Key management in wireless sensor networks

Lanfranco Lopriore
Dipartimento di Ingegneria dell'Informazione, Università di Pisa, via G. Caruso 16, 56126 Pisa, Italy. E-mail: lanfranco.lopriore@unipi.it

Abstract — We refer to a distributed architecture consisting of sensor nodes connected by wireless links and organized in a tree shaped hierarchy. We present a paradigm for the management of the cryptographic keys used by nodes to communicate, and we consider the problems connected with key generation, distribution, and replacement. In our paradigm, names are assigned to nodes by using a uniform scheme, which is based on the position of the given node in the node hierarchy. Each node holds a hierarchical key to communicate with its ancestors, and a level key to communicate with its siblings. A single, publicly-known parametric one-way function is used to assign hierarchical keys to nodes, in an iterative procedure that starts from the key of the root of the node hierarchy, and proceeds downwards to the lowest hierarchical levels. A similar procedure is used to generate the level keys. The total memory requirements for key storage are extremely low. The number of keys exchanged in a key replacement process is kept to a minimum. Dynamic access control is fully supported, whereby new nodes can be added to the node hierarchy, and existing nodes can be evicted from the hierarchy.

Keywords: cryptographic key; key replacement; parametric one-way function; tree shaped hierarchy; wireless sensor network.

1 INTRODUCTION

We refer to a distributed architecture consisting of sensor nodes connected by wireless links. In an architecture of this type, stringent limitations exist on the hardware complexity, computational power and energy consumption of each sensor node [1], [2], [3]. Consequently, the network design is largely different from that of a traditional wired-line or wireless network. The number of messages exchanged between the network nodes must be kept to a minimum, to save energy. Low hardware complexity implies the absence of any form of memory management, and even of the traditional separation between the two processor modes, a privileged mode and a user mode with memory access limitations [4], [5], [6]. This situation is unlikely to change in the near future. Rather than supporting the addition of new sophisticated hardware functionalities, advances in integration technology are likely to be used to reduce the system size and cost [7], [8], [9].

Thus, in the primary memory of a sensor node, no separation exists between the address spaces of the different software components and the supervisor. Every piece of software has unlimited access to all memory resources. Consequently, it is impossible to confine sensitive information items, such as cryptographic keys; every software routine can read and possibly modify all these critical information items.

1.1 The protection model

In a classical protection system paradigm, a set of active entities, the *subjects*, generates access attempts to a set of passive entities, the protected *objects* [10], [11], [12]. When a subject issues an access to a given object, the access terminates successfully only if that subject holds the corresponding access privileges. A *protection domain* is a set of access privileges for the protected objects. A subject executed in a given protection domain can access the objects included in that domain, with the corresponding access privileges. A subject can switch domain; an action of this type produces a change in the access privileges that the subject can exercise successfully. In a possible approach, a *key* is associated with each domain. Possession of a given key implies possession of the access privileges included in the corresponding domain.

We shall refer to an implementation of our protection paradigm based of symmetric keys and conventional cryptography, superior to public key cryptography in both terms of low computation requirements and low energy costs [13], [14], [15], [16]. We shall model a wireless sensor network as a distributed protection system where each sensor node is a subject, and each message exchanged between the nodes is a protected object. A protection domain associated with a given key includes all the messages encrypted by using that key. A node that owns a given key can generate new messages in the corresponding domain and can read the messages in that domain. A node that holds two or more keys can use these keys to read and write messages in the corresponding domains. The utilization of different keys corresponds to a domain switch.

A cryptographic key has a *name* and a *value*. The name identifies the domain, the value grants access permission to that domain. A message consists of a header and a body. Besides the control information items that are necessary, for instance, for message routing, the header contains the name of a cryptographic key. The message body is encrypted by using the value of this key.

Two nodes, a sender node S and a receiver node R, can communicate by message exchanges if they share a domain. This means that both nodes hold a key for that domain. Node S will use the key to create a new message. The message header will contain the key name, the body will be encrypted by using the key value. On receipt of the message, node R reads the key name in the message header, and uses the key value to decrypt the body. Let $N_1, N_2, \ldots$ be the nodes in the routing path from S to R. Suppose that these intermediate nodes have no access privilege for the domain of the messages exchanged by S and R, as they do not possess the key. They can read the message header, as is necessary for message routing, for instance. However, they cannot decrypt the message body, as they do not possess the key. In fact, the key creates a communication channel between S and R through the intermediate nodes $N_1, N_2, \ldots$.

An *application* is the result of the joint actions of two or more sensor nodes, the *application members*, which cooperate in the same task by message exchange. In our protection model, an application consists of sensor nodes that hold access privileges for the same protection domain in the form of a key, the *application key.* They will use this key to create messages in this domain, and to read the messages in this domain. If the routing path between nodes S and R includes a node N which is part of the same application, then N is in the position of decrypting the messages, as it holds the necessary access permission (i.e. the application key). This is not a protection violation, as the nodes of the same application are considered mutually trustworthy.

Each node can host only a single application. This is a consequence of the lack of address space separation between the software components running in a given node. In fact, a software routine that is executed in a node supporting two applications would be in the position of using the cryptographic keys of both these applications. Consequently, the messages exchanged by the members of one application will be known to the members of the other application; this is a security violation we are aimed at avoiding.

1.2 Tree shaped node hierarchy

The protection model based on nodes (subjects), messages (protected objects) and keys (protection domains) allows us to make the physical topology of the sensor network independent of the logic topology, expressed in terms of relationships between the nodes. At the physical level, we shall not hypothesize any specific network configuration. In fact, the network configuration is subject to change dynamically, e.g. new wireless links can be generated between the network nodes, and existing links can disappear, as a consequence of node mobility, or variations of the wireless transmission strength of the nodes [17].

In contrast, at the logical level, we shall hypothesize that the nodes are structured in a tree shaped hierarchy, whereby each node has only one parent node [18], [19]. In an example of an organization of this type, the sensor nodes at the lowest hierarchical level are partitioned into applications. In each application, a node acts as an *application server*, which can be more resource rich than a traditional sensor node, and is responsible for collecting data from all the other sensor nodes of that application. The applications are grouped to form application categories. In each category, a *category server* is aimed at transforming the collected data into a compact form suitable for transmission to the *base station.* In turn, the base station is responsible for the final presentation and delivery of the results of the environmental observations of the entire network. Thus, we have a tree shaped hierarchy of four levels. At level 0, the base station embodies the root. At level 1, we have a node for each application category, which is the server of that category. Level 2 includes all application servers. Finally, at level 3, the sensor nodes are in contact with the external environment. Messages flow across the hierarchical levels. An application server receives messages from the sensor nodes of its application, a category server communicates with all the application servers in its category, and finally, the base station communicates with all the category servers.

In a previous paper [20], a protection system paradigm based on subjects, objects, and access authorizations was considered with reference to security classes organized into a tree shaped hierarchy. A key derivation mechanism was proposed, which allows a subject that holds the key of a given class to transform this key into the keys of the descendent classes. In this paper, with reference to tree shaped wireless sensor networks, we consider the protection problems connected with the generation, distribution, and replacement of the cryptographic keys. In our model, each node is assigned a key to communicate with its ancestors. This key is called the *hierarchical key*, or *h-key* for short. Furthermore, each node shares a *level key*, or *v-key*, with all its siblings; the v-key is used to communicate with the siblings (at the lowest hierarchical level, the v-key used for communication between the sensor nodes of a given application is called the *application key*). The root is assigned a key, called the *base key* and denoted by h_{base}. This key is never distributed to other nodes, and is never used for message transmission. It is aimed at generation of all the other keys.

The rest of this paper is organized as follows. Section 2 illustrates our key model with special reference to key generation. A uniform scheme to assign names to nodes is introduced, based on the position of the given node in the node hierarchy. A single, publicly-known parametric one way function, the *key generation function*, is used to assign an h-key to each node in an iterative procedure that starts at the base key. A similar procedure is used to generate the v-keys. Section 3 considers the problems inherent in dynamic key management, i.e. the process of replacing the keys of all the nodes of the network, or of a subset of these nodes. Section 4 deals with dynamic access control, i.e. the ability to add new nodes to the node hierarchy, and to remove existing nodes from the hierarchy. Related problems are access privilege distribution and revocation. Section 5 discusses our approach to key management in tree shaped wireless sensor networks from a number of viewpoints, which include cryptographic key forging, the network traffic caused by a rekey, the memory requirements for key storage, and the relation of our work to previous work. Section 6 gives concluding remarks.

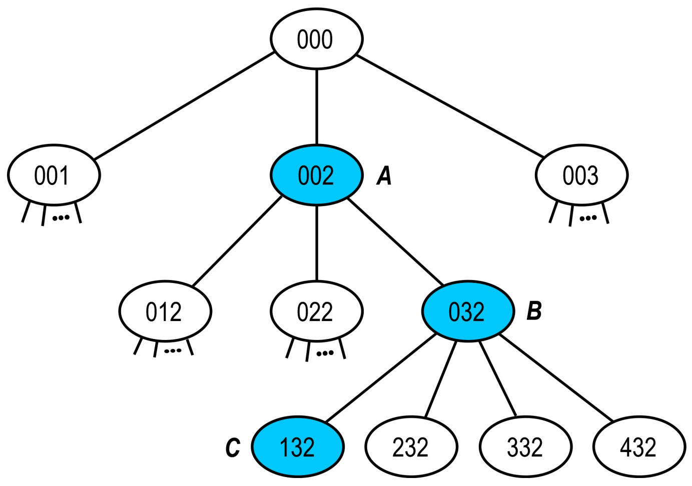

Figure 1: A four-level node hierarchy. The size of a node name is 12 bits, which are partitioned into three subnames, i.e. $q = 3$. The size of a subname is four bits, i.e. $p = 4$, and each node can have up to 15 children. In the hierarchy, the names of the nodes are shown in hexadecimal notation.

2 THE KEY MODEL

2.1 Node names

In our hierarchical model, each node can have up to $2^p - 1$ children, where quantity p is network specific. The children are numbered starting from 1. We partition the name N of a given node into q subnames, and the size of each subname is p bits. Thus, $N = (n_{q-1}, n_{q-2}, \ldots, n_0)$, where n_i denotes a subname, and the size of N is $p \cdot q$ bits. The subnames indicate the path from the root of the hierarchy to the given node, starting from subname n_0 that identifies a child of the root. The path terminates at the first subname which is cleared (i.e. all 0's).

Figure 1 shows an example of a four-level node hierarchy. In this example, we have $p = 4$, that is, each node can have at most 15 children. The size of a node name is 12 bits, which are partitioned into three subnames of four bits, so we have $q = 3$. A node name is denoted by three hexadecimal digits, one digit for each subname. The root is denoted by 000. The root has three direct children, namely 001, 002 and 003. The second child 002, labelled A in the figure, has three children, and their names are 012, 022, and 032. Thus, for instance, node 032, labelled B, is the third child of the second child of the root. This node has four children. The first child is labelled C; its name 132 indicates the path from the root to the second child, then to the third child, and finally, to the first subsequent child.

Thus, the most significant non-zero subname of the name of a given node indicates the position of this node between its sibling. Furthermore, the node name indicates the names of the direct ancestors, e.g. the name of the parent node can be obtained by clearing the

Table 1: Cryptographic keys.

Key name: (c, v, N)	where c identifies a class, v identifies a version, and N is a node name
Hierarchical key (*h-key*)	$v = 0$ denotes an h-key. N is the name of the node for which the h-key was generated. The h-key of a given node is used by that node to communicate with its ancestors.
Level key (*v-key*)	$v > 0$ denotes a v-key. N is the name of the parent node. Each node shares a v-key with its siblings to communicate with the siblings.

most significant non-zero subname. In the example of Figure 1, node 132 is the first child of node 032, for instance.

2.2 Cryptographic keys

As seen in Section 1, each key has a name and a value. A key name consists of a triple (c, v, N), where the c and v fields denote the *class* and the *version* of the key, and N denotes a node (Table 1). When the system is generated, the class is 0 for all keys. The class is incremented by 1 at each total rekey. The total rekey causes the replacement of base key h_{base} (the key of the root). Consequently, all h-keys and v-keys are replaced, as they descend from the base key; this issue will be analysed in depth shortly.

In an h-key name, the node name identifies the node for which the h-key was generated. The version number is always 0. In the v-key of the children of a given node, the node name is that of the parent node. The version number is set to 1 when the system is generated, and is incremented by 1 at each subsequent replacement of the v-key value.

For simplicity, we shall hypothesize that the size of the class and version fields of a key name is equal to the size of a subname, i.e. p bits. In the representation of a key name, the class and the version precede the node name. For instance, if a node name consists of three subnames ($q = 3$) and the size of a subname is four bits ($p = 4$), then the size of a key name is 20 bits, where the four most significant bits specify the class, and the four subsequent bits specify the version. Thus, 10032 is the h-key in class 1 of node 032 (labeled B in the node hierarchy of Figure 1), where the most significant digit indicates the class, and the subsequent digit indicates the version (equal to 0 for an h-key). Furthermore, 12032 is the second version of the v-key in class 1 of the children of node 032.

2.3 Hierarchical keys

All keys, both h-keys and v-keys, descend from base key h_{base} by an iterative procedure, which is based on application of a parametric one way function, the key generation function.

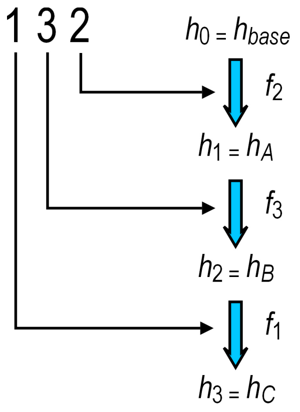

Figure 2: Utilization of the key generation function to derive the h-keys of the nodes A to C of Figure 1 from base key h_{base}.

Function f is *one way* if, given a value x, it is computationally easy to evaluate $f(x)$, but given a value y, it is computationally unfeasible to find a value x so that $y = f(x)$ [21], [22]. Function $f_n(x)$ is a *parametric one way function* if, given a parameter n and a value y, it is computationally unfeasible to find a value x so that $y = f_n(x)$ [23]. The effort connected with the design and implementation of an effective parametric one way function can be minimized by using a good cryptosystem as a base [24], [25]. For instance, we can encrypt parameter n using value x as the key, i.e. $f_n(x) = E_x(n)$ [26].

The key generation function is a parametric one way function having the form $f_n(h)$, where argument h is an h-key, and parameter n is a subname. Let $N = (n_{q-1}, n_{q-2}, \ldots, n_0)$ be a node name expressed in terms of its subnames. H-key h_N of node N derives from base key h_{base}. We have $h_{r+1} = f_{n_r}(h_r)$, $r = 0, 1, \ldots, c-1$, where c is the index of the least significant subname of N which is cleared, $h_0 = h_{base}$, and $h_N = h_c$. If no subname of N is cleared, then $c = q$.

Figure 2 shows the evaluation of the h-keys of nodes A to C in the node hierarchy of Figure 1. In this example, each node can have up to 15 children ($p = 4$). A node name is partitioned into three subnames ($q = 3$), so we have $N = (n_2, n_1, n_0)$, and each subname corresponds to a hexadecimal digit. The hexadecimal name of node C is 132. In our iterative procedure to generate h-key h_C of node C, we have $h_0 = h_{base}$. Subname n_0 is 2, so we have $h_1 = f_2(h_0)$. Subname n_1 is 3, and consequently $h_2 = f_3(h_1)$. Subname n_2 is 1, thus $h_3 = f_1(h_2)$. This terminates the procedure, and $h_C = h_3$. It is important to note that the intermediate h-keys correspond to the nodes in the path from the root to node C. Thus we have $h_A = h_1 = f_2(h_{base})$, $h_B = h_2 = f_3(h_A)$ and $h_C = h_3 = f_1(h_B)$.

A node at a given level in the hierarchy can use its own h-key to generate the h-keys of all its descendants. To this aim, the node executes the iterative key derivation procedure, described above. This means, for instance, that node A in Figure 1 can generate the h-key $h_B = f_3(h_A)$ of node B (its third child), and the h-key $h_C = f_1(h_B) = f_1(f_3(h_A))$ of node

C (the first child of B).

2.4 Level keys

Let h_N be the h-key of a given node N. Version 1 of the v-key shared by the children of N is given by $f_{2^p}(h_N)$, and version v of this v-key is given by $f_{2^p+v-1}(h_N)$. Thus, quantities $f_0(h_N)$ to $f_{2^p-1}(h_N)$ are the h-keys of the children of N, and quantities $f_{2^p}(h_N)$ to $f_{2^{p+1}-2}(h_N)$ are the versions of the v-key of these children.

With reference to the node hierarchy of Figure 1, where $p = 4$, version 1 of the v-key of the children of the root is given by $f_{16}(h_{base})$, and version 2 by $f_{17}(h_{base})$, for instance. Version 1 of the v-key of the children of node A is given by $f_{16}(h_A)$, that is, $f_{16}(f_2(h_{base}))$, and version 2 is given by $f_{17}(f_2(h_{base}))$.

Each given node N is in the position to evaluate the v-key of its own children. To this aim, N maintains a *v-key version counter* whose contents indicate the next version. This counter is set to 1 at system initialization, and is incremented by 1 at each subsequent change of the v-key version. Let v be the value of this counter at a given time. The corresponding v-key at that time is given by $f_{2^p+v-1}(h_N)$, where h_N denotes the h-key of node N.

It is worth noting that node N may be unable to evaluate the v-keys of the nodes that are not its direct children, as N not necessarily knows the versions. This fact poses no restriction on the ability of N to communicate with its descendants, as it can use the h-keys. In fact, the v-keys are only aimed at communication between siblings, e.g., within the boundaries of a given application, between the sensor nodes that are members of that application.

3 DYNAMIC KEY MANAGEMENT

Dynamic key management is the process of replacing the keys of all the nodes of the network, or of a subset of these nodes, periodically or on demand, according to variations of the network state [27]. This is a peculiar aspect of wireless sensor networks. Periodic rekeying can be useful to safeguard secrecy and resilience to attacks or failures, for instance [28], [29], [30].

When the hierarchical system is generated, the class is 0 for all keys. The class is incremented by 1 at each total rekey. The root chooses base key h_{base} at random. For each child node, the root uses the key generation function f to produce its h-key, which, for the i-th child, is given by $f_i(h_{base})$. The h-key is delivered to the child. Furthermore, the root generates the first version of the v-key of its children, given by $f_{2^p}(h_{base})$, and distributes this v-key to the children. In turn, each child of the root generates the h-keys of its own

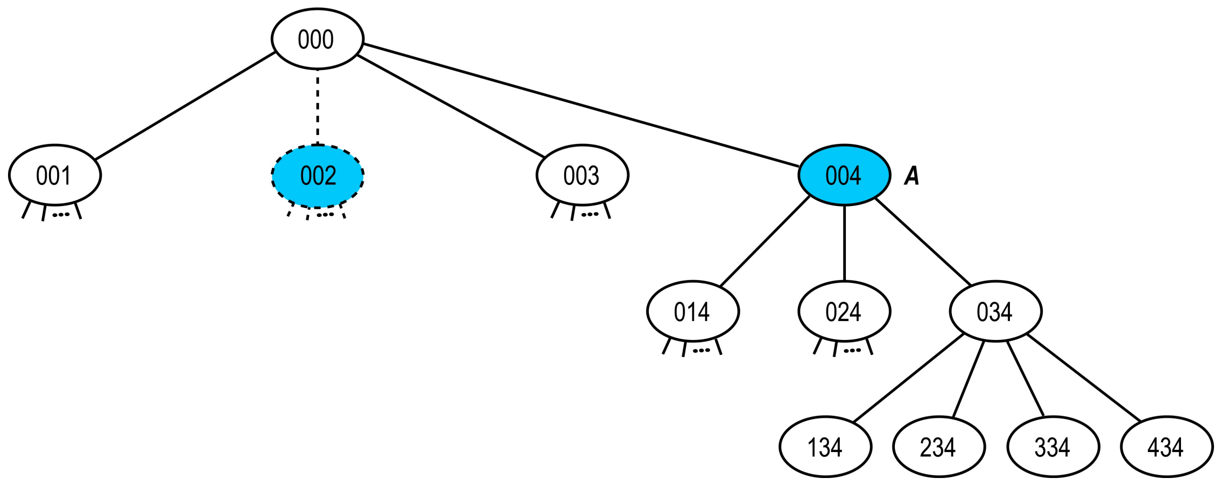

Figure 3: Partial rekey of the node hierarchy shown in Figure 1, obtained by renaming node A.

children, and the first version of their v-key. These keys are delivered to the children. This iterative key generation process propagates down to the lowest level of the hierarchy.

3.1 Partial rekey

Our system supports a form of partial rekey, whereby new keys are assigned to the nodes in a subtree of the node hierarchy. This functionality takes advantage of the fact that, for a given base key, the name of a node determines both the h-key of that node and the v-key of its children. The partial rekey is obtained by changing the name of the node that is the root of the subtree to be rekeyed (the *subroot*). The subroot is moved to the first position after its siblings: its previous position is discarded. Consequently, all the nodes in the subtree are renamed.

Figure 3 shows an example of a subtree renaming with reference to the node hierarchy of Figure 1. Node A is moved to the first position after its siblings, and its previous position is discarded. The name of A was 002; in the new position, it is 004. All the descendants of A are renamed accordingly. The parent of node A, i.e. the root, generates a new h-key for A, given by $f_4(h_{base})$. In turn, A uses this h-key to generate the h-keys of its children, and their v-key. These keys are distributed to the children. The process is iterated down to the lowest level of the hierarchy.

As seen in Section 2.1, a node can have no more than $2^p - 1$ children, where quantity p is network specific. Our partial rekey mechanism tends to exhaust available child numbers. In a situation of this type, a solution is a total network rekey.

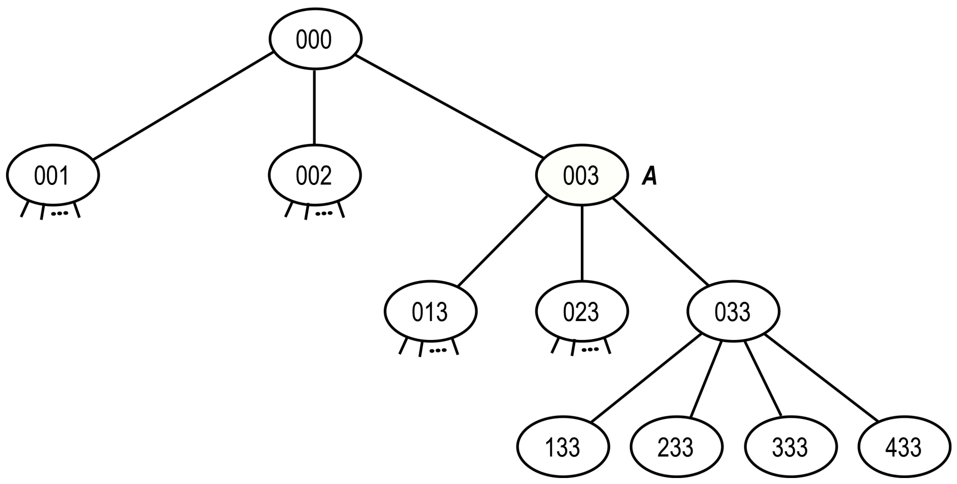

Figure 4: The node hierarchy of Figure 3 after a total rekey. The discarded position in the second hierarchical level has been eliminated. The subsequent nodes and their subtrees have been renamed.

3.2 Total rekey

A total rekey is a form of key replacement that involves all h-keys and v-keys. It is similar to the key derivation and distribution process, which takes place when the system is generated. The rekey implies a change of the class, which is incremented by 1. Consequently, a new base key h_{base} is chosen at random. All v-key version counters are set to 1, so that, for a node whose h-key is h, the v-key of its children is given by $f_{2^p}(h)$. Furthermore, the nodes in every given level of the hierarchy are renumbered consecutively, superseding any previous action of node renaming, generated, for instance, by a partial rekey. In this way, a total rekey prevents exhaustion of available node names and v-key versions.

Figure 4 shows the effects of a total rekey on node names in the node hierarchy of Figure 3. The discarded position in the second hierarchical level has been eliminated, and consequently, all the subsequent nodes and their descendants have been renamed.

A total rekey starts at the root, which uses the new base key to generate the h-keys of its children, and their v-key. In the name of each new key, the class field contains the new class number. The keys are sent to the children, which propagate the key derivation process down to the lowest level of the hierarchy. On termination, all keys belong to the new class.

3.3 V-key replacement

Suppose that a sensor node is evicted from a given application. We must revoke all the access privileges held by that node, to prevent it from taking advantage of these access privileges any longer. In particular, we must revoke the v-key shared by the evicted node and its siblings, and we must distribute a new v-key to the siblings. So doing, we prevent the evicted node from decrypting the new message traffic, for instance, if it is in a routing path connecting the siblings (*forward secrecy*). A similar v-key replacement action is also necessary when a new node is added to an application, to prevent that node from decrypting the old message traffic, if it has recorded this traffic (*backward secrecy*) [31], [32].

The replacement of a v-key is accomplished by the parent node, e.g., in the example of Figure 1, by node 032 for the v-key of its children nodes, 132 to 432. The parent node uses its own h-key h and v-key version counter v to generate the new v-key, given by $f_{2^p+v-1}(h)$ (see Section 2.4). Then, the v-key version counter is incremented by 1, and the new v-key is transmitted to all the children, or part of them, according to the specific implications of the event that caused the v-key replacement (e.g., when a given node is deleted, this node will be excluded from the v-key redistribution).

It is worth noting that a v-key replacement has no consequence on the h-keys. In fact, the h-keys descend from the node names, and these are not affected by the replacement.

3.4 Rekey messages

Let us suppose that node S holds h-key h_S and encrypts a message M by using this key. Then, S sends the message to node R, which holds an h-key h_R for the same node (i.e. it specifies the same node name). Let c_{h_S} denote the class of h_S, and c_{h_R} denote the class of h_R. If $c_{h_S} = c_{h_R}$, then these two keys are identical, R can decrypt the message, and the message transmission and delivery are successful. If $c_{h_S} < c_{h_R}$, then h_S is outdated. This means that either node S encrypted the message before updating the key, or a rekey message was lost, or S is no longer part of the network and it does not participate in the rekey. Node R cannot discriminate between these situations, so it discards the message and sends a negative acknowledgement to S.

If $c_{h_S} > c_{h_R}$, then h_R is outdated. In this case, node R sends quantity c_{h_R} to its parent node P asking for an updated key. P compares c_{h_R} with the class of its own h-key, c_{h_P}. If $c_{h_R} < c_{h_P}$, then P uses h_P, the position of R between its siblings, and the key generation function to derive a new h-key for R, as has been illustrated in Section 2.3. The new h-key is finally transmitted to R. If $c_{h_R} \geq c_{h_P}$, P propagates the key update request to its own parent. Propagation proceeds upwards in the class hierarchy, in the direction of the root, until a node is reached (possibly the root), which holds a key of a class newer than c_{h_R}. The rekey starts in this node and proceeds downwards, involving all intermediate nodes until the original node R is reached.

The considerations outlined above can be extended to the case that the original message M has been encrypted by using a v-key. In this case, the classes are compared first, and then (in the case of a class match) the versions, to determine whether the v-keys of the two communicating nodes are both updated, or a rekey is needed. The rekey is obtained by transmitting a request to the parent node, which uses its own h-key and v-key version counter to generate the v-key of its children. This new v-key is delivered to the children.

We may conclude that our rekey mechanism is able to cope with losses of rekey messages. A node that receives a message encrypted by using a given key is in the position to detect whether it holds an outdated key, by comparing the classes, and possibly the versions. The rekey process involves the ancestor nodes at higher hierarchical levels. This feature is especially important for a reliable application rekeying in an unreliable network [28].

4 DYNAMIC ACCESS CONTROL

Dynamic access control is the ability to add new nodes to the node hierarchy, and to delete existing nodes from the node hierarchy [33]. Related problems are access privilege distribution and revocation.

4.1 Adding a node

Let us refer again to the node hierarchy introduced in Section 1, featuring a base station, category servers, application servers, and sensor nodes grouped into applications. When a new category is generated, subname n_0 of the name of the category server identifies its position among the category servers. Similarly, when a new application is added to an application category, subname n_1 of the name of the application server identifies its position among the application servers in that category. Finally, when a new sensor node joins an application, subname n_2 identifies its position among the sensor nodes of that application.

Figure 5 shows the addition of new nodes to the node hierarchy of Figure 1. In this example, a new application category is introduced; its category server is named 004, where $n_0 = 4$ denotes the position assumed by this node after the existing application servers. Furthermore, a new application is added to the hierarchy; its server is named 042, where $n_1 = 4$ denotes the position of this application in its category. Finally, a new sensor node is added to the application whose server is node 032. The new node is assigned name 532, where $n_2 = 5$ denotes the position of the new node in the application.

When a node is added to the network, it is necessary to grant this node access privileges congruent with the position it assumes in the hierarchy. These access privileges include the ability to communicate with its siblings and ancestors. To this aim, the parent node uses its own h-key and the name of the new node to generate the h-key of the new node. Furthermore, as seen in Section 3.3, the v-key of the siblings of the new node are replaced, for backward secrecy. The parent node uses the contents of its own v-key version counter

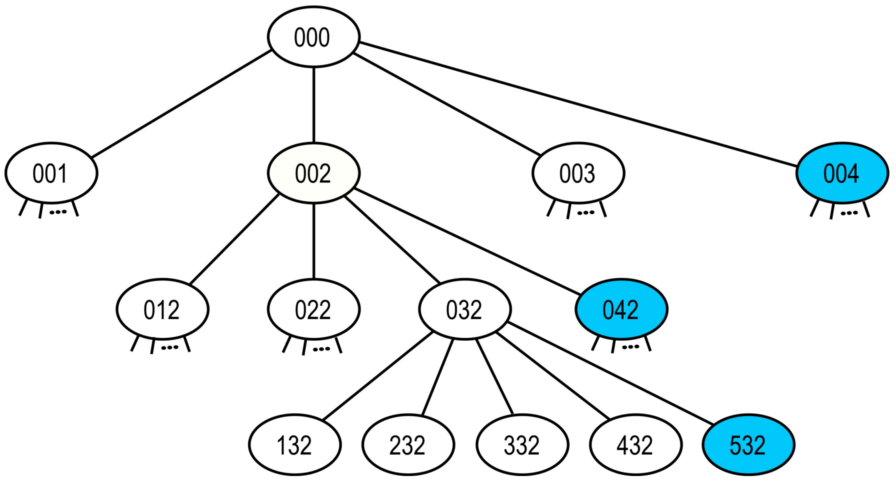

Figure 5: Modification of the node hierarchy shown in Figure 1, by the addition of a category server (004), an application server (042), and a sensor node (532).

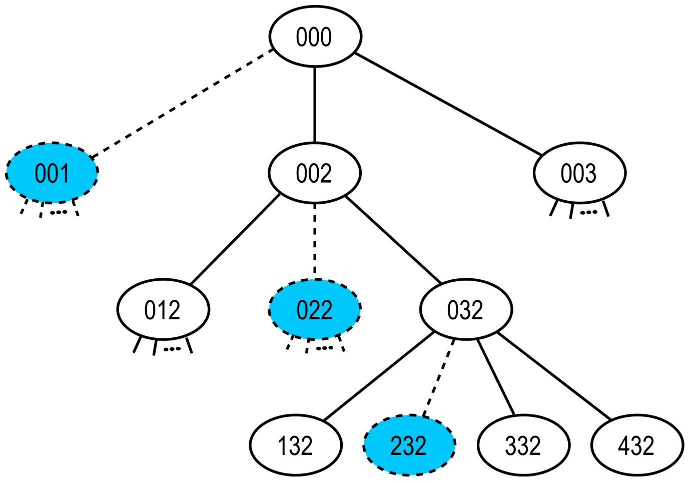

Figure 6: Modification of the node hierarchy shown in Figure 1, by elimination of the application category whose server is 001, the application whose server is 022, and sensor node 232.

to generate a new version of the v-key, which is distributed to the children, including the new node.

4.2 Deleting a node

When a node is deleted, the corresponding node name is discarded. Figure 6 shows the deletion of the application category whose server is node 001, of the application whose server is node 022, and of sensor node 232 at the lowest hierarchical level. Consequently, node names 001, 022, and 232 are discarded.

Node names cannot be reused. In fact, the iterative procedure for key derivation uses the name of a given node to generate the h-key of that node. It follows that, if a node name is reused for a different node, the new node is assigned the h-key of the previous node. Consequently, the new node is allowed to access old messages exchanged by the previous node, if it has recorded these messages. This is a security hole we are aimed at avoiding. In the example of Figure 6, the names 001, 022, and 232 of the deleted nodes cannot be used to identify new nodes. Of course, a solution is a total network rekey, causing all network nodes to be renamed (see Section 3.2). In this case, any previous action of node deletion is superseded. The rekey generates all keys from scratch, and these keys are in a new class. The class discriminates the old keys, which become useless.

A related problem is connected with the fact that, as seen in Section 2.1, a node cannot have more than $2^p - 1$ children, where quantity p is network specific. Repeated actions of node replacement (the deletion of a child of a given node followed by the addition of a new child to that node) may lead to exhaustion of available node numbers. For adequate values of p, this will be rarely the case. A solution is a total network rekey, which removes all discarded positions and restores the ability of the node numbering procedure to add new nodes to the hierarchy.

Let us first consider the elimination of a leaf node, e.g. an application member at the lowest hierarchical level. In Figure 6, this is the case for node 232. This node possesses an h-key, which it received from its parent node. Utilization of this h-key is restricted to messages exchanges with the ancestor nodes, i.e. application server 032, category server 002, and the root. Revocation of the h-key will be simply obtained by instructing the ancestors to discard any message encrypted by using this h-key. The leaf node also holds a v-key, which it shares with the other nodes of the same application. It is necessary to switch these nodes to a new v-key, for forward secrecy. This v-key replacement action will be accomplished as has been illustrated in Section 3.3.

Let us now consider the deletion of a node at an intermediate hierarchical level. In Figure 6, this is the case for nodes 001 and 022. The ancestors of the deleted node will be instructed to discard any message encoded by using the h-key of that node. A v-key replacement action will take place involving the siblings of the deleted node, for forward secrecy. The nodes in the subtree of the deleted node will be redistributed in the network. For each of them, the actions involved in a node addition will be carried out.

5 DISCUSSION

5.1 Cryptographic keys

Hierarchical keys are used for secure communications between a node and its server, so that another node in the routing path between that node and the server cannot read the messages. Examples are the communications between a sensor node member of a given application and the application server, between an application server in a given category and the category server, or between a category server and the base station.

An important h-key functionality is related to key distribution. Let us consider the case that a sensor node has been evicted from a given application, for instance. As seen in Section 3.3, the v-key (application key) must be replaced, for forward secrecy. The application server generates a new v-key using its own v-key version counter and the key generation function (see Section 2.4). The new v-key is transmitted to each application member in a message encrypted by using the h-hey of that member. Of course, the new v-key cannot be transmitted by using the old v-key. In fact, if the evicted node is in the routing path between the application server and an application member, it can capture the message. It possesses the old v-key, and can decrypt the message to read the new v-key.

V-keys make communications possible between members of the same application, between application servers in the same application category, and between category servers. For instance, let us consider two nodes A_1 and A_2 that are members of the same application A. When A_1 sends a message to A_2, a node in the same application, which is in the routing path between A_1 and A_2, can capture the message and decrypt its contents, as this node holds the v-key. This is not a security hole, as we hypothesized that the members of the same application are mutually trustworthy. On the other hand, a node which is not a member of application A will not be able to decrypt the message, as it does not hold the v-key.

A node cannot be granted more than a single v-key, e.g. the v-keys of two different applications, as this would allow this node to forward the messages exchanged between the members of one application to the members of the other application. This is a security hole we are aimed at avoiding. Consequently, v-keys cannot be used for communications between the members of two different applications, as this would imply that a node possesses the v-keys of both of them. Instead, inter-application interactions are obtained via the application servers, which can communicate by taking advantage of the v-key of their application category. If the applications are in different categories, category servers will be used.

5.2 Network traffic

As seen in Section 1, in a sensor network the amount of message traffic is a significant parameter, which must be kept low to comply with the stringent requirement to save energy. In our approach, the number of messages exchanged in a key replacement process is kept to a minimum. In a total rekey, each node sends only two keys to each child, the h-key of this child and the v-key shared by this child and its siblings. The process is initiated by the root, and propagates down to the lowest hierarchical level. In a partial rekey, the propagation initiates at the parent of the subroot involved in the rekey, which sends the new h-key and v-key to the subroot.

5.3 Memory requirements

As anticipated in Section 1, in a sensor node stringent limitations exist on the amount of available memory. It follows that the memory space for cryptographic key storage is a significant issue.

A key has a name and a value. The size of a key value descends from the overall security requirements of the system, e.g. 128 bits. A key name consists of a class field, a version field and a node name. If the size of the class and version fields is 8 bits, we can have up to 256 classes, and up to 256 versions for each class. The node name is partitioned into subnames. For instance, in a hierarchical organization featuring a base station, category servers, application servers and sensor nodes, the node name is partitioned into three subnames, namely n_0 that designates a category, n_1 that identifies an application in the category, and n_2 that identifies a sensor node within the application. The three subnames can have different sizes, e.g. if the size of n_0 and n_1 is 4 bits and the size of n_2 is 8 bits, we can have up to 15 categories each consisting of up to 15 applications, and each application can include up to 255 members. In a large network of this type, the size of a key is 20 bytes, 4 bytes for the key name and 16 bytes for the key value.

In our approach, an application member holds two keys, that is, an h-key to communicate with its ancestors, and a v-key to communicate with the other members of the same application. An application server needs a v-key to communicate with the other application servers in the same category, and an h-key to communicate with the ancestors. A further requirement would be an h-key for each application member, to communicate with that member. In the server of a large application with many members, the resulting memory requirements for h-key storage would result prohibitive. In fact, in our approach, the application server keeps a single h-key in memory, i.e. its own h-key. When it needs to communicate with a given application member, it generates the h-key of this member from its own h-key h by using key generation function f. As seen in Section 2.3, the h-key of the n-th application member is given by $f_n(h)$. Similar considerations can be made for the nodes at higher hierarchical levels, up to the base station that has to store a single key, the base key h_{base}. All the h-keys and the v-keys can be generated starting from h_{base} by iterated applications of the key generation function.

We may conclude that the memory requirements for key storage are extremely low, and represent a negligible fraction of the total memory resources of each network node.

5.4 Key forging

The root is assigned a first base key at system generation; this base key is in class 0. The base key changes at each variation of the class, e.g. a total rekey. Base keys must be large and sparse. In this way, if a malevolent node tries to use a key chosen at random for the base key, or for a key derived from the base key, the probability of success is vanishingly low.

Each node can take advantage of its own h-key and the key generation function to derive the h-keys of its descendants. Let us now consider the case of a node N that tries to amplify its own h-key h_N by transforming it into the h-key h_M of a node M at a *higher* hierarchical level (that is, M is an ancestor of N). This means that h_M *precedes* h_N in the key generation procedure that uses the key generation function to derive h_N from the base key. But the key generation function is one way. It follows that it is impossible to invert this function, and the h-key amplification attempt is destined to fail.

5.5 Relation to previous work

The problem of key generation in hierarchic organizations has been considered in depth in the past. Objects can be grouped into security classes that are structured hierarchically. Class hierarchies can be classified according to the number of direct ancestors (parents) and direct descendants (children) [10]. In a directed acyclic graph, each class can have many parents and many children. In a tree shaped hierarchy, each class can have a single parent and many children. Finally, in a linear hierarchy each class can have a single parent and a single child. In this paper, we have considered tree shaped hierarchies, for their prominence in the ambit of wireless sensor networks.

Hassen et al. [34] classify the key management schemes into two broad categories, *independent-keys* and *dependent-keys*. In an approach in the independent-keys category, a subject in a given security class holds the key of that class and the keys of all the descendant classes. This means that more keys must be maintained by a subject in a class that is not a leaf class, and this is especially true for subjects in higher level classes, which have to administer a large number of keys. A situation of this type is prone to severe security issues [35]. High costs in terms of network traffic are connected with an action of partial or total rekey, as the new key of each given class must be delivered to all its ancestors. In wireless sensor networks, the number of messages exchanged between the network nodes must be kept to a minimum to save energy. Memory is a scarce resource, so the memory requirements for key storage are a significant issue.

In contrast, in the dependent-keys approach, a subject in a given class holds a single key, the key of that class, and takes advantage of universally known functions and/or parameters to derive the keys of the descendant classes. To this aim, complex theoretical cryptographic techniques have been used, e.g. prime number fundamental properties [35], [36], [37], [38].

Our proposal is in the dependent-keys category. A publicly-known parametric one way function, the key generation function, allows each node that is not a leaf of the hierarchy to derive the h-keys of all its descendants. A seen in Section 2.3, the effort to design and implement an effective parametric one-way function can be kept to a minimum, by using a good symmetric cryptosystem as a base.

A peculiar aspect of wireless sensor networks is that every given node needs to communicate with other nodes at the same hierarchical level. In an application, this is the case for the application members, for instance. We take advantage of different keys, h-keys and v-keys, for different functionalities. Each node holds an h-key to communicate with its ancestors, and a v-key to communicate with its siblings. In particular, h-keys are used in v-key replacement. When a parent node sends the new v-key to a given child node, the v-key transmission message is encrypted by using the h-key of that child node.

The idea of using different keys for different functionalities is certainly not new [14]. For instance, in [39] each node holds an individual key to communicate with the base station, a pairwise key shared with each of its immediate neighbouring nodes, a cluster key shared with all the neighbouring nodes, and a global key shared with all the network nodes. It has been reputed that key differentiation can improve the overall system security, and can facilitate key management [31].

6 CONCLUDING REMARKS

With reference a distributed architecture consisting of sensor nodes connected by wireless links, we have presented a key management paradigm in the hypothesis that, at a logical level, the nodes are organized in a tree shaped hierarchy. We have considered the protection problems connected with the generation, distribution, and replacement of the cryptographic keys. In our paradigm:

- Names are assigned to nodes by using a uniform scheme based on the position of each given node in the node hierarchy.
- Each node possesses an h-key to communicate with its ancestors, and a v-key to communicate with its siblings.
- A single, publicly-known parametric one way function, the key generation function, is used to assign an h-key to each node, in an iterative procedure that starts from the base key at the root of the node hierarchy, and proceeds downwards to the lowest hierarchical levels. A similar procedure is used to generate the v-keys.

We have obtained the following results:

- Each node has to store only two keys, an h-key and a v-key. Thus, the total memory requirements for key storage are extremely low, and represent a negligible fraction of the memory resources of each node. This is true even for resource-scarce sensor nodes, at the lowest hierarchical level.

- Nodes are prevented from generating the keys of the ancestors by the one way property of the key generation function.
- The number of keys exchanged in a key replacement process is kept to a minimum, and is equal to two keys for each node (an h-key and a v-key). In fact, in the dependent-keys approach [34], each node is able to generate the keys of its descendants autonomously, starting from its own h-key, and taking advantage of the key generation function.
- Each key name specifies a class to support total rekeys. If the key is a v-key, the version supports v-key replacement. Forms of partial rekey, whereby new keys are assigned to the nodes in a subtree of the node hierarchy, are implemented by changing the name of the subroot, and consequently renaming all the nodes in the subtree.
- Dynamic access control is fully supported. When a node is added to the network, the parent node uses its own h-key and the name of the new node to generate the h-key of the new node. When a node is deleted, its h-key is discarded by the ancestors. V-key replacements will be used to enforce forward secrecy, or, in the case of a node deletion, backward secrecy.

REFERENCES

[1] C.-Y. Chen and H.-C. Chao, “A survey of key distribution in wireless sensor networks,” *Security and Communication Networks*, vol. 7, no. 12, pp. 2495–2508, 2014.

[2] G. Dini and L. Lopriore, “Distributed storage protection in wireless sensor networks,” *Journal of Systems Architecture*, vol. 61, pp. 256–266, May–June 2015.

[3] V. C. Gungor and G. P. Hancke, “Industrial wireless sensor networks: challenges, design principles, and technical approaches,” *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, 2009.

[4] M. O. Farooq and T. Kunz, “Operating systems for wireless sensor networks: a survey,” *Sensors*, vol. 11, no. 6, pp. 5900–5930, 2011.

[5] R. Kumar, A. Singhania, A. Castner, E. Kohler, and M. Srivastava, “A system for coarse grained memory protection in tiny embedded processors,” in *Proceedings of the 44th Annual Conference on Design Automation*, (San Diego, California, USA), pp. 218–223, IEEE, June 2007.

[6] L. Lopriore, “Hardware support for memory protection in sensor nodes,” *Microprocessors and Microsystems*, vol. 38, pp. 226–232, May 2014.

[7] A. Dunkels, B. Grönvall, and T. Voigt, “Contiki – a lightweight and flexible operating system for tiny networked sensors,” in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors*, (Tampa, Florida, USA), pp. 455–462, IEEE, November 2004.

[8] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, *et al.*, “TinyOS: an operating system for sensor networks,” in *Ambient Intelligence*, pp. 115–148, Springer-Verlag, New York, 2005.

[9] L. Lopriore, “Memory protection in embedded systems,” *Journal of Systems Architecture*, vol. 63, pp. 61–69, February 2016.

[10] H. R. Hassen, H. Bettahar, A. Bouadbdallah, and Y. Challal, "An efficient key management scheme for content access control for linear hierarchies," *Computer Networks*, vol. 56, no. 8, pp. 2107–2118, 2012.

[11] L. Lopriore, "Password management: distribution, review and revocation," *The Computer Journal*, vol. 58, pp. 2557–2566, October 2015.

[12] R. S. Sandhu and P. Samarati, "Access control: principles and practice," *IEEE Communications Magazine*, vol. 32, pp. 40–48, September 1994.

[13] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Proceedings of the 2003 Symposium on Security and Privacy*, (Oakland, California, USA), pp. 197–213, IEEE, May 2003.

[14] C.-M. Chen, X. Zheng, and T.-Y. Wu, "A complete hierarchical key management scheme for heterogeneous wireless sensor networks," *The Scientific World Journal*, no. 816549, 2014.

[15] F. Hu and N. K. Sharma, "Security considerations in ad hoc sensor networks," *Ad Hoc Networks*, vol. 3, no. 1, pp. 69–89, 2005.

[16] J. Zhang and V. Varadharajan, "Wireless sensor network key management survey and taxonomy," *Journal of Network and Computer Applications*, vol. 33, no. 2, pp. 63–75, 2010.

[17] G. Dini and L. Lopriore, "Key propagation in wireless sensor networks," *Computers & Electrical Engineering*, vol. 41, pp. 426–433, January 2015.

[18] X. Chen and P. Yu, "Research on hierarchical mobile wireless sensor network architecture with mobile sensor nodes," in *Proceedings of the 3rd International Conference on Biomedical Engineering and Informatics*, vol. 7, (Yantai, China), pp. 2863–2867, IEEE, October 2010.

[19] A. K. Das, P. Sharma, S. Chatterjee, and J. K. Sing, "A dynamic password-based user authentication scheme for hierarchical wireless sensor networks," *Journal of Network and Computer Applications*, vol. 35, no. 5, pp. 1646–1656, 2012.

[20] L. Lopriore, "Key management in tree shaped hierarchies," *Information Security Journal: A Global Perspective*, published online: 7 Sep 2018.

[21] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk, "Cryptographic hash functions: a survey," tech. rep., Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Australia, 1995.

[22] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, pp. 770–772, November 1981.

[23] W. Trappe, J. Song, R. Poovendran, and K. J. Liu, "Key management and distribution for secure multimedia multicast," *IEEE Transactions on Multimedia*, vol. 5, no. 4, pp. 544–557, 2003.

[24] R. C. Merkle, "One way hash functions and DES," in *Proceedings of the 9th Annual International Cryptology Conference – Advances in Cryptology*, (Santa Barbara, California, USA), pp. 428–446, Springer, August 1989.

[25] B. Preneel, R. Govaerts, and J. Vandewalle, "Hash functions based on block ciphers: a synthetic approach," in *Proceedings of the 13th Annual International Cryptology Conference*, (Santa Barbara, California, USA), pp. 368–378, Springer, August 1993.

[26] R. S. Sandhu, "Cryptographic implementation of a tree hierarchy for access control," *Information Processing Letters*, vol. 27, no. 2, pp. 95–98, 1988.

[27] X. He, M. Niedermeier, and H. De Meer, "Dynamic key management in wireless sensor networks: a survey," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 611–622, 2013.

[28] F. Kausar, S. Hussain, J. H. Park, and A. Masood, "Secure group communication with self-healing and rekeying in wireless sensor networks," in *Proceedings of the Third International Conference on Mobile Ad-Hoc and Sensor Networks*, pp. 737–748, Beijing, China: Springer, December 2007.

[29] T. Pham and P. A. Watters, "The efficiency of periodic rekeying in dynamic group key management," in *Proceedings of the Fourth European Conference on Universal Multiservice Networks*, (Toulouse, France), pp. 425–432, IEEE, February 2007.

[30] S. Rafaeli and D. Hutchison, "A survey of key management for secure group communication," *ACM Computing Surveys*, vol. 35, no. 3, pp. 309–329, 2003.

[31] X. Chen, K. Makki, K. Yen, and N. Pissinou, "Sensor network security: a survey," *IEEE Communications Surveys & Tutorials*, vol. 11, pp. 52–73, second quarter 2009.

[32] M. A. Moulavi and H. Parvar, "Agent based bandwidth reduction for key management in hierarchical group communication," in *Proceedings of the 2nd International Conference on Communication Systems Software and Middleware*, (Bangalore, India), pp. 1–5, IEEE, January 2007.

[33] C. Yang and C. Li, "Access control in a hierarchy using one-way hash functions," *Computers & Security*, vol. 23, no. 8, pp. 659–664, 2004.

[34] H. R. Hassen, A. Bouabdallah, H. Bettahar, and Y. Challal, "Key management for content access control in a hierarchy," *Computer Networks*, vol. 51, no. 11, pp. 3197–3219, 2007.

[35] L. Harn and H.-Y. Lin, "A cryptographic key generation scheme for multilevel data security," *Computers & Security*, vol. 9, no. 6, pp. 539–546, 1990.

[36] 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, 1983.

[37] J. Crampton, K. Martin, and P. Wild, "On key assignment for hierarchical access control," in *Proceedings of the 19th IEEE Computer Security Foundations Workshop*, (Venice, Italy), pp. 1–14, IEEE, July 2006.

[38] A. De Santis, A. L. Ferrara, and B. Masucci, "Cryptographic key assignment schemes for any access control policy," *Information Processing Letters*, vol. 92, no. 4, pp. 199–205, 2004.

[39] S. Zhu, S. Setia, and S. Jajodia, "Leap+: Efficient security mechanisms for large-scale distributed sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 4, pp. 500–528, 2006.