S^2RP: a Secure and Scalable Rekeying Protocol for Wireless Sensor Networks

Gianluca Dini and Ida Maria Savino
University of Pisa
Dipartimento di Ingegneria dell'Informazione
via Diotisalvi 2, 56100 PISA Italy
{g.dini, ida.savino}@iet.unipi.it

***Abstract*— Nowadays, small, low-cost sensor nodes are being widely used to build self-organizing, large-scale, wireless networks for various applications, such as environmental surveillance, health monitoring and so on. Given its unique features, protecting a wireless sensor network is a difficult challenge.**

In this paper, we present S^2RP, a secure and scalable rekeying protocol for sensor networks. S^2RP aims at a trade-off between security and resource consumption while it guarantees an authentic distribution of keys that preserves both forward and backward security. The proposed protocol is efficient in terms of communication overhead as it reduces the number and the size of rekeying messages. It is efficient in terms of computation overhead as it guarantees the necessary level of confidentiality and authenticity of rekeying messages by only using symmetric ciphers and one-way hash functions. It follows that S^2RP meets the reduced capabilities of sensor nodes, results scalable, and particularly attractive for large and/or highly dynamic groups.

I. Introduction

The research community has recently acknowledged the importance of Wireless Sensor Networks (WSNs) and has proposed their application in several fields of research. WSNs will enable a wide range of applications to sense and control the physical world, such as civil and military surveillance, health monitoring and so on [1].

A major benefit of WSNs is that they can perform in-network processing to reduce large streams of raw data into useful aggregated information. However, protecting WSNs poses unique challenges. First of all, unlike traditional wired networks, an adversary with a simple radio receiver/transmitter can easily eavesdrop as well as inject/modify packets in a WSN. Second, in order to make WSN economically viable, sensor nodes are limited in their energy, computation, storage, and communication capabilities. Furthermore, sensor nodes typically lack adequate support to tamper-resistance. Therefore, the fact that WSN can be deployed over a large, unattended, possibly hostile area exposes each individual sensor node to the risk of being compromised.

In a WSN, sensor nodes cooperate towards a given application according to the *group communication* model [2]. According to this model, a sensor node becomes a new member of the group by explicitly joining it. As a member of the group, the sensor node may broadcast messages to the other members. Later on, the node may voluntarily leave the group, or, if compromised, may be forced to.

In order to protect group communication, group members share a *group-key*. They use this key to encrypt messages broadcast within the group so that anyone that is not part of the group can neither access nor inject/modify messages. When a sensor node joins the group, it must not be able to decipher previous messages encrypted with an old key even though it has recorded them (*backward security*). When a sensor node leaves, or is forced to leave, the group, the sensor node must be prevented from accessing the group communication (*forward security*).

In this model, backward and forward security are provided via rekeying [3]. When a member joins or leaves the group, the current group-key must be revoked and a new one must be distributed in order to guarantee both backward and forward security. This *reactive* approach has the advantage that a new node can immediately join the group and a compromised member can be promptly forced to leave as soon as it is discovered. However, the implementation of a reactive approach in a WSN poses two severe challenges. First, upon receiving new keying material, every sensor node must be able to immediately and efficiently verify its authenticity. Unfortunately, techniques based on public key cryptography, e.g., digital signatures, that are customary used to achieve broadcast authentication in traditional wired networks,

1-4244-0507-6/06/$20.00 ©2006 IEEE

cannot be used. Second, reactive rekeying may incur in a high communication overhead, especially in large and/or dynamic groups. Therefore, due to the severe resource limitations of sensor nodes, the communication overhead of the rekeying protocol has to be kept low.

In this paper we take up these challenges and present S^2RP, a Secure and Scalable Rekeying Protocol conceived for large, highly dynamic WSNs. The protocol is secure because it fulfills both the forward and the backward security requirements. The protocol is scalable because it requires $\mathcal{O}(\log n)$ messages to revoke the current group-key and distribute a new one, where n is the number of sensor nodes in the WSN. Furthermore, the protocol immediately authenticates new group-keys by only means of symmetric ciphers and one-way hash functions which are several orders of magnitude more efficient than public-key cryptography.

S^2RP has the merit of taking the problem of key revocation in WSN to the same level of importance as key distribution. The importance of key revocation stems from the observation that, if the cryptographic algorithms do not expose the secret keys, then secret keys can only be compromised by compromising sensor nodes. It follows that the ability to revoke keys translates into the ability to remove compromised nodes [4]. Actually, by revoking all keys of a compromised node, it is possible to remove the presence of that node from the WSN. Despite its importance, the problem of key revocation in WSN has received relatively little attention. Key revocation of pairwise keys has been studied in [5], [6] whereas key revocation of group-keys has been studied in [7].

S^2RP follows the centralized approach and has the additional merit of improving on the centralized schemes proposed so far [7], [5]. In a centralized revocation scheme, a *Key Management Service* has the task of revoking current keying material and redistributing new ones to all sensor nodes but the one that leaves, or is forced to leave, the group. In the centralized schemes proposed so far, the Key Management Service unicasts rekeying messages to sensor nodes that remain in the group. This requires $\mathcal{O}(n)$ messages. As it is shown in the paper, S^2RP accomplishes the same task with $\mathcal{O}(\log n)$ messages.

The protocol levers on two basic mechanisms: Logical Key Hierarchy and key-chains. *Logical Key Hierarchy* (LKH) is a technique for secure and scalable group rekeying that has been originally proposed for conventional wired networks and that allows us to achieve a group rekeying protocol where the number of messages is a logarithmic function of the network size [8], [9], [10]. *Key-chain* is an authentication mechanism based on the Lamport's one-time passwords [11] and that has been already profitably employed for efficient key authentication in wireless sensor networks [7]. Although, these mechanisms are not new, another merit of this work consists in proposing their integration. Actually, we prove that compounding them compounds their strengths and, therefore, their integration is conducive to improve scalability and security of group rekeying in WSNs. We originally proposed this integration in a preliminary work [12].

The paper is organized as follows. Section II introduces the system architecture. Section III illustrates the key-chain mechanism for key authentication. Section IV gives an overview of S^2RP and explains how it deals with the group operations join and leave. Sections V and VI give more details about the operational phases of the protocol. Section VII discusses the types of messages necessary for rekeying. Section VIII presents a performance analysis based on an early prototype and, finally, in Section IX we expose our concluding remarks.

II. SYSTEM ARCHITECTURE

We consider a WSN in which sensor nodes cooperate towards a given application according to the *group communication* model. In this model, a sensor node becomes a new member of the group by explicitly joining it. As a member of the group, the sensor node may broadcast messages to the other members. Later on, a node may leave the group wither voluntarily, when it terminates its mission, or be forced to, if compromised. After leaving a group, a node cannot send messages to, or receive messages from, that group, or join it again.

Members of a group can be further sub-grouped into *clusters*. A sensor node becomes member of a cluster when the node joins the group and remains member of that cluster as long as the node is member of the group. Sensor nodes may be clustered according various criteria. For example, clusters may be defined on geographical basis. The area over which the sensor nodes are distributed can be divided into grids. Sensor nodes that happen to be in the same grid are "close" to one another and belong to the same cluster. As a further example, sensor nodes can be clustered according to their functionality, e.g., their sensing function. It follows that sensor nodes that sense the same physical quantity (e.g., light, humidity, vibrations) are included in the same cluster.

In order to protect group communication from both a passive and an active external adversary, group members

share a *group-key* they use to encrypt messages within the group. Similarly, sensor nodes in the same cluster share a *cluster-key* to encrypt messages within the cluster. In this model, forward and backward security should be provided via rekeying, in which both the group-key and the cluster-keys are changed and distributed whenever a sensor node joins or leaves the network.

The group of sensor nodes is managed by a *Group Controller* (*GC*) that is composed of three main components: a *Group Membership Service* (*GMS*), a *Key Management Service* (*KMS*), and an *Intrusion Detection System* (*IDS*). The *GMS* component maintains the membership of the group by keeping track of sensor nodes that join and leave the group. A sensor node wishing to cooperate towards the WSN application joins the group as a member by invoking the join operation. Later on, the sensor node may decide to terminate its collaboration and explicitly leave the group by invoking the leave operation.

As individual sensor nodes are exposed to attackers, the *IDS* component probes/monitors network activities to uncover compromised nodes. Upon detecting a compromised sensor node, *IDS* forces the sensor to leave the group by invoking the leave operation and specifying the sensor node identifier as argument.

Whenever a sensor node joins, leaves, or is forced to leave the group, the group-key has to be renewed in order to guarantee the backward and forward security requirements. *KMS* is the component that is responsible to perform such a rekeying task. Upon handling a change in the group membership, *GMS* activates a rekeying by invoking the rekeying operation of *KMS* and specifying the kind of event, join or leave, that gave rise to the membership change. However, *KMS* gives also support to a periodic rekeying aimed at reducing the amount of encrypted material available to an adversary.

In a centralized approach, *GC* can be implemented by a more powerful computing node than sensor nodes. *GC* may well be a computing node such as a PC, a workstation, or a server, with plentiful of computational, storage, communication and power resources. Furthermore, we reasonably assume that *GC* is either tamper-resistant or physically protected and thus we exclude that it can be compromised by an external adversary. In the rest of the paper we detail the *KMS* component.

III. KEY-CHAIN: A MECHANISM FOR AUTHENTICATION

S^2RP achieves the authentication by employing one-way hash functions (OWHFs). A OWHF H is an hash function with the following additional properties: (a) given an input m, it is easy to compute $H(m)$ (*easy computation*); (b) given an output y for which the corresponding input is not known, it is difficult to find a preimage m such that $y = H(m)$ (*preimage resistance*); and (c) given an input m it is computationally infeasible to find a second preimage m such that $H(m\ ) = H(m)$ (*2nd-preimage resistance*) [13].

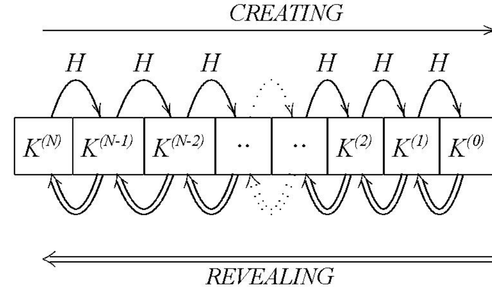

Fig. 1. Key-chain

We consider a sender S that broadcasts symmetric keys and a receiver R that must be able to verify the authenticity of the received keys. S uses the OWHF H for authentication purposes as follows. S computes a chain of symmetric keys so that each element in the key-chain is the image of the next one under H [11]. In detail, S randomly chooses a seed s and then builds the key-chain as follows: $s, H(s), H^2(s), \ldots, H^N(s)$, where $H^i(s)$ corresponds to applying i times H on s and $H^0(s) = s$. We define $K^{(i)} = H^{N-i}(s)$. Therefore, $K^{(N)} = H^0(s) = s$. So, the key-chain generated by applying H on the seed $K^{(N)}$ is defined as follows:

$$Chain(K^{(N)}N, =, H)\{K^{(i)} \mid K^{(i)} = H^{N-i}(K^{(N)}), 0 \leq i \leq N\}$$

Properties (a)–(c) of OWHF guarantee that anybody who knows the key $K^{(j)}$ can compute all the previous keys $K^{(i)}, 0 \leq i \leq j$, but cannot compute any of the later keys $K^{(i)}, j+1 \leq i \leq N$.

S reveals the chain elements in reverse order with respect to creation. Assume that S reveals a given key $K^{(i)}$ of the chain to R through some predefined authenticated channel. Assume now that, later, S reveals $K^{(j)}$, $i < j \leq N$, to R over an insecure channel without additional communication overhead. R can verify the authenticity of $K^{(j)}$, i.e., it comes from S, by applying $j - i$ times H to $K^{(j)}$ and checking that the result is equal to $K^{(i)}$. That is, $K^{(i)} = H^{j-i}(K^{(j)})$. It must be noticed that, if S initially transfers the *chain-head* $K^{(0)}$ to R, then R can authenticate all keys in the key-chain.

We define the *current-key*,$K^{(cur)}$, the last-revealed key that belongs to the chain. That is, $K^{(cur)}$ corresponds to $K^{(i)}$ if S has already revealed $K^{(i)}$ but $K^{(i+1)}$ is still secret. Moreover, we define the *next-key*, and denote it by $K^{(nxt)}$, as the next key to be revealed. Of course, if $K^{(cur)} = K^{(i)}$ then $K^{(nxt)} = K^{(i+1)}$.

IV. Overview of S^2RP

Every sensor node has a *private-key*, a symmetric key that it secretly shares with *KMS*. *KMS* uses this key to securely unicast rekeying material to the sensor node. We denote by K_{s_x} the sensor-specific key of node s_x.

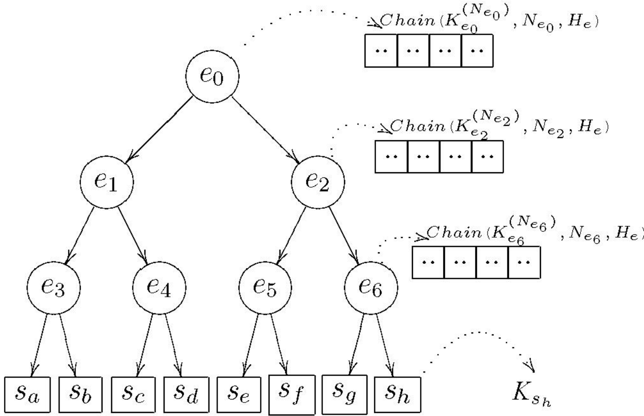

Fig. 2. Eviction-tree with m=2 and h=3

In order to preserve the forward secrecy, S^2RP maintains a hierarchical structure of symmetric keys, called *eviction-tree*. The internal nodes are indexed in the Breadth-First Search fashion, as shown in Figure 2, with the root as node e_0. Tree nodes are associated with encryption keys as follows. *KMS* associates the private-key of every group member with a leaf of the eviction-tree, while each internal node e_j is associated with the key-chain $Chain(K_{e_j}^{(N_{e_j})}, N_{e_j}, H_e)$.

Let $K_{e_j}^{(cur)}$ be the current-key of $Chain(K_{e_j}^{(N_{e_j})}, N_{e_j}, H_e)$. We define $Keys^{(cur)}$ the set of the current-keys $K_{e_j}^{(cur)}$ of every internal nodes that *KMS* stores. Every sensor node s_x stores a subset of $Keys^{(cur)}$, $KeySet(s_x)$, defined as follows. Let $Path(s_x)$ be the set of internal nodes lying on the path from the root to the leaf associated with s_x. So, the key-set $KeySet(s_x)$ is composed of the current-keys associated with the internal nodes in $Path(s_x)$:

$$KeySet(s_x) = \{K_{e_j}^{(cur)} \mid e_j \in Path(s_x)\}$$

At any time, the key-set in each group member is composed of h keys, where h is the path length. In the case of balanced tree h is $h = \lceil \log_m(n) \rceil$, where n is the network size and m is the tree ariety. In particular, the current-key associated with the tree root, $K_{e_0}^{(cur)}$, belongs to the key-set of every group member.

This key hierarchy allows us to define the following form of *clustering*. Let e_j be an internal node and let $L(e_j)$ be the set of leaves belonging to the subtree rooted at e_j. We define a cluster associated to e_j, and denote it by Clu_j, as the set of sensor nodes whose keys are associated to leaves in $L(e_j)$. For example, with reference to Figure 2, the set of nodes $\{e_a, e_b, e_c, e_d\}$ forms the cluster Clu_1. The clusters are at most $\sum_{k=1}^{h-1} m^k = \frac{m^h - m}{m-1}$, while every group member could take part to at most $h-1$ clusters. It must be noticed that the current key $K_{e_j}^{(cur)}$ associated with node e_j is shared only by members of cluster Clu_j.

The key server *KMS* uses the eviction-tree to guarantee forward security as follows. When a sensor node s_l leaves the group communication, all keys in $KeySet(s_l)$ are *compromised* and must be renewed. Consequently, *KMS* updates $Keys^{(cur)}$ as follows. For each internal node e_j in $Path(s_l)$, *KMS* replaces the current-key $K_{e_j}^{(cur)}$ with the next-key $K_{e_j}^{(nxt)}$ in the key-chain $Chain(K_{e_j}^{(N_{e_j})}, N_{e_j}, H_e)$. Then, *KMS* broadcasts $K_{e_j}^{(nxt)}$ to nodes having e_j in their associated path, except the leaving node s_l. In Section VIII we show that *KMS* accomplishes this task by means of $O(\log n)$ messages. It follows that after its leaving, s_l holds neither the current group-key nor the current cluster-keys.

The eviction-tree is not suitable to preserve the backward security. Actually, when a new sensor node joins the group, it is associated with a leaf of the eviction-tree and receives the current-keys of the internal nodes belonging to its path. However, as soon as the new member receives these keys, it becomes able to calculate all the previous keys in the corresponding key-chains by repeatedly applying H_e.

In order to preserve the backward secrecy, *KMS* defines an additional key, K_J, that we call *join-key*. This symmetric key is shared by all group members and it is securely renewed when a new member joins the group.

The group members calculate the group-key by mixing the join-key K_J and the current-key associated with the tree root $K_{e_0}^{(cur)}$. Backward security is now guaranteed because the new member does not know the previous join-keys, and thus is not able to recover the corresponding previous group-keys. Similarly the cluster-keys can be obtained by mixing the join-key and the current-key of internal nodes to preserve both the forward and backward secrecy. For instance, the cluster-key of cluster Clu_j is

obtained by mixing K_J and $K_{e_j}^{(cur)}$.

When a new member joins the group, the join-key stored by all members must be securely renewed. In a simple approach, *KMS* could generate a fresh join-key and unicasts it to all members encrypting with their private-keys. It follows that this solution requires $\mathcal{O}(n)$ messages. An alternative, more efficient approach lets each member locally calculate the new K_J so reducing communication costs and saving energy. More in detail, *KMS* broadcasts the command of renewing the join-key to the members already in the group. The group members verify the authenticity of command and then apply a OWHF on the current join-key to obtain the new one. Thus, *KMS* needs only to securely unicast the new join-key to the joining member.

In the latter approach every group member has to be sure of the authenticity of the renewing command, i.e., it actually comes from *KMS*. Once again, the proof of authentication is based on the key-chain technique. To this purpose, *KMS* builds a key-chain $Chain(K_C^{(N_C)}, N_C, H_c)$[1] where $K_C^{(l)}$ corresponds to the l-th command of renewing K_J.

V. The Initialization Phase

In the initialization phase, *KMS* initializes sensor nodes that form the initial membership. Such initialization takes place via off-line methods. *KMS* assigns an identifier and a private-key to each group member.

The private-key for sensor node s_a is generated as follows: $K_{s_a} = f(MK, s_a)$, where f is a secure pseudo-random function and *MK* is a master key known only by *KMS*. In this scheme *KMS* needs only to keep *MK* in storage. When *KMS* needs to communicate with s_a, it computes K_{s_a}.

As mentioned in Section IV, *KMS* precomputes the key-chain $Chain(K_{e_j}^{(N_{e_j})}, N_{e_j}, H_e)$ for each internal node e_j of the eviction-tree. The length of a key-chain could depend on the internal node with which it is associated. Let $level(e_j)$ be the level of the internal node e_j in the eviction-tree. That is, $level(e_j) = \lceil \log_m (j(m-1)+m) \rceil$. If $level(e_j) < level(e_i)$, the subtree rooted at e_j is composed of more leaves than the e_i's subtree. *KMS* reveals an element in the key-chain associated with e_j whenever a leaf that belongs to its subtree is removed. For this reason, the chains associated with lower-level internal nodes are consumed more quickly. Therefore, usually, the chain associated with an internal node is longer than the ones of its children. That is, if e_j and e_i are two internal nodes and e_i is a child (descendant) of e_j, then $level(e_j) < level(e_i)$ and $N_{e_j} \geq N_{e_i}$.

During the initialization phase, each group member securely receives the chain-heads of the nodes belonging to its path. Hence, $Keys^{(cur)}$ initially contains the chain-heads associated with the internal nodes.

As an example, let us consider a given node s_a. With reference to Figure 2, *KMS* associates K_{s_a}, the private-key of sensor node s_a, with the leftmost leaf of the eviction-tree. It follows that $Path(s_a) = \{e_0, e_1, e_3\}$ and thus the key-set in s_a is composed of the chain-heads associated with the internal nodes in $Path(s_a)$, i.e. $KeySet(s_a) = \{K_{e_0}^{(0)}, K_{e_1}^{(0)}, K_{e_3}^{(0)}\}$.

Furthermore, *KMS* defines the join-key, K_J, that is randomly chosen and generates a chain of command-keys $Chain(K_C^{(N_C)}, N_C, H_c)$. Besides, *KMS* securely distributes K_J and $K_C^{(0)}$ to every group members.

VI. The Group Communication Phase

During the group communication phase the group membership could change and *KMS* must guarantee the forward and backward security by means of rekeying. Moreover, *KMS* may periodically refresh both the group-key and cluster-keys as needed.

A. Secure group communications

At any time, a group member s_x stores the join-key and its $KeySet(s_x)$. The group members use these keys to locally generate the group-key K_{grp} and the cluster-keys K_{clu_i} as follows.

The group-key: Every group member stores the join-key, K_J, and the current-key associated with the tree root e_0, $K_{e_0}^{(cur)}$. To preserve only the forward security, the group members use $K_{e_0}^{(cur)}$ as group-key, i.e., $K_{grp} = K_{e_0}^{(cur)}$. In fact this key is renewed and efficiently transmitted to group members whenever a node leaves, or is forced to leave, the group communication.

To preserve both the forward and backward security, the group members locally generate the group-key by mixing K_J and $K_{e_0}^{(cur)}$. That is, $K_{grp} = M(K_J, K_{e_0}^{(cur)})$ where M is a mixing function. Backward security is guaranteed because new members do not know the previous values of join-keys, and thus are not able to recalculate the corresponding previous group-keys.

The cluster-keys: The current-keys $K_{e_i}^{(cur)}$ that are associated with the internal nodes (except e_0), are shared only by members of cluster Clu_i.

Hence, the members use these keys to construct the cluster-keys similarly to the group-key. To preserve only

[1] H_c and H_e could be the same OWHF

the forward security, the cluster members use the $K_{e_i}^{(cur)}$ as the cluster-key K_{clu_i}. To achieve also the backward security, a cluster member mixes K_J and $K_{e_i}^{(cur)}$ to locally generate the cluster-key K_{clu_i} as $K_{clu_i} = M(K_J, K_{e_i}^{(cur)})$, where M is the mixing function.

B. Managing group membership: a member leaves the group

When a member s_l leaves, or is forced to leave, the group, *KMS* must prevent s_l from accessing future communication. The keys that s_l holds become compromised and *KMS* has to renew the keys belonging to *KeySet*(s_l) to preserve the forward security.

First, *KMS* identifies the compromised internal nodes that belong to *Path*(s_l). Then, *KMS* updates the eviction-tree by removing the leaf associated with K_{s_l}. For each compromised internal node e_j and for each child of its, *KMS* generates a rekeying message. If the e_j's child is a leaf, the message contains $K_{e_j}^{(nxt)}$ encrypted with the private-key of the correspondent group member. If the e_j's child is an internal node e_c and $K_{e_c}^{(cur)}$ is not compromised, the rekeying message contains $K_{e_j}^{(nxt)}$ encrypted with $K_{e_c}^{(cur)}$. If $K_{e_c}^{(cur)}$ is compromised, $K_{e_j}^{(nxt)}$ is encrypted with $K_{e_c}^{(nxt)}$. After this procedure, *Keys*$^{(cur)}$ is updated as follows: for each compromised node e_j, $K_{e_j}^{(cur)}$ is replaced by $K_{e_j}^{(nxt)}$. As the private-key held by s_l is not used to encrypt any new key, and all its known current-keys are changed, s_l is no longer able to access the group messages.

With reference to Figure 2 , let us assume that the sensor node s_d leaves the group. It follows that keys $K_{e_0}^{(cur)}$, $K_{e_1}^{(cur)}$, and $K_{e_4}^{(cur)}$ are considered compromised and *KMS* has to renew each of them. The new keys are respectively $K_{e_0}^{(nxt)}$, $K_{e_1}^{(nxt)}$, and $K_{e_4}^{(nxt)}$ so that $K_{e_i}^{(cur)} = H_e(K_{e_i}^{(nxt)})$. Then, *KMS* generates and broadcasts the following rekeying messages:

M_1. *KMS* $\not\to s_c : E_{K_{s_c}}\left(K_{e_4}^{(nxt)}\right)$
M_2. *KMS* $\not\to s_c : E_{K_{e_4}^{(nxt)}}\left(K_{e_1}^{(nxt)}\right)$
M_3. *KMS* $\not\to \{s_a, s_b\} : E_{K_{e_3}^{(cur)}}\left(K_{e_1}^{(nxt)}\right)$
M_4. *KMS* $\not\to \{s_a, s_b, s_c\} : E_{K_{e_1}^{(nxt)}}\left(K_{e_0}^{(nxt)}\right)$
M_5. *KMS* $\not\to \{s_e, s_f, s_g, s_h\} : E_{K_{e_2}^{(cur)}}\left(K_{e_0}^{(nxt)}\right)$

where $\not\to$ denotes the broadcast communication operator.

In the case of balanced m-ary tree, *KMS* needs to broadcast $m \lceil \log_m(n) \rceil - 1$ messages (see Section VIII-B).

Upon receiving the re-keying messages, the group members decrypt them with appropriate keys in order to get the new keys. Then, the group members can immediately verify the authenticity of the new keys by applying the OWHF H_e. With reference to Figure 2 , let us consider s_c, for example. Upon receiving messages M_1, M_2, and M_4, s_c performs the following actions. Initially, s_c decrypts M_1 with its private-key K_{s_c} and checks that $K_{e_4}^{(cur)} = H_e(K_{e_4}^{(nxt)})$. Then, s_c decrypts M_2 with $K_{e_4}^{(nxt)}$ and checks that $K_{e_1}^{(cur)} = H_e(K_{e_1}^{(nxt)})$. Finally, s_c decrypts M_4 with $K_{e_1}^{(nxt)}$ and checks that $K_{e_0}^{(cur)} = H_e(K_{e_0}^{(nxt)})$.

C. Managing group membership: a new member joins the group

In order to preserve the backward security, all group members share the join-key K_J. This symmetric key is securely renewed whenever a new node joins the group in such a way it cannot obtain the previous ones.

When a new node s_n joins the group, *KMS* broadcasts the command of renewing K_J. Let us suppose that, when s_n joins the group, all members that are already in the group store a copy of $K_C^{(l)}$ and that the message conveying the command carries $K_C^{(l+1)}$. It follows that $K_C^{(cur)} = K_C^{(l)}$ and $K_C^{(nxt)} = K_C^{(l+1)}$. Upon receiving the new key $K_C^{(nxt)}$, the group members can immediately verify the authenticity of the command. If the check is successful, every member locally calculates the new join-key by simply applying H_j to the current join-key, i.e., $K_J \leftarrow H_j(K_J)$. Moreover, the members already in the group refresh their current command-key with the newly received value $K_C^{(nxt)}$, i.e., $K_C^{(cur)} \leftarrow K_C^{(nxt)}$. Contextually, KS needs only to unicast $K_C^{(nxt)}$ to s_n and the new value of the join-key encrypted with K_{s_n}.

Furthermore, *KMS* needs to update the eviction-tree creating a new leaf associated with K_{s_n}. Then, *KMS* securely unicasts *KeySet*(s_x) to s_n. With reference to Figure 2 , assume that *KMS* inserts the leaf associated with s_n as a child of e_3. Then, *KMS* unicasts s_n the messages containing respectively $K_{e_0}^{(cur)}$, $K_{e_1}^{(cur)}$, and $K_{e_3}^{(cur)}$. These keys are encrypted with the private-key of the joining node, K_{s_n}.

D. Periodic key refreshing

Periodically *KMS* refreshes both group-key and cluster-keys. So doing, *KMS* makes cryptanalysis less attractive because the keys will be invalidated regularly and thus the material encrypted with those keys that an adversary can collect is limited.

In our protocol, the group-key K_{grp} and cluster-keys K_{clu_i} are function of the join-key K_J. So, when a node renews K_J, automatically updates K_{grp} and K_{clu_i}. Therefore, in order to periodically refresh K_{grp} and K_{clu_i}, *KMS* needs only to periodically broadcast $K_C^{(nxt)}$, i.e., the

Type	*Destination*	*Indexes*	*Keys*
T_{E_b}	BRDcst	$j \mid\mid i \mid\mid c \mid\mid l$	$E_{K_{e_c}^{(l)}}\left(K_{e_j}^{(i)}\right)$
T_{E_u}	UNIcst: s_x	$j \mid\mid i$	$E_{K_{s_x}}\left(K_{e_j}^{(i)} \mid\mid h(K_{e_j}^{(i)})\right)$
T_{C_b}	BRDcst	l	$K_C^{(l)}$
T_{C_u}	UNIcst: s_x	l	$E_{K_{s_x}}\left(K_C^{(l)} \mid\mid h(K_C^{(l)})\right)$
T_{J_u}	UNIcst: s_x	-	$E_{K_{s_x}}(K_J \mid\mid h(K_J))$

TABLE I

CLASSIFICATION OF MESSAGES.

command to refresh K_J. Upon receiving a message conveying $K_C^{(nxt)}$, every group member verifies its authenticity by ascertaining that $K_C^{(cur)} = H_c(K_C^{(nxt)})$, sets $K_C^{(nxt)}$ as the new current command-key, and, finally, computes the new values for K_J,K_{grp} and K_{clu_i}.

E. Key Reconfiguration

The command key-chain and the key-chain associated with any internal node e_j have a limited length. When all keys belonging to a key-chain have been revealed, the key-chain has run out, and *KMS* has to reconfigure it as follows. *KMS* builds a new key-chain as specified in Section III. Then, *KMS* takes different actions according to whether the key-chain to be reconfigured is the command key-chain or is a key-chain in the eviction-tree. In the former case, *KMS* unicasts the chain-head $K_C^{(0)}$ to each sensor node that is member of the group. In the latter case, assuming it is the key-chain associated with node s_j that has to be reconfigured, *KMS* unicasts $K_{e_j}^{(0)}$ to all sensor nodes rooted at e_j.

VII. CLASSIFICATION OF REKEYING MESSAGES

In order to guarantee the forward and backward security, *KMS* needs five types of rekeying messages. Table I shows them.

KMS broadcasts messages typed T_{E_b} when a sensor node, e.g. s_l, leaves, or is forced to leave, the group communication. Let e_j be a node belonging to $Path(s_l)$ and $K_{e_j}^{(i)}$ be the next-key associated with e_j. *KMS* has to broadcast $K_{e_j}^{(i)}$ by means of m messages typed T_{E_b}, one for each child of e_j. Let e_c be one of these children and $K_{e_c}^{(l)}$ be the key associated with it. The message typed T_{E_b} contains $K_{e_j}^{(i)}$ encrypted with the key $K_{e_c}^{(l)}$ in order to guarantee the confidentiality. Since the group members can verify the authenticity by applying the OWHF H_e, *KMS* sends only the encrypted key without appending a digital signature for guaranteeing authenticity. The message contains the indexes j and c corresponding to the internal nodes e_j and e_c respectively. Hence, the length of index j, or c, limits the number of internal nodes so that the eviction-tree is composed of at most $2^{\|j\|}$ internal nodes, where $\|.\|$ is the bit-length. The indexes i and l correspond to the position of the keys in the key-chains associated with the internal nodes e_j and e_c respectively. Consequently, the indexes limit the length of the corresponding key-chains so that $N_{e_j} \leq 2^{\|i\|}$ and $N_{e_c} \leq 2^{\|l\|}$.

A message of type T_{E_u} contains the i-th key of $Chain(K_{e_j}^{(N_{e_j})}, N_{e_j}, H_e)$. *KMS* unicasts this message to s_x when s_x joins the group communication or looses messages of type T_{E_b}. Moreover, when all N_{e_j} keys belonging to e_j chain have been revealed, *KMS* needs to unicast this type of message ($i = 0$). Before unicasting this message to s_x, *KMS* must verify that e_j is included in $Path(s_x)$. The authenticity of messages typed T_{E_u} is guaranteed by encrypting the message with the private key and using an MDC h. When *KMS* unicasts a message containing a key K to s_x, he computes the hash value of K, appends it to the key, and encrypts the pair $(K \mid\mid h(K))$ by using the private-key K_{s_x}. The receiver s_x decrypts the message with its private-key and separates the recovered key K' from the recovered hash H'. Then s_x computes the hash function h on K' and compares it with H'. If these quantities agree, K' is accepted as being authentic. The encryption protects the appended hash so that it is infeasible for an attacker without K_{s_x} to alter the message without disrupting the correspondence between K and its hash value.

KMS broadcasts the group members the message of type T_{C_b} to command renewing the join-key both periodically and whenever a new member joins the group. This message contains the key $K_C^{(l)}$ belonging to $Chain(K_C^{(N_C)}, N_C, H_c)$. The index l specifies the position of the key in the key-chain and it limits the length of key-chain. That is, $N_C \leq 2^{\|l\|}$ where $\|l\|$ is the bit-length of index l. *KMS* neither encrypts the command-key nor appends a digital signature as a proof of its authenticity.

KMS unicasts s_x the message typed T_{C_u} when s_x joins the group communication or looses messages of type T_{C_b}. Moreover, when all command-keys have been revealed, *KMS* needs to unicast this type of message to all group members ($l = 0$). The authenticity of this message is guaranteed by encrypting the message with the private key and using an MDC h as well as in the case of type T_{E_u}.

Finally, *KMS* unicasts the new joining member s_x the message typed $\mathtt{T}_{\mathrm{J_u}}$. This message contains the current-value of join-key K_J. The authenticity of this message is guaranteed by using the private key and the MDC h as well as in the case of type $\mathtt{T}_{\mathrm{E_u}}$.

VIII. Performance evaluation

In this section we present experimental results and analyze storage, communication, and computing overhead of S^2RP. We have implemented an early prototype of the S^2RP protocol for a WSN composed of sensor nodes of the Tmote Sky class [14]. These sensor nodes are powered with two AA batteries and equipped with a 16-bit 8MHz MSP430 microcontroller, 48 Kbytes of ROM, 10 Kbytes of RAM, and IEEE 802.15.4 radio interface. We run TinyOS as operating system [15].

Our implementation, referred to as *MoteCrypt*, uses SkipJack [16] or RC5 [17] as symmetric cipher, and SHA-1 [18] as hash function. We borrowed the TinySec implementation of these algorithms [19]. However, as for the moment TinySec does not support Tmote Sky, we had to port the machine-dependent parts of such implementations onto our platform. Although the porting was carried out in the NesC language for fast prototyping, the initial resulting implementation was satisfactory.

The structure of messages in MoteCrypt is based on the packet format of TinyOS (Table III) The fields Dst_addr and Grp correspond to the destination address and the group identifier respectively. Field $Type$ specifies the appropriate handler function to extract and interpret the message on the receiver. Field $Other$ is reserved for TinyOs, while Pld_len is the length of payload, Cnt and $Data$. The field $Data$ contains the cryptographic key that *KMS* unicasts to a specific-sensor node s_x or broadcasts to group members. Cnt contains the index specifying the position of the key in the chain and in some cases other information used to extract the key. For example, in message of type $\mathtt{T}_{\mathrm{E_b}}$, Cnt contains the position of key in the key-chain, the index of the internal node associated with the key, and the indexes that locate the key used for encryption. Finally, the rekeying message contains Cyclic Redundancy Code (CRC) that is used by receivers to detect transmission errors.

A. Computing overhead

Table II reports the total amount of time (in milliseconds) employed by each sensor node to process a rekeying message, verify the authenticity of a received key, and to refresh the corresponding group-key or cluster-key. The table highlights the three main operations contributing to the computation: decrypting the message, verifying the authenticity of the received key, and installing the group-key and/or cluster-keys. In particular, to verify the authenticity of a key each sensor node applies the hash-function. In the case of SHA-1 this requires 3.91 ms. This is an improvement with respect to LKH, that uses the digital signature for key authentication. Furthermore, depending on the cipher, installing a new key requires a preprocessing that makes decryption/encryption faster. This preprocessing requires 6.45 ms with RC5 and 1.22 ms with SkipJack.

Type:	$\mathtt{T}_{\mathrm{E_b}}$	$\mathtt{T}_{\mathrm{E_u}}$	$\mathtt{T}_{\mathrm{C_b}}$	$\mathtt{T}_{\mathrm{C_u}}$	$\mathtt{T}_{\mathrm{J_u}}$
RC5	3.85	5.12	-	5.12	5.12
SHA-1	3.91	3.91	2×3.91	3.91	3.91
Preproc.	6.45	6.45	$h \times 6.45$	-	$h \times 6.45$
Total(h=3)	14.81	15.76	26.97	9.60	28.15

Type:	$\mathtt{T}_{\mathrm{E_b}}$	$\mathtt{T}_{\mathrm{E_u}}$	$\mathtt{T}_{\mathrm{C_b}}$	$\mathtt{T}_{\mathrm{C_u}}$	$\mathtt{T}_{\mathrm{J_u}}$
SkipJack	1.46	1.93	-	1.93	1.93
SHA-1	3.91	3.91	2×3.91	3.91	3.91
Preproc.	1.22	1.22	$h \times 1.22$	-	$h \times 1.22$
Total(h=3)	7.24	7.47	11.55	5.88	9.53

TABLE II
COMPUTATIONAL COST TO PROCESS A REKEYING MESSAGE (IN MILLISECONDS).

As shown in the table, upon receiving a message of type $\mathtt{T}_{\mathrm{E_u}}$ and $\mathtt{T}_{\mathrm{E_b}}$, each group member decrypts the message and calculates the hash value to verify the authenticity. Then it calculates the corresponding communication key. That is, if the received key corresponds to the tree root ($j = 0$), the group member calculates and preprocesses the new group-key K_{grp}, otherwise it calculates and preprocesses the corresponding cluster-key K_{clu_j}. When a member receives a message of type $\mathtt{T}_{\mathrm{C_b}}$, it verifies the command authenticity and renews the join-key by means of the hash function. Then, it changes both the group-key and the cluster-keys. When a member receives a message of type $\mathtt{T}_{\mathrm{C_u}}$, it has only to decrypt the messages and to verify the key authenticity without renewing the group-key and cluster-keys. When a member receives a message of type $\mathtt{T}_{\mathrm{J_u}}$, it decrypts the message and verifies its authenticity. Then, it updates its stored join-key and preprocesses both the renewed group-key and the cluster-keys.

B. Communication overhead

In this section, we analyze communication overhead of S^2RP from two points of view: the number of messages for key management and the size (in bytes) of rekeying messages.

S^2RP increases scalability by reducing the number of messages to revoke the group-key and cluster-keys when a sensor node leaves, or is forced to leave, the group communication. As discussed in the Introduction, so far key revocation has required a number of messages that is $\mathcal{O}(n)$ and thus it is not scalable in a large scale WSN. In contrast, in S^2RP the number of rekeying messages depends on the path length h and the tree ariety m. Actually, for each node on the path (h nodes), we need a rekeying message of each one of the node's child (m children). The exception is made for the internal node at the $h-1$ level. In this case, *KMS* has to transmit $m-1$ unicast messages, one for each leaf, except the one associated the leaving node. As $h = \lceil \log_m(n) \rceil$, it follows that the number of messages is $m\lceil \log_m(n) \rceil - 1$.

In order to preserve the backward security, our proposed approach requires only to broadcast the message that conveys the command-key. Furthermore, *KMS* unicasts $h+2$ keys to the new member that are respectively the renewed join-key, the command-key and the key-set of the receiver (h keys).

Rekeying messages are broadcast. It follows that S^2RP would greatly benefit from an underlying efficient and scalable broadcast protocol. Notwithstanding, S^2RP itself gives a fundamental contribution to scalability as it reduces the number of rekeying messages from $\mathcal{O}(n)$ to $\mathcal{O}(\log(n))$.

Dst addr	Grp	Type	Pld len	Other	Cnt	Data	CRC
2	1	1	1	3	2	0-27	2

Type:	T_{E_b}	T_{E_u}	T_{C_b}	T_{C_u}	T_{J_u}
Header	10	10	10	10	10
Cnt	2-4	2	2	2	2
Data	10	20	10	20	20
Tot.(bytes)	22-24	32	22	32	32

TABLE III

FORMAT OF REKEYING MESSAGES AND COMMUNICATION OVERHEAD PER PACKET(IN BYTES).

Table III shows the size of rekeying packet types. It must be noticed that the field $Data$ in messages of type T_{E_b} or T_{C_b} contains only the key without a digital signature or a MDC to guarantee the authenticity. Hence, the communication overhead per packet is reduced with respect to LKH. Furthermore, in order to achieve semantic security we must use a fresh Initialization Vector (`IV`) for each encryption that is sent in the same packet with the encrypted data. In order to reduce the communication overhead introduced by `IV`, we reuse some of fields in the packet header [19]. In particular, `IV` is given by the concatenation of Dst_addr, Grp, $Type$, Pld_len, Cnt, and $padd$, where $padd$ is one byte of the $Other$ field.

C. Storage overhead

Although memory space is a very scarce resource for the current generation of sensor nodes, storage is not an issue in our scheme. In fact, a sensor node s_x needs to store the private-key, h keys belonging to its key-set *KeySet*(s_x), the join-key, and the current command-key. Furthermore, it needs to store one group-key and $h-1$ cluster-keys. In order to save storage, a sensor node could avoid storing the group-key and cluster-keys and build them as needed by mixing the join-key with a key of its key-set. However, so doing, the node would incur every time in the cost of installing a new key (see Section VIII-A). In the case of binary eviction-tree and a WSN composed of 1024 sensor nodes, the path length h is 10 and every sensor node has to store 23 keys. Assuming a key size of 10 bytes, keys totally require 230 bytes of storage.

As to storage overhead at *KMS*, it stores one master key *MK* to generate n private-keys. In fact, *KMS* computes the private-key K_{s_a} as follows: $K_{s_a} = f(MK, s_a)$, where f is a secure pseudo-random function and s_a is the node identifier.

Furthermore, *KMS* needs to store a key-chain for each internal node. To avoid storing the entire key-chain, we can exploit the optimization algorithm by Coppersmith and Jakobsson [20] to trade storage and computation cost. The algorithm requires $\log_2(N)$ memory cells to store a chain composed of N keys. Thus, if we assume for simplicity that every node internal node is associated with a fixed-length chain of N keys, *KMS* has to store $\left(\frac{m^h-1}{m-1} \times \log_2 N\right)$ keys. With reference to the previous example, *KMS* has to store only 8 keys for each chain of 256 keys and thus 8184 keys for the whole entire eviction-tree. For a key of 10 bytes, *KMS* requires about 80Kbytes to store all the keys associate with the eviction tree.

Finally, the memory requirement for the code storage is 40932 bytes and 39788 bytes by using SkipJack or

RC5 as encryption algorithm respectively. The code size includes the operating system TinyOS, the cryptographic primitive SkipJack or RC5, and the hash function SHA-1. Moreover, the memory requirement for static data is 2176 and 2932 bytes by using SkipJack or RC5 as encryption algorithm respectively.

IX. Conclusions

With reference to group communication in WSNs, we have presented S^2RP a scalable and secure rekeying protocol that allows every member to locally authenticate a key distributed over insecure broadcast channels. The proposed solution has the following merits.

- The protocol guarantees the forward and backward security when the group membership changes.
- The protocol has a reduced communication overhead in terms of the number of messages necessary to distribute new keys. In particular, when a sensor node leaves, or is forced to leave, the group, the number of messages necessary to distribute a key is a logarithmic function of network size.
- The protocol has reduced computational overhead as it uses only one-way hash functions and symmetric ciphers to guarantee confidentiality and authenticity of the rekeying messages.

The reduced communication and computational overhead meets the limited capabilities of the sensor nodes, improves the protocol scalability and gives rise to energy savings that increase the network lifetime.

Acknowledgements

This work has been supported by the Commission of the European Communities under Sixth Framework Programme Project IST-004536 "Reconfigurable Ubiquitous Networked Embedded Systems"(RUNES). We are grateful to the students I. S. La Porta and M. Dell'Unto who participated in coding the prototype.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 293–422, March 2002.

[2] B. Sinopoli, C. Sharp, L. S. S. Schaffert, and S. S. Sastry, "Distributed Control Applications Within Sensor Networks," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1235–1246, August 2003.

[3] D. M. Wallner, E. G. Harder, and R. C. Agee, "Key management for multicast: issues and architecture," IETF, RFC 2627, 1999.

[4] H. Chan, V. Gligor, A. Perrig, and G. Muralidharan, ""on the distribution and revocation of cryptographic keys in sensor networks"," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 3, pp. 233–247, July-September 2005.

[5] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *10th ACM Conference on Computer and Communication Security (CCS'03)*, Washington D.C., USA, October 27–30 2003, pp. 41–57.

[6] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *IEEE Symposium on Security and Privacy (SP'03)*, 11–14 May 2003, pp. 197–213.

[7] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security suite for sensor networks," in *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking (MOBICOM-01)*. New York: ACM Press, July 16–21 2001, pp. 189–199.

[8] S. Rafaeli and D. Hutchison, "A Survey of Key Management for Secure Group Communication," *ACM Computing Surveys*, vol. 35, no. 3, pp. 309–329, September 2003.

[9] C. K. Wong, M. G. Gouda, and S. S. Lam, "Secure Group Communications using Key Graphs," *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 16–30, February 2000.

[10] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The VersaKey Framework: Versatile Group Key Management," *IEEE Journal on Selected Areas of Communications (Special Issue on Middleware)*, vol. 17, no. 9, pp. 1614–1631, August 1999.

[11] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, November 1981.

[12] G. Dini and I. M. Savino, "An efficient key revocation protocol for wireless sensor networks," in *Proceedings of IEEE WOWMOM'06*, Niagara-Falls, Buffalo-NY, 26–29 June 2006.

[13] Alfred J. Menezes and Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, October 1996.

[14] Moteiv, "Tmote sky," `http://www.moteiv.com/`.

[15] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System Architecture Directions for Networked Sensors," in *Proceedings of the Ninth Symposium on Architectural Support to Programming Languages and Operating Systems (ASPLOS)*, Cambridge, Massachusetts, United States, November 2000, pp. 93–104.

[16] National Institute of Standards and Technology (NIST), "SKIPJACK and KEA algorithm specifications," 1998.

[17] R. L. Rivest, "The RC5 encryption algorithm," in *Proceedings of the Second International Workshop on fast Software Encryption*, B. Preenel, Ed., vol. LNCS 1008. Leuven, Belgium: Springer-Verlag, December 14–16 1994, pp. 86–96.

[18] National Institute of Standards and Technology, *FIPS PUB 180-1: Secure Hash Standard*. Gaithersburg, MD, USA: National Institute for Standards and Technology, Apr. 1995. [Online]. Available: http://www.itl.nist.gov/fipspubs/fip180-1.htm

[19] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: A link layer security architecture for wireless sensor networks," in *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, United States, November 3–5 2004, pp. 162–175.

[20] D. Coppersmith and M. Jakobsson, "Almost optimal hash sequence traversal," in *Financial Cryptography*, 2002, pp. 102–119.