An Efficient Post-Deployment Key Establishment Scheme for Heterogeneous Sensor Networks

Paul Loree and Kendall Nygard
Department of Computer Science
North Dakota State University
Fargo, ND 58105, USA
Email: {paul.loree, kendall.nygard}@ndsu.edu

Xiaojiang Du
Department of Computer and Information Sciences
Temple University
Philadelphia, PA 19122, USA
Email: dux@temple.edu

Abstract - **Establishing pair-wise keys in wireless sensor networks is a challenging task due to the hostile environment and limited resources in sensor nodes. Many key management schemes have been proposed for sensor networks. However, most existing schemes are designed for homogeneous sensor networks where all of the nodes have the same or similar capabilities. Research has shown that better performance can be achieved by forming a heterogeneous sensor network. In this paper, we present an efficient post-deployment key management scheme designed for heterogeneous sensor networks. The scheme does not assume any prior knowledge about sensor deployment and location. Our scheme takes advantage of a few powerful high-end sensor nodes and achieves efficient and effective key establishment in a sensor network. The performance evaluation shows efficiency and effectiveness measures for our scheme.**

I. Introduction

Securing communications in wireless sensor networks is an essential requirement for many applications, such as operations in military battlefields and in security monitoring. To ensure that the data received from sensors in such situations are valid readings from the area and have not been compromised by an attacker, the messages sent between nodes must be protected. Typical small sensor nodes have limited resources. For instance the Mica2 mote only has a 8 MHz CPU and 4K bytes of EEPROM [1][2]. Another security challenge in wireless sensor networks is that after deployment sensor nodes may be captured and compromised by an adversary. Effective key management is the pre-requisite of other security operations.

Many key management schemes have been proposed for sensor networks. We discuss the related work in Section II. However, most existing schemes are designed for homogeneous sensor networks where all of the nodes are modeled to have similar capabilities. In this paper, we present an efficient key management scheme designed for heterogeneous wireless sensor networks.

II. Related works

In [5], Eschenauer and Gligor first proposed a pre-distribution based key management scheme. The scheme requires considerable memory to store keys. In [4] and [7], the authors enhanced the pre-distribution based key management scheme by allowing pair-wise keys to be created after deployment if the nodes share keying information from the same key space. These schemes will be discussed further in Section III. The work in [3] and [9] describes improved sensor key management by utilizing deployment knowledge. However, these methods require a substantial amount of knowledge prior to sensor deployment. In [10], Zhu et al. designed a key management scheme, called LEAP, which assumes that the network is secure for a short period after sensor deployment, and then a preloaded global key is used to bootstrap key management.

A post-distribution key management scheme called SBK was proposed in [6], and uses the key space models described in [4] and [7]. The SBK scheme preloads each node with a set of parameters to be used after deployment to establish pair-wise keys between neighboring nodes. Each node elects itself with some probability to become a service node and generate a key space. The key space is used by neighboring nodes to establish pair-wise keys. The SBK scheme does not require *a priori* deployment knowledge but still achieves high connectivity between neighboring nodes. However, under SBK, the service nodes run extensive computation and die out after key management.

In this paper, we adopt a Heterogeneous Sensor Network (HSN) model. The HSN model consists of a small number of powerful High-end sensors (H-sensors), and a large number of small Low-end sensors (L-sensors). H-sensors have better capabilities than L-sensors in terms of communication, computation, energy supply, storage space, and other aspects. In our research, we take advantage of the strong capabilities of H-sensors for designing efficient and effective key management. Our scheme also does not require *a priori* knowledge before sensor deployment and allows for the nodes to be randomly distributed in an area.

The rest of the paper is organized as follows: Section III describes the models for generating key spaces. Section IV details our post-deployment key establishment scheme. Section V presents performance evaluation of our scheme and Section VI concludes our paper.

III. Models for generating key spaces

There are two key spaces models that will work with our scheme for establishing pair-wise keys with neighboring nodes. The first model for generating key space is polynomial based [7]. This model requires the usage of a bivariate symmetric *n*-degree polynomial such that

$$f(x, y) = f(y, x) = \sum_{i,j=0}^{n} a_{ij} x^i y^j \qquad (1)$$

over a finite field F_q, where q is a prime that is large enough to be used in a cryptographic key. The pair-wise shared key between two sensor nodes can be obtained by plugging in node IDs (say i and j) in the polynomial, i.e., $f(i, j) = f(j, i)$. Further information regarding generating the key space $f(x, y)$ can be found in [7].

The other model for generating key space is based on the usage of a public symmetric matrix, *G*, of dimensions $(n + 1) \times (n + 1)$ and a private matrix, *D*, with the same dimensions, where *D* is over a finite field *GF(q)*, and q is a large prime. *D* must also be symmetric. These matrices are used to generate another matrix *A*:

$$A = (D \cdot G)^T \qquad (2)$$

The key for a sensor, say *i*, can be obtained by giving sensor *i*, the *i*th row of *A* and the *i*th column of *G*. Neighboring nodes compute a pair-wise key by exchanging the column of *G* they were given to find their key, k_{ij}, which is found in the *i*th row and *j*th column of the matrix, *K*, computed by

$$K = A \cdot G \qquad (3)$$

Additionally, if the matrix G is properly designed, then only a seed is transmitted between nodes instead of the entire column (e.g., using a Vandermonde matrix [4]). However, this requires that the entire column be regenerated by the receiving node. For further information on the generation of matrices *D, G* we refer the readers to [4].

According to the work in [4] and [7], these key spaces have a property in that they are n-collusion resistant in that as long as no more than n sensors are compromised the entire key space remains secure. Additionally, storage space required for each sensor node is approximately $(n + 1)\ log\ q$ for the polynomial key space and $(n + 2)\ log\ q$ for the matrix key space, with a properly developed matrix *G*.

IV. The Post-Deployment Key Management Scheme

This section presents our post-deployment key management scheme. We first describe our network and security model of which we make our assumptions. The following subsection describes our algorithms which will be used by the H-sensors and L-sensors after being deployed to generate and distribute keying information that will be used to establish pairwise keys between neighboring sensor nodes.

A. The Network and Security Model

We consider a large Heterogeneous Sensor Network (HSN). We assume no prior knowledge of sensor deployment (location). For example, sensor nodes may be randomly deployed in the field. The network is comprised of two types of nodes: a large number of tiny, low-cost and low-end sensors (L-sensors), and a few powerful high-end sensors (H-sensors, e.g., PDAs). H-sensors have better computational capability, larger storage space, more energy supply, and longer transmission ranges than L-sensors. All of the nodes are loosely time synchronized prior to deployment and are preloaded with the parameters that are listed in Table I. Nodes are not assumed to be tamper resistant, and if compromised then all the stored information will be released to the attacker.

In our work we consider two security models. In the first model, it is assumed that the network is secure for a short amount of time immediately after node deployment, and during that time period the bootstrapping phase establishes pair-wise keys for sensor nodes. In the second model, we do not assume there is a short secure period immediately after node deployment, and we present a scheme to securely establish pair-wise keys for sensor nodes.

B. The Key Management Scheme

In this subsection, we present the post-deployment key management scheme that we designed for HSNs. There are two algorithms loaded onto each sensor prior to deployment. Additionally, there are two different algorithms used for key space generation, one of which will be loaded onto H-sensors and another onto L-sensors. There are also various parameters loaded onto the sensors as shown in Table I. Each H-sensor is loaded with a predefined maximum cluster size value λ_1. This is the maximum number of neighboring L-sensors to which it will distribute keys. Each L-sensor is loaded with a similar threshold λ_2. Due to the additional resources of H-sensors, λ_1 is usually much larger than λ_2. If the value λ_1 is no more than the value n in the key space models, then the entire cluster becomes perfectly secure against node capture attacks. This issue is discussed in subsection VI.C.

Each sensor is also preloaded with a maximum bootstrapping time limit t_w and a unique node ID. The L-sensors are also preloaded with a probability value P, which will be used if an L-sensor needs to elect itself to generating key space, e.g., if the L-sensor is out of the range of any H-sensor. Finally, if the bootstrapping phase is not assumed to be secure then additional parameters such as large primes, p and q, are loaded onto each sensor. These large primes are used to establish a secure communication channel between nodes by using the Rabin's cryptosystem [8].

TABLE I
PRELOADED PARAMETERS

λ_1	Maximum number of nodes in a H-sensor cluster (H-sensors only)
λ_2	Maximum number of nodes in a L-sensor cluster (L-sensors only and $\lambda_2 \ll \lambda_1$)
t_w	Maximum waiting time
ID	Unique sensor node ID
P	Probability for an L-sensor becoming cluster head (L-sensors only)
p, q	Large primes for generating key space

Rabin's cryptosystem allows a secure communication channel to be established between the cluster heads and the nodes in the network. This cryptosystem is computationally asymmetric and requires only one modular squaring operation to encrypt a message using the product of the primes, p and q. In order to decrypt the message however it is computationally as expensive as RSA decryption. In order to use Rabin's cryptosystem the broadcast message sent by the cluster heads would include the product of the prime numbers it selected prior to deployment. The receiving node would use this product to encrypt its key request message, padded with a predefined pattern needed by the cryptosystem. The cluster head would then decrypt the message using its preloaded primes, p and q, and strip off the defined padding to read the message from the sending node.

We first describe the algorithm loaded onto each H-sensor. Fig. 1 shows the algorithm. When an H-sensor first starts the bootstrapping phase, it generates a key space based on one of the two models (polynomial and matrix based models), as seen in line 2, and stores this information. If the polynomial based model is used, the genKeySpace function would randomly generate n coefficients to be used as a_{ij} in Equation 1. These coefficients would then be sent to the nodes within range of the H-sensor during the keying information distribution phase.

Similarly, in the case of the matrix based key space model the H-sensor would first construct $(n+1)^2$ elements for the public matrix *G* and the elements for a symmetric private matrix *D* with the same dimensions. If the matrix *G* is constructed by a Vandermonde matrix, then only the seed of this matrix would need to be generated and disseminated by the H-sensor and the remaining rows of the matrix can be generated by L-sensors. After generating the matrices *D* and *G*, the H-sensor can construct the matrix *A* by using the transposition of the product of the matrices *D* and *G*. During the keying information distribution phase the H-sensor would send each L-sensor in its cluster the appropriate row from the matrix *A* and column or seed from the matrix *G*.

H-sensor Algorithm

```
1: function HSensor ($\lambda_1$, $t_w$, $k_0$)
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
```

Fig.1: Algorithm used by H-sensors

Once the keying information is generated, the H-sensor will pick a random time to wait, to attempt to avoid collisions with other H-sensors within broadcast range, before broadcasting to its area. This wait time is implemented by the while loop started on line 4. During the wait time the H-sensor listens for any neighboring H-sensor nodes to broadcast that they have completed their key space generation.

If a broadcast is heard from another H-sensor it requests keying information from that H-sensor in order to establish a pair-wise key with it. This pair-wise key is then used for communication directly between two neighboring H-sensors. After the wait time has elapsed the H-sensor runs the key distribution function (details are given in Fig. 3), on line 11, which will broadcast to its cluster that it has completed its key space generation and will start accepting keying information requests from L-sensors.

Fig. 2 shows the algorithm used by L-sensors. The first thing an L-sensor does when starting its bootstrapping phase is to set the wait time for it to listen for H-sensors, by setting *w* to half the maximum wait time, as shown on line 2. This is the maximum amount of time that the L-sensor can wait before keying information is distributed. On line 3, the L-sensor sets its Boolean value, *elected*, to false. This variable is used if the L-sensor does not hear any H-sensors in range and elects itself to generate its own key space and distribute to its neighboring L-sensors. Line 4 starts the while loop in which the L-sensor listens for any broadcasts from nearby H-sensors. If an H-sensor broadcast is heard, the L-sensor requests keying information from it, as shown on line 7. After requesting keying information the L-sensor will set its probability value, *P*, to zero so that it will not elect itself in the future.

The next phase of the algorithm runs if no H-sensor broadcast is heard during the wait time. In this case the L-sensor will attempt to elect itself as a Low Powered Cluster Head (LPCH). On line 12, the L-sensor checks if it is eligible to become an LPCH, i.e., determines if its probability is greater than zero. If it is not eligible, then the L-sensor skips the rest of the algorithm and terminates the function. If it is eligible, it resets the wait time to a random value that is less than half the remaining bootstrapping phase time limit, as shown on line 13. During this wait time if the L-sensor has not been elected, it attempts to elect itself based on the probability value (on line 15). If it is elected to be an LPCH, the L-sensor generates its key space, similar to the process used in an H-sensor. However, due to the limited resources of an L-sensor, the degree of the polynomial or dimensions of the matrix generated may be smaller than those of an H-sensor, and hence can only distribute to a smaller number of neighboring L-sensors. If the L-sensor does not elect itself and has not heard any neighboring nodes broadcast, the algorithm will increase the probability of the L-sensor being elected in the next round.

On line 22 the L-sensor listens for neighboring L-sensors that may have elected themselves as LPCHs. If another L-sensor has become an LPCH, then the L-sensor will request keying information from it. On lines 24-25, if the listening L-sensor has also become an LPCH it then deletes its keying information and sets its variable *elected* to false before sending a request to the neighboring LPCH. After requesting the keying information it terminates the function. However, if it does not hear a nearby LPCH it will start the distribution function on line 30, and then terminates the L-sensor function. These L-sensors cluster head (LPCH) may run out of power due to the extensive computational and communicational overheads needed to generate and distribute key space models. Furthermore, if Rabin's cryptosystem is also required for security reasons during distribution of keying information, then the RSA-like decryption computational overhead also reduces a LPCH's lifetime. One way to reduce the number of LPCHs in the network is to deploy sufficient H-sensors in order to obtain the best coverage in the area using H-sensors.

L-sensor Algorithm

```
1:  function LSensor ($\lambda_2$, $t_s$, P)
2:  w $\leftarrow$ $t_w$ / 2
3:  elected $\leftarrow$ 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 $\leftarrow$ 0 > remove possibility of becoming
                            LPCH
9:   end if
10:         elapse(w)
11:  end while
12:  if P > 0
13:         w $\leftarrow$ random($t_w$ / 2)
14:         while w > 0 do
15:                 elected $\leftarrow$ BecomeLPCH(P) > w/ probability P
16:                 if elected do
17:                         keys $\leftarrow$ genKeySpace()  > construct key
                            space for $\lambda_2$  nodes
18:                  else
19:                         increase probability chance
20:                 end if
21:         listen for neighboring node BROADCAST
22:         if BROADCAST heard
23:                 if elected
24:                         delete key space generated
25:                         elected $\leftarrow$ 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
```

Fig. 2: Algorithm used by L-sensors

In Fig. 3, we describe the key distribution function. This function is loaded onto each sensor. When a sensor calls the function to distribute keying information it will broadcast its ID to all nodes in range, as shown on line 2. If the network is not assumed to be secure during the bootstrapping phase, then the broadcast message will also contain information for establishing a secure connection between the sender and the receivers. For example, if the Rabin's cryptosystem [8] is used, then the broadcast message contains the product of the two large primes p and q. The receivers then encrypt their own keys that they have generated and encrypt this key with this product before sending it to the cluster head, and the key will be used for future communications. The Rabin's cryptosystem allows for computationally cheap encryption of a message, which is suitable for L-sensors. However, the computation of decryption using the Rabin's cryptosystem is comparable to RSA [8].

A sensor then starts the loop on line 3 which accepts key requests from nearby sensors until all of the generated keying information is used by the nearby sensors. Because H-sensors are expected to continue functioning after the bootstrapping phase, they would save one piece of the keying information for itself, which will be used to create pair-wise keys with its one-hop neighbors. As keying information is distributed to requesting nodes, the used keying information is stored in the variable *keyed*, as shown on line 6. On line 7, the cluster head node will uni-cast to the requesting node by sending the ID of the requesting sensor along with the keying information. If the message must be secured, the keying information could first be encrypted with the requesting sensor's key. Finally, once all the keying information is allocated to the cluster members, the cluster head will delete its key space information for security purpose.

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 $\leftarrow$ an unused key share
6:                  keyed $\leftarrow$ keyed $\cup$ {KeyingInfo}
7:                  send(ID, $k_0$(KeyingInfo))
8:          end if
9:  end while
10:  end function
```

Fig. 3: The key distribution function

V. Performance Evaluation

In this Section, we present the performance evaluation of the post-deployment key management scheme. We analyze the communication overhead, storage and computation overhead, and the resilience of the scheme in subsection A, B, and C, respectively.

A. Communication Overhead

We now discuss the communication overhead of the key management scheme, and compare it with the SBK scheme in [6]. Fig. 4 shows the message exchange between a cluster head and a sensor node in the cluster under our scheme. Initially, a broadcast is sent by the cluster head to all sensors within transmission range. Each sensor that is able to hear the broadcast then requests keying information from the cluster head by a unicast message. Finally the cluster head returns the keying information to the sensor if the request is accepted.

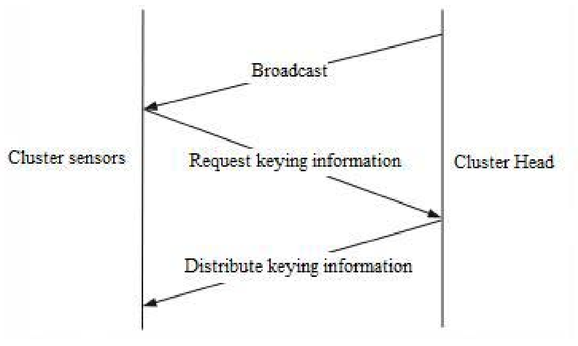

Fig. 4: Message exchanges between a cluster head and sensors

The total possible network-wide communication overhead is given by the following equation:

$$H(w+\lambda_1)+L(w+\lambda_2)+\bar{h}H\lambda_1+d \qquad (4)$$

In this equation, H is the total number of H-sensors in the network which will be distributing the keying information, w is the total number of broadcast messages sent by a cluster head, and λ_1 is the total number of nodes that request keying information for the H-sensor cluster head. Each H-sensor broadcasts w messages and then distributes keying information to λ_1 sensors in its cluster using unicast. This is shown as the first term in Equation (4). An L-sensor cluster head has similar communication overhead, shown as the second term in Equation (4), where L is the total number of L-sensors that elect themselves cluster heads, and λ_2 is the maximum number of sensors in a L-sensor cluster.

There is a total of $H\lambda_1$ keying information requests from L-sensors in the network to their H-sensor cluster heads. Note that the communication from an L-sensor to an H-sensor may be multi-hops, since an L-sensor has a shorter transmission range than an H-sensor. Each of these requests is forwarded on average $\bar{h}$ hops from the L-sensor to its cluster head. This overhead is shown as the third term in Equation (4). Finally, each L-sensor may establish pair-wise keys with d neighbor nodes, where d is the average node degree (i.e., average number of neighboring nodes).

The communication overhead of the SBK scheme [6] is given in Equation (5), where s is the total number of nodes elected to be the service nodes (similar to cluster heads), and other variables have the same meanings as in Equation (4).

$$s(w+\lambda_2)+\bar{h}s\lambda_1+d \qquad (5)$$

We now compare the communication overhead of our scheme and the SBK scheme [6]. In our scheme, there are many fewer broadcast messages being sent throughout the network due to the longer transmission range of H-sensors. For example, suppose 10,000 nodes are uniformly distributed over an area of 1000 by 1000 meters and that the transmission range of L-sensors is approximately 25 meters. Supposed the transmission range of an H-sensor is five times that of the L-sensors in the network. In the SBK scheme, which uses L-sensors, 510 nodes would need to be elected as service nodes for distributing keying information to other nodes in the network. In our scheme, with the addition of using H-sensors in the network, only 20 H-sensors would be needed for distributing keying information to the other nodes in the network. This means that the H in Equation (4) is much smaller than the s in Equation (5), meaning that our scheme uses significantly less broadcast messages than the SBK scheme, and hence has much smaller communication overhead.

Below in Fig. 5, we show the number of cluster heads required by the network to support a 1000 by 1000 meter area given the transmission range of the cluster head. This shows that if the transmission range of the sensor increases the number of cluster heads needed decreases. Due to the limited lifetime of L-sensors and high computational and communicational costs, the 510 elected service nodes in the SBK scheme are expected to die out shortly after the bootstrapping phase. With the addition of H-sensors in the network in our scheme we expect only the cluster heads that have been formed using elected L-sensors to die out after the bootstrapping phase, which is expected to be very small as long as sufficient H-sensors have been deployed in the network.

B. Storage and Computation Overhead

In this subsection, we analyze the storage and computation overhead of our scheme, and compare those with several other key management schemes. Under our scheme, each node in the cluster may receive at most one piece of keying information from a cluster head; each H-sensor may distribute keying information to at most λ_1 nodes; and each L-sensor cluster head may distribute to at most λ_2 nodes. The storage upper bound for a sensor to store its key shares is $H(n+1)\log q$ for the polynomial-based key space, and $H(n+2)\log q$ for the matrix-based key space, where n is the degree of the polynomial or dimensions of the matrix used.

The total storage space is the sum of the storage space of H H-sensors and L L-sensors. We expect L to be small in an HSN since most L-sensors are covered by one or more H-sensors. In addition, with the powerful computational capability, H-sensors should be able to use a large value of n, facilitating key management for more L-sensors. This also increases the security of H-sensor clusters, which is discussed in subsection C.

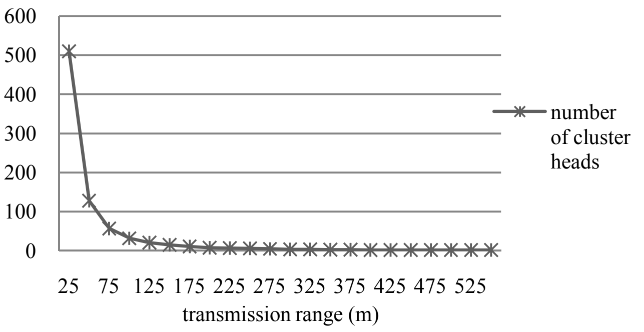

Fig. 5: Number of cluster heads required for a 1000 by 1000 meter area.

Table II compares the number of keys stored under our scheme with three existing key management schemes: the SBK scheme [6], the E-G scheme [5], and the LEAP scheme [9]. In Table II, d represents the average node degree, while S is the number of keys pre-loaded in a sensor prior to deployment. Table III gives an example of key storage requirements with d = 20. As shown in Table III, the E-G scheme needs to store 150 keys in order to establish pair-wise keys with 90% of neighboring nodes, while our scheme, the SBK and LEAP schemes are able to establish pair-wise keys with 100% connectivity between neighboring nodes. Furthermore, our scheme requires a small storage space for the keys. This shows that our scheme is at least as efficient as these existing schemes [6, 5, and 10] in terms of storage overhead.

TABLE II
KEYS STORED

Ours	SBK	E-G	LEAP
d	d	S	$d+3$

TABLE III
KEYS STORED WITH ≥ 90% CONNETIVITY

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

Next we discuss the computation overhead. In our scheme, establishing pair-wise keys with neighboring nodes requires a total of $n+1$ modular multiplications for either of the key space models. For the matrix-based key space model, if a seed is used for generating columns in G, then each L-sensor (that receives the keying information) needs additional $hl(n-1)$ modular multiplications for generating the columns. Our scheme requires the same amount of computation as the SBK scheme in [6].

C. Resilience Against the Node Compromise Attack

We now discuss the resilience of our scheme against the node compromise attack [7]. As mentioned in Section III, both the two key space models hold the property of being n-collusion resistant, meaning that $n+1$ or more nodes (of the same key space) must be captured in order to compromise the pair-wise keys established in the cluster. For better resilience against the attack, one can select a cluster size (e.g., λ_1) that is the same or smaller than the key space size n, i.e., $\lambda_1 \leq n$. This means that only when all the node in the cluster are captured, the key space can be compromised. Thus, if one or more nodes in a cluster are not captured, then the key space is secure, and is resilient against the node compromise attack.

Suppose that an n-degree polynomial is used for generating the key space, and this key space is used by N nodes, where $N > n$. When $n+1$ (or more) nodes are compromised, the coefficients of the polynomial can be determined by the attacker, and the keys in the remaining $N-(n+1)$ nodes can be computed by using their node IDs. Based on the above discussion, one can achieve better resilience against the node compromise attack by using higher polynomial degree n. Similar results can be shown for the matrix-based key space model.

VI. CONCLUSIONS

In this paper, we presented an efficient post-deployment key management scheme for Heterogeneous Sensor Networks (HSNs). The scheme does not require prior deployment (location) knowledge. The scheme achieves high efficiency and effectiveness by utilizing powerful H-sensors, such as larger memory space, better computation capability, and longer transmission range. Most communications and computations are done by the H-sensors. Only a small number of L-sensors become cluster heads and sacrifice themselves for key management. The performance evaluation shows that our scheme has small communication, storage and computation overhead, and achieves strong resilience against the node compromise attack, compared to several other existing key management schemes

ACKNOWLEDGEMENT

This research was supported in part 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.

REFERENCES

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

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

[3] W. Du, J. Deng, Y.S. Han, S. Chen, and P. Varshney, "A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge," *Proc. IEEE INFOCOM '04*, Mar. 2004.

[4] W. Du, J. Deng, Y.S. Han, and P.K. Varshney, "A Pairwise Key Predistribution Scheme for Wireless Sensor Networks," *Proc. 10th ACM Conf. Computer and Comm. Security* (CCS '03), 2003.

[5] L. Eschenauer and V.D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," *Proc. Ninth ACM Conf. Computer and Comm. Security* (CCS '02), 2002.

[6] 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.

[7] D. Liu and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," *Proc. 10th ACM Conf. Computer and Comm. Security* (CCS '03), 2003.

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

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

[10] 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.