01 Apr 2007

Energy-efficient Group Key Management Protocols for Hierarchical Sensor Networks

Biswajit Panja

Sanjay Kumar Madria
Missouri University of Science and Technology, madrias@mst.edu

Bharat Bhargava

Recommended Citation

B. Panja et al., "Energy-efficient Group Key Management Protocols for Hierarchical Sensor Networks," *International Journal of Distributed Sensor Networks*, Taylor & Francis Group, Apr 2007.
The definitive version is available at https://doi.org/10.1080/15501320701205225

International Journal of Distributed Sensor Networks, 3: 201–223, 2007

ISSN: 1550-1329 print/1550-1477 online
DOI: 10.1080/15501320701205225

Taylor & Francis
Taylor & Francis Group

Energy-Efficient Group Key Management Protocols for Hierarchical Sensor Networks

BISWAJIT PANJA and SANJAY MADRIA

Department of Computer Science, University of Missouri-Rolla, MO

BHARAT BHARGAVA

Department of Computer Science, Purdue University, West Lafayette, IN

In this paper, we describe a group key management protocol for hierarchical sensor networks where instead of using pre-deployed keys, each sensor node generates a partial key dynamically using a function. The function takes partial keys of its children as arguments. The design of the protocol is motivated by the fact that traditional cryptographic techniques are impractical in sensor networks because of associated high energy and computational overheads. The group key management protocol supports the establishment of two types of group keys; one for the nodes within a group (intra-cluster), and the other among a group of cluster heads (inter-cluster). The protocol handles freshness of the group key dynamically, and eliminates the involvement of a trusted third party (TTP). We have experimentally analyzed the time and energy consumption in broadcasting partial keys and the group key under two sensor routing protocols (Tiny-AODV and Tiny-Diffusion) by varying the number of nodes and key sizes. The performance study provides the optimum number of partial keys needed for computing the group key to balance the key size for security requirements and the power consumption. The experimental study also concludes that the energy consumption of SPIN [9] increases rapidly as the number of group members increases in comparison to our protocol. Similarly the pre-deployed key approach requires more communication time in comparison with this protocol. We have implemented this protocol using MICA2 motes and repeated most of the experiments which are done in simulation and we found out that the obtained results are very close to the observations made using the simulator.

Keywords Group Key Management; Hierarchical Sensor Networks; Partial Keys

1. Introduction

Sensor networks [1, 6] have become an important area of research because of their applications in military and disaster relief. The most limiting factors of a sensor node are its battery capacity and available memory. Thus, the energy and storage conservation are two important issues at the node level and at the network level.

Security is one of the most important issues in distributed ad hoc sensor networks. For example, a wireless sensor network uses a radio frequency (RF) channel [20], which is not secure. It is also difficult to prevent an adversary sensor node from compromising the security of sensor networks because of untraceable sensor nodes and less physical protection. To control information access in a hierarchical sensor environment, only authorized sensors should have the cryptographic keys by which they can decode the disseminated

Address correspondence to Biswajit Panja, Department of Computer Science, University of Missouri-Rolla, Rolla, MO 65401; E-mail: bptfc,madrias@umr.edu

information. Thus, a group key management is required for such a hierarchical environment as it can implement different access control policies at each level and provide mechanisms for secure group communication by eliminating the compromised nodes. In the traditional cryptographic techniques for security, every sensor node would need a {private, public} key pair which is impractical because of high energy consumption and scalability.

In this paper, we propose a group key management protocol using a hierarchical architecture consisting of different groups with a unique group key. Using this approach, multi-level security can be achieved to secure the group of sensors at different levels. There are two types of group keys: intra-cluster, and inter-cluster. The intra-cluster group key is used for encryption/decryption of messages inside a group of sensor nodes, whereas the inter-cluster group key is used among groups of cluster heads. In literature, there are two approaches [10, 12] for group key management: centralized and decentralized. In the centralized approach, a key is distributed among a group of sensor nodes by a trusted third party (TTP) [5]. In the decentralized method, a group member is selected for computing and distributing the key to other members. The centralized approach is simple but more vulnerable to attacks [12]. TTP provides a single point of attack for both security and fault intolerance bottleneck. In the centralized approach, to compute the group key, a centralized key server needs to be present for every subset of the group. Thus, it is not practical to implement a TTP in dynamic sensor networks. Therefore, a decentralized key distribution approach more appropriate and is adopted in this paper.

Most of the security requirements of Ad-Hoc/sensor networks are achieved using pre-deployed keys [1, 4, 11]. Matt et al. [1] showed the total number of keys that are to be generated for groups in sensor networks of different sizes. According to their scheme [1] as the network size grows, the number of needed pre-deployed keys increases exponentially. Therefore, the memory requirement increases with the number of nodes. But the memory is not sufficient to store many pre-deployed keys using smart sensor nodes [9], which makes the pre-deployed key approach impractical.

The two most important advantages of dynamic partial keys over pre-deployed keys are that

1. sensor nodes need not store too many keys and
2. the dynamic key may not be compromised because it changes frequently.

In our proposed scheme, every sensor node in a group generates a partial key dynamically, and used it for computing the group key in a bottom up fashion. Once the sensor network is deployed, it is organized in a hierarchical fashion. After that, a cluster head (leader) gets information from all of its group members to know their position (level, location in the sub-tree) and in response it sends a message. It also requests the leaf nodes (initiator) to compute their partial keys by generating random numbers as their partial keys, because they have no descendents. In this protocol the partial keys are computed using the function associated with each node and the partial keys of their descendents as arguments. The function is expressed as $f(k_1, k_2) = \alpha^{k_1 \oplus k_2} \bmod p$, where p is the prime number, α is the primitive root of P, and k_1, k_2 are the partial keys (k_1, $k_2 < p$). The function uses the partial keys of its children as the argument *f(partial key of child1, partial key of child2)*. The cluster head computes the group key using an optimum number of partial keys. The decision for choosing a number of partial keys is based on the key size for the security requirements and the corresponding energy consumption.

In this paper, we modified the Tree Based Group Diffie-Hellman (TGDH) protocol [12] for group key management using a general tree structure. In addition, the Diffie-Hellman protocol [14] is used for computing the group key. Using the modified TDGH, a new group key management scheme is presented. The experiments, with a sensor network

environment created in NS-2[16] and TinyOS [22] are performed using two sensor routing protocols (Tiny-AODV and Tiny-Diffusion) by varying nodes, key sizes, and energy consumption. It is observed that Tiny-AODV is slower than Tiny-Diffusion in terms of broadcasting the partial keys and the group key. Also, the Tiny-AODV takes more time to re-establish a route and re-send the partial keys. The experiments for energy computation and delivery of partial keys helped in selecting the optimum partial key and group key sizes. It is observed that a 300-bits group key size would be reasonable considering associated memory and communication overheads. In the experiments, the optimum (energy, security) group key size is 300 bits, which can be computed from 15 partial keys of 20 bits each. It is known [15] that decrypting 300-bits group key needs 2^{300} micro seconds, with the decryption rate of 1 bit per micro second.

We performed experiments to compute the energy consumption concluded that the protocol consumes a very small amount of energy (approximately 0.245 joule) in the process of computing 15 partial keys, broadcasting them, computing, and broadcasting the group key. The energy consumption is very small compared to the total available energy of 4,61700 joule [20] (for 15 nodes with two batteries each having 15,390 joule per battery). The proposed protocol conserves energy and communication with respect to SPIN [9] and pre-deployed key protocol. To save communication cost and energy usage, the re-keying of the group is done by the cluster head by sending a message to the sensor nodes, which contains information for adding or removing certain partial keys to generate the new group key. To guarantee that all the nodes in a group received the information, they send a reply (REP) message. If the cluster head does not get the REP from every node, it re-broadcasts.

2. Related Work

Kim et al. [12] proposed a group key agreement protocol called the Tree Based Group Diffie-Hellman protocol [TGDH], which is based on the Diffie-Hellman key exchange. In this protocol, a distributed key agreement has been considered rather than a centralized group key agreement. Different group agreement protocols have been proposed. In a centralized group key distribution, one key server generates keys and distributes them to the group, while in the decentralized approaches the key is computed dynamically. The basic requirements for group key agreement protocols [1, 10, 11] are key freshness, group key secrecy (forward and backward), and key independence.

Kong et al. [11] addressed the security in mobile ad-hoc networks. In this paper, secret shares of the key are distributed among the hosts, so that no certification authority is needed to implement the robustness. In the papers [11, 13], the idea of threshold secret sharing and secret updates have been used. No particular entity in the network holds the whole key; instead each holds part of the key. In the approach [11], the authors considered only one-hop networks where K nodes are present. The K entities jointly provide the system security. The number of intruders has to be K in a group in order to get the key to attack the system. Secret share updates are used to maintain the freshness of the key. By periodically changing the key, the system achieves more resistance from the intruder or eavesdropper. The disadvantage of this scheme[11] is using a TTP at the time of deployment, which is a single point of failure.

In SPIN [9] two security concepts are used: SNEP (Secure network encryption protocol) and micro TESLA (Time, efficient, streaming, loss-tolerant authentication protocol). The advantages of SNEP [9] are low communication overhead and semantic security. A DES-CBC chaining algorithm is used to maintain data confidentiality in SPIN, and a MAC is used to keep messages unaltered. A special counter is used to maintain the

sequence of messages. The counter value will never be the same, so the encrypted message is different for the same data. The disadvantages of SPIN are that

1. it is based on one-to-one communication but nodes in sensor networks work in groups and
2. it does not consider the security in hierarchical structure and clustering which are important for sensor network applications such as in military.

Eschenauer et al [4] presented a key management scheme which has selective distribution and revocation of keys in sensor nodes. Their scheme is based on the probabilistic distribution of the key, which guarantees that two neighboring nodes will have at least one common key in their key ring. This key is used by the neighboring nodes to encrypt/decrypt messages. The disadvantage of this approach [11] is that, pre-deployed keys are comparatively easy to forge than a dynamic key. Also, each node needs more memory to store many keys, and therefore, using it is impractical for sensor nodes.

3. Hierarchical Sensor Network Model

In this section, the hierarchical architecture of a sensor network with multiple levels consisting of sensor nodes, cluster heads, and relay nodes is described. There are two types of sensor groups: one is a group of sensor nodes lead by a cluster head, and the other is a group of cluster heads with one cluster head acting as the head of that group. Figure 1 shows the architecture, and Fig. 2 shows the group of cluster heads. This self-reconfigurable [8] sensor network can reconfigure the network according to the requirement for sensing coverage. In this model, each sensor group collects data from a particular geographical area and sends the data to the nearest sensor nodes. If the neighboring nodes are relay nodes, they forward those data using the appropriate routing path. Finally, the cluster head aggregates the data and forwards that to its upper level cluster head.

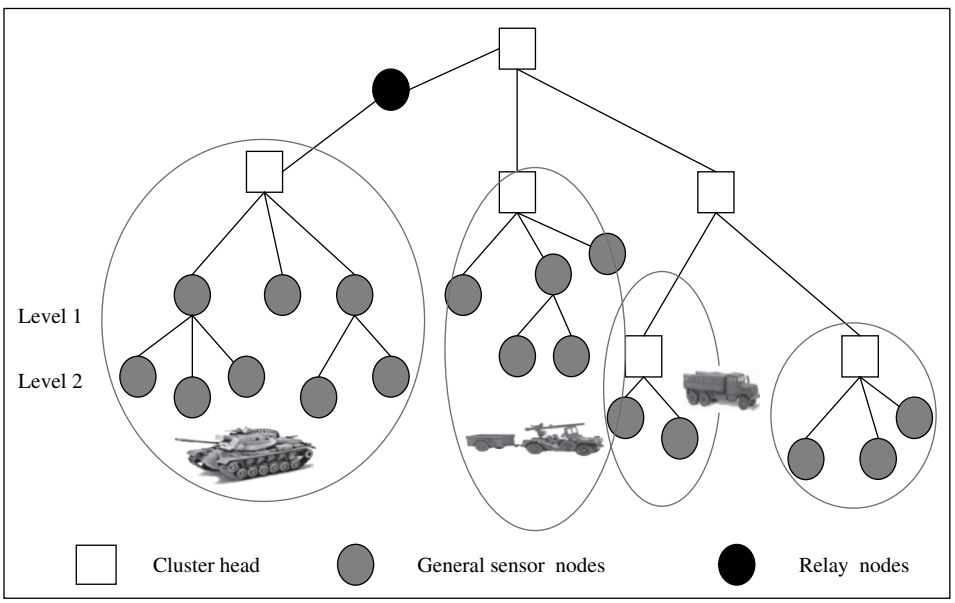

FIGURE 1 Hierarchical architecture.

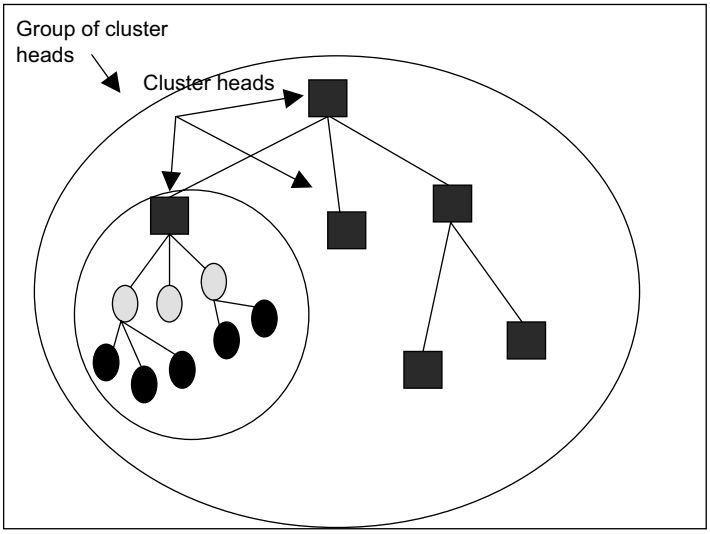

FIGURE 2 Group of cluster heads.

Three different types of identifications are used in this model: a unique identification (ID) for each sensor node, each cluster head, and each group of clusters. The assignments of the IDs are done in the following ways:

- The IDs of the sensor nodes are given by the cluster head of that particular group.
- The IDs of the cluster heads are given by the Head of Cluster Heads [HCH] of a particular geographical area. HCH is the cluster head, which is responsible for leading the group of cluster heads.
- Also, the IDs of the group of sensor nodes are given by HCH.

The identification of the nodes, cluster heads, and group of sensor nodes depends on the level (as shown in Fig. 1) and location. The sensor nodes, those at level one would be <1, 0>, <1, 1>, <1, 2>, etc. For level two, the IDs would be <2, 0>, <2, 1>, etc. When a sensor node joins or leaves a group the identification has to be re-assigned.

4. Partial Key Computation

In this section, we describe the scheme for computing the partial key in each sensor node. For this purpose, we assume that after the self-organization, the cluster head of a particular group finds out the position ($Pos_{<l,v>}$) (l is the level of node and v is the position of the node from the left in the sub-tree) of its group members. Then the cluster head sends a message ($Rep_{<init,node>}$) to the leaf nodes to compute the partial keys. The function *f(partial key of child1, partial key of child2)* is used to compute the partial keys in each node. As the leaf nodes do not have any decedents, they generate random numbers as their partial keys. Then, it uses a simple approach to compute the partial keys of non-leaf nodes. The parents of the leaf nodes compute their partial keys using a function *f()*. The function is $f(k_1, k_2) = \alpha^{k_1 \oplus k_2} \bmod p$, where p is the prime number, α is the primitive root of P, and k_1, k_2 *are* the partial keys ($k_1, k_2 < p$). The arguments of the function are the partial keys of their children. Then, using a bottom up approach, all non-leaf sensor nodes can compute their partial keys.

If the tree is binary, then the function for computing the partial key K is $f(K_{<l+1,\ 2v>}\ K_{<l+1,\ 2v+1>})$, where l is the level in the tree, and v is the position of the node from the left. For example in Fig. 3, to compute the $K_{<2,0>}$ we need the function $f(K_{<3,0>}\ K_{<3,1>})$. The computation of $K_{<2,1>}$ is not possible using $f(K_{<3,2>}\ K_{<3,3>})$ as children now have different parents. If the function $f(K_{<l+1,\ 2v+1>}\ K_{<l+1,\ 2v+2>})$ is used then we are able to compute $K_{<2,1>}$. From this we observe that the level l does not change, but the position v changes depending on the number of children. In a non-binary tree, for calculating a key for the parent node, the algorithm needs to count the siblings in the left part of the sub-tree. It then computes m where $m = siblings - 2$ so the function would be $f(K_{<l+1,\ 2v+m>}\ K_{<l+1,\ 2v+1+m>})$.

The purpose of the function f is to produce new partial keys using the partial keys of its children. The function f must have the following properties:

1. f can be applied to a block of data of size which can be handled by sensor nodes.
2. f produces a fixed length partial key.
3. $f(x, y)$ should be easy to compute in sensor nodes.
4. For any given integer value t, it is computationally infeasible to find x, y such that $f(x, y) = t$.

The first three properties are required to guarantee that the sensor networks are able to process the data and operations, because the sensor nodes contain less processing, storage, and communication power than distributed and mobile ad-hoc networks. The function should produce a fixed length partial key to guarantee that at the time of computing the group key, the cluster head knows how many partial keys are received without any loss (since the length of the partial keys is known apriori to the cluster head). If the partial key length is a variable, then it is not possible to know if the cluster head has received all the partial keys without any loss. The fourth property is to make the partial key computation a one way function so that the child nodes cannot act as parent nodes. This also helps guard against any insider attack.

All of the functions operate using the following principles: the input partial keys are considered as a sequence of *n-bit* blocks; the input is processed using one block at a time iteratively. An n-bit partial key is produced using the bit-by-bit XOR of every partial key. This can be written as follows:

$\mathrm{Pi} = \mathrm{p}_{i1} \oplus \mathrm{p}_{i2} \oplus \ldots\ldots\ldots \oplus \mathrm{p}_{in}$ where Pi is the ith partial key, n is the number of children for that parent node, and p_{ij} is the ith node in jth sub-tree. The advantage of this function is that it is difficult to decode. When the nodes know the function $(f(k_1, k_2) = \alpha^{k_1 \oplus k_2} \bmod p)$ then they do not need to analyze the known function. The

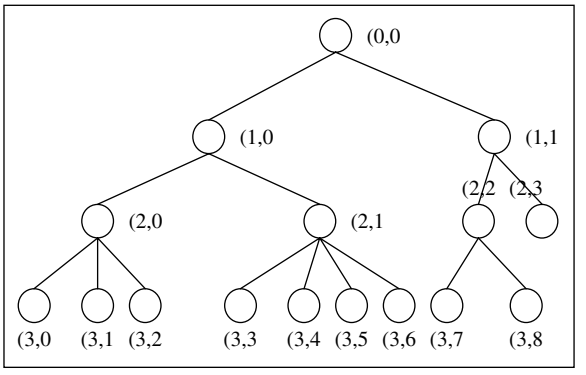

FIGURE 3 Dynamic partial key computation.

Algorithm:

1. if (tree is binary) then

2. { $f(K_{<l+1, 2v>}\ K_{<l+1, 2v+1>})$ }

3. else if (it is not binary), then

4. { $f(K_{<l+1, 2v+m>}\ K_{<l+1, 2v+1+m>})$

m = [If there is sub-tree left side of node, left siblings – (number of sub-tree)x 2 – 2] }

5. if (no sub-tree in the left side of the node) then

6. { $f(K_{<l+1, 2v>}\ K_{<l+1, 2v+1>})$ }

7. else if (sub-tree is there in the left side of the node),

then

8. { $f(K_{<l+1, 2v+m>}\ K_{<l+1, 2v+1+m>})$ }

FIGURE 4 Algorithm for dynamic partial key computation.

complexity of the protocol is $O(logn + mn)$, because the partial keys and the group key are computed in a tree structure. For n nodes, it takes $O(logn)$ time, and computing the m partial keys from n nodes takes $O(mn)$ time.

Initially, a one time symmetric key is used for generation and verification of the MAC. This key is also used for encryption/decryption of partial keys, intermediate keys and group key. Once the group key is computed, then the symmetric key is discarded. Figure 4 provides the algorithm for computing the partial keys in a tree which is based on the position of the nodes in that group.

To guarantee that the partial keys are not altered by an intruder at the time of broadcast, the following process is followed: Initially, the leaf nodes broadcast the partial keys p_{ij}. A MAC is computed using the partial keys and one time symmetric key. It is attached to the partial keys so that the parent nodes can verify the authenticity of the partial keys. The identification (*ID*), level, and position of the leaf nodes are attached with the partial keys so that the parent nodes can keep track of children for the purpose of node failure and re-organizing the network. Once the parents of the leaf nodes receive the partial keys they send a reply message (*REP*) with a timestamp as an acknowledgement. The leaf nodes also send a message which contains the approximate battery power left, so that the parent node can calculate the approximate time the leaf nodes will be alive. After that, the parent nodes compute the partial keys and send them to their ancestors. The process repeats till the root (cluster head) computes its partial key.

5. Group Key Computation

We have used the multi-party Diffie-Hellman [14] and TGDH protocol [12] to propose a new group key computation method for sensor networks. To accomplish this proposition, the leaf nodes work as the initiators and the cluster head as the leader. Starting from the initiator sensor nodes, every sensor node contributes its partial key for computing the group key. The leader node accumulates all the partial keys for computation of the group key. This is a bottom up approach, as partial keys are accumulated from leaf

nodes to the parent nodes. In the following subsection, we show the group key computation with and without using the blind factor. The blind factor is a unique number generated by a sensor node.

5.1 Group Key Computation without Blind Factor

We use the following approach for group key computation without using the blind factor [1]. As the leaf nodes act as the initiators, they first broadcast their partial keys. The parent nodes of the leaf nodes get the partial keys and then add their own partial keys and rebroadcast it. As it is a bottom up approach the cluster head will have all the partial keys, and it will compute the group key using its partial key contribution. After that, the cluster head broadcasts the group key.

Initially a pre-deployed one time symmetric key is used to encrypt and decrypt the partial keys and the group key. Thereafter, the group key is used for a similar purpose. The identification of the nodes is attached with the encrypted partial keys. The sensor nodes check the identification before decrypting the partial keys, as the parent nodes need the partial keys of their children only. In this way they can have early rejection of packets, which saves communication and computation cost. The other group members cannot compute the group key because they cannot get the partial key of the cluster head, since it does not broadcast its partial key.

In Fig. 5, the leaf nodes are $M_1, M_2, \ldots, M_9$. To start, M_1 computes the partial key g^{S1} and broadcasts it. The parent node M_{10} gets the partial keys from M_1 and other children. Here, g is a generator of the multiplicative group Z_P^* (i.e. the set $\{1, 2 \ldots p-1\}$, p is the prime) [15] and $S1$ is a randomly chosen secret number for member M_1. Likewise, member M_2 computes g^{S2} and broadcasts it, and the parent M_{10} gets the partial keys. In this way, member M_{10} receives g^{S1S2S3}, and raises the power by S_{10} to get *the intermediate key (IK)*. Here, g^{S10} is the partial key contribution of M_{10}.

In the following paragraphs, we discuss two types of group keys: the intra-cluster and the inter-cluster. The intermediate keys in M_{10}, M_{11}, and M_{12} are $IK1 = g^{S1\ S2\ S3\ S10}$, $IK2 = g^{S4\ S5\ S6\ S11}$, *and* $IK3 = g^{S7\ S8\ S9\ S12}$, respectively. The intermediate keys are encrypted

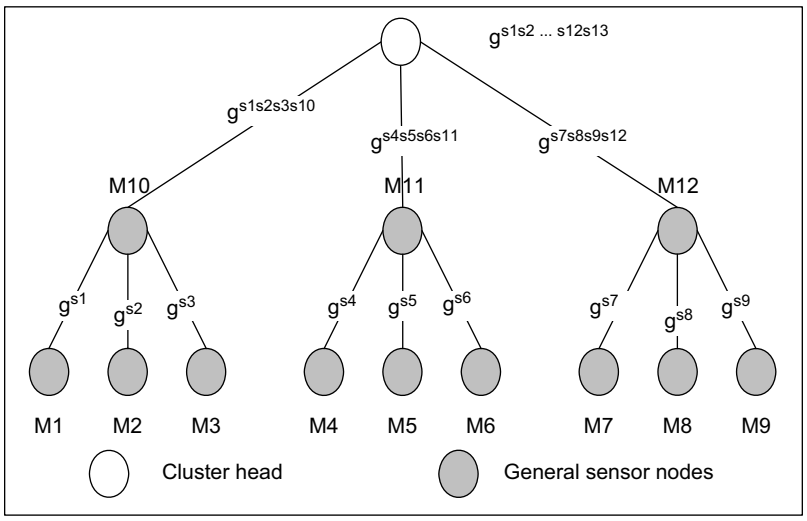

FIGURE 5 Intra-Cluster key computation.

using a one time symmetric key, as explained earlier. The cluster head computes the group key *K*, using *IK1*, *IK2*, and *IK3* and its contribution g^{s13}.

$$K = g^{S1\,S2\,S3\,S10\,S4\,S5\,S6\,S11\,S7\,S8\,S9\,S12 s13}\,[intra-cluster\ group\ key]$$

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

For inter-cluster encryption/decryption a different group key is computed. The inter-cluster group key is not known to the general sensor nodes. Figure 6 shows the computation of the inter-cluster group key. The intermediate key in C_7, C_8, C_9 are $IK1_{inter} = g^{C1C7}$, $IK2_{inter} = g^{C2C3C8}$, and $IK3_{inter} = g^{C4C5C6C9}$.The head of the cluster heads (HCH) computes the inter-cluster group key C using intermediate keys $IK1_{inter}$, $IK2_{inter}$, $IK3_{inter}$, and its contribution g^{c10}.

$$C = g^{C1C7C2C3C8C4C5C6C9c10}\,[inter-cluster\ group\ key]$$

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

5.2 Group Key Computation using Blind Factor

We can compute the group key using a blind factor. The advantage of using a blind factor is that an attacker will not be able to get the group key when the cluster head broadcasts the group key. In Fig. 5, the intermediate keys are $IK1 = g^{S1\,S2\,S3\,S10}$, $IK2 = g^{S4\,S5\,S6\,S11}$ and $IK3 = g^{S7\,S8\,S9\,S12}$. After computation of *IK1, IK2*, and *IK3*, the parent nodes $M_{10,}$ $M_{11,}$ M_{12} broadcast the intermediate keys. The children of $M_{10,}$ $M_{11,}$ and M_{12} are interested in those keys, as they need to remove their contribution from the *IK*. Then they insert a randomly chosen blind factor *B*. The keys, after inserting the blind factor, are as follows.

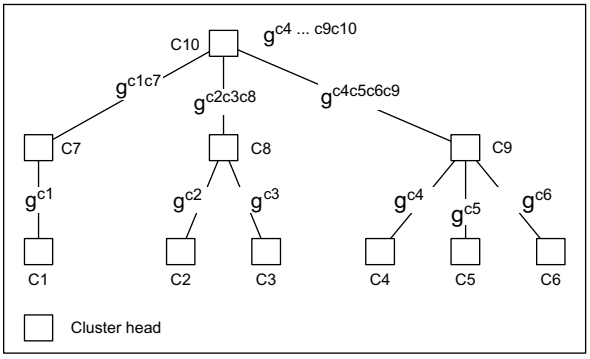

FIGURE 6 Inter-Cluster key computation.

$IKB1 = g^{B1\,S2\,S3\,S10}$, IKB2 = $g^{S1\,B2\,S3\,S10}$, and IKB9 = $g^{S7\,S8\,B9\,S12}$. The cluster head gets the broadcasted keys *IKB1,…, IKB9*. The cluster head then computes the group key *K*, using *IKB1…IKB9* and its contribution g^{s13}.

$$K = g^{B1\,S2\,B3\,S10\,S4\,S5\,S6\,S11\,S7\,S8\,S9\,S12S13}$$

After the group key computation, the cluster head broadcasts the group key with the blind factor. Now the authentic sensor node can recognize its blind factor. Each member unblinds its blind factor that it received from the cluster head. They reinsert their original contribution S_i *(i = 1…n)* for getting the group key. The same method is used to compute the inter-cluster group key. A symmetric key is used for encryption and decryption of partial keys. The cluster head uses the same symmetric key for encryption of the group key.

5.3 Updating Group Key

Our key management protocol provides a scalable approach for updating the group key for large dynamic groups. Section 3 shows that as groups become large and dynamic, re-keying the group on each membership change becomes unsustainable. One of the approaches to keep the key fresh is by re-keying the group key at some fixed intervals; this approach is computationally expensive as the partial keys and group key will be computed again. Another approach for updating the key would be to send a message from the cluster head to its group members consists of instructions to remove or add a certain partial key from the group key in order to get the new group key. The group key *K*, which is described in Section 5.1 is:

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

For example, the cluster head sends an encrypted message to its group member for removing *S10* from their group key. The new group key *K* would be:

$$K = g^{S1\,S2\,S3\,S4\,S5\,S6\,S11\,S7\,S8\,S9\,S12s13}\,[New\ group\ key]$$

To guarantee that all the sensor nodes received the message to update the part of the group key, nodes send an acknowledgment *(ACK)* message to the cluster head. If the cluster head receives the reply *(REP)* from all the nodes, then it sends the next message that "the new group key is in effect now." If the cluster head does not receive the reply message *REP* from all its group members, then it resends (broadcast) the message until it can get a reply message or it can come to know events, such as a particular sensor node does not have battery power left for communication. To keep track of all the partial keys, the cluster head can use a primary index by sorting the IDs of its group member nodes.

The HCH keeps track of all the partial keys of its group members and of all the cluster heads. Likewise, the cluster head of each group keeps track of the partial keys of its group members, so that it can send a message for removing or adding a certain part from the group key. To confuse intruders, instead of sending the information of the partial key directly, an associated message is sent, so that if an intruder wants to get the group key, it has to first decode the information to know the information associated with the key and then it needs to get all other partial keys.

5.4 Authentication of Nodes

When the partial keys (p_{ij}) are computed, an intruder can listen to the messages. But it will not be able to compute its partial key, as it may not know the function, since the functions are not broadcasted. To change the function frequently, we compute a new function after a certain period of time. For computing the new function, we use the previous function to guarantee that the new function in every node is the same. This way it can protect the network from an attacker whose intension is to compute the partial keys and act as an authentic group member. To prevent an intruder node from modifying partial keys, a MAC is used along with the partial keys to guarantee that the partial keys are from authentic nodes. Message→ MAC + Message, where *MAC = Hash(Message, ID, location, key*). Initially, the symmetric key, and later the group key are used for computing and verifying the MAC.

5.5 Analysis: Energy and Security Level with Respect to Key Size

This subsection explains the analysis of balancing the energy consumption verses security level by choosing the appropriate key sizes in the proposed protocol. See Table 1 for the notations used.

Let $L_f \in L_{vt}$ and $L_v = 1, 2, \ldots\ldots\ldots n$ [$L_v = 0$ is the root]. The parent key K_p is computed from the function $f(K_{lf}, K_{lf+1})$ where $f(k) = \alpha^k \bmod p$. The energy remaining in each sensor node is computed using their levels in the architecture. It is known that the energy consumption in a circuit is: Power $P = V^2 .f. C$

If the power (energy) consumption at each level is P then the total power in all groups is:

$$\text{Total power} = \sum P \times L_v \times N_g$$

$$\text{Energy remaining} = P_{total} - \sum P \times L_v \times N_g$$

The energy required at each level will compute the total energy needed in the computation of the group key, and is given by $\int_1^n L_v = \sum P \times L_v \times L_n$

TABLE 1 Notations

L_f	Leaf level nodes in the hierarchy	P_{total}	Total energy in the network (joule)
L_{vt}	Leaf level	$P_{consumed}$	Consumed energy for group key computation
K_{lf}	Partial keys in the leaf nodes	G	Group of sensors in the network
P_k	Pre-deployed key in the network, can be symmetric or asymmetric	N_g	Number of groups in the network
K_p	Parent key of the sub-tree	S	Size of key
P_n	Parent nodes of each sub-tree	L_n	Number of levels in each group
L_v	Levels in the hierarchy as in Fig. 1	C_n	Number of partial keys considered for the group key
$f()$	Function with the partial keys of children as arguments	V	Battery voltage (volt)
E	Energy consumption (joule)	C	Capacitance (farad)
		f	Frequency (Hz)

By considering the partial keys at each node level and at their parent level energy consumption would be:

$$\prod_{Lf}^{Pn}(L_v \le K_p \times L_n) \;\; \exists K_p \in k_{lf} + P_n$$

If n partial keys are considered to form the group key then the energy consumption with respect to key size would be: $P_{consumed} = (K_{lf} + K_p) \times C_n \times E$

The energy consumption for the group key must be less than the total available energy. Thus, $(K_{lf} + K_p) \times C_n \times E \le P_{total} \le E \times N_g$

It is assumed that there exists more than one group in the network. If the partial key size varies from 20 to 100 bits then the energy consumption would be:

$$E = \sum_{s=20byte}^{100} (K_{lf} + K_p) \times C_n \times S$$

The partial key sizes are based on the group key size, the number of partial keys considered for the group key, and the available energy. The energy consumption for computing the partial keys and the group key proportional to the security requirement in terms of key sizes, can be expressed as follows:

$$\forall C_n \in G,$$

$$\exists K_p, K_{lf} \mid C_n \times (K_p + K_{lf}) \times E \propto Sec$$

Where *Sec* is the time to the decrypt the key at a decryption rate of 1 bit per micro second. It is observed that, as the key size increases the security and energy consumption would increase, and it is not possible to have $P_{consumed} \ge P_{total}$.

If $\dfrac{(L_f \times L_n \times E) \cong (L_{vt} \times L_n \times E)}{P_{consumed}} \le 1$, then the cluster head can choose the partial keys from the leaf nodes. From the above, we can find the number of partial keys to be used for balancing the security level based on key size and the corresponding energy consumption. Table 2 gives 3 cases where the number of partial keys and their sizes are varied to observe the variation of energy consumption using the given formulas. These cases are used in the simulation. Based on the analytical model and later in section 6 it is confirmed by the simulation that 15 partial keys would be optimum for the group key in a group of 100 nodes. The group key size ranges from 300 to 450 bits.

To see how close the result of this analysis is with the simulation (Section 6), we take the following example. If 15 partial keys are used for the group key construction then theoretically the group key computation scheme should consume 0.15534 joule, as each

TABLE 2 Balancing energy and the key size

Examples	No of partial keys	Key size (bits)	Energy consumption
Case-1	n	$n \times 20$	$\Sigma E \times n$
Case-2	n + 10	$n \times 30$	$\Sigma E \times (n + 10)$
Case-3	n + 20	$n \times 40$	$\Sigma E \times (n + 20)$

TABLE 3 *Parameters for simulation*

Routing Protocols	Tiny-AODV,Tiny-Diffusion
Area	2000×2000 meter
Number of nodes per group	50
Channel	Single (wireless)
Simulation time	160 sec
Transmitting power	0.175 mW
Receiving power	0.175 mW
Idle	0.0 W
Initial energy	0.5 Joule
Sensing power	0.00000175 mW

partial key computation takes 0.010356 joule. In the simulation results we observed that the total energy consumption is 0.245 joule. The reason it takes more energy because some energy is involved in the routing, network settling time, and in re-broadcast, which are ignored in the analytical analysis. (Table 3)

6. Performance Evaluations and Observations

For performance analysis, we have implemented the group key management protocol in TinyOS[22] and NS-2. To create a sensor network environment in NS-2 [16], we plugged in sensor agent, energy model, and multi-channel model developed by the United States Navy [19]. The experiments are performed using two sensor routing protocols Tiny-AODV [22] and Tiny-Diffusion [23] by varying the number of nodes and partial key sizes.

We have developed an application-layer approach for generating and broadcasting the partial keys. The partial keys are generated using a random number generator. The sensor nodes use the UDP transport agent to broadcast the partial keys to the appropriate cluster head. The cluster head computes the group using the method as described in Section 5. It chooses a certain number of partial keys in order to compute the group key. It also stores the identification of the nodes along with the partial keys so that it can save memory by discarding the partial keys received earlier.

In order to measure how fast the partial keys are delivered to the cluster head, we did experiments using two sensor routing protocols: Tiny-AODV [22] and Tiny-Diffusion [23]. The sensor nodes, which do not fall in a 10 meter range, generate partial keys so that the cluster head can get the partial keys from different regions in its group. Then, they establish a route to the cluster head using the routing protocols and send the partial keys using that route. The reasons for restricting the number of sensor nodes and broadcasting the partial keys are to reduce communication overhead, and energy consumption. Figure 7 shows the time taken by the network to send the first 25 partial keys to the cluster head using the Tiny-AODV and Tiny-Diffusion routing protocols. The algorithm for generating and broadcasting partial keys and computing the group key was kept the same in both cases.

We observed that the Tiny-AODV takes approximately 54 seconds to deliver the first 25 partial keys to the cluster head; whereas Tiny-Diffusion takes almost 15 seconds. This experiment is performed to determine the effectiveness of routing protocols in terms of how fast the partial keys can reach the destination. After that, the group key is computed in the cluster head. Then it is broadcasted to all its group members. We found that both the routing protocols take a comparably small amount of time for broadcasting partial keys.

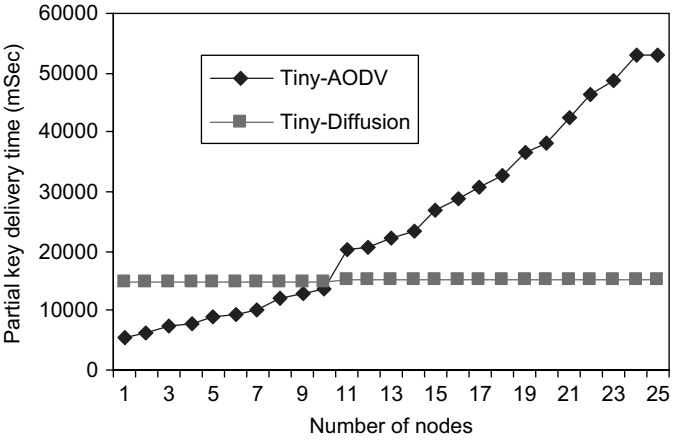

FIGURE 7 Partial key delivery to the cluster head.

Figure 8 shows the time taken by the network to broadcast the group key generated based on the collection of partial keys using the Tiny-AODV and Tiny-Diffusion routing protocols. After obtaining the result from the experiments, it is observed that the group members with IDs 1 through 25 receive the group key in approximately 7.57 seconds if Tiny-AODV is used; whereas if we use Tiny-Diffusion then it takes approximately 11.82 seconds. The reason is that, Tiny-Diffusion uses a routing table for creating routes and therefore, it takes more time to deliver the initial packets compared to Tiny-AODV, because of its proactive features [17]. There is a very small time difference in broadcasting the group key using Tiny-AODV and Tiny-Diffusion. Thus, delivering the partial keys (as shown in Fig. 7) to the cluster head using Tiny-AODV takes more time than Tiny-Diffusion because of the partial key broadcast from every node, and then delivering it to the cluster head requires many-to-one communication. On the other hand as Tiny-Diffusion uses flooding, all the nodes in a group have the knowledge about all its group members. In Tiny-AODV the cluster head is unable to get all the partial keys in one route discovery as the routes are updated frequently. The Tiny-Diffusion

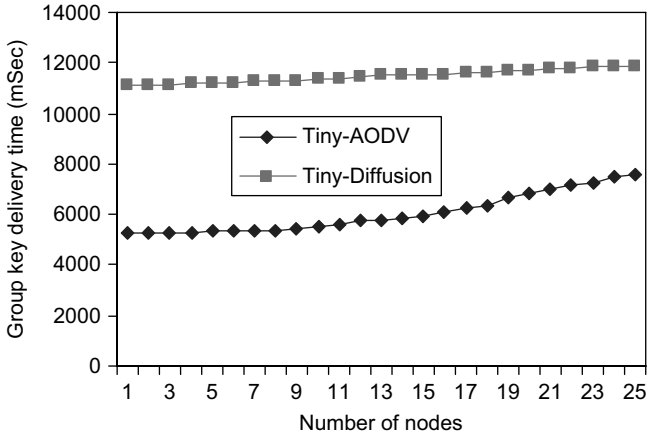

FIGURE 8 Broadcasting group key.

takes more time for broadcasting the group key as it uses time to create a routing table and then only it can deliver the group keys. After observing both the scenarios, we selected Tiny-Diffusion for our key management protocol for delivering keys, as it has better overall performance.

In Fig. 9, we present the results for time for broadcasting partial keys of size 20 and 30 bits. The group key in the cluster head is computed using the first 15 partial keys. The group key size are 300 and 450 bits, respectively. We observed that the time taken to accumulate partial keys of size 300 and 450 bits remains almost same. Theoretically, the time required to decrypt 300 bits key size is 2^{300} micro seconds, at a rate of 1 bit decrypt per micro second. Therefore, the forgery time for a group key size of 300 bits is quite high.

We performed another experiment to analyze the energy consumption in the proposed model. As a reminder, the power consumed by the sensor nodes for transmission and reception is set at 175 mW, for sensing 0.00000175 mW, and the initial energy of the general sensor nodes is kept at 0.5 joule. The cluster head consumes the same power for transmission and reception, but its initial energy is set to 2.5 joules. Figure 10 shows the

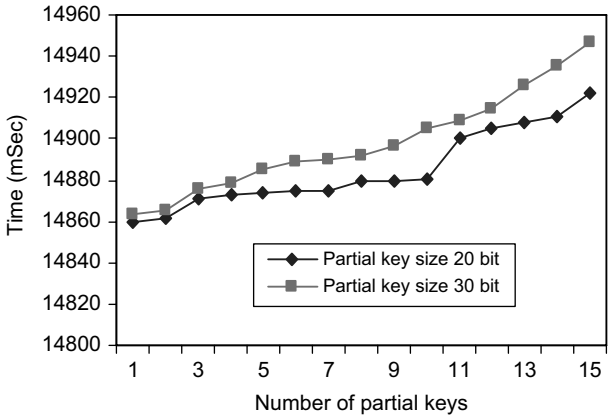

FIGURE 9 Time taken for different key sizes.

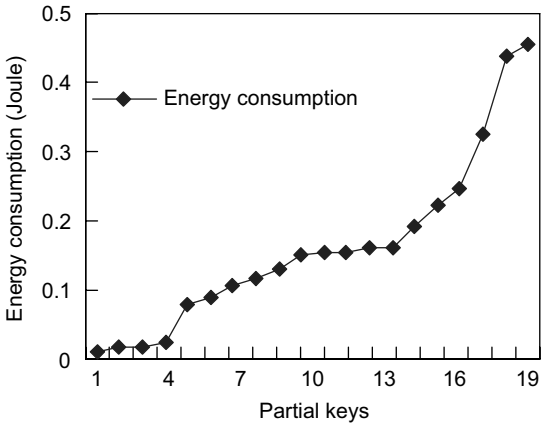

FIGURE 10 Energy consumption verses the number of partial keys.

energy consumption graph for generating 20 partial keys and delivering those to the cluster head. This also includes the energy consumption for communication before delivering the partial keys. From this experiment we observe that changing the number of senders has an impact on the energy consumption, and this experiment helps to decide the number of partial keys that should be chosen in order to balance the energy consumption and security. The optimum number of partial keys is 15, with respect to the current configuration. Although the total power consumption for generating partial keys, delivering, computing the group key, and broadcasting back which is not shown in the figure, it is approximately 0.245 joule.

One way to protect the network from intruders is by updating (re-keying) the group keys frequently so that an intruder cannot get enough time to forge a group key. In our model, we update the group key using the technique described in Section 5.3. As explained earlier, in order to save communication and computation cost, instead of re-computing the partial keys and group key, the cluster head sends a message to its group members for removing or adding certain partial keys from the group key to obtain the new group key. In Fig. 11, the time taken by the network to broadcast a message for re-keying the group key for 100 nodes is approximately 3.1 seconds. From this, we conclude that re-keying the group using this method is less time consuming than re-computing the group key.

In Fig. 12, the energy consumption by SPIN [9] and our group key management protocol are compared. Though SPIN is used for one-to-one node communication, here it is used for group communication. It is observed that SPIN takes more time for communicating within a group. We observed that the energy consumption of SPIN increases exponentially with the increase in the number of nodes in a group. Though initially the dynamic group key management protocol consumes energy exponentially, as the number of nodes increases it tends to consume comparatively less energy. The reason is that in this protocol the initial activities like dynamic partial key computation and then the group key computation are involved with significant amounts of communication and computation costs. Once the first group key is computed, then it only needs to broadcast or receive a message for refreshing the key, which is less power consuming.

From Fig. 13, it is observed that the pre-deployed keys take more time in searching the associated keys (as hundreds of keys are stored) for encryption/decryption, authentication, and communication in comparison to the dynamic key approach.

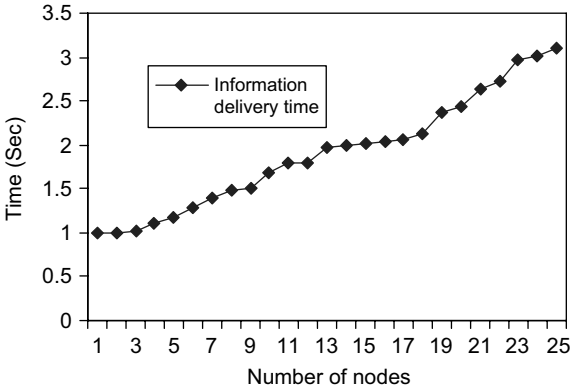

FIGURE 11 Updating the group key.

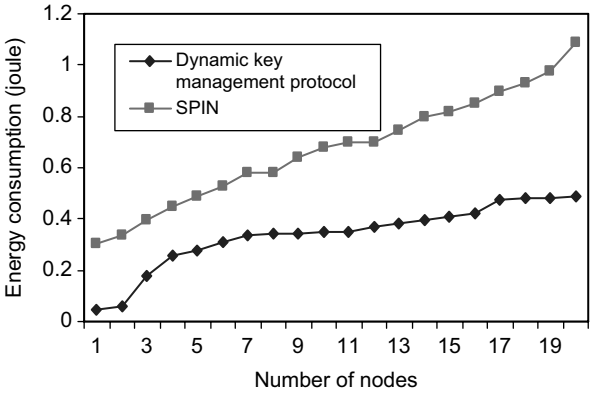

FIGURE 12 Comparison of dynamic key management protocol with SPIN.

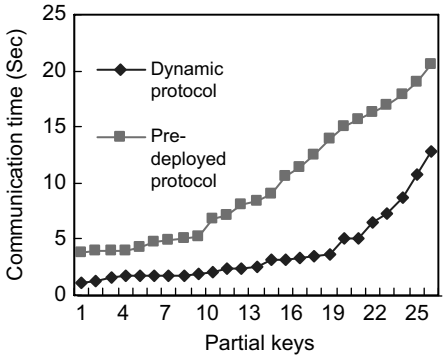

FIGURE 13 Comparison of dynamic key management protocol with pre-deployed key management protocol.

7. Hardware Implementation using MICA2 Motes

For performing experiments in the sensor network consist of MICA motes, we create a self-configurable structure of the group hierarchy as shown in Fig. 2. Thus, each node has a flag indicating whether it is a cluster head; this flag influences the behavior of the node (i.e., which messages it responds to). Each node is also pre-programmed with a list of children: this list includes the ID of each child, its state, and its partial key (if known by the parent). Additionally, each group has its own broadcast address, that is, where TinyOS [10] normally recognizes one broadcast address, in our implementation there could be several. The upshot of this is that a node can broadcast a message to everyone in its group (including the cluster head), but a cluster head can also broadcast messages to its neighboring cluster heads.

When the head cluster head (ID = 0) goes online, it periodically begins broadcasting an initialization message to all nodes in the hierarchy. Any node, upon receiving this message, will send its partial key to its parent if it has computed its partial key; otherwise, it will do nothing. Each leaf node automatically has a partial key, generated randomly. The system is structured such that each node will continue sending its partial key to its parent

until the head cluster head computes its partial key; thus, it is assured that every node will be able to compute its partial key eventually, with partial keys propagating from the leaf nodes up to the cluster heads, regardless of how many packets are lost.

As described earlier, each node records the partial keys of its children. Upon receiving a "partial key" message, the node determines which child it belongs to, and stores it in that slot. When it has received enough partial keys, it computes its own partial key, and prepares to send it upon receiving the initialization message. If the node is a cluster head, it then has enough information to compute the group key, which it does. At this point, it begins broadcasting the group key to its subordinate nodes. It continues broadcasting the group key as long as it receives messages from nodes that do not have the group key; if no such messages are received for some length of time (say, 10 seconds), it assumes that everyone has the group key, and switches to it. The reason for this method is that, in this implementation, the cluster head can only hold two keys: the regular group key, and the cluster-head group key. It cannot therefore switch to the new key until all of its descendents have received it. Further implementations may be based on new modifications to TinySec [8,10] wherein a node can communicate using an arbitrary number of keys. The current implementation simply modifies TinySec so that it switches between two keys (if the node is a cluster head) depending on whether the message is intended for the regular group or for the cluster-head group.

In order to compare memory usage for computing the group key we performed the following experiment. We used the shortest path routing algorithm to deliver the partial keys to the cluster head. Figure 14 shows the comparison of memory for the partial key sizes of 20 bits and 30 bits. The group key sizes for 5 nodes were 100 bits and 15 bits. The difference is a result of the fact that the node must store:

1. the partial keys of its children,
2. its own partial keys, and
3. the partial keys to be used in constructing the group key, all of which depend on the partial-key size.

In order to measure how fast the partial keys are delivered to the cluster head, we did experiment using the shortest path routing protocol. The sensor nodes, which do not fall in a 10 meter range, generate partial keys so that the cluster head can get the partial keys from different regions in its group. Then, they establish a route to the cluster head using

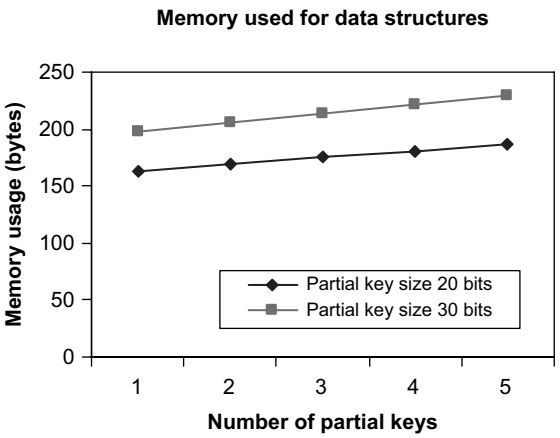

FIGURE 14 Memory usage for partial keys.

the routing protocols and send the partial keys. The reasons for restricting the number of sensor nodes and broadcasting the partial keys are to reduce communication overhead, and energy consumption. Figure 15 shows the time taken by the network to send the first 30 partial keys to the cluster. The algorithm for generating and broadcasting partial keys and computing the group key was kept same in both cases.

Figure 16 shows the time taken by the network to broadcast the group key generated based on the collection of partial keys. After obtaining the result from the experiments, it is observed that the group members whose IDs are 1 through 30 receive the group key in approximately 4.21seconds. The time, as seen by some cluster head, is measured which has elapsed between the point of having its partial key, and the point at which all nodes in the group are reported to have the group key, versus the number of nodes in the group. Note that this includes the time taken to compute the group key, as well as the time taken to ensure that all nodes have the group key and, if not, to resend it.

In Fig. 17(a,b), we present the results for time consumption for broadcasting partial keys of size 20 and 30 bits. The group key size would be 100 and 150 bits, respectively. We observed that the time taken to accumulate partial keys of size 100 and 150 bits

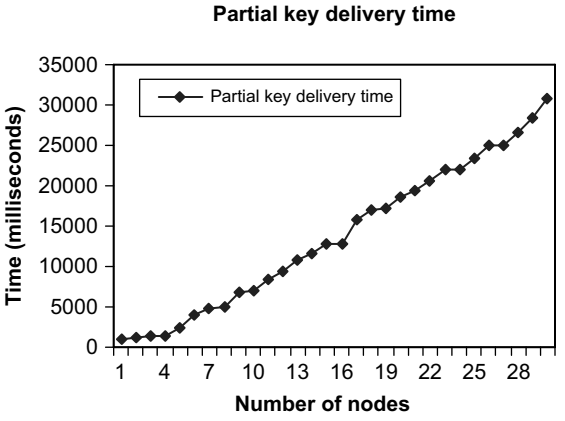

FIGURE 15 Partial key delivery time.

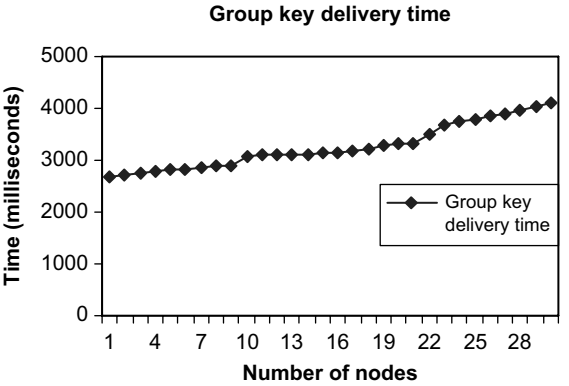

FIGURE 16 Group key delivery time.

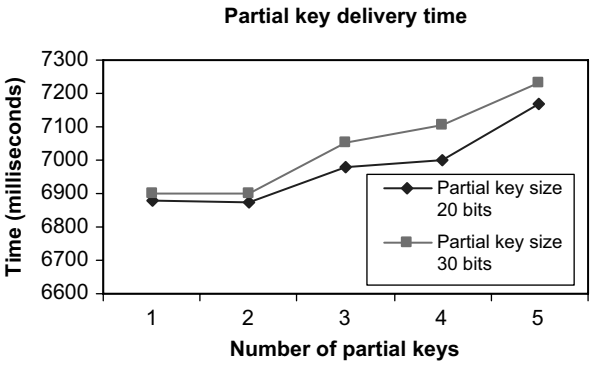

FIGURE 17 (a) Partial key delivery time by changing the size.

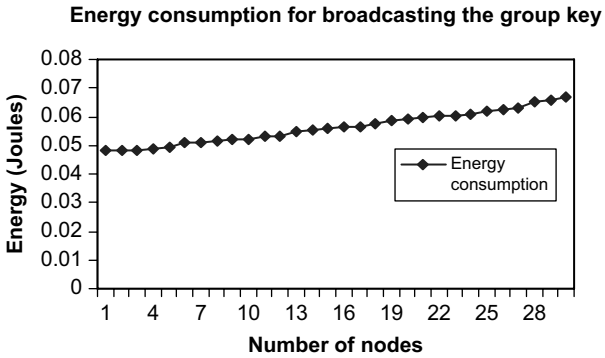

FIGURE 17 (b) Energy consumption.

remains almost same. Theoretically, the time required to decrypt 300 bits key size is 2^{300} micro seconds, at a rate of 1 bit decrypt per micro second. Therefore, the forgery time for the size of 300 bits group key is quite high.

We performed another experiment to analyze the energy consumption in the proposed model. As a reminder, the power consumed by the sensor nodes for transmission and reception is set at 175 mW, for sensing 0.00000175 mW, and the initial energy of the general sensor nodes is 0.5 joule. The cluster head consumes the same power for transmission and reception, but its initial energy is set to 2.5 joules. Figure 18 shows the energy consumption graph for generating 30 partial keys and delivering those to the cluster head. This also includes the energy consumption for communication before delivering the partial keys. From this experiment, we observe that changing the number of senders has an impact on the energy consumption, and this experiment helps to decide the number of partial keys that should be chosen in order to balance the energy consumption and key size for security.

8. Conclusions and Future Work

In this paper, we have described a group key management protocol where the partial keys are computed dynamically instead of using pre-deployed keys. A group key is generated based on the partial keys of the group members. The dynamic partial keys have some

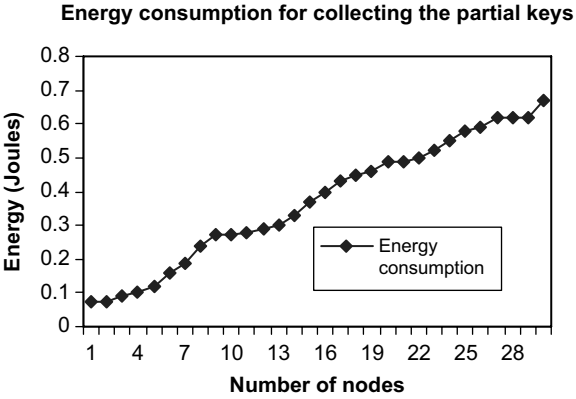

FIGURE 18 Energy consumption.

advantages over pre-deployed keys, as many pre-deployed keys need to be stored to provide secure communication, which is not feasible in sensor nodes because of the limited memory. It is easy to re-compute the partial keys as each node uses a function with the partial keys of its children as arguments to compute its partial key. Using a detailed simulation study it is shown that the proposed protocol is able to compute the partial keys and the group key within a very small time period. From experiments, we observed that the energy consumption for generating the partial keys and the group key is very small compared to the total available energy. In another experiment for comparison with SPIN [9] regarding energy consumption showed that the energy consumption of SPIN increases exponentially as the number of group nodes increase, but our proposed protocol consumes a very small amount of energy after the first group key computation. We have implemented this protocol using Mica2 sensor motes and presented the results. We observed that obtained results are very close to the simulation results.

One natural direction for future research is to model a top-down key management technique and compare it with the proposed bottom-up approach. A possible approach can be loading the sensor nodes with a pre-deployed {private, public} key pair, and instead of keeping the private key in the sensor nodes, split the key (as partial keys) using a function and distribute those among the sensor nodes in the hierarchy rooted at that node (so that an attacker will not be able to get the private key only by attacking that node). In this model, when a sensor wants to decode some data which is encrypted by other sensor nodes, it must accumulate the partial-keys to formulate its private key. The partial keys can be lost because of sensor node failures. A fault-tolerant key recovery technique is needed by which private keys can be recomputed using only some of the partial keys.

Acknowledgement

This research is partially supported by NSF grant EIA-0323630.

About the Authors

Sanjay Kumar Madria received his Ph.D. in Computer Science from the Indian Institute of Technology, Delhi, India in 1995. He is an Associate Professor, Department of Computer Science, at the University of Missouri-Rolla, USA. Earlier he was Visiting Assistant

Professor in the Department of Computer Science, Purdue University, West Lafayette, USA. He has published more than 100 journal and conference papers in the areas of mobile and sensor computing and Mobile P2P among others. He was PC Chair of Secure and Reliable Mobile workshop held in New Orleans, Oct. 2001. Dr. Madria has given tutorials on mobile computing in many international conferences. He received the JSPS (Japanese Society for Promotion of Science) invitational fellowship in 2006. His research is supported by grants from NSF, DOE, UM research board, and a grant from industry. He is an IEEE Senior Member.

Biswait Panja completed his Ph.D. in 2006 from the University of Missouri-Rolla in the area of Security in Sensor Networks. Currently he is working as an Assistant Professor of Computer Science at Morehead State University. His research interests are in secure sensor networks. While pursing his PhD at the University of Missouri-Rolla he received Outstanding Graduate Teaching Assistant, Outstanding Graduate Research Showcase, and Outstanding Computer Science Graduate student awards.

Bharat Bhargava is a professor in the Department of Computer Science with a courtesy appointment in the School of Electrical & Computer Engineering at Purdue University. Professor Bhargava is conducting research in security and privacy issues in distributed systems. This involves host authentication and key management, secure routing, and dealing with malicious hosts, adaptability to attacks, and experimental studies. In the 1988 IEEE Data Engineering Conference, he and John Riedl received the best paper award for their work on "A Model for Adaptable Systems for Transaction Processing." Bhargava is a Fellow of the Institute of Electrical and Electronics Engineers and of the Institute of Electronics and Telecommunication Engineers. He has been awarded the charter Gold Core Member distinction by the IEEE Computer Society for his distinguished service. He has several NSF funded projects. In addition, DARPA, IBM, Motorola, and CISCO are providing contracts and gift funds.

References

1. David W. Carman, Peter S. Kruus, and Brian J. Matt. Constraints and approaches for distributed sensor network security. *NAI Labs Technical Report #00–010*, September 2000.
2. H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks, in *IEEE Symposium on Security and Privacy*, Berkeley, California, May 11–14, 2003.
3. Wenliang Du, Jing Deng, Yunghsiang S. Han, Shigang Chen and Pramod Varshney. A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge. *To appear in IEEE INFOCOM*, 2004.
4. L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks, in *Proceedings of the Ninth ACM Conference on Computer and Communications Security*, Washington, DC, USA, 2002.
5. Y. Amir, G. Ateniese, D. Hasse, Y. Kim, C. Nita-Rotaru, T. Schlossnagle, J. Schultz, J. Stanton, and G. T. Sudik. Secure group communication in asynchronous networks with failures: Integration and experiments. *ICDCS 2000*, Apr. 2000.
6. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking, pp. 56–67*, Boston, MA, Aug. 2000. ACM Press.
7. Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod Varshney. A Pairwise Key Predistribution Scheme for Wireless Sensor Networks. In *Proceedings of the Tenth ACM Conference on Computer and Communications Security (CCS)*, Washington DC, October 27–31, 2003.
8. Lakshminarayanan Subramanian, Randy H. Katz. An Architecture for Building Self-Configurable Systems, *IEEE MobiHoc*, 2000.

9. A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *In Proceedings of Mobicom*, 2001.
10. Michael Steiner, Gene Tsudik, Michael Waidner. Key Agreement in Dynamic Peer Groups. *IEEE Transactions on Parallel and Distributed Systems **11** 8*, 769–780, 2000.
11. P. Jiejun Kong, Petros Zerfos, Haiyun Luo, Songwu Lu, Lixia Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks. *International Conference on Network Protocols (ICNP)*, 2001.
12. Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *ACM CCS 2000*, November 2000.
13. A. Shamir. How to Share a Secret *Communications of the ACM, 22(11):612–613*, 1979.
14. Whitfield Diffie and Martin E. Hellman. Privacy and authentication. An introduction to cryptography. *Proceedings of the IEEE*, **67** 3, 397–427, March 1979.
15. William Stallings. Network Security Essentials Applications and Standards.
16. http://www.isi.edu/nsnam/ns/
17. Charles E. Perkins and Elizabeth M. Royer. Ad hoc On-Demand Distance Vector Routing. in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, February 1999, pp. 90–100.
18. Charles Perkins. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, 1994.
19. http://pf.itd.nrl.navy.mil/projects.php?name=nrlsensorsim
20. http://www.xbow.com/Products/Wireless_Sensor_Networks.htm.
21. Sencun Zhu, Sanjeev Setia, Sushil Jajodia, LEAP: efficient security mechanisms for large-scale distributed sensor networks, *Proceedings of the Tenth ACM Conference on Computer and Communication Security*, 2003.
22. http://www.tinyos.net/
23. http://www.cens.ucla.edu/~mmysore/Design