Efficient Hierarchical Key Generation and Key Diffusion for Sensor Networks

Mohamed Shehab
School of Electrical and
Computer Engineering
Purdue University
West Lafayette, IN, USA
shehab@ecn.purdue.edu

Elisa Bertino
Department of Computer
Sciences and CERIAS
Purdue University
West Lafayette, IN, USA
bertino@cs.purdue.edu

Arif Ghafoor
School of Electrical and
Computer Engineering
Purdue University
West Lafayette, IN, USA
ghafoor@ecn.purdue.edu

Abstract— **Sensor networks are designed with the assumption that nodes are willing to collaborate. However, the open collaboration of nodes introduces privacy and security issues. Therefore, ensuring privacy in wireless sensor networks is a challenging task. Based on a multilevel security paradigm, in this paper we present a hierarchical key generation and distribution protocol for wireless sensor networks. We show by simulation results that our key generation scheme outperforms the existing hierarchical key generation schemes thus it is suitable for sensor networks with limited computation and energy capabilities. Furthermore, we present an energy efficient key diffusion protocol. We also discuss the possible security threats involved with the proposed protocol and provide suitable solutions to such threats.**

I. Introduction

Sensor networks are characterized by a large number of wireless nodes distributed over a region for sensing and monitoring purposes. One of the main assumptions when designing sensor networks is that nodes are willing to collaborate with one another to process, aggregate and forward the collected data. However, the data collected by sensors varies in security, sensitivity and importance levels, thus the collaboration of nodes introduces several security and privacy challenges [1]. For example, sensors located close to military locations produce highly sensitive data while sensors located at less critical locations produce less sensitive data. The open collaboration among these sensors can cause privacy violations. Therefore, security techniques should be in place to prevent security breaches by managing the collaboration among the sensor nodes.

In order to address this problem, we propose an approach based on a multi-level security model [2]. The proposed approach assigns each sensor an access class from a partially ordered set of access classes, thus resulting in a hierarchical organization of sensors. The security class of each sensor is decided based on the contextual information of both the sensor and the collected data. The hierarchical access structure represents the sensor network security policy and controls the data flow among the sensor nodes. To implement the proposed approach, several issues have to be addressed. An important requirement is that hierarchical keys should be used for data encryption. This allows nodes at high access classes to decrypt data sent by lower access classes. The generation and distribution of the hierarchical key should be suitable for the sensor network environment. Other issues include the design of special data retrieval, data fusion and querying techniques that adhere to the hierarchical access control paradigm to prevent security and privacy breaches.

In this paper, we propose a hierarchical key generation and distribution scheme that addresses the sensor network computational, energy and power limitations. The proposed key generation algorithm offers a novel, scalable and efficient technique for establishing hierarchical keys for sensor networks with low computational and energy requirements. The proposed algorithm is based on low cost hashing functions that enable the efficient key generation. Moreover, we propose an optimized version of the hierarchical key generation algorithm for tree shaped hierarchies. Consequently, a hybrid technique can be used to further optimize key generation for any hierarchical structure.

We also propose a hierarchical key distribution protocol that enables the diffusion of the hierarchical keys in an efficient and systematic manner. The technique allows the diffusion of keys from high access levels to lower access levels and ensures the secure dissemination of the hierarchical keys in the network. Moreover, the structures generated for key distribution are used for data collection, data fusion and query propagation

The rest of the paper is organized as follows. In Sections II and III we discuss the related work and the preliminaries. Our hierarchical key generation algorithm is discussed in Section IV. The optimized key generation technique for tree hierarchical structures and the hybrid technique for general hierarchical structures are presented in Section V. We then show simulation results comparing several key generation techniques to our proposed technique in Section VI. Our hierarchical key diffusion protocol is presented in Section VII. In Section VIII we discuss possible security threats and present suitable solutions to such threats. Concluding remarks and future work are added in Section IX.

II. Related Work

Several key management protocols proposed in literature [3][4][5] provide hierarchical key generation and management.

However, most of the available techniques depend on exponentiation which is a computationally expensive operator. In this section we review some of the available techniques to show the high dependency on exponentiation.

In [5] the author proposes a time bound cryptographic key assignment scheme. This scheme uses ideas similar to RSA [6]. Given a set of access classes $C_1, C_2, \ldots, C_m$ and a binary relation $\leq$ that partially orders the set of access classes, the algorithm generates for every class C_i a key K_i, a public number e_i and a private number d_i. The numbers $e_1, e_2, \ldots, e_m$ are randomly generated so that they are relatively prime to the Euler totient function $\phi(n_1)$. Where n_1 is the product of two large primes. The private number d_i is computed such that $e_i d_i \ mod \ \phi(n_1) = 1$. Consequently, for every class C_i a key K_i is computed using the following relation $K_i = a^{\prod_{C_k \leq C_i} d_k} \ (mod \, n_1)$, where a is a randomly selected integer such that $1 < a < n_1$. If $Ci \leq Cj$ then K_j could be used to generate K_i, that is, $K_i = a^{\prod_{C_k \leq C_j, C_k \not\leq C_i} e_k} \ (mod \ n_1)$. Therefore exponentiation is the main operator used in this technique.

Akl et al. [3] proposed a technique that assigns to each access class C_i an integer t_i such that if $C_i \leq C_j$ then $t_j | t_i$ (t_j divides t_i). A special one way function f_m is defined having the property $f_{mz} = f_m \circ f_z$, where m and z are integers and $\circ$ denotes the composition operator. A random number K_0 is chosen, the other keys are calculated using the relation $K_i = f_{t_i}(K_0)$. If $C_i \leq C_j$ then $t_i = z t_j$ for some integer z and K_i can be computed by using K_j, z and by applying the composition property $K_i = f_{t_i}(K_0) = f_{z t_j}(K_0) = f_z \circ f_{t_j}(K_0) = f_z(K_j)$. The function $f_m(K) = K^m \ (mod \, M)$ is chosen so that the above calculation is only possible when $C_i \leq C_j$. The number M is the product of two large primes p and q. M is made public. Using the selected function f_m, the key generation procedure translates to $K_i = K_0^{t_i} = [K_0^{t_j}]^{\frac{t_i}{t_j}} = [K_j]^{\frac{t_i}{t_j}} \ (mod \ M)$. This technique depends on exponentiation for the generation of the hierarchical keys. Moreover, the generation of the integer t_i for all the access classes is difficult especially with complex hierarchical structures.

Ray et al. [4] proposed a cryptographic solution to implement access control in a hierarchy that also uses exponentiation extensively in generating hierarchical keys; the ideas are similar to RSA [6].

The previously proposed solutions [3][5][4] require exponentiation that is computationally expensive and require the provisioning of public keys which adds to the computational and communication overhead. As a result such techniques are not appropriate for sensor networks. Our proposed hierarchical key generation technique does not use exponentiation. Instead, it uses the secure hashing operator which is computationally less expensive. Furthermore, our technique does not require any public information. As a result it eliminates the computational and communication costs involved in the exchange of public information.

III. Preliminaries

A. Cryptographic hash functions

A cryptographic hash function maps strings of arbitrary length into strings of a fixed length called *message digests*. Given a cryptographic hash function h and an input string x it is computationally easy to compute the message digest $h(x)$. On the other hand, it is infeasible to compute the original input string x from the message digest $h(x)$. As it is infeasible to compute h^{-1}, cryptographic hash functions are one-way functions. These functions are also designed to be collision resistant, which means that given a hash function h, it is infeasible to find two strings x and x' such that $h(x) = h(x')$. In addition to the above properties cryptographic hash functions are also designed to have some randomness-like properties, independence of input/output, and unpredictability of the output when parts of the input are unknown. Such properties are required for the security of these functions. Hash functions are used for generating message authentication codes and pseudo-random numbers.

B. One way hash chains

A one way hash chain $(d_0, d_1, \ldots, d_n)$, d_0 is randomly chosen. Given a one way hash function h, the complete one way hash chain is computed according to the following relation $d_i = h(d_{i-1})$ for $i > 0$. Note that if d_i is known, then d_j where $i \leq j \leq n$, can be easily computed by repeatedly applying the one way function. Figure 2 shows an example hash chain. The chain has the one-way property because of the complexity of computing the chain in the opposite direction, which requires solving the inverse hash function. One way chains are used in many applications including one time passwords [7], and secure and efficient authentication protocols for sensor networks [8].

C. Ordering and hierarchical access control structure

Let S be a set of access classes (classes for short) where $S = C_1, C_2, \ldots, C_n$ and $\leq$ be a binary relation that partially orders the set S. The ordering $C_j \leq C_i$ is used to indicate that class C_i dominates class C_j. Based on the ordering relation the set of classes can be organized according to an *access hierarchy*, which adequately presents a description of the system policy. A hierarchy can be defined by a directed acyclic graph (V, E), where the vertex set V represents the security classes while the edges set E represents the partial ordering among the vertices. A path from C_i to C_j exists if and only if $C_j \leq C_i$.

The set $Children(C_i)$ denotes the vertex nodes that are directly connected to C_i, such that $C_k \in Children(C_i)$ if and only if $(C_i, C_k) \in E$. If C_i is a leaf node then $Children(C_i)$ is empty. The set $Parents(C_i)$ represents the set of vertices such that $C_k \in Parents(C_i)$ if and only if $(C_k, C_i) \in E$. If C_i is a root node then $Parent(C_i)$ is empty. In a hierarchy nodes could have multiple parents. Figure 1 shows an example of a hierarchy.

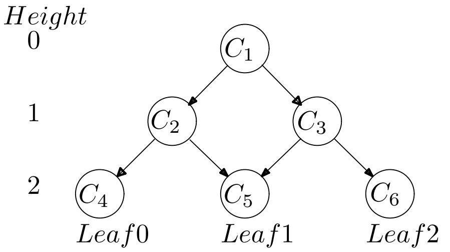


Fig. 1. A hierarchy with 6 access classes and 3 leaves

IV. Our scheme

A *hierarchical key* is a key that is associated with a class C and allows one to decrypt all information classified at C and at all classes lower than C in the access hierarchy. To implement the hierarchical key management and generation scheme we use secure cryptographic hashing functions as our main operator. This assures security and reduces the computational requirements for both key generation and management. The following notation is used to simplify the discussion that follows. Let H be a hierarchy describing n access classes C_i, $1 \leq i \leq n$. Let K_i be the hierarchical key assigned to access class C_i and R_i is the residual set[1] for access class C_i. Our approach is to generate a hash chain for every leaf node in the hierarchy. The hash chain starts by a random number at the root; intermediate hash values are stored at intermediate nodes while the final hash value is stored at the leaf nodes. Figure 2 graphically represents the hash chain generation, and illustrates the notation used.

$$D_0 \longrightarrow h(D_0) \longrightarrow h(h(D_0)) \longrightarrow \ldots\ldots \longrightarrow h(\ldots(h(D_0))\ldots)$$

$$D_0^0 \longrightarrow D_0^1 \longrightarrow D_0^2 \longrightarrow \ldots\ldots \longrightarrow D_0^m$$

Fig. 2. Hash chain generation from root to leaf; the one above is the hash chain, whereas the one below is the residual set chain.

The hash chain can be computed easily in only one direction which is from root to leaf; the opposite direction is infeasible because it would require the computation of the inverse hash function, which is computationally infeasible. As a result of such approach a parent node has enough information to compute the hash values of all the nodes below it, however children nodes cannot compute the hash values of parent nodes.

The hash values computed for every node are stored in the corresponding residual set. The residual sets R_i, $1 \leq i \leq n$ are generated by populating the hash chains for every leaf node. Figure 3 describes the detailed algorithm used to generate the residual set for each node in the hierarchy.

The generation of the residual set involves two functions $Generate\_Residual$ and $Add\_Residual$, assuming we have access to a secure random number generator and a secure cryptographic hash function. The function $Generate\_Residual$ calls $Add\_Residual$ for every leaf node in the hierarchy. $Add\_Residual$ is a recursive function that climbs the hierarchy from every leaf node to the root node. This function populates the residual sets of all the parent nodes in the path to the root node. Note that a node may have multiple parent nodes, however $Add\_Residual$ handles this by populating each parent node with the corresponding hash values. After applying the algorithm to the hierarchy in Figure 1 the generated residual sets are reported in Figure 4.

We can observe that leaf nodes have $|R_i| = 1$, while parent nodes can have residual sets with higher cardinality depending on the number connected leaves. Another observation is that if there are m leaves then the cardinality of the root's residual set is m. We define the *inclusion property*: let C_i and C_j be two access classes, with residual sets R_i and R_j respectively. If $C_i \in Parent(C_j)$ then $R_j \subseteq h(R_i)$. This indicates that R_j could be completely computed using the information in R_i.

The algorithms discussed so far generates the residual set for every node in the hierarchy. Residual sets at the top of the hierarchy can easily be used to generate the residual sets of nodes lower in the hierarchy by following the correct hash chains, while the opposite direction is infeasible due to the requirement of computing the inverse hash function. As a result the generation of the hierarchical key should depend on the computed residual sets. Our approach is to hash the entries of the residual set to generate the hierarchical key. The hierarchical key inherits the properties of the residual set.

The $Generate\_Key$ function generates the hierarchical key by computing the hash value of the concatenation of all the elements in the residual set if $|R_i| > 1$, otherwise key is equal to the residual set. This condition guarantees that it is infeasible for nodes in the lower levels to compute the upper level keys. Figure 5 describes the detailed algorithm used to generate the keys.

The complexity of the algorithm is embedded in the generation of the residual sets. However, the generation of residual sets uses the hashing operator which is a computationally low cost operator. Key generation from residual sets is a transformation procedure that also uses hashing as the main operator. Furthermore, the derivation of lower level keys from high level keys is achieved by generating a hash chain.

The addition of a new security class to an existing hierarchy

```
Generate_Residual(H)
  for each $j \in leaves(H)$
     do $Add\_Residual(H, j, j)$

Add_Residual(H, i, j)
  if $i == H.root$
     then $int\_hash = Generate\_Random(seed)$
  else $int\_hash = Add\_Residual(H, i.parent, i)$
  $R_i = R_i \cup int\_hash$
  for all $k \in Parents(j)$
     $R_k = R_k \cup int\_hash$
  return $hash(int\_hash)$
```

Fig. 3. $Generate\_Residual$ and $Add\_Residual$, algorithms that generate the residual set

[1] Residual set is the set containing information required to generate the key. Residual set R_i is the information required to generate key K_i.

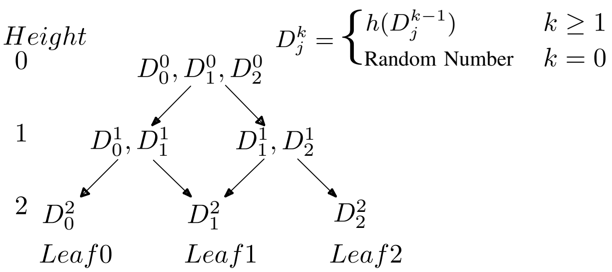


Fig. 4. The residual set generated at each of the access levels after generating the hash chains starting from each leaf using the hierarchy in Figure 1

is easily accommodated by recomposing the hash chain starting at the new node. The only issue we see in our technique is the storage requirement of the residual sets. We address this issue by proposing an optimization technique that reduces the size of the residual set. This technique is discussed in the following section.

V. Algorithm optimization

The hierarchical key generation algorithm could be optimized by reducing the size of the resulting residual sets. In this section we start by presenting an optimization technique that is used for tree shaped hierarchies. Then we propose a hybrid technique that is used to optimize residual sets for any hierarchical structure.

A. Tree shaped hierarchies

Our optimization technique exploits properties of cryptographic hash functions, namely the independence of input/output and the one-way property. Assume the hierarchy is a tree and that each node has only one parent. Then, our proposed optimization technique ensures that $|R_i| = 1$ for $1 \leq i \leq n$. It assigns to each access class C_i the residual value of $h(R_{parent(i)} \times 2^{m_i})$, where m_i is the order of node i and is determined as follows: Let $Parent(i)$ have a total of l children nodes (including node i); if we order the children nodes from left to right and assign to each node a unique value from $0\ to\ l-1$, then m_i is the order of node i in such ordering. The only exception to such assignment is the root node where its residual set is assigned a random value. This technique ensures that each node will only need to store one value in its residual set, which is the optimal size for the residual set. Access classes cannot compute the residual set of other nodes at the same height due to unpredictability and randomness properties of the cryptographic hash functions. Our approach exploits the interesting property of cryptographic hash functions that if $h(x)$ is known and $f(x)$ is a deterministic function, where $f(x) \neq x$, then $h(f(x))$ is not predictable by knowing only $h(x)$. The powers of 2 can be efficiently computed by using left shifts which is a computationally low cost operation.

```
$Generate\_Key(H)$
  Initialize $R\ and\ K\ to\ Null$
  $Generate\_Residual(H)$
  for each $node\ i$
    if $|R_i| > 1$
     then $K_i = h(concatenate\ elements\ of\ R_i)$
    else $K_i = R_i$
  return $(K, R)$
```

Fig. 5. $Generate\_Key$, to generate the hierarchical key.

```
$Generate\_Residual\_Tree(H, Random\_Num)$
  $H.root.R = Random\_Num$
  $Populate\_Children(H, H.root)$

$Populate\_Children(H, p)$
  $m = p.num\_children$
  $k = m - 1$
  while $k \geq 0$
    $p.child[k].R = h(2^k \times p.R)$
    $Populate\_Children(H, p.child[k])$
    $k = k - 1$
```

Fig. 6. $Generate\_Residual\_Tree$ and $Populate\_Children$, algorithms that generate optimized residual set for tree hierarchy

Figure 6 reports the detailed algorithm used to generate the residual set for each node in the tree hierarchy. The function $Generate\_Residual\_Tree$ starts by assigning a random number to the residual set of the root of the hierarchy. Then $Populate\_Children$ is called to recursively populate the residual sets of all the nodes below the root node. The application of $Generate\_Residual\_Tree$ on an example hierarchy is shown in Figure 7.

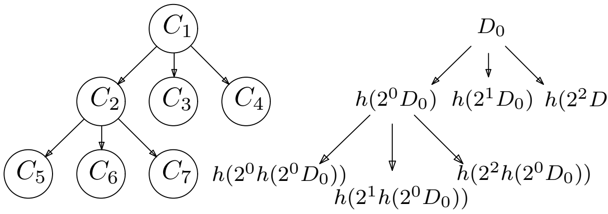


Fig. 7. A three level hierarchy example showing the optimization adopted in the generation of the residual set and hierarchical keys for a tree.

B. Hybrid approach

The hybrid technique uses both the $Generate\_Residual$ and $Generate\_Residual\_Tree$ to optimize the residual set of complex hierarchical structures. A complex hierarchy H is partitioned into tree partitions and non-tree partitions. Tree partions are maximal trees with nodes that are single parents and do not share the parenthood of any other node. The tree partitions $H_{t_1}, H_{t_2}, \ldots, H_{t_n}$ are replaced by super nodes $S_{t_1}, S_{t_2}, \ldots, S_{t_n}$. The new hierarchy is referred to H_S. Performing $Generate\_Residual(H_S)$ generates the residual set for all the nodes in H_S including the super nodes. For every super set S_{t_i} the function $Generate\_Residual\_Tree(H_{t_i}, S_{t_i}.R)$ is called with arguments H_{t_i} as the input hierarchy and the residual set of S_{t_i} as the random number. This populates the tree partitions with the optimized residual set. Figure 8 shows the pseudo code for the hybrid algorithm.

Figure 9 shows an example that explains the several steps followed by the hybrid technique. The hybrid approach provides an optimal residual set for a complex hierarchy.

VI. Simulation results

In this section we present some experimental results that compare the computational overhead and memory usage of our proposed hierarchical key generation algorithm to other hierarchical key generation algorithms in literature.

A. Computational overhead

The experimental results presented in this section show the superiority of our proposed algorithm with respect to the exponentiation dependent algorithms. In these experiments the Wen-Guey technique [5] is used to represent the exponentiation based hierarchical key generation techniques. A binary tree is used as the input access class hierarchy for the system. The hierarchy size represents the number of node in the hierarchy, which is related to the height of the binary tree.

In the first experiment the key size was fixed to 512 bits and hierarchy size was varied. The execution time for the Wen-Guey technique, our proposed technique and our optimized version were measured. This experiment captures the behavior of the key generation algorithms with respect to different hierarchy sizes. Figure 10 shows the simulation results for a key size of 512 bits. Our proposed techniques are always below the Wen-Guey technique for all the hierarchy sizes. We carried out other experiments with different key size values. However, these experiments show the same trends as in the results reported.

The second experiment uses a fixed hierarchy size while varying the key size. It analyses the performance of the key generation technique for different key sizes. Figure 11 shows the simulation results for a hierarchy of size 31. As the key size increases the running time of the Wen-Guey increases exponentially, while the running time of our proposed algorithms does not show considerable increase.

The presented evaluation results show that our proposed techniques have a low computational cost even for large hierarchy sizes and large key sizes. All such properties make our proposed technique the best candidate for the sensor network environment. Having a computationally low cost algorithm reduces the costs required for key generation and re-keying operations.

```
Algorithm: Hybrid algorithm pseudo code
Input: Hierarchy $H$
Output: Residual sets and Hierarchical keys
 Step 1:
 Generate the maximal tree partitions
 $H_{t_1}, \ldots, H_{t_n}$
 Step 2:
 Generate $H_S$ by replacing each $H_{t_i}$ by a
 supernode $S_{t_i}$
 Step 3:
 Compute $Generate\_Residual(H_S)$
 Step 4:
 for each supernode $S_{t_i}$
    $Generate\_Residual\_Tree(H_{t_i}, S_{t_i}.R)$
```

Fig. 8. Pseudo code describing the hybrid algorithm.

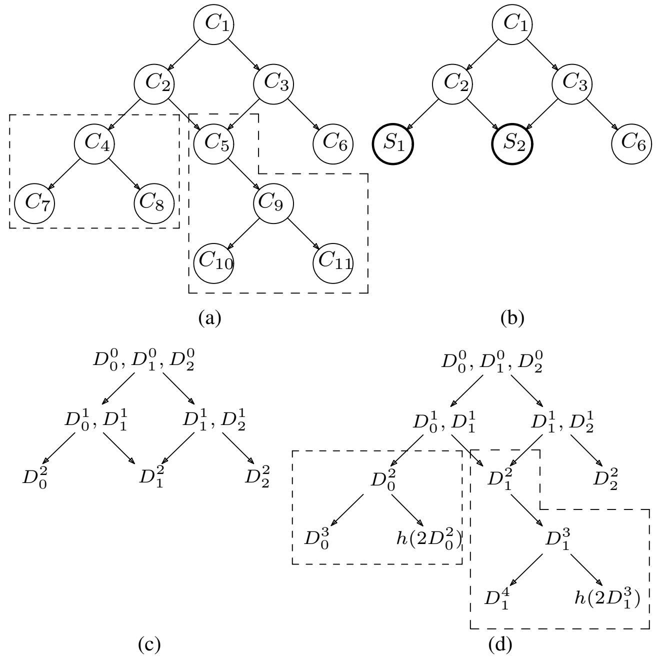


Fig. 9. Complex hierarchy and steps involved in the hybrid approach. (a) The tree partitions are located in the hierarchy and are shown by the dotted boxes. (b) Tree partitions are replaced by super nodes S_1 and S_2. New hierarchy H_S. (c) Output after $Generate\_Residual(H_S)$ showing the populated residual sets for H_S. (d) After running $Generate\_Residual\_Tree$ for every tree partition.

B. Memory requirements

The memory requirements mainly include the memory required to store the keys, the residual sets and public key information. Assuming the access class is a full binary tree with with N nodes and height $h = \lfloor logN \rfloor$. Table I reports the storage requirement for Wen-Guey, and our proposed techniques.

TABLE I
Hierarchical key storage requirements.

Technique	Key storage	Public keys storage
Wen-Guey	$O(N)$	$O(N)$
Hash Tree	$O(NlogN)$	None
Optimized Hash Tree	$O(N)$	None

VII. (HKDP): Hierarchical key diffusion protocol

The previous sections discussed the approaches required to efficiently generate the hierarchical keys. However, the challenge to be addressed next is how to efficiently and securely deliver the generated key to the corresponding sensor nodes. The hierarchical keys cannot be embedded in the sensor nodes before deployment because the security access class of a node is dependent on the context of the node and the

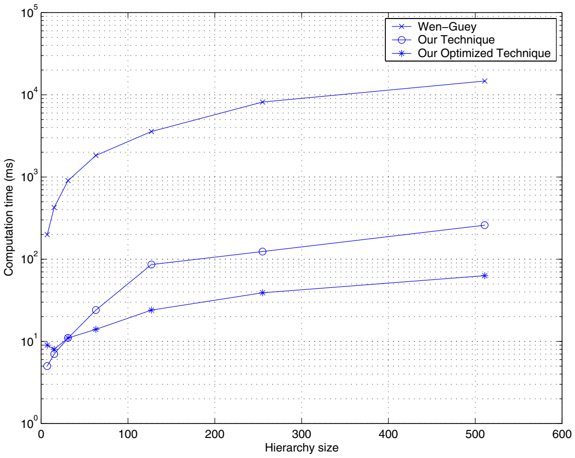


Fig. 10. Simulation of computation time w.r.t variable hierarchy sizes, the key size fixed to 512 bits.

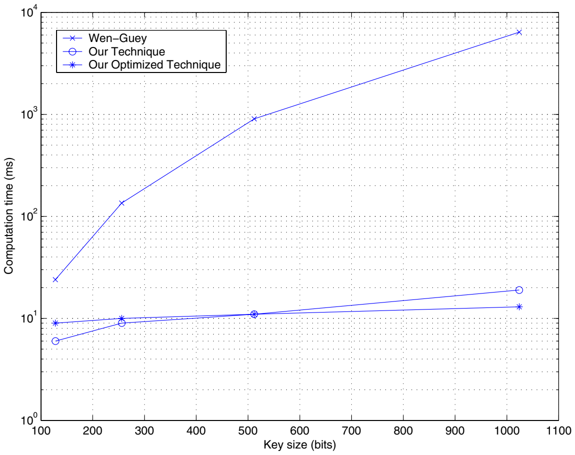


Fig. 11. Simulation of computation time w.r.t variable key sizes, the hierarchy size fixed to 31.

sensed data. In addition, broadcasting the hierarchical keys is not a feasible solution because it defeats the purpose of having secret keys. As a result, a key distribution protocol is needed to manage the secure distribution of the generated keys. In this section we propose a hierarchical key distribution protocol that accounts for sensor network energy constraints by using energy efficient algorithms. We will refer to our proposed key distribution protocol as the hierarchical key diffusion protocol (HKDP). The data structures developed for HKDP are also useful for data collection, data fusion and query propagation in sensor networks. HKDP is built over the existing network routing protocols which enables it to function easily over already designed architectures.

We assume that the sensor network is partitioned into groups called *clusters*; each node is a member of only one cluster. Every cluster has a unique identifier and a cluster head. The cluster head is responsible for the management of all the nodes in the cluster. A base station is located at the border of the network for global network management and data collection. The system can be easily extended to accommodate multiple coordinated base stations. HKDP is organized into the following four phases:

A. Phase 1: Access class assignment

The access class of each node is decided based upon the contextual information of both the node and collected data. For example, contextual information includes the node location, node energy level, node computational capabilities, data quality, and data sensitivity. Appendix I reports examples contextual information that could be used in access class assignment. A trusted security module in the sensor node collects the required context information and then computes the appropriate access class from a set of access classes ordered in a hierarchy. The relation mapping context information to access classes is dependent on the system policy.

A cluster could contain nodes belonging to different access classes. We define the access class of the cluster as the *least upper bound (lub)*[2] of the access classes of all the nodes in the cluster. The cluster head takes the same access class as the cluster.

B. Phase 2: Connecting clusters with the same access classes

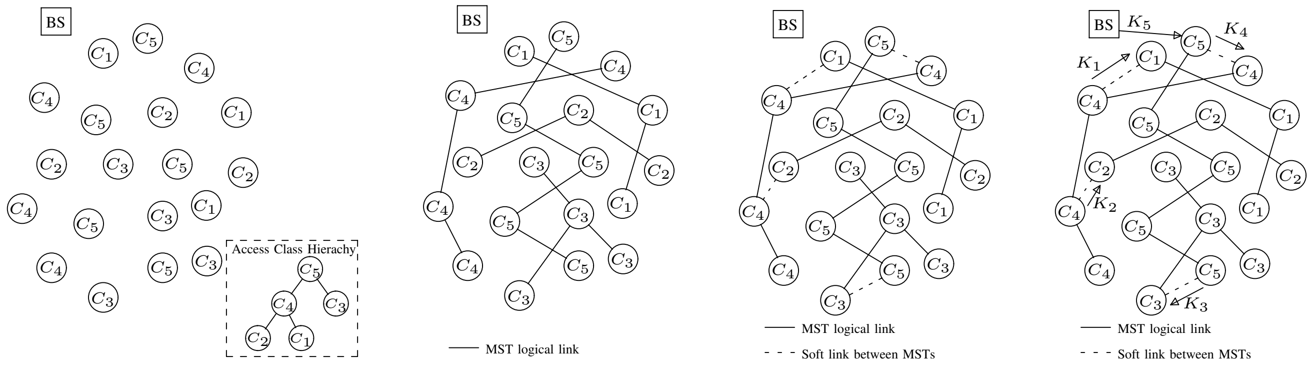
Clusters belonging to the same access class should be connected to enable efficient exchange of information among such clusters. Clusters communicate with one another not only for key distribution but for several other reasons such as data collection, data fusion and query propagation [9]. A distributed *minimum spanning tree* (MST) is generated to provide connectivity between clusters with the same access class. Clusters at C_i form MST_i. To generate the MST, energy efficient multicast tree generation schemes [10], [11] are used. However, the detailed MST generation technique is out of scope of this paper.

To maintain the generated MST, each cluster in the MST is only required to store the addresses of its parent cluster and its descendant clusters in the MST. Furthermore, to ensure reliability and connectivity of the clusters in the MST, proactive techniques [12] are used for the reconstruction of the MST if nodes fail, move or decide to leave the network.

C. Phase 3: Soft links and connecting $MSTs$ of related access classes

To enable the collaboration of clusters belonging to different access classes, a means for connecting such clusters is required. However, this collaboration should satisfy the constraints imposed by the hierarchical access control policy. For this reason, we cannot connect any clusters together. Using the fact that high level hierarchical keys can be used to generate lower level hierarchical keys, high level clusters should be connected to clusters that they dominate. Let C_i and C_j be two classes such that $C_i \leq C_j$. Suppose that clusters at access class C_j wish to collaborate with clusters at C_i. Clusters

[2] This is consistent with the notion that a set of access classes being partially ordered set. Here we assume that a cluster cannot have two or more *lubs*, which may happen when we have classes that are not comparable under partial order. However this problem is easily addressed by letting clusters take more than one access class.



(a) Phase 1 (b) Phase 2 (c) Phase 3 (d) Phase 4

Fig. 12. The different phases of the hierarchical key diffusion protocol.

at C_i could be connected to clusters at C_j by connecting MST_i and MST_j. To connect MST_i and MST_j links should be generated between them, these links are referred to as soft links $SLink_{ij}$.

The generation of soft links between MST_i and MST_j is formulated as follows. Let MST_i and MST_j be two $MSTs$; let $k \in MST_i$ and $l \in MST_j$ be two clusters and let a_{kl} be the cost of connecting them. The problem is to choose the pair (k, l) such that a_{kl} is minimal, such link is the soft link $SLink_{ij}$. The cost a_{kl} is derived from several metrics such as distance, energy level, hop count and other routing metrics.

To generate $SLink_{ij}$ nodes in MST_j we compute the minimum cost to connect to MST_i. The cluster with the minimum link cost is selected and the soft link $Slink_{ij}$ is generated. The overall generated tree formed by connecting the $MSTs$ at different access levels is referred to as a *diffusion tree*. The diffusion tree is used for key distribution and several other applications such as data collection, data fusion and query propagation.

D. Phase 4: Hierarchical key diffusion

To ensure the security of the transmitted key, we assume a public key infrastructure (PKI) existing between each cluster pair. However, the PKI system is only used during the exchange of hierarchical keys to avoid the high computation costs of using the PKI.

The *diffusion tree* is the main backbone that enables the efficient distribution of the hierarchical key. The base station generates the hierarchical keys, and sends only the highest level key(s) to clusters operating at the highest access class. In this design the base station is only required to know the location of the high level clusters; this reduces the cost of knowing the location of clusters at all access levels. The highest level key is propagated to the corresponding high access class MST. Furthermore, the lower level key is computed by the high access class clusters and is injected into the low access class MST via the soft links. Consequently, the key diffuses in a hierarchical manner from class to class until all the clusters are populated with their corresponding key. Moreover, once a cluster receives its corresponding hierarchical key the cluster head computes and distributes the keys required by the nodes inside the cluster.

Figure 12 illustrates through an example the various steps involved in assigning the access levels, building the $MSTs$, soft links and key diffusion. Figure 13 shows the timeline of key diffusion. When no clusters operate at high access levels the base station forwards the corresponding keys to lower level clusters. This ensures the successful key diffusion for different access class distributions.

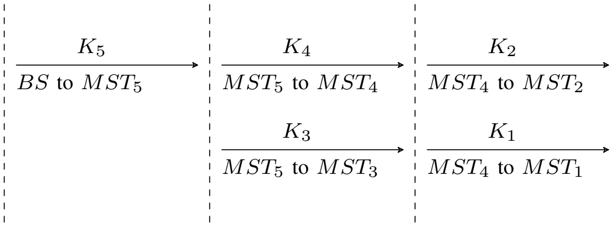


Fig. 13. Hierchical key diffusion in the network.

VIII. Security analysis

In this section we discuss the security attacks that could be performed on the proposed key diffusion protocol. We also provide suitable solutions that eliminate the threats caused by such attacks. We assume the sensor nodes are rational and are willing to collaborate with each other. This assumption is needed to ensure the functionality of the basic network operations.

- *Compromised Nodes:* Suppose an attacker captures a sensor node and is able to acquire the keys stored in the node's memory. If the node is at access class C_i, then by capturing this node the key K_i is exposed and all the keys K_j such that $C_j \leq C_i$ are also exposed. However, to cope with this threat we make the assumption made in [13], that the sensor nodes are tamper-proof. Another solution to this problem is to periodically distribute new keys. Keys derived from nodes are thus valid only up to the next periodic key update.
- *Incorrect access class request:* Suppose a node cheats by requesting an access class that does not match its

context. However, the cheating node's context information could be easily estimated by the neighboring nodes. Consequently, neighboring nodes can also estimate the correct access class of the cheating node. Note that we are assuming that cheating nodes do not cooperate with one another.

- *Eavesdropping:* Let X, Y and Z be nodes and C_X, C_Y and C_Z be their access classes respectively. Let the access classes be ordered such that $C_Z \leq C_Y \leq C_X$. Suppose nodes X and Y are communicating and node Z is trying to eavesdrop. Nodes X and Y encrypt the data they exchange by using K_Y as the session key. However, node Z will not be able to decrypt the data because K_Y cannot be derived from K_Z. Therefore all data transmissions are secure against eavesdropping threats. Moreover, hierarchical keys are transmitted using the PKI system which ensures key secrecy during key diffusion.
- *Unauthorized communication:* Let $C_X \not\leq C_Y$ and $C_Y \not\leq C_X$, thus C_X and C_Y are not comparable under the partial order. As a result of such ordering it is not possible to determine K_X from K_Y or K_Y from K_X. This renders the communication between X and Y impossible using neither K_X nor K_Y. Thus unauthorized hierarchical communication between nodes is prevented.
- *Denial of service attacks:* A denial of service attack could be caused by repeatedly requesting re-keying. There are several steps involved in re-keying namely key generation and key diffusion. Most of the cost of re-keying is attributed to key diffusion. The repeated unnecessary re-keying generates traffic that congests the network and depletes the sensors' energy. This problem is addressed by using a proactive approach which involves periodic re-keying instead of allowing nodes to reactively request re-keying.
- *Covert channels:* Let X, Y and Z be nodes and C_X, C_Y and C_Z be their access classes respectively. Let the access classes be ordered such that $C_Z \leq C_Y \leq C_X$. Suppose nodes X and Y are communicating and node Z is intercepting such communication. Node Z is not able to encrypt the communication channel. However, the fact that node Z knows that X and Y are communicating generates a covert channel. This covert channel cannot be avoided due to collaborative nature of sensor networks. Sensor nodes are required to cooperate to perform the routing operations and this covert channel would always exist if Z is in the routing path between X and Y.

To enhance the security of the system, nodes that do not behave correctly are removed from their corresponding $MSTs$, thus isolating them from the network. Furthermore, keys are not forwarded to such nodes during re-keying operations. This completely isolates the node because all the data exchanges in the network are encrypted. A special case is when cluster heads are misbehaving. In this case, reclustering is needed after they are isolated to elect new cluster heads. This ensures both connectivity and security of the network. Clearly, this technique enhances the system fault tolerance.

IX. Conclusion and future work

In this paper, we have presented a hierarchical key generation algorithm that is suitable for sensor networks with computational and energy constraints. We have proposed a residual set optimization technique for tree hierarchies and a hybrid optimization technique for general shaped hierarchies. Our simulation results show that our proposed hierarchical key generation algorithm is computationally less expensive than all the exponentiation based key generation algorithms. Moreover, our proposed algorithm requires no public information, which reduces the communication costs required for key derivations.

Furthermore, we have presented the key diffusion protocol. The presented protocol uses energy efficient algorithms in all its phases to satisfy the energy limitations of wireless sensor networks. We further have discussed the security threats involved with the proposed protocols and provided suitable solutions to such threats.

In future, one direction we plan to pursue is the development of policies and techniques to specify access class assignments to data and sensor nodes. We also plan to investigate the problem of integrity in sensor networks.

Acknowledgment

Portions of this work have been supported by the sponsors of the Center for Education and Research in Information Assurance and Security (CERIAS) at Purdue University, and the National Science Foundation under NSF Grants No. 0430274 and II-0242419.

References

[1] B. Thuraisingham, "Secure Sensor Information Management," *IEEE Signal Processing*, May 2004.

[2] D. Bell and L. LaPadula, "Secure Computer Systems: Mathematical Foundations," *Technical Report MTR-2547, MITRE Corporation*, vol. I, Mar 1973.

[3] S. G. Akl and P. D. Taylor, "Cryptographic Solution to a Problem of Access Control in a Hierarchy," *ACM transactions on Computer Systems*, vol. 1, no. 3, pp. 239–248, Aug 1983.

[4] I. Ray, I. Ray, and N. Narasimhamurthi, "A Cryptographic Solution to Implement Access Control in a Hierarchy and More," in *SACMAT 02: Proceedings of the Symposium on Access Control Models and Technologies*, June 2002.

[5] W.-G. Tzeng, "A Time-Bound Cryptographic Key Assignment Scheme for Access Control in a Hierarchy," *IEEE Transaction on Knowledge and Data Engineering*, vol. 14, no. 1, Jan/Feb 2002.

[6] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb 1978.

[7] L. Lamport, "Password Authentication with Insecure Communication," *Communication of ACM*, vol. 24, no. 11, pp. 770–772, Nov 1981.

[8] A. Perrig, R. Canetti, D. Song, and J. Tygar, "Efficient and Secure Source Authentication for Multicast," in *NDSS '01: Proceedings of the Network and Distributed System Security Symposium*, 2001.

[9] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, Aug 2000.

[10] B. Wang and S. Gupta, "G-REMiT: An Algorithm for Building Energy Efficient Multicast Trees in Wireless AdHoc Networks," in *Proceedings of the Second IEEE International Symposium on Network Computing and Applications*, 2003.

[11] M. Gerla, C.-C. Chiang, and L. Zhang, "Tree multicast strategies in mobile, multihop wireless networks," *Mobile Networks and Applications*, vol. 4, no. 3, 1999.

[12] M. Yang and Z. Fei, "A Proactive Approach to Reconstructing Overlay Multicast Trees," in *Proceedings of INFOCOM*, March 2004.

[13] R. D. Pietro, L. V. Mancini, and S. Jajodia, "Providing secrecy in key management protocols for large wireless sensor networks," *Journal of AdHoc Networks*, vol. 1, no. 14, 2003.

[14] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, Feb 2001.

Appendix I
Context Information

Context is any information that can be used to characterize the situation of an entity [14]. In sensor networks an entity includes sensor node, base station or any object that interacts in the network. Context information is used to assign access classes to nodes in the sensor network; such assignment is dependent on the system security policy. Here we provide some useful contextual information used to assign access classes to sensor nodes.

- *Node location:* Node location implies the sensitivity of the sensed data. For example nodes located close to military locations are assigned high level access classes because such nodes sense highly sensitive data.
- *Node energy level:* Sensor nodes depend on batteries as their primary source of energy. The energy level is one of the factors that indicate the node's capability of surviving in the network, thus high energy nodes are assigned high access classes.
- *Number of neighboring nodes:* This parameter is an indication to connectivity of the node. Nodes with large number of neighbors are good cluster head candidates. Cluster heads are responsible for several organizational tasks thus they are assigned to high access classes.
- *Relative distance between neighboring nodes:* This metric indicates the relative distance between neighboring nodes. It is related to power required for radio transmission.
- *Errors:* Errors including number of transmission errors, noisy channel errors, retransmission errors, collisions encountered. All such errors indicate the reliability of the node in the network.
- *Date/time*: The access class of nodes depends on the temporal dimension. Data sensed at certain periods of time is assigned to different access classes. For example consider sensors deployed at the entrance of a bank for surveillance purposes; at night the data sensed is assigned a high access class as such data may indicate possibilities of a robbery.
- *Data quality:* Data quality is captured by analyzing the processing, sampling and sensing errors. The level of trustworthiness of data is indicated by this metric. High quality data is assigned to high security access classes. For example a sensor taking photos with an object blocking its view produces low quality data.
- *Data sensitivity and resolution:* As more information could be extracted from high resolution data, such data is assigned high access levels.