2009

Multicast Encryption Infrastructure for Security in Sensor Networks

Richard R. Brooks

Brijesh Pillai

Matthew Pirretti

Michele C. Weigle
Old Dominion University

Repository Citation

Brooks, Richard R.; Pillai, Brijesh; Pirretti, Matthew; and Weigle, Michele C., "Multicast Encryption Infrastructure for Security in Sensor Networks" (2009). *Computer Science Faculty Publications*. 14.
https://digitalcommons.odu.edu/computerscience_fac_pubs/14

Original Publication Citation

Brooks, R.R., Pillai, B., Pirretti, M., & Weigle, M.C. (2009). Multicast encryption infrastructure for security in sensor networks. *International Journal of Distributed Sensor Networks, 5*(2), 139-157. doi: 10.1080/15501320601062114

International Journal of Distributed Sensor Networks, 5: 139–157, 2009
Copyright © Taylor & Francis Group, LLC
ISSN: 1550-1329 print / 1550-1477 online
DOI: 10.1080/15501320601062114

Multicast Encryption Infrastructure for Security in Sensor Networks

R. R. BROOKS[1], BRIJESH PILLAI[1], MATTHEW PIRRETTI[2], and MICHELE C. WEIGLE[1]

[1]Holcombe Department of Electrical and Computer Engineering, Department of Computer Science, Clemson University Clemson, SC
[2]Computer Science and Engineering Department, The Pennsylvania State University University Park, PA

Designing secure sensor networks is difficult. We propose an approach that uses multicast communications and requires fewer encryptions than pairwise communications. The network is partitioned into multicast regions; each region is managed by a sensor node chosen to act as a keyserver. The keyservers solicit nodes in their neighborhood to join the local multicast tree. The keyserver generates a binary tree of keys to maintain communication within the multicast region using a shared key. Our approach supports a distributed key agreement protocol that identifies the compromised keys and supports membership changes with minimum system overhead. We evaluate the overhead of our approach by using the number of messages and encryptions to estimate power consumption. Using data from field tests of a military surveillance application, we show that our multicast approach needs fewer encryptions than pair-wise keying approaches. We also show that this scheme is capable of thwarting many common attacks.

Keywords Sensor Network Security; Multicast Security; Group Key Management

1. Introduction

Wireless technology has seen remarkable growth in the past decade [1, 2]. Low cost, low powered sensors with reasonable processing power have become a reality. A sensor network is a dense mesh of low cost devices that collect and propagate information to a remote user community. Security is a major concern as adversaries can manipulate these sensors to disseminate incorrect information [3, 4]. Secure information exchange between two nodes requires a secure communications pipe between the nodes.

Point-to-point communication schemes use a separate encryption key for every pair of nodes. In this type of environment, each node must either maintain a minimum of *(n–1)* encryption keys to securely communicate with *n* partner nodes, or use a secure index structure for key discovery.

Secure packet forwarding can be done in two ways. One way is to decrypt and re-encrypt the packet at every hop. If a message takes multiple hops to reach the recipient,

This work was supported by, or in part by, the U.S. Research Laboratory and the U.S. Army Research Office (W911NF-05-1-0226).

Address correspondence to R. R. Brooks, Holcombe Department of Electrical and Computer Engineering, Clemson University, 313- C Riggs Hall, PO Box 340915, Clemson, SC, 29634-0915, USA. E-mail: rrb@acm.org

which is typical for sensor networks, a large number of encryptions and decryptions will drain the node power reserves. Another way is using the scheme in [5], where a packet is encrypted once for every recipient and decrypted locally by each recipient. This assumes that the routing information is kept as plain text. In the seminal document on sensor network security [6], the NAI Labs studied many key management protocols for sensor networks. They found traditional secret key protocols inflexible for managing group membership and refreshing cryptographic keys. They also found RSA based public key protocols too power hungry for use in sensor networks. It has yet to be established whether or not elliptic curve public key cryptography also has excessive power needs.

Since public key approaches appear unsuited to use in sensor networks, authentication has been attempted by placing secret keys on the devices. To mitigate the security damage caused by secret keys being exposed when the nodes are physically compromised, Eschenauer et al. [7] proposed random key predistribution (RKP). In RKP a large pool of keys is generated and each node is provided with a small number of keys randomly chosen from the pool. Nodes with keys in common can communicate securely. In that paper, they explain how to revoke keys on compromised nodes; however, they do not discuss how to detect when these nodes are compromised.

Many extensions to RKP have been proposed. In [5], the nodes authenticated using RKP use multiple disjoint communications paths to negotiate new point-to-point keys. This has a number of drawbacks:

(i) the number of communications required seems excessive and
(ii) as with public key systems each node has a key for each partner, limiting the scalability of the approach.

Another extension to RKP that establishes keys for point-to-point communications is given in [8]. They improve on the results in [5] but do not solve the scalability issue associated with pair-wise key management schemes.

Unfortunately, most sensor network applications are data-centric and map poorly to point-to-point communications schemes [9]. Applications are interested in data describing geographic regions, thus the specific node used to collect data is of absolutely no importance. The multicast key management approaches given in [10] and [11] map better to this application domain. These approaches were developed for use in mobile communications systems and use binary trees to efficiently distribute and refresh secret keys. Unfortunately, the approaches in the literature do not deal adequately with the problem of node authentication for sensor networks.

In this paper we show how to use RKP to authenticate nodes and initialize a multicast key management infrastructure in a sensor network. In our approach, a network of nodes deployed at random using the authentication approach in [7] self-organize into a set of multicast regions. Our distributed system is capable of identifying and counteracting several important classes of attacks. We use data from a military sensor network prototype [9] to quantify the power requirements of this approach and find that, for that specific application, it requires about 10% of the energy budget required by existing point-to-point security methods.

Figure 1 summarizes our approach. RKP bootstraps network security. Authenticated sensor nodes collaborate to create secure multicast regions. Geographic regions have local multicast keys. Local key sharing allows applications to support data-centric concepts and reduces communications overhead. Keys are refreshed periodically. We show how to identify and remove compromised nodes from the network.

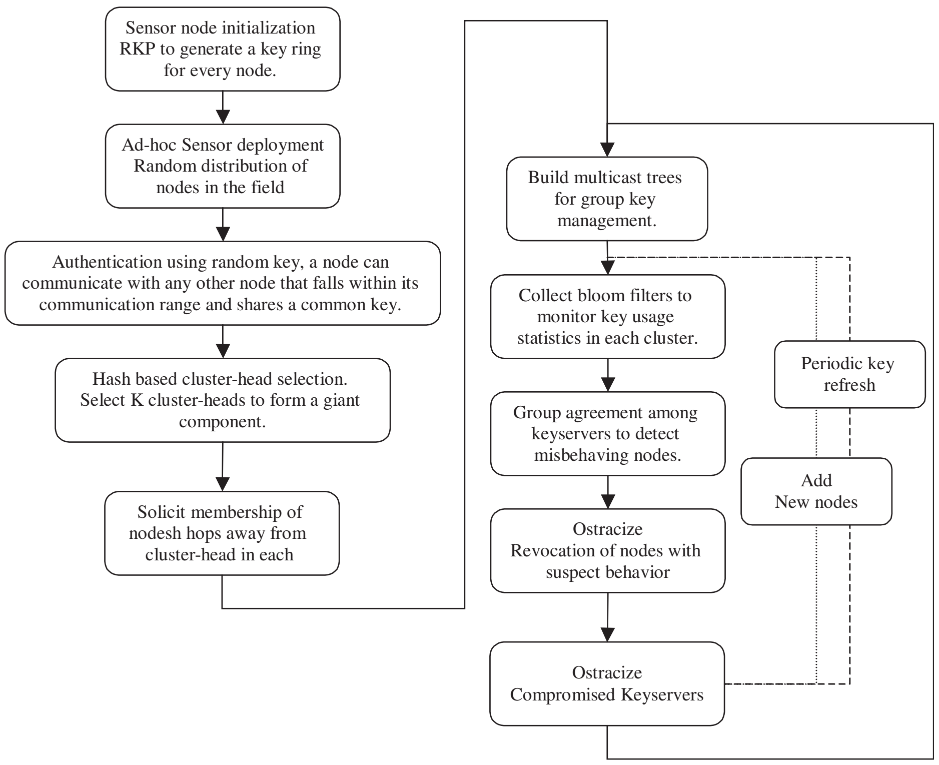
Sensor node initialization RKP to generate a key ring for every node.
Ad-hoc Sensor deployment Random distribution of nodes in the field
Authentication using random key, a node can communicate with any other node that falls within its communication range and shares a common key.
Hash based cluster-head selection. Select K cluster-heads to form a giant component.
Solicit membership of nodesh hops away from cluster-head in each
Build multicast trees for group key management.
Collect bloom filters to monitor key usage statistics in each cluster.
Group agreement among keyservers to detect misbehaving nodes.
Ostracize Revocation of nodes with suspect behavior
Ostracize Compromised Keyservers
Periodic key refresh
Add New nodes

Figure 1. Flowchart of sensor network security approach presented in this paper.

The paper is organized as follows. Section 2 gives an overview of our approach and explains how the network self-organizes to support multicast communications. Section 3 describes our key management policies and quantifies system overhead. Overhead is studied in terms of both the number of encryptions and the number of packet exchanges required. Section 4 explains how our key agreement protocol removes cloned nodes and keyservers from the network. In Section 5 we show the power consumption overhead for encryptions and wireless communications incurred. We analyze the security of this approach and show how it can be used to counter common attacks in Section 6. We conclude in Section 7 with a summary of our approach and its viability.

2. Multicast Tree Management

The approach we present uses two separate tree structures to organize communications among the nodes in the multicast region:

(i) a *communications tree* (Fig. 2a) that defines the connections used to transmit packets between nodes, and
(ii) a *binary key tree* (Fig. 2b) that is used to group nodes so that they share common keys which allows the number of encryptions to be kept to a minimum.

To avoid confusion, we will use these terms throughout the article to distinguish between these two structures.

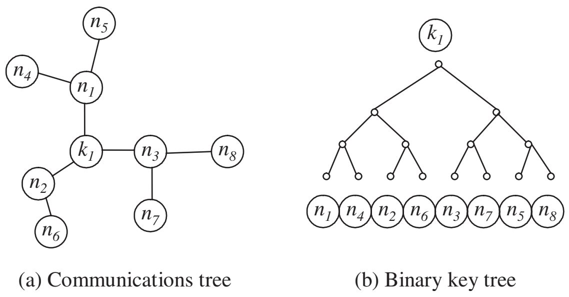

Figure 2. Tree structures (Communications tree and binary key tree).

The work presented here builds on the results [12] that provide optimizations for group re-keying in multicast environments. In this approach, the multicast keys are managed using a rooted multicast key tree [10, 11]. Each sensor node is associated with a leaf node on the key tree. All nodes attached to the key tree share a common symmetric key for encrypting/decrypting data. The data encryption key and a set of key encryption keys are managed by a unique keyserver node. Each of the *n* sensor nodes in the key tree has a unique Key Encryption Key (KEK) and a set of approximately log *n* Shared Key Encryption Keys (SKEKs). KEKs and SKEKs are used only for key management. A binary key tree is used to manage the SKEKs to minimize both the number of keys stored on end nodes and the number of messages sent over the communications tree to refresh and/or revoke keys.

The keyserver solicits every node within *h* hops to join its multicast region. To reduce unnecessary multi-hop data transmissions, during initialization the keyserver constructs a communications tree which groups the nodes it successfully recruits by geographic proximity. The keyserver constructs and securely transmits a unique KEK to each node. This can be done using Shamir's no-key protocol, which requires three messages [13]. The keyserver then constructs, encrypts, and securely transmits the appropriate SKEKs to the sensor nodes. There is one SKEK for each non-leaf node of the key tree, forcing each sensor node to store approximately log *n* SKEKs. Given this key structure, the keyserver can securely communicate with any subset of nodes grouping the multicast region by choosing the minimal set of KEKs and SKEKs.

The first step in our approach is determining the membership of each multicast region by choosing the keyserver. To determine the number of keyservers *k* and number of hops *h* to use for a given network we use the approach in [14] and [15], which uses concepts from the percolation theory to find the values of these two parameters necessary for the system to self-organize into a viable communications substrate. Keyservers maintain the security of the multicast region by controlling group formation and membership.

Using our approach, there is the threat that network security could be compromised if malicious nodes collude to be chosen as keyservers. They could then undermine the keyserver election scheme and skew the process to favor the election of malicious nodes. We avert this possibility by using the secure selection scheme in [16]. This selection scheme ensures that all nodes have an equal chance of becoming a keyserver. When selecting the keyservers (cluster head in [16]), each node:

- Generates a random number *n.*
- Calculates a hash value from the number *h(n).* Any hashing algorithm ranging from modulo arithmetic to SHA1 could be used.

- Broadcasts the hash value to the participating nodes. This commits the node to its number without revealing its value.
- Waits an an agreed-upon time-out period for every participating node to broadcast its hash value.
- Broadcasts a list of all nodes that have transmitted hash values.
- Matches the lists it receives to its local list and requests a hash value from any missing node.
- Waits on an agreed-upon time-out period, and then broadcasts its random number.
- Verifies the random numbers against pre-committed hash values to ensure integrity.

The keyserver is then chosen using an agreed-upon criteria based on these random numbers. For example, the node whose id satisfies (1) becomes the keyserver.

$$Mod\left(\left(\sum_{i=1}^{n} r_i\right), n\right) + 1 \tag{1}$$

where r_i is the pseudo random number generated by node i of the n nodes participating in the selection process. Note that collusion to foil this approach to favor any given node effectively requires the cooperation of all participating nodes.

The second step of the process is creating a binary key tree structure rooted at the keyserver. We use this binary tree structure to manage a set of KEKs that are used when refreshing keys to guarantee the security of the new keys. Let n_c be the number of nodes in a given multicast group, or cluster. Each node is a member of the set $S = \{m_0, m_1, \ldots, m_{n_c-1}\}$. The keyserver is node m_k. We construct a binary key tree T_b where each member m_i is associated with the leaf nodes of T_b. Every node in T_b represents a key K_{ij} where i is the level of the node and j indicates the number of the node *(j^{th} node on level i)*. The key acts as a KEK for all nodes within its sub-tree. Each sensor node in the multicast group stores $\log n_c$ KEKs and a session key. Each KEK represents a group of sensor nodes that are associated with the leaf nodes of the KEK in the key tree. Since finding the optimal tree is NP-complete [14], no tractable polynomial time algorithm exists for solving this problem.

We construct the key tree by using a two-pass protocol, where phase

(i) creates an initial tree and phase
(ii) groups nodes to assign tree nodes to sets of physical sensor nodes.

Phase (i) constructs the initial tree T_{bd} which is used to construct the binary key tree T_b in Phase (ii)

1. Initialize T_{bd} by making the keyserver the root node. It is both root and leaf at this point.
2. Each node not in the tree T_{bd}, that is one hop from a leaf node, becomes its child.
3. Repeat step 2, *(h–1)* times where h is the maximum number of hops from the keyserver to any sensor node in the multicast group. Techniques for computing the parameter h are given in [14].

Phase (ii) constructs the binary key tree T_b from the initial tree T_{bd}. (We will illustrate this phase with the example shown in Fig. 3.)

1. Assign any leaf node of the tree T_{bd} as n_t. (Let n_t be node 5 as shown in Fig. 3b.)
2. Count the number of siblings of n_t (*child nodes of its parent node*). Node 5 has 2 siblings: nodes 6 and 7.

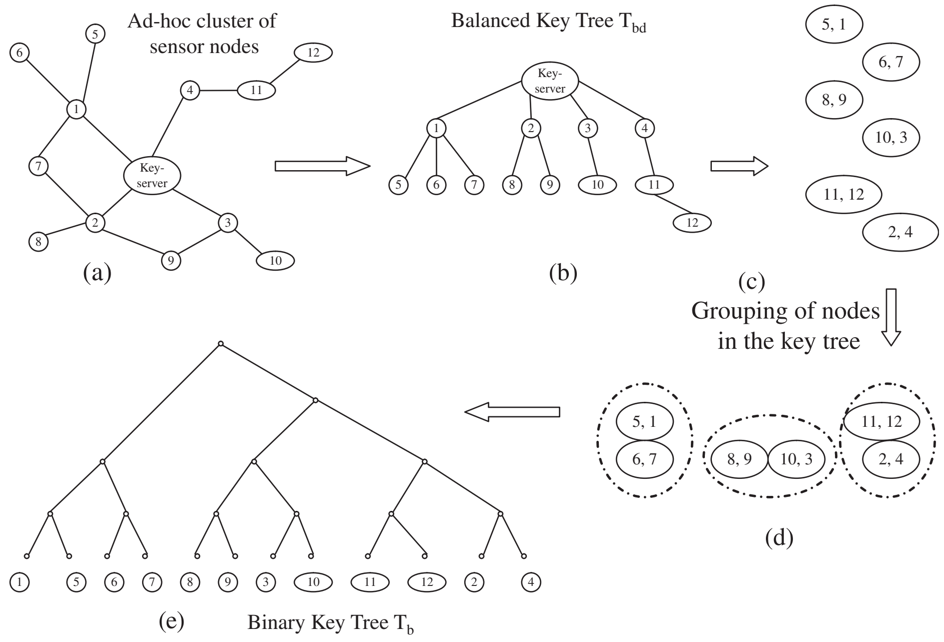
Ad-hoc cluster of sensor nodes
Key-server
(a)
Balanced Key Tree T_{bd}
Key-server
(b)
5, 1
6, 7
8, 9
10, 3
11, 12
2, 4
(c)
Grouping of nodes in the key tree
(d)
(e)
Binary Key Tree T_b

Figure 3. From the ad hoc network (a), we create an initial tree (b). We group nodes (c and d). The result is a binary tree (e) that groups nodes by the keys they will have in common.

a. if (even) group parent with n_t, (node 1 gets grouped with node 5), group all siblings into pairs, (nodes 6 and 7 are grouped together)
b. if (odd) do not group parent node group all siblings and n_t into pairs

3. Remove nodes that were grouped in step 2 from consideration and repeat steps 1 and 2 until all nodes are grouped. (The iterations group nodes 8 & 9, 10 & 3, 11 & 12, 2 & 4 as shown in Fig. 3c.)
4. Fuse grouped nodes into single nodes, reconstruct a balanced tree of the fused nodes and repeat steps 1, 2 and 3. (Fig. 3d shows how nodes 5, 1, 6, 7 get grouped.)
5. Repeat step 4 until only one node exists in the initial tree. Construct the key tree according to the grouping of nodes at every iteration in step 4. (Further iterations group nodes 8, 9, 10, 3 into one group and 11, 12, 2, 4 into another finally to form the binary tree as shown in Fig. 3e.)

Figure 3 shows how to construct a binary key tree from the ad hoc network cluster. This protocol creates an efficient tree for key management within the multicast group. An *n-ary* tree with N nodes will contain ($\log_n N + 1$) levels. A binary tree is used, since binary search trees require a minimal number of operations to distinguish subsets of nodes [17]. To minimize the amount of network traffic using this approach, nodes need to be grouped in a manner that reflects the physical layout of the multicast communications tree. One message-hop is the transmission of one message over one hop in the network. When physical neighbors are leaves on the same key sub-tree, the number of message-hops needed to transmit a new key to those nodes is minimal.

If *msg*(a) is the number of hops from the key server to node a, and *msg(a,b)* indicates the number of hops from the key server to nodes a and b, then

$$msg(a,b) = msg(a) + msg(b) - msg(a \bigcap b).$$

The grouping of nodes a and b is optimal when *msg(a,b)* is minimum (i.e. when the number of hops is shared by nodes the paths from the keys a and b is maximized). This occurs at:

$$\min(msg(a,b)) = \min(msg(a), msg(b)) + 1.$$

For example, consider the two leftmost leaf nodes on the binary key tree T_b in Fig. 3e. They correspond to nodes 1 and 5 in the communications tree. To change the KEK for the key tree node that has nodes 1 and 5 as children takes a total of two message-hops. If the same tree nodes had been mapped to physical nodes 6 and 12, the same operation would require 3 hops to node 12 and 2 hops to node 6, or 5 message-hops total.

Our protocol is a heuristic that provides good results in most cases. Some simulations found multiple trees with equal numbers of message-hops to every node. For the cluster of 10 nodes in Fig. 4a, our protocol gives the binary tree as shown in Fig. 4b. Another possible configuration is shown in Fig. 4c. However, we see by counting that both Fig. 4b and Fig. 4c require 82 message-hops. This is calculated by:

- 3 messages from the keyserver to each node in the cluster (45 message-hops for both Fig. 4b and Fig. 4c).
- 1 encryption for each parent node and transmission to both children.
  - For Fig. 4b: $k_{1,2}$ 2 message-hops, $k_{6,7}$ 4 message-hops, $k_{4,8}$ 5 message-hops, $k_{3,9}$ 3 message-hops.
  - For Fig. 4c: $k_{1,2}$ 2 message-hops, $k_{5,7}$ 3 message-hops, $k_{4,8}$ 5 message-hops, $k_{3,9}$ 3 message-hops.
- 1 encryption for each higher level parent node and transmission to all children.
  - For Fig. 4b: $k_{5,6,7}$ 4 message-hops, $k_{1,2,5,6,7}$ 5 message-hops, $k_{4,8,3,9}$ 5 message-hops, $k_{1,2,3,4,5,6,7,8,9}$ 9 message-hops.

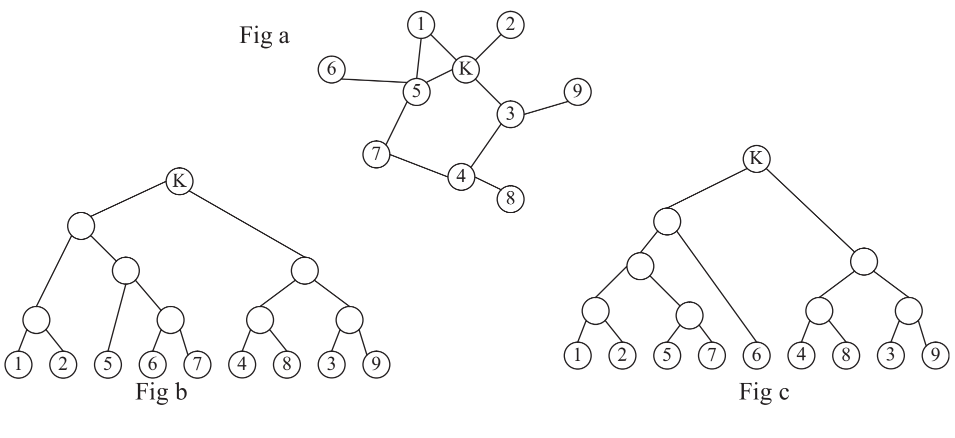

Figure 4. Example solutions to the tree generation process.

- For Fig. 4c: $k_{1,2,5,7}$ 4 message-hops, $k_{1,2,5,6,7}$ 6 message-hops, $k_{3,4,8,9}$ 5 message-hops, $k_{1,2,3,4,5,6,7,8,9}$ 9 message-hops.

3. Group Key Management

Key management is performed by the keyserver, which coordinates the following activities:

a. Initial keying.
Every sensor node elects the keyserver in a distributed and secure manner by the scheme in [16]. The keyserver establishes a unique private key with each member of the multicast group using Shamir's protocol [13] which sets up a shared key over an untrusted channel by exchanging 3 messages. The keyserver then generates a key-encryption key (KEK) for pairs of nodes each at level (log n_c–1) of the binary key tree. It encrypts the key for level j with the private key for level j+1 of the tree. Finally, the session key denoted by the root of the binary tree is established. This session key can now be used to secure multicast data communications.

b. Member ostracism.
If a single node is found to be insecure [18], a new multicast key is created and sent to the rest of the multicast tree. If node m_l leaves the cluster, its former sibling replaces their parent node. All keys with member m_l are compromised and must be re-keyed. When a single node is compromised, the entire log n_c keys associated with it are compromised. However, since the node's sibling is promoted one level in the tree structure, one less key needs to be replaced. Removing isolated nodes is the worst-case scenario, since it is often the case that sets of nodes will have some SKEK in common. The existence of a common key allows the keyserver to group nodes in a way that reduces both the number of encryptions and the number of message-hops.

c. Member join operation.
A join operation is a request from a sensor node to join the cluster. It is not efficient to re-key the entire tree to perform a single join. Instead, the new sensor node is attached to leaf nodes without siblings when they are available. If n_c was even, then there are usually no isolated nodes. In this case, the new sensor node attaches itself to the nearest sensor node. Over time the key tree may become unbalanced and a complete re-keying of the multicast region will be necessary. In our approach, we assume that the process does not have to keep previous transactions secret from new members. If that is the case, a new session key needs to be distributed to all nodes in the multicast region.
The overhead for encryptions, messages, and the number of hops required for each operation is derived in Section 5.

4. Group Agreement Protocol

Random key predistribution security schemes are well-suited for use in sensor networks due to their low overhead. However, the security of a network using pre-distributed keys can be compromised by cloning attacks. Chan and Perrig [4] explain how clones can falsify sensor data, extract data from the network, and/or stage denial of service attacks.

In this section, we expand the approach in [18] for detecting cloning attacks on sensor networks using random key predistribution. In that approach, clone detection is done by a central authority outside the network. We show here how this process can become an organic (in military usage) part of the network.

Each sensor node makes a counting Bloom filter of the keys used by its neighbors to authenticate themselves [18]. A Bloom filter is essentially a hashing function that supports membership queries. Counting Bloom filters provide information on the number of times an element exists in the set. For our purpose, Bloom filters simultaneously reduce the volume of data transmitted and avoid sending key values over the network. Each keyserver collects key usage statistics within its cluster. Equations that define threshold values for cloned key use in sensor networks can be found in [18]. If a key is used more than the threshold value, sensor nodes authenticated by using that key are assumed to be cloned. Since counting Bloom filters are transmitted within the cluster, any keyserver can only recognize the keys within its key ring. Hence, a second round of message exchanges between keyservers is necessary so that the key usage for every key is available to every keyserver.

The threshold for the number of times a key is used for authentication is based on the probability that two nodes share a key and the probability that they share a connection. The expected number of times a key is used for communications in the network is:

$$\mu_k = \frac{\sum_{j=1}^{n} M_j * \left(\frac{N_j}{M_j}\right)}{P} \tag{2}$$

with variance:

$$\nu_k = \frac{\sum_{j=1}^{n} M_j * \left(\frac{N_j}{M_j} - \mu_k\right)^2}{P} \tag{3}$$

where M_j is the probability a key is on exactly j nodes, P is the total number of keys in the system, and N_j is the expected number of times a key on exactly j nodes is used.

We determine whether or not a key is cloned by comparing U_i (the number of times key i is used to establish connections, which we collect by using Bloom filters) to μ_k computed using (2) [4]. If U_i is significantly higher than μ_k it is likely that the key is being used by cloned nodes. Our approach monitors key usage and detects statistical deviations in key use that indicate cloning attacks. The system can recover from a cloning attack by terminating connections using cloned keys. In [18] we detail the clone detection protocol and derivation of equations (2) and (3).

If any keyserver turns out to be subverted, it gives rise to a problem similar to the Byzantine Generals Problem. Barborak [19] provides a survey of related research that contains algorithms for forcing agreement as long as well-known criteria (described in Section 4.1.4 of [19]) are satisfied. By applying these algorithms, our group key agreement protocol can ensure security as long as less than one-third of the keyservers are compromised.

Since compromised nodes may incorrectly report key usage statistics and adversaries may collude to cause legitimate nodes to be detected as cloned nodes and ostracized from the network, our group agreement protocol must be immune to internal subversion. The protocol in Section 4.1 reaches agreement on the set of cloned keys and can tolerate subversion of less than 1/3 of the keyservers. The limit of 1/3 is the theoretical bound on distributed consensus in the presence of malicious parties [19].

4.1. Group Key Agreement Protocol

To reach consensus on the set of cloned keys in the network:

1. Each node transmits to its keyserver the counting Bloom filter for the keys used by the node.
2. The keyserver transmits the counting Bloom filters from all the nodes within its multicast region to every other keyserver using an authenticated channel. These channels are established using Shamir's 3 pass protocol [13].
3. Every keyserver now has counting Bloom filters for every multicast region. Each keyserver computes key usage statistics for keys in its key ring to identify cloned nodes.
4. The keyservers use a Byzantine agreement protocol [19] to reach consensus on key usage statistics. The protocol from [20] comes to a correct consensus as long as less than one-third of the keyservers are faulty or compromised. It also degrades gracefully as long as fewer than ½ of the inputs are malicious.

- The keyserver computes a vector v of usage statistics from the Bloom filters provided by the keyservers.
- The vector v is sorted, and the lowest and the highest τ values are discarded to give rise to a new vector v' containing $(k - 2*\tau)$ entries, where τ is the number of adversaries the network can tolerate as keyservers.
- The key usage value is the mean of the vector v'.
- This protocol is guaranteed to be correct when $k \geq 3\tau + 1$. Performance degrades slowly until $k = 2\tau + 1$, at which point the majority of the information is false and the opponents control the network.

Suppose that the system is designed to tolerate up to $\tau = 10$ subverted keyservers. The key usage reported by the subverted keyservers is questionable and should not be taken into account in the decision process. We discard 20 values (the first 10 and last 10 from the sorted vector v of key usage statistics) so that values with highest deviation from the mean do not affect the clone detection protocol. Any adversary that launches a Byzantine attack will have to report values near to those reported by the legitimate nodes if their values need to be a part of the decision-making process. The protocol is thus robust against Byzantine attacks. The average number of nodes in a single multicast group is n_c. Hence, $(n_c–1)$ messages have to be transmitted for the keyserver to collect the counting Bloom filters from every node in its multicast group. This amounts to $(k * (n_c–1))$ for the entire network of k keyservers.

A secure private key has to be established between every pair of keyservers to exchange the compressed counting Bloom filters. Using Shamir's 3 pass protocol $3k(k-1)/2$ messages have to be transmitted to setup a secure connection between every pair of keyservers. We require additional $k(k-1)$ messages to exchange counting Bloom filters.

Total number of messages for group agreement = $k(n_c - 1) + 5(k^2 - k)/2$ (Derived in [14])

5. Message Overhead and Power Consumption

We now compute the number of encryptions, messages, and hops required by our approach. We denote the key associated with the j^{th} node on i^{th} level of the binary key tree as $K_{i,j}$.

The results tabulated in Table 1 for system overhead for every membership operation are derived below.

a. Initial keying.

Number of encryptions:

At level (log n_c), 0 encryptions are needed. Shamir's protocol requires no encryption. At each of (log n_c)–1 levels from level 0 to level (log n_c)-1, a KEK is established using 2 encryptions.

The number of KEKs at these levels $= \sum_{i=0}^{(\log n_c)-1} 2^i = 2^{\log n_c} - 1 = n_c - 1$

$$\text{The total number of encryptions} = 2 * (n_c - 1) + 0 = 2(n_c - 1) \qquad (4)$$

Number of messages exchanged:

At level (log n_c), 3 messages are exchanged using Shamir's protocol to setup the private key. At each of (log n_c)–1 levels from level 0 to level (log n_c)–1, a KEK is established and it takes two messages to transmit to both the branches. Thus for each KEK, we have 2 messages.

$$\text{Total number of messages} = 2 * (n_c - 1) + 3 * (n_c - 1) = 5(n_c - 1) \qquad (5)$$

Number of hops:

Let n_i denote the number of nodes exactly i hops from the root node such that $(n_c - 1) = \sum_{i=1}^{h} ni$. At level (log n_c), each of the 3 messages required to set a private key using Shamir's protocol require h_i hops where h_i is the number of hops to member m_i from the root node, *i.e.* the keyserver. The number of hops to set up keys at level $(\log n_c) = 3 * \sum_{i=1}^{h} (ni * i)$. At level (log n_c)–1, again we transmit one message to each of the members of the multicast group. The number

Table 1
Overhead for group membership operations

	Total Encryptions	Total messages transmitted	Total hops required
Initial keying	$2(n_c - 1)$	$5n_c - 2$	$n_c - 1 + (4 + \acute{\alpha} * ((\log n_c) - 2)) * \sum_{i=1}^{h} (ni * i)$
Member ostracism	$2 * ((\log n_c) - 1)$	$2 * ((\log n_c) - 1)$	$n_c + \acute{\alpha} * n_c/2^j * ((\log n_c) - 2) * \sum_{i=1}^{h} (ni * i)$
Member join	2	$5 + ((\log n_c) - 2)$	$\acute{\alpha} * h * (5 + ((\log n_c) - 2))$

of hops to set up keys at level $(\log n_c) - 1 = \sum_{i=1}^{h} (ni * i)$. At level 0, the session key for the entire cluster has to reach all the members. The number of hops to set up the session key at level $0 = \sum_{i=1}^{h} ni = (n_c - 1)$. At the remaining (log n_c)–2 levels *(level 1 to level (log n_c)–2)*, each level i has 2^i nodes. At each node we transmit two messages to their respective left and right branch, which have to reach nodes m_i each of which are h_i hops away *($i = 0, \ldots, n_c$–1)*. However, the number of hops is reduced according to our ability to group nodes so that paths from the root to neighboring nodes overlap. We define a variable $\acute{\alpha}$ which represents this effect. Consider a physical layout where there is no overlap. Every sensor node in the multicast group has a separate path from the keyserver. This worst case estimate can be calculated by setting $\acute{\alpha}$ to 1. In the best case scenario, the number of hops at *(level 1 to level (log n_c)–2)* would be h. The average number of hops at *(level 1 to level (log n_c)–2)* = $\acute{\alpha} \sum_{i=1}^{h} (ni * i)$

$$\text{The total number of hops} = (n_c - 1) + (4 + \acute{\alpha} * ((\log n_c) - 2)) * \sum_{i=1}^{h} (ni * i) \quad (6)$$

b. Member ostracism.

Number of encryptions:
(log n_c)–1 keys have to be replaced, each requiring 2 encryptions.

$$\text{Total number of encryptions} = 2 * ((\log n_c) - 1) \quad (7)$$

Number of messages exchanged:
For the replacement of (log n_c)–1 keys, each replacement requires sending one message to each of the key's two child nodes.

$$\text{Total number of messages} = 2 * ((\log n_c) - 1) \quad (8)$$

Number of hops:
Each message must reach a subset of nodes within the cluster which consist of all leaf nodes associated with the new KEK. The session key at level 0 *($K_{0.0}$)* has to be encrypted with its child node *($K_{1.0}$ and $K_{1.1}$)* and then transmitted to the set of all leaf nodes served by $K_{1.0}$ and $K_{1.1}$. In general, the size of the subset of nodes is $n_c/(2^{(j-1)})$ where j denotes the level of the KEK in the key tree. The average number of hops to deliver 2 *((log n_c)–1) messages, where each message reaches $n_c/(2^{(j-1)})$ nodes = [2 * ((log n_c)–1)] * [$n_c/(2^{(j-1)})$] = n_c((log n_c)–1)/2^j. The average number of hops at each level = $\acute{\alpha} \sum_{i=1}^{h} (ni * i)$. With every change, the session key has to be rekeyed, which requires n_c hops to reach all nodes

within the cluster. The number of hops required to transmit remaining ((log n_c)–2) KEKs = $\acute{\alpha} * n_c/2^j * ((\log n_c)-2) * \sum_{i=1}^{h}(ni * i)$

$$\text{The total number of hops} = n_c + *n_{c/2^j} * ((\log n_c)-2) * \sum_{i=1}^{h}(ni * i) \qquad (9)$$

c. Member join operation.

Number of encryptions:

When a node joins a leaf node, it creates an extra parent node and a KEK associated for that node.

$$\text{Hence, a total of 2 extra encryptions are required} \qquad (10)$$

Number of messages exchanged:

As the case for encryption, we require 2 messages to transmit that KEK. Also, the new node should establish its private key (3 messages) and receive existing ((log n_c)–2) KEKs in the path to the root node *(keyserver)* in the keytree.

$$\text{Total number of messages} = 5 + ((\log n_c) - 2) \qquad (11)$$

Number of hops:

Since the join occurs on any of the members less than h hops away, the total number of hops required to transmit the messages to the new member and its sibling can be a maximum of (5+((log n_c)–2))*h and minimum of (5 + ((log n_c)–2)).

$$\text{The total number of hops} = *h * (5 + ((\log n_c) - 2)) \qquad (12)$$

The overhead calculations for the number of encryptions and messages are of the order of the multicast region size (n_c) and for some operations proportional to its logarithmic value which scales better with the number of nodes. We compute power requirements for our scheme using data from NAI Labs [6]. Table 2 summarizes the results we get by applying the NAI Labs empirical values to our scheme. We consider networks using the SA-110 StrongARM and ARC-3 processors using AES (Advanced Encryption Standard) and RSA encryption. Symmetric key cryptography algorithms like AES consume less power than asymmetric algorithms like RSA. For the sake of argument, we calculate the power overhead incurred when the random key predistribution protocol preloads each sensor node with 50 keys and that the average message length is 900 bits.

The StrongARM 110 processor consumes 108 nJ for each 128-bit operation [6]. The StrongARM consumes 12 times the energy of the ARC-3 [6]. Hence the power consumption for a 128-bit operation on ARC-3 can be estimated as 9 nJ.

Table 2
Power consumption for network initialization

	RSA–SA 100	*RSA-ARC 3*	*AES-SA 110*	*AES-ARC 3*
Keyserver $2(n_c-1)$ encryptions	*$10.72(n_c-1)$ mJ*	*$0.90(n_c-1)$ mJ*	*$30.52(n_c-1)$ μJ*	*$1.12(n_c-1)$ μJ*
Each sensor node $\log(n_c)$ decryptions	*$110.42\log(n_c)$ mJ*	*$9.2\log(n_c)$ mJ*	*$15.26\log(n_c)$ μJ*	*$0.56\log(n_c)$ μJ*
Bluetooth Tx & Rx	*Keyserver $4(n_c-1)$ Tx & (n_c-1) Rx*		*Each sensor node 1 Tx & $(2+\log(n_c))$ Rx*	
Total power consumption for 900-bit messages	*$0.42(n_c-1)$ mJ*		*$(150+60\log(n_c))$ μJ*	

RSA encryption roughly requires 7,056 128-bit operations and decryption requires 145,408 128-bit operations. Therefore, RSA requires 5.36 mJ to encrypt a message of 900 bits and decryption consumes 110.42 mJ on StrongARM 110 processor. RSA encryption on ARC-3 requires 0.45 mJ and decryption 9.20 mJ.

AES requires less energy than RSA. AES encryption and decryption overhead is nearly identical. Encrypting 900 bits on a StrongARM processor would require 15.3 μJ whereas the ARC-3 requires only 0.6 μJ. These estimates are based on 0.00217 mJ/128 bit encryption for StrongARM and 0.00008/128 bit encryption for the ARC-3 processor [6].

Our protocol requires $2(n_c-1)$ encryptions by the keyserver and $\log(n_c)$ decryptions at each node. Table 2a gives the power consumption for encryption and decryption key initialization at any keyserver and sensor node.

Transmission costs are usually 1.5 times the energy required for reception. The energy consumed by idling and receiving data is nearly identical. Assuming that only transmission requires extra energy, we estimate the cost for transmitting 900-bit messages within a range of 50 meters. Radio communications using Bluetooth expend 10^{-7} joules per bit for transmission.

Private key establishment with each sensor node requires two message transmissions by the keyserver and one by the sensor node. Each sensor node receives two messages and has to transmit one. Also, for each KEK in the binary key tree other than leaf nodes, the keyserver transmits two messages and every sensor node receives $\log n_c$ such transmissions. There exist (n_c-1) such KEKs. Thus, we have a total of $4(n_c-1)$ transmissions and (n_c-1) receptions at the keyserver and one transmission and $(2+\log n_c)$ receptions at each of the (n_c-1) sensor nodes in the multicast group.

Table 2b tabulates the total power consumption for transmission and reception during key initialization.

Thus for a sample network of 100 nodes scattered in unit square with a communication radius of 0.2, the ideal cluster size [14] is a 2-hop radius containing 26 nodes on an average. Assuming AES encryption on ARC-3 processor, the keyserver consumes 28 μJ for encryption and 10.5 mJ for communication, whereas every sensor node in the multicast consumes 2.6 μJ for decryption and 0.43 mJ for communication.

ColTraNe [21] is a surveillance application that tracks military targets using a sensor network. We use the data collected from our field test of this network to estimate the power overhead required by our approach. Our multicast approach requires fewer encryptions and less energy consumption than pair-wise keying approaches. One tracking method used in the field tests required 911 packets to be exchanged between 70 sensor nodes. A pair-wise keying approach would require one encryption per packet to securely transmit the data. AES encryption on a MC68328 Dragonball processor requires 205.68 mJ energy to encrypt the 911 packets that contain a total of 254,552 bytes of data.

Our multicast approach requires fewer encryptions. In the worst-case each packet would have to be encrypted for all 12 multicast regions. This would require only 95 encryptions. The total power consumption for encrypting data drops to 17.76 mJ. A more detailed analysis of these results is available in [14].

6. Security Analysis

Applications need to guarantee that adversaries cannot subvert the sensor network, and our approach can be used to neutralize many important classes of attacks. In Byzantine attacks malicious nodes try to force disagreement among legitimate nodes. In Sybil attacks [22] compromised nodes try to maintain multiple identities. This can be used either to stage Byzantine attacks or drain power from legitimate network nodes. Cloning attacks create multiple copies of legitimate nodes. They can disrupt a network by inserting false detections, dropping packets, modifying data, and eavesdropping. In this section we describe these attacks and show how they fail against our multicast encryption infrastructure.

6.1. Byzantine Attack

In a Byzantine attack, a compromised node forwards manipulated information to cripple the decisions of legitimate nodes and, in certain cases, also to influence the result. Assume that a cloned node C reports incorrect information about its key usage by transmitting randomly generated Bloom filters. In this case, when a keyserver has to calculate the key usage statistics, it will discard the extreme τ values after sorting as in the Byzantine agreement protocol. False information from malicious nodes gets thrown out.

Consider a network of N nodes scattered randomly on unit area. Let c of the n nodes be adversaries. The probability that an illegitimate node gets selected as the keyserver is the same as any other node in the network, i.e., $c/_n$. If we pick k nodes at random as keyservers, the expected number of cloned keyservers is $(c.k)/_n$. According to the group agreement protocol in Section 4.1, to secure the network against a Byzantine attack, we have to introduce redundancy in the network. To tolerate c clones in the entire network, we have to select $max\left\{ (3c.k)/_n + 1, k \right\}$ keyservers.

6.2. Sybil Attack

The Sybil attack [13, 22] is an attack on a network where one compromised node acts as multiple nodes (i.e., has multiple identities.) For example, multiple Sybil identities can drain resources from legitimate nodes by sending spurious traffic. Alternatively, Sybil nodes can skew voting algorithms, like the Byzantine Agreement. Sybil attacks are known to affect routing, data aggregation, and detection of cloned nodes in sensor networks. Our approach counteracts these attacks, since they make it easier for the compromised nodes to be identified.

Lemma. A Sybil attack from any node picked as a keyserver on the sensor network makes it easier to detect that adversary.

Proof. We select k nodes out of N nodes distributed randomly in a square field of area A as keyservers. Hence, the density of keyservers in the field is k/A.

In the case of a Sybil attack, a compromised keyserver reports the existence of multiple keyservers. This leads to an increase in the density of reported keyservers. We have a prior threshold on the density of keyservers in the sensor network. Since the keyservers are selected at random, their density should remain within a predictable variance from the mean. A Sybil attack will increase the keyserver density when the multicast region is contained entirely in the field. Alternatively, compromised keyservers can report a region outside the regular field to create Sybil nodes without changing the node distribution. To counter this, it is advisable to restrict deployments to predefined regions.

The key usage statistics will also support the detection of the Sybil nodes. Since the Sybil nodes require its keys to be used multiple times, they quickly fall into the category of suspicious cloned nodes that will be ostracized by the network.

Consider a network of 1000 nodes spread over unit area with 100 keyservers picked at random. Hence, the average density of the nodes is 1000 and that of the keyservers is 100. Assume a node located at location (0.1, 0.1) is a Sybil node. It reports the false existence of 6 keyservers around it, within a range of 0.1. Hence the density of the keyservers around the Sybil node increases to $6/(\pi*0.1*0.1) = 192$. Such a high density of the reported keyservers implies that the node is a Sybil node. To avoid detection of this increased density, the Sybil node would have to report its Sybil keyservers to be located outside our bounded area, say locations (−0.05, −0.1) which would in turn make that node void, since it lies outside the area of the network.

6.3. Cloning Attacks

In a cloning attack, the adversary fills the network with copies of compromised nodes. The cloned nodes then launch a variety of attacks ranging from forwarding erroneous data to manipulating the messages in the network. It also attempts to drain other legitimate nodes of its limited power. Our previous work [18] shows that by using statistical analysis of the key usage in the network such clones can be easily detected. They can be ostracized from the network by discarding the keys used by that node. A key that is used very often indicates unusual activity by the nodes using it.

Each node transmits a counting Bloom filter of the used keys. The keyservers come to a consensus on which keys are cloned on the basis of pre-defined thresholds. Any key that is used more than a given threshold times is a candidate to be a suspected cloned node. If the mean usage and variance of a key is above the specified threshold, the keyserver calculates a confidence on the suspected cloned key. The cloned nodes are removed from the network by terminating all connections that use the set of cloned keys.

Simulations on a 250-node network support the predicted equations. With a key pool size of 1000, each node randomly picking a key ring of 50 keys and with an assumption that

10% of the keys are already cloned, analysis predicts that any key used more than 100 times on average indicates a cloned node. Simulations showed that cloned nodes used the same keys an average of 100 times. In a network of 250 nodes, if there are at least 20 clones, the cloned nodes can be detected with a false positive rate of 70%.

A re-encryption of only log n_c keys to form new key encryption keys restores the credibility of the network with the suspected node thrown out of the network. Since a single point of failure does not affect connectivity, the network still maintains a giant component and is able to perform its task.

7. Conclusions

In this paper we present a fully distributed approach to sensor network security. The proposed approach allows the network to organize itself into a set of security zones. We have shown how to organize and maintain the security regions. In doing so, we quantified both the energy savings attained and showed that the approach is able to tolerate many classes of attacks.

We start with an *ad hoc* network of sensor nodes deployed using random key predistribution. Each node is equipped with a random sampling of keys from a large key pool that they use for authentication. Once the nodes have authenticated their neighbors using the random key predistribution approach, we create multicast regions. Some nodes are chosen at random to work as keyservers and maintain security within their local multicast region.

We use a secure selection scheme to randomly select the keyservers. The keyserver nodes establish local multicast groups using the protocol described in Section 2. This approach has no single point of failure and is resilient to collusion among nodes.

The key agreement protocol presented in Section 3 secures the network from cloning attacks and internal corruption. Keyservers use a Byzantine agreement protocol to identify and ostracize compromised nodes. Group key management reduces the network overhead due to encryption and message exchanges, which results in significant energy savings. We use statistics from standard commercial hardware to quantify the energy savings provided by our approach.

Our approach uses key usage statistics to detect cloned nodes. This keeps the false positive rates within manageable bounds. If the number of clones is below this threshold, they will not be detected. This means that applications that use our approach need to be designed to tolerate inputs from a small number of malicious participants.

Further research is desirable in developing more efficient protocols for two portions of the proposed framework. The first aspect is keyserver selection. The current approach requires $O(n^2)$ messages. It would be useful to have a localized server selection protocol, which requires less network bandwidth. The second issue is the use of Byzantine agreement for securing against malicious keyserver nodes. Byzantine agreement protocols tend to require a large number of messages. It may be possible to develop local agreement protocols that provide similar safeguards.

About the Authors

Dr. R.R. Brooks has a Ph.D. in Computer Science from Louisiana State University, and a B. A. in Mathematical Sciences from The Johns Hopkins University. He is currently an associate professor of electrical computer engineering at Clemson University in Clemson, South Carolina. He previously was the head of the Distributed Systems Department of the Pennsylvania State University Applied Research Laboratory. Dr. Brooks was PI of the

Mobile Ubiquitous Security Environment (MUSE) Project sponsored by ONR as a Critical Infrastructure Protection University Research Initiative (CIP/URI). He is author of *Disruptive Security Technologies with Mobile Code and Peer-to-peer Networking* from CRC Press. He was co-PI of a NIST project defining security standards for networked building control systems. He has had other research projects funded by ARO, ONR, and DARPA. His Ph.D. dissertation received an exemplary achievement certificate from the Louisiana State University graduate school. His current research concentrates on distributed strategic systems for network security and national defense. He has a broad professional background with computer systems and networks. This includes being technical director of Radio Free Europe's computer network for many years. His consulting clients include the French stock exchange authority and the World Bank. While with the World Bank he expanded their internal network to sub-Saharan Africa, Eastern Europe, and the former Soviet Union.

Brijesh Pillai received his M. S. degree in Computer Engineering from the Holcombe Department of Electrical and Computer Engineering in Spring 2006. His thesis topic was *Network Embedded Support for Sensor Network Security.* It proposed solutions for countering cloning and Sybil attacks in sensor networks.

Michele C. Weigle is an Assistant Professor of Computer Science at Clemson University. She received her Ph.D. from the University of North Carolina at Chapel Hill in 2003. Her research interests include network protocol evaluation, network simulation and modeling, Internet congestion control, and mobile ad-hoc networks.

Matthew Pirretti is a Ph.D. candidate with the Computer Science and Engineering (CSE) Department of the Pennsylvania State University in University Park, PA. He has his B.S. and M. S. degrees in CSE from Penn State. His research interests include sensor network security issues.

References

1. I. F. Akyildiz W. Su, Y. Sankarasubramaniam and E. Cayirci, ''A survey on sensor networks,'' *IEEE Communications*, vol. 40, no. 8, pp. 102–114, Aug. 2002.
2. Malik Tubaishat and Sanjay Madria, ''Sensor Networks – An Overview'', *IEEE Potentials*, vol. 22, no. 2, pp. 20–23, April/May 2003.
3. D. W. Carman, ''Data security perspectives,'' in *Distributed Sensor Networks*, (eds.) S. S. Iyengar and R. R. Brooks, Chapman and Hall,, Boca Raton, FLA, 2005.
4. H. Chan and A. Perrig, ''Security and privacy in sensor networks,'' *IEEE Computer Magazine*, pp. 103–105, 2003.
5. H. Chan, A. Perrig and D. Song, ''Random key predistribution schemes for sensor networks,'' *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pp. 197–214, 2003.
6. D. W. Carman, P. S. Kraus and B. J. Matt, ''Constraints and approaches for distributed sensor network security (final),'' *NAI Labs Technical Report #00–010*, September 1, 2000.
7. L. Eschenauer and V. D. Gligor, ''A key-management scheme for distributed sensor networks,'' *Proceedings of the 9th ACM Conference on Computer and Communications Security CCCS*, 2002.
8. Liu D and Ning P, ''Establishing pairwise keys in distributed sensor networks,'' *Proceedings of the 10th ACM Conference of Computer and Communication Security CCCS* 2003.
9. S. S. Iyengar and R. R. Brooks, eds., *Distributed Sensor Networks*, Chapman and Hall, Boca Raton, FL, 2005.
10. R. Poovendran and J. S. Baras, ''An information theoretic analysis of rooted-tree based secure multicast key distribution schemes,'' *Lecture Notes in Computer Science*, vol. 1666, pp. 624–638, August 1999.

11. S. Dahlman, *Key management schemes in multicast environments*, Pro gradu Thesis, Computer Science Dept, University of Tampere, 2001.
12. S. Zhu, S. Setia and S. Jajodia, ''Performance Optimizations for Group Key Management Schemes for Secure Multicast,'' *Technical Report, George Mason University*, 2002.
13. A. J. Menezes, P. C. van Oorschot and A. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, 1996.
14. B. Pillai, Network *Embedded Support for Sensor Network Security*, Masters Thesis, ECE Dept, Clemson University, May 2006.
15. R. R. Brooks, B. Pillai, S. Rai and S. Racunas, ''Mobile Network Analysis Using Probabilistic Connectivity Matrices,'' *IEEE Transactions on Systems Man and Cybernetics, Part C*, Accepted for Publication, Nov. 2005.
16. M. Pirretti, N. Vijaykrishnan, M. Kandemir and R. R. Brooks, ''Realistic Models for Sensor Networks Using Key Predistribution Schemes,'' *Innovations and Commercial Applications of Distributed Sensor Networks Symposium*, Bethesda, MD (October 2005)
17. Sara Baase, *Computer Algorithms – Introduction to Analysis and Design*, Addison Wesley, New York, 1988.
18. R. R. Brooks, P. Y. Govindaraju, M. Pirretti, N. Vijaykrishnan and M. Kandemir, ''On the Detection of Clones in Sensor Networks Using Random Key Predistribution,'' Submitted for review.
19. M. Barborak, M. Malek and A. Dahbura, ''The Consensus Problem in Fault-Tolerant Computing,'' *ACM Computing Surveys*, vol 25, no. 2, pp. 171–220, June 1993.
20. R. R. Brooks and S. S. Iyengar, *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall PTR, Upper Saddle River, NJ, 1998.
21. R. R. Brooks, D. Friedlander, J. Koch and S. Phoha, ''Tracking Multiple Targets with Self-Organizing Distributed Ground Sensors,'' *Journal of Parallel and Distributed Computing Special Issue on Sensor Networks*, vol. 64, no. 7, pp. 874–884, August 2004.
22. John R. Doucear, ''The Sybil attack,'' *In Proc. of the IPTPS02 Workshop*, Cambridge, MA (USA), March 2002.
23. Harney H. and C. Muchenhirn, ''Group Key Management Protocol (GKMP) Specification,'' RFC 2093, July 1997.
24. T. Hardjono, B. Patel, and M. Shah, ''*Intra-domain group key management protocol*.'' IETF Internet draft, September 2000.