 needed in the HSN model. SBK requires a three hop TTL in order to decrease below the number of L-sensors needed for key management in our scheme. Furthermore, we almost completely eliminate the need for L-sensors to be elected after a sufficient number of H-sensors have been deployed. As shown below in Figure 13, when we include 150 or more H-sensors into the network to handle the key management for the network less than nine L-sensors must be elected.

Table 13 shows the means of elected L-sensors from the results of each test of our scheme under simulation and their 95% confidence intervals. In comparison with the SBK scheme we can see that when as few as 50 H-sensors are included our scheme needs to only elect a few more L-sensors than the best results achieved from the SBK scheme. For instance, when we deploy 50 H-sensors we have a mean election of 244 L-sensors in the network compared to 218 L-sensors elected in the WSN model of SBK. Furthermore when H-sensors are included in an unmodified version of SBK the SBK scheme requires an average of 113 more L-sensors to be elected in the best case scenario than our scheme.

We also can see that when there is a sufficient number of H-sensors deployed in our scheme we virtually eliminate the need of electing L-sensors. For instance when 150 H-sensors are deployed only nine L-sensors are needed to be elected in our scheme. This is further improved as we increase the number of H-sensors. As we can see below when we include 225 or more H-sensors in the network we only need to elect one or two L-sensors in the entire network to handle key management for areas not covered by the H-sensors.

Since fewer nodes are needed to be used for key management in our scheme this also improves the overall network communication overhead since fewer nodes will need to broadcast messages compared to the SBK scheme. For example, under perfect communication, when 150 H-sensors are deployed in our network we need a minimum of 159 broadcast messages to be used to announce key space information is available. However, in a one hop SBK scheme it would require a minimum of 1400 broadcast messages when there is no collisions on the wireless medium in the WSN model and 1380 broadcast messages in the HSN scheme. In the best case scenario of SBK we see that it still requires more broadcast messages needed than in our scheme. For instance in the WSN model of the SBK scheme it requires a minimum of 218 broadcast messages compared to the 159 broadcast messages in our scheme when there are 150 H-sensors present. Only when we deploy 225 H-sensors in our network does it require more broadcast messages than the SBK scheme. When 225 H-sensors are deployed using our scheme we would then require a minimum of approximately 227 broadcast messages under perfect conditions on the wireless medium. Furthermore since transmitting data on the wireless medium requires the most energy to the sensors we have improved the overall energy needs of the network and assist in increasing the lifetime of the network compared to the SBK scheme.

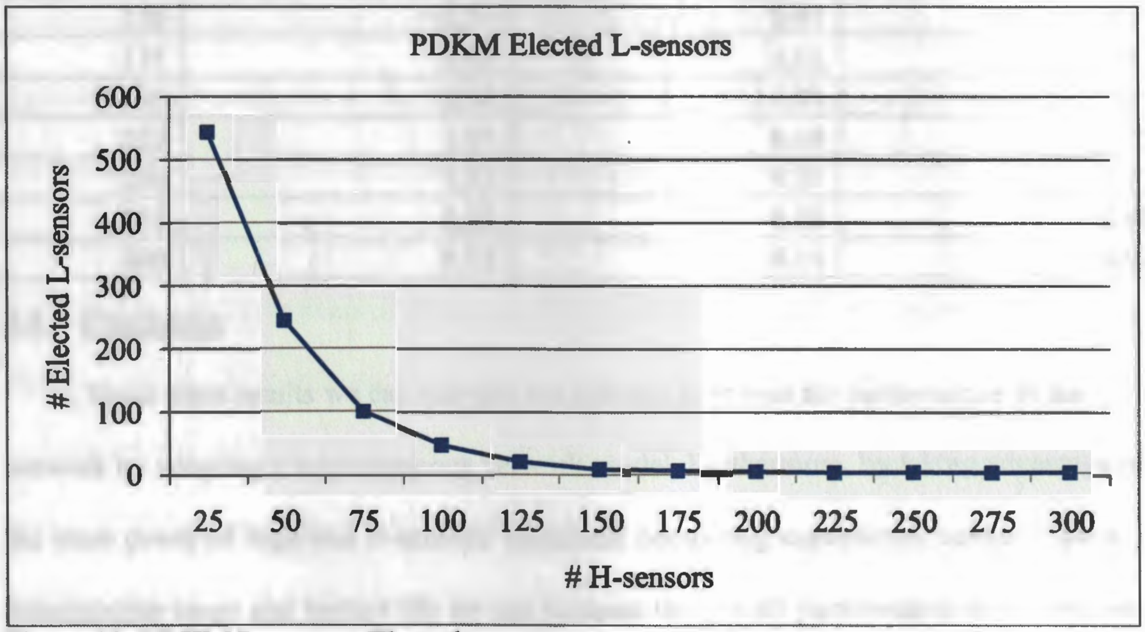

Figure 13. PDKM L-sensors Elected

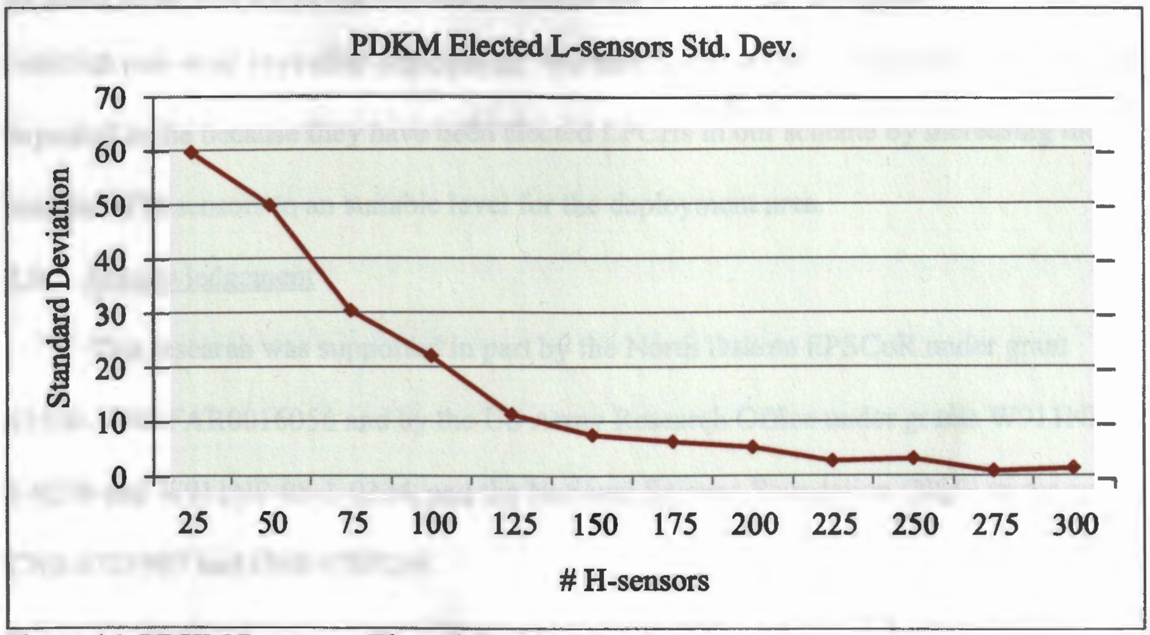

Figure 14. PDKM L-sensors Elected Standard Deviation

Table 13. PDKM L-sensors Elected Means and 95% Confidence Intervals

H-Sensors	Mean	Lower	Upper
25	542.85	524.39	561.31
50	243.97	228.57	259.37
75	99.27	89.89	108.66
100	46.11	39.29	52.92
125	19.93	16.46	23.40
150	8.27	5.97	10.57
175	5.93	4.01	7.85
200	3.55	1.95	5.15
225	1.41	0.60	2.23
250	1.31	0.37	2.25
275	0.23	0.00	0.48
300	0.53	0.14	0.92

2.8. Conclusion

From these results we can see that our scheme improves the performance of the network by adopting a heterogeneous network model. Furthermore, by taking advantage of the more powerful high-end H-sensors' additional computing capabilities, memory space, transmission range and battery life we can increase the overall performance of the network by using these sensors in the network to handle the key management processes needed to establish pair-wise keys after deployment. We also decrease the number of L-sensor nodes expected to die because they have been elected LPCHs in our scheme by increasing the number of H-sensors to an suitable level for the deployment area.

2.9. Acknowledgement

This research was supported in part by the North Dakota EPSCoR under grant 43500-2740-FAR0016056 and by the US Army Research Office under grants W911NF-07-1-0250 and W911NF-08-1-0334, and the National Science Foundation (NSF) under grants CNS-0721907 and CNS-0709268.

CHAPTER 3. CONCLUSION

We have presented a set of algorithms for providing key generation and distribution for heterogeneous sensor networks which improves the overall performance of the network. By taking advantage of H-sensors in the network we can increase the network connectivity of the network since H-sensors should be able to survive the bootstrapping phase of the network. Additionally due to the increased memory space, processing capabilities and transmission range we can form larger clusters around H-sensors which also increases the resiliency of the network from node capture attacks.

In this study we accomplished several objectives and tasks. The first objective was to design a new set of algorithms for key management for a heterogeneous sensor network. To accomplish this objective we have designed an algorithm for key generation for more powerful H-sensors present in the network. We have also presented an algorithm for L-sensors which allows us to take advantage of the more powerful nodes present in the network. This algorithm also includes a secondary function that also allows for a backup method of key management that an H-sensor is not present in the deployment area still requiring key management. Lastly we have presented a method for distributing keying information from the cluster heads to the rest of the network.

The second objective of this study was to analyze our method and to compare our scheme to existing schemes previously proposed. In our analysis of our scheme we show that we have improved the communication overhead by reducing the number of broadcast messages required throughout the network. We have also shown that our scheme has better pair-wise key storage requirements than the E-G scheme and the LEAP scheme while matching the key storage requirements needed by the SBK scheme. Furthermore we have shown through our analysis that as long as the number of nodes compromised is less than the degree of the polynomial or the size of the matrix used for key space generation that our scheme remains perfectly secure against node capture attacks.

The last objective we have accomplished in this study is we have shown through simulation that our scheme allows for better network connectivity than the E-G and SBK scheme. To accomplish this objective we have designed four simulators written in the C# programming language. Along with simulating our scheme on a randomly deployed wireless sensor network, we have also simulated the E-G scheme, the SBK scheme and a modified version of the SBK scheme which included H-sensors being deployed in an HSN. Each of these simulators have been run forty times for each test to obtain our results. We adjusted a parameter for each test which would increase the network connectivity for each of the key management schemes. From our results we were able to show that our scheme achieved a higher level of network connectivity than the E-G and SBK scheme with near perfect network connectivity being obtained.

The first scheme we compared our scheme to was a pre-distribution scheme for homogeneous sensor networks which preloads a set of keys from a larger pool of generated keys onto each node prior to deployment. In our simulation results we were able to achieve better connectivity than the E-G scheme until a very large number of keys are preloaded onto each sensor. At this point the E-G scheme performs as well as our scheme in obtaining near perfect network connectivity.

We have also shown that our scheme performs better than a post-deployment scheme proposed for homogeneous sensor networks. In the SBK scheme a similar method of key management is accomplished. However, from our simulation we have shown that the SBK scheme functions best when only when a homogeneous sensor network is deployed. Furthermore, we have shown that we almost always achieve better network connectivity with our scheme than the SBK scheme. This is due to SBK relying on L-sensors to provide key management to the network which they consider casualties of the network after bootstrapping and would die soon after they have finished with their key management duties.

We have also compared the number of nodes needing to be used for key management processes in our scheme with the SBK scheme. From these results we can see that our scheme requires many fewer nodes to be used for key management than the SBK scheme. We have also shown that the number of broadcast messages being sent over the network can be significantly reduced in our scheme when a heterogeneous sensor network is being deployed. This further increases the lifetime of the network since communication transmission requires a significantly large amount of battery power for sensors in the network. By reducing the number of messages needing to be sent we reduce the amount of power required to handle key management in the network compared to the SBK scheme.

REFERENCES

[1] Atmel Corporation. [Online]. http://www.atmel.com

[2] Crossbow Technology Inc. [Online]. http://www.xbow.com

[3] X. Du, M. Guizani, Y. Xiao, and H. Chen, "Two Tier Secure Routing Protocol for Heterogeneous Sensor Networks," *IEEE Transactions on Wireless Communications*, vol. 6, no. 9, pp. 3395-3401, Sept. 2007.

[4] L. Eschenauer and V. D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," in *9th ACM Conf. Computer and Communications Security*, Washington, DC, 2002.

[5] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," in *IEEE Symposium on Security and Privacy*, Berkeley, CA, 2003.

[6] S. Zhu, S. Setia, and S. Jajodia, "LEAP+: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," *ACM Transactions on Sensor Networks*, vol. 2, pp. 500-528, Nov. 2006.

[7] W. Du, J. Deng, Y. S. Han, S. Chen, and P. Varshney, "A Pairwise Key Predistribution Scheme for Wireless Sensor Networks," in *10th ACM Conf. Computer and Communications Security*, Washington, DC, 2003.

[8] D. Liu and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," in *10th ACM Conf. Computer and Communications Security*, Washington, DC, 2003.

[9] R. Blom, "An Optimal Class of Symmetric Key Generation," in *Conference on the Theory and Applications of Cryptographic Techniques*, Paris, Fr, 1984.

[10] C. Blundo et al., "Perfectly-Secure Key Distribution for Dynamic Conferences," in *12th Annual International Cryptology Conference on Advances in Cryptology*, Santa Barbara, CA, 1992.

[11] W. Du, J. Deng, Y. S. Han, S. Chen, and P. Varshney, "A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge," in *IEEE INFOCOM*, Hong Kong, CN, 2004.

[12] Z. Yu and Y. Guan, "A Key Pre-Distribution Scheme Using Deployment Knowledge for Wireless Sensor Networks," in *4th Int'l Symposium on Information Processing in Sensor Networks*, Los Angeles, CA, 2005, pp. 261-268.

[13] D. Liu, P. Ning, and W. Du, "Group-Based Key Predistribution in Wireless Sensor Networks," in *4th Workshop on Wireless Security*, Cologne, DE, 2005.

[14] L. Zhou, J. Ni, and C. V. Ravishankar, "Efficient Key Establishment for Group-Based Wireless Sensor Networks," in *4th ACM Workshop on Wireless Security*, Cologne, DE, 2005.

[15] H. Chan and A. Perrig, "PIKE: Peer Intermediaries for Key Establishment in Sensor Networks," in *IEEE INFOCOM*, Miami, FL, 2005.

[16] F. Liu, X. Cheng, L. Ma, and K. Xing, "SBK: A Self-configuring Framework for Bootstrapping Keys in Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 7, July 2008.

[17] P. Loree, K. Nygard, and X. Du, "Efficient Post-Deployment Key Establishment Scheme for Heterogeneous Sensor Networks," in *IEEE GLOBECOM*, Honolulu, HI, 2009.

[18] M. Rabin, "Digitalized Signatures and Public-Key Functions as Intractable as Factorization," MIT Laboratory for Computer Science, Cambridge, MA, 1979.

[19] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," in *IEEE INFOCOM*, Hong Kong, CN, 2004.

[20] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock Synchronization for Wireless Sensor Networks: A Survey," *Ad Hoc Networks*, vol. 3, no. 3, pp. 281-323, May 2005.