LARK: a Lightweight Authenticated ReKeying scheme for Clustered Wireless Sensor Networks

GIANLUCA DINI
University of Pisa, Italy
IDA M. Savino
Elsag Datamat S.p.A., Italy

Group communication has proven to be a powerful paradigm for designing applications and services in Wireless Sensor Networks (WSNs). Given the tight interaction between WSNs and the physical world, a security infringement may translate into a safety infringement. Therefore, in order to fully exploit the group communication paradigm we need to secure it. Traditionally, this requirement has been formalized in terms of *backward* and *forward security* and fulfilled by means of *rekeying*. In WSNs, group rekeying becomes particularly a complex problem because communication takes place over an easily-accessible wireless medium and because sensor nodes have severe limitations in terms of computing, storage, energy and tamper-resistance capabilities for cost reasons.

In this paper we present a *Lightweight Authenticated ReKeying* (LARK) scheme for clustered WSNs. LARK guarantees backward and forward security, is scalable in terms of communication overhead and efficient in terms of computing overhead for key authentiticy verification. LARK achieves security, efficiency, and scalability by exploiting two basic well-known mechanisms, namely *key graph* and *key chain*, and integrating them in an original way. LARK supports a general group model where groups can be hierachical and partially overlapping. In contrast to other WSN group rekeying schemes, LARK considers grouping a tool for designing and implementing applications and services rather than for network management. Consequently, LARK receives a group topology reflecting the application needs and manages rekeying at single-group level. In the paper we describe LARK, formally argue that it meets the backward and forward security requirements, and, finally, evaluate its performance in terms of communication, computing and storage overhead in limited-resources sensor nodes.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; K.6.m [**Management of Computing and Information Systems**]: Miscellaneous—*Security*

General Terms: Algorithms, Design, Security

Additional Key Words and Phrases: Authentication, key management, lightweight security, sensor networks

1. INTRODUCTION

The *group communication model* has proven to be a suitable and effective paradigm to design and implement applications and services in WSNs. Group communication naturally supports and promotes *in-network processing*, a fundamental technique for elaborating the wealth of data provided by WSNs in an efficient and scalable way [Intanagonwiwat et al. 2000]. Furthermore, many WSN applications can be abstracted as pursuit-evasion game (PEG) applications where the

Authors' addresses: G. Dini, Dipartimento di Ingegneria della informazione: Elettronica, Informatica, Telecomunicazioni, University of Pisa, Largo Lazzarino 1, 56126 Pisa, Italy, email: gianluca.dini@ing.unipi.it; I. M. Savino, Divisione Difesa, Spazio e Ambiente, Elsag Datamat S.p.A., Via Laurentina 760, 00143 Rome, Italy, email: ida.savino@elsagdatamat.com.
This work has been partially supported by EU FP6 Project RUNES (Grant Agreement no. IST-004536-RUNES), EU FP7 Project CHAT (Grant Agreement no. FP7-224428), and EU FP7 Network of Excellence CONET (Grant Agreement no. FP7-2007-2-224053), and by Research Project 2007 funded by Cassa di Risparmio di Livorno, Lucca e Pisa.

WSN increases the sensing capabilities of the pursuers [Sinopoli et al. 2003]. In PEG applications, the group communication model is a natural paradigm [Park and Shin 2004].

Given the tight interaction of WSN with the physical environment, security infringements may cause safety infringements with possible consequences in terms of damages, injures and, even, death. Hence, if we wish to exploit the advantages the group communication model brings about it is crucial to protect communication within groups. The need for secure group communication is generally formalized in requiring that when a sensor node joins a group, it must not be able to access group communication prior its joining (*backward security*) and that when a sensor node leaves, or is forced to leave, the group, the sensor node must be prevented from accessing any further group communication (*forward security*) [Zhou et al. 2005]. These requirements can be fulfilled by *rekeying*. Intuitively, a *group key* is distributed to all group members which use it to encrypt and decrypt broadcast messages. When a new member joins or a current member leaves the group, the current group key is revoked and a new one distributed [Zhou et al. 2005]. In such scenario, key revocation has the same level of importance as key distribution. In fact, compromised sensor nodes have to be logically removed from the network communication and, usually, the ability to logically remove them translates into the ability to revoke keys [Chan et al. 2005; Wallner et al. 1999]. Actually, if cryptographic algorithms do not expose the secret keys, then the secret keys can only be compromised by compromising the device itself. It follows that by revoking all keys of a compromised device, it is possible to remove the logical presence of that device from the network.

Group rekeying has been largely investigated in traditional networks [Rafaeli and Hutchison 2003; Wong et al. 2000]. However, WSNs pose 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. These constraints exclude any traditional solution based upon public-key encryption, for example. Furthermore, economical reasons exclude also adequate support for tamper-resistance. Therefore, the fact that a WSN can be deployed over a large, unattended, possibly hostile area exposes each individual sensor node to the risk of being compromised.

Several solutions to group rekeying in WSNs have been proposed so far, including [Choudhary et al. 2007; Eltoweissy et al. 2004; Eltoweissy et al. 2005; Park and Shin 2004; Perrig et al. 2001; Son et al. 2007; Wang and Ramamurthy 2007; Younis et al. 2006; Zhu et al. 2006]. As a common denominator, all these systems use grouping as a network topology control technique [Younis et al. 2006]. A WSN typically features a single base station collecting data from many homogeneous sensors and co-ordinating activities. WSN applications often require only an aggregate value to be reported to the base station. In this case, sensor nodes in different regions of the field can collaborate to aggregate their data and provide more accurate reports about their local regions. In order to support data aggregation through efficient network organisation, nodes are partitioned into a number of small groups called *clusters*. Each cluster has a coordinator, referred to as a cluster head, that is responsible for coordinating the nodes within the cluster, and communicating with the base station and other cluster heads.

An emerging class of decentralised WSN architectures is characterised by multiple base stations, different applications on the same hardware, and heterogeneous nodes. These architectures find their realisation in WSNs with actuation capabilities where nodes are not only capable of sensing the environment but also acting on it [Akyildiz and Kasimoglu 2004; Sinopoli et al. 2003]. In these architectures, applications are composed of many collaborating tasks, each affecting only a portion of the system where the notion of neighbourhood from physical may become logical [Mottola and Picco 2006a; 2006b]. These applications may range from localisation facilities to control systems in tunnels or buildings, interactive museums, and home automation [Årzén et al. 2007; Petriu et al. 2000]. Design and implementation of these applications may benefit from the group communication paradigm. However, in these applications, group membership may be based on logical basis rather than physical proximity, and change frequently, perhaps even continuously, as tied to sensor reading. Groups may be organized hierarchically and rooted at different collecting nodes (roots). Furthermore, although groups are logically distinct on service or task basis, they may overlap as one or more sensor nodes may concur to implement two or more services or tasks. As it turns out, clustering is not adequate anymore to capture this richer and more dynamic group model. In other words, group topology has to be defined according to applicative requirements rather than by a network management technique.

In this paper we cope with the problem of group rekeying in WSN by considering grouping as a means to model WSN applications [Akyildiz and Kasimoglu 2004; Park and Shin 2004; Sinopoli et al. 2003] and consequently, we assume that the group topology is determined and fixed by the application needs. According to this approach, we consider a WSN where sensor nodes that cooperate for the same task/service are *logically* placed in the same group. Several groups may coexist at the same time, they can partially overlap and be hierarchically organized. Sensor nodes may join and leave a group dynamically and, possibly, frequently.

With reference to this quite general model, we propose LARK, a group rekeying scheme that guarantees secure communication at the single group level in highly dynamic WSNs. This is achieved by assigning every group a secret key, having it shared by all group members, and by properly revoking and redistributing that group key whenever a member joins or leaves in order to fulfill the backward and forward security. This reactive approach based on rekeying has the advantage that a new node can immediately join a 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, cannot be used. Second, reactive rekeying may incur 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.

We take up these challenges by proposing a *centralized* key distribution and revocation scheme that levers on two basic mechanisms: key-graphs and key-chains. *Key-graph* is a mechanism to specify groups that has been originally proposed for conventional wired networks and that allows us to achieve an efficient group rekeying protocol [Wong et al. 2000]. *Key-chain* is an authentication mechanism based on Lamport's one-time passwords [Lamport 1981] which has already been profitably employed in WSNs [Park and Shin 2004; Perrig et al. 2001]. We formally prove that compounding them compounds their strengths by guaranteeing both forward and backward security. Furthermore, we prove their integration is conducive to improve scalability and efficiency of group rekeying in WSNs. Such integration have been proposed in a preliminary work [Dini and Savino 2006]. Intuitively, the proposed scheme retains the scalability properties of the original proposal [Wong et al. 2000] and makes key authenticity verification more efficient from two viewpoints: first, verification is based on symmetric ciphers and hash functions that are orders of magnitude more computationally efficient than public-key cryptography; second, key-chain provides "auto-verifiable" key materials, thus no additional proof of authenticity is necessary so making the size of rekeying messages smaller. Furthermore, the resulting scheme is scalable because it exhibits a communication overhead that is between $O(n)$, in the case of a "flat" group [Eschenauer and Gligor 2003; Park and Shin 2004; Perrig et al. 2001], and $O(\log n)$, in the case of a balanced key hierarchy [Rafaeli and Hutchison 2003; Waldvogel et al. 1999; Wong et al. 2000], with n the number of sensor nodes. The communication overhead of the proposed scheme depends on the current group configuration.

The paper is organized as follows. Section 2 gives the problem statement and Section 3 describes the system architecture. Section 4 introduces key-chains as a basic mechanism for key authenticity in rekeying. Section 5 explains how LARK performs rekeying after a joining or leaving event in order to guarantee the forward and backward security. Section 6 presents both a formal security analysis of LARK and a performance evaluation in terms of computing, communication and storage overhead. Performance analysis has been carried out by means of a prototype on TinyOS [Hill et al. 2000] on TMote Sky sensor nodes [Moteiv ; Polastre et al. 2005]. Section 7 discusses related work. Finally, in Section 8 we make our final considerations.

2. PROBLEM DEFINITION

A WSN typically features a single base station co-ordinating activities and collecting data from many homogeneous sensors. Habitat monitoring is a common example application [Mainwaring et al. 2002]. Several WSN applications require only an aggregate value to be reported to the base station. In this case, sensor nodes in different regions of the field can collaborate to aggregate their data and provide more accurate reports about their local regions. In this network setting, *clustering* is an effective network topology control technique to support data aggregation and increase scalability and lifetime [Younis et al. 2006]. However, clustering is not sufficient to capture the requirements of the new decentralised architectures that are rapidly emerging and find their realisation in WSNs with actuation capabilities, referred to as Wireless Sensor and Actor networks (WSANs) [Akyildiz and Kasimoglu 2004]. In these architectures, the grouping strategy is lead by application specific requirements rather than the network management level.

In contrast to mainstream WSNs, characterised by a single application gathering and reporting data, these new decentralised architectures are composed of multiple base stations, different applications running on the same hardware, and heterogeneous nodes that not only observe and gather data from the environment, but are also capable of affecting it by performing a variety of actions. Actor and sensor nodes may be even integrated in the same node (e.g., robots). Applications of these architectures range from localisation facilities to control systems in tunnels or buildings, interactive museums, and home automation [Årzén et al. 2007; Petriu et al. 2000]. They are composed of many collaborating tasks running on the same hardware, each affecting only a portion of the system. For instance, an application for monitoring and control in tunnels or building can be decomposed in at least three main tasks, i.e., structural monitoring, environment monitoring, and response to extreme events such as fire [Dermibas 2005]. To realise the latter functionality, for instance, the nodes controlling water sprinklers must monitor nearby temperature sensors and smoke detectors and take appropriate measures when and where needed. Mobile robots equipped with sensing and actuating capabilities may interact with the surrounding sensor nodes to help first rescue teams [Årzén et al. 2007]. The application logic now resides in the network: including a central base control loop degrades system performance and reliability without any sensible advantage [Akyildiz and Kasimoglu 2004].

In this class of applications, sensor nodes can be profitably grouped on the basis of application specific requirements. Sensor nodes may be grouped according to the task they cooperate for. As different tasks may use the same sensor nodes, the corresponding groups may overlap. For instance, the environment monitoring task and the response to extreme events task may use a common set of temperature sensors. Groups may be further sub-grouped. For instance, sensor nodes cooperating for a given task may be sub-grouped according to their sensing function, so including in the same sub-group those sensing the same physical quantity (e.g., light, humidity, smoke), or geographically, putting those physically "close" in the same sub-group. In addition to this physical notion of neighbourhood, applications may greatly benefit from a logical one [Mottola and Picco 2006b]. With logical neighbourhood the set of nodes in the communication range of a given device are determined by applicative information. For instance, a water sprinkle may be interested in the sensor nodes that have reported a temperature greater than $100°$ C and lying two hops from the sprinkle. The logical neighbourhoods of two water sprinkles may overlap for reliable coverage purposes. Of course, the resulting hierarchical, multi-rooted group model with overlapping groups may raise performance and lifetime issues due to the traffic and energy necessary for its management. Mottola and Picco have recognised these issues and proposed a novel routing strategy specifically tailored for the logical neighbourhood programming model [Mottola and Picco 2006a]. LARK gives a similar contribution in terms of group key management.

LARK assumes a group topology defined by the application level and protect group communication from an external adversary by letting sensor nodes in the same group share a *group-key* to encrypt messages within the group. Hence, anyone that is not part of the group can neither access nor inject/modify messages. When a sensor node leaves a group, it must be prevented from accessing the group communication (*forward security*). When a sensor node joins a group, it must not be able to decipher previous messages encrypted with an old key even though it has recorded

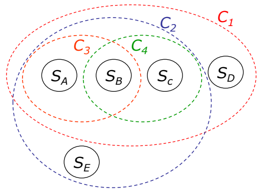


Fig. 1. Example of grouping.

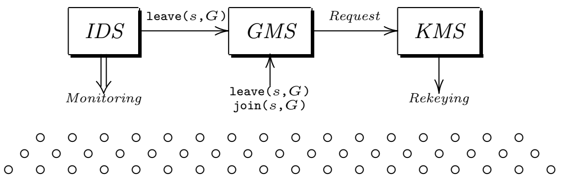


Fig. 2. Sensor Network Controller *WSNC*.

them (*backward security*). In this model, forward and backward security are fulfilled via rekeying, that is the current group-key is revoked and a new one is distributed whenever a sensor node joins or leaves the group. In performing rekeying, LARK must be scalable in terms of communication overhead and efficient in terms of key authentication computing and storage overhead.

3. SYSTEM ARCHITECTURE

In our system a sensor node becomes member of a group by explicitly joining it. As a member of the group, the node may broadcast messages to the other members. Later on, a node may voluntarily leave the group or be forced to leave if compromised. After leaving a group, a node cannot be member of the group and cannot send messages to, or receive messages from, that group.

Groups may overlap, that is a sensor node may belong to one or more groups. A group can be further composed of other groups. More in detail, let G_i and G_k be two overlapping groups so that $G_i \cap G_k \neq \emptyset$. If $G_k \subseteq G_i$, then G_i *contains* G_k, or equivalently G_k is a *sub-group* of G_i. It follows that each sensor node s that is member of G_k is also member of G_i. On the other hand, there could be a member of group G_i that is not included in the group G_k. It follows that when a sensor node s leaves a group G, then it leaves all the sub-groups of G containing s (*cascade leave*). When a sensor node s joins a group G, then s joins all the groups containing G (*cascade join*). With reference to Figure 1, the group G_1 contains the sensor node s_d and groups G_3 and G_4. Hence, G_1 includes the sensor nodes belonging to its sub-groups, i.e., s_a, s_b and s_c. With reference to this example, if sensor node s_c leaves G_1, it also leaves G_4. If a new sensor node s_e joins G_3, it also joins the group G_1.

The system is managed by a *WSN Controller* (*WSNC*) 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 groups by keeping track of sensor nodes that join and leave the groups. A sensor node wishing to cooperate in a specific group invokes the join operation. We assume that, upon joining the system, a node is genuine, i.e., it is not compromised and its integrity is guaranteed. 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 [Roman et al. 2006; Wang et al. 2008; Zhang et al. 2008]. Upon detecting a compromised sensor node, *IDS* forces the sensor node to leave every group the sensor node belongs to by invoking the leave operation and specifying the identifiers of the sensor node and the group as arguments. After that the node cannot re-join anymore.

Whenever a sensor node joins or leaves a group, the group-key has to be renewed in order

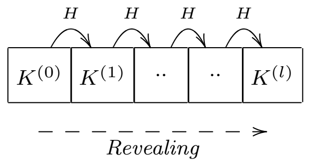


Fig. 3. In a Key-Chain, keys are created and revealed in the same order.

to guarantee the backward and forward security requirements. *KMS* is the component that is responsible for performing 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. In addition to this event-based rekeying policy, *KMS* can also give support to a periodic rekeying aimed at reducing the amount of encrypted material available to an adversary.

In a centralised approach, *WSNC* can be implemented by a more powerful computing node than sensor nodes. *WSNC* 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 *WSNC* will be not compromised. Although workstation and server security are still research issues, the literature provides a number of established techniques and methodologies to keep workstations and servers secure. Good starting readings are [Anderson 2008; Cole 2009], for example. In the rest of the paper we detail *KMS*.

4. KEY AUTHENTICITY

The proposed protocol achieves efficient key authenticity by employing *chains of keys*. This mechanism allows us to efficiently verify the authenticity of a key by simply computing a lightweight cryptographic primitive such as an hash function. A chain of keys is an ordered set of a predefined number $l > 0$ of symmetric keys. We denote by $K^{(i)}$ the i-th key of the chain with $0 \leq i < l$. The key server *KMS* constructs and stores a chain of keys, and then distributes the keys in increasing order. That is, *KMS* distributes $K^{(i)}$ if and only if all the keys $K^{(j)}$ with $j < i$ are already distributed. So, firstly *KMS* distributes the key $K^{(0)}$, defined as *chain head*. Furthermore, if the i-th key $K^{(i)}$ is the last distributed key, the next key that has to be distributed is $K^{(i+1)}$. In the rest of the paper we refer to the last distributed key as the *current-key* of the chain and the next key that has to be distributed as the *next-key*. More in detail, we denote by $head(X)$, $curr(X)$ and $next(X)$ the head, the current-key and the next-key of key chain X, respectively.

In order to guarantee key authenticity, we consider two types of chain of keys, referred to as the *Key-Chains* and the *Inverted Key-Chains*. The keys belonging to these types of classes are linked to each other by means of a one-way hash function [Menezes et al. 1996]. A one-way hash function H is a function that has the following properties: given an input x, it is easy to compute the *image* y so that $y = H(x)$, whereas given y it is computationally unfeasible to find the *preimage* x so that $y = H(x)$. Furthermore, given an input x and its image $y = H(x)$ it is computationally unfeasible to find a second preimage z such that $H(z) = y$. Example of one-way hash functions are SHA-1 [National Institute of Standards and Technology 1995] and MD5 [Rivest 1992].

In Section 4.1 and 4.2, we introduce Key-Chains and Inverted Key-Chains, respectively. In Section 4.3, we provide a preliminary intuition of the usage of Key-Chains and Inverted Key-Chains to manage a group key. Such an intuition will be refined in the next sections.

4.1 Key-Chain

A *Key-Chain Ch* is a chain of l symmetric keys such that each element in the chain is the *preimage* of the next one under a one-way hash function (Figure 3).

In order to compute the Key-Chain *Ch*, *KMS* randomly chooses the chain-head $K^{(0)} = r$, where r is a random secret. Then, *KMS* iteratively applies the hash function H in order to calculate the other elements of the chain. More in detail, the i-th key $K^{(i)}$ of the chain is the hash image of the

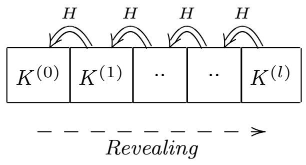


Fig. 4. In an Inverted Key-Chain, keys are created and revealed in the opposite order.

previous one $K^{(i-1)}$. That is, $K^{(i)} = H(K^{(i-1)}) = H^i(r), 0 < i < l$.

$$Ch = chain(r, H, l) = \{K^{(i)} \mid K^{(0)} = r \ \wedge K^{(i)} = H^i(r), i \in (0, l)\}$$

The Key-Chain guarantees the key authenticity. In fact, if a sensor node s knows the i-th key $K^{(i)}$ of the chain Ch, then s is able to calculate the next keys $K^{(j)}, j > i$ by iteratively applying $j - i$ times the hash function H to $K^{(i)}$. If KMS transfers the key $K^{(i)}$ to node s in a way guaranteeing authenticity and secrecy, then all the keys $K^{(j)}, j > i$, can be considered authentic. So, if the KMS initially transfers the chain-head, then s can locally calculate all the keys of the chain Ch. The secrecy of the transfer is necessary to prevent an adversary from intercepting the chain-head and thus compromising the whole key-chain.

4.2 Inverted Key-Chain

An *Inverted Key-Chain ICh* is a chain of l symmetric keys such that each element is the *image* of the next one under a one-way hash function (Figure 4).

In order to compute the Inverted Key-Chain, KMS randomly chooses a random secret r so that $K^{(l-1)} = r$. Then, KMS iteratively applies the hash function H so that $K^{(i)} = H(K^{(i+1)}) = H^{l-i}(r), 0 \leq i < l - 1$. In other word, the key $K^{(i+1)}$ is the hash preimage of $K^{(i)}$.

$$ICh = Ichain(r, H, l) = \{K^{(i)} \mid K^{(l-1)} = r \wedge K^{(i)} = H^{l-i}(r), i \in [0, l-1)\}$$

A node s can verify the authenticity of the received keys by applying the hash function H, as in Lamport's one-time password [Lamport 1981]. Let us assume that KMS transferred the i-th key $K^{(i)}$ to s in a way guaranteeing authenticity and secrecy. Then, KMS can now transfer the next key $K^{(i+1)}$ to s in a way guaranteeing only secrecy, i.e., without using authentication mechanisms such as HMACs or digital signatures [Rivest et al. 1978]. Secrecy of transfer prevents an adversary from eavesdropping the key. The properties of the one-way hash function guarantee that nobody who knows $K^{(i)}$ is able to compute the next ones because $K^{(i+1)}$ is the hash preimage of $K^{(i)}$. However, the sensor node s can verify the authenticity of $K^{(i+1)}$ by simply applying the hash function H to $K^{(i+1)}$ and checking that the result is equal to $K^{(i)}$. That is, $K^{(i)} = H(K^{(i+1)})$. Finally, if KMS initially transfers the chain head $K^{(0)}$ to node s, then s is able to verify the authenticity of any key $K^{(i)}$ belonging to ICh by checking if $K^{(0)} = H^i(K^{(i)})$.

4.3 Guaranteeing backward and forward security

In this section we provide an intuitive description of the use of Key-Chains and Inverted Key-Chains as basic mechanisms to provide backward and forward security. For the sake of simplicity we abstract away fundamental aspects, namely secure and efficient transport of keys, that will be faced with in the next sections in great detail. In contrast we will focus on the different kind of keys and their roles.

Let us consider a group G of sensor nodes and let us assume that a *Joining Key-Chain* Ch_J and a *Leaving Inverted Key-Chain* ICh_L have been defined for the group. Furthermore, let us suppose that each group member holds the current keys of the two key chains, namely the *joining-key* $curr(Ch_J)$ and the *leaving-key* $curr(ICh_L)$. In order to guarantee backward and forward security, group members encrypt messages by using the group-key $K_G = \mu(curr(Ch_J), curr(ICh_L))$, where μ is a mixing function.

When a sensor node leaves, all the remaining members in G receive the next leaving-key $next(ICh_L)$ from KMS in a way guaranteeing secrecy, i.e., in a way that prevents any other subject, including the leaving sensor node, from eavesdropping the key. The property of the one-way hash function guarantees that the leaving node who knows the current leaving-key $curr(ICh_L)$ is not able to compute the next one because $next(ICh_L)$ is the hash preimage of $curr(ICh_L)$. When the remaining members receive the next leaving-key $next(ICh_L)$, they verify its authenticity, compute the new group key, and then start encrypting the messages using that key. It follows that the leaving node is not able to access to the future communication.

When a new sensor node joins the communication, all the sensor nodes calculate the next joining-key $next(Ch_J)$ of Ch_J locally by applying H. The joining node receives the value of the next joining-key $next(Ch_J)$ from KMS in a way guaranteeing secrecy and authenticity (e.g., encrypted by means of an *a priori* shared key). At this point, every member can compute the new group key. As the current joining-key $curr(Ch_J)$ is the hash preimage of the next one $next(Ch_J)$, the property of the one-way hash function guarantees that the joining node receiving $next(Ch_J)$ is not able to compute the current joining-key $curr(Ch_J)$ and thus it cannot have access to the previous communication.

It is important to notice that the group members need to be informed that a sensor node is joining so that they can calculate the next-key $next(Ch_J)$. In other words, they need an authenticated command from KMS that "triggers" this computation. In order to fulfil this task we use an additional Inverted Key-Chain, the *Trigger Inverted Key-Chain* ICh_T. Let us assume that each group member holds the current *trigger-key* $curr(ICh_T)$ of ICh_T. When a sensor node joins the group, KMS computes the next trigger-key $next(ICh_T)$ and sends it to the group members. Upon receiving this key every group member verify its authenticity and, if the verification succeeds, computes $next(Ch_J)$ as described above. It is worthwhile to notice that the trigger-key only conveys an authenticated triggering signal and thus it is not necessary to encrypt it. Furthermore, the property of the one-way hash function guarantees that the knowledge of the current trigger-key does not make it possible to derive the next ones. The trigger-key is transferred to the joining node together with the joining-key.

5. LIGHTWEIGHT AUTHENTICATED REKEYING SCHEME

The mechanisms based on the Key-Chains and Inverted Key-Chains guarantees an efficient verification of the key authenticity. In this section, we describe the Lightweight Authenticated ReKeying scheme (LARK) aimed at efficiently distributing the new group key whenever a sensor node leaves or joins a group.

In order to perform $LARK$, every sensor node secretly shares a symmetric sensor-key with KMS. KMS uses this key to securely unicast rekeying material to the sensor node as necessary. Furthermore, each sensor node stores all the group-keys of the groups to which it belongs. We denote by K_s the specific sensor-key of node s, and by K_G the key of group G.

5.1 Overview of the Graph Theory

In this section, we give an overview of the graph theory concepts that are relevant to this work. A *directed graph* or *digraph* is a pair $\mathcal{G} = (V, E)$ of sets such that $E \subseteq V^2$. The elements of V are the *vertices* of the graph, and the element of E are its *edges*. The edge $\langle v_i, v_k\rangle$ starts from v_i and ends on v_k so that $\langle v_i, v_k\rangle$ is said to be directed from v_i to v_k. With reference to $\langle v_i, v_k\rangle$, the vertex v_i is said to be the *direct predecessor* of v_k, and v_k the *direct successor* of v_i. Let us define $Pred_D(v_i)$ and $Succ_D(v_i)$ the set of direct predecessors and successors of vertex v_i respectively.

$$\begin{aligned} Pred_D(v_i) &= \{v_k \mid \exists \langle v_k, v_i\rangle \in E\} \\ Succ_D(v_i) &= \{v_k \mid \exists \langle v_i, v_k\rangle \in E\} \end{aligned} \tag{1}$$

A *directed path* from a vertex v_0 to v_k is a not-empty graph $\mathcal{P}(v_0, v_k) = (V', E')$, where $V' = \{v_0, v_1, \ldots, v_k\}$ and $E' = \{\langle v_0, v_1\rangle, \langle v_1, v_3\rangle, \ldots, \langle v_{k-1}, v_k\rangle\}$. In other words, the path $\mathcal{P}(v_0, v_k)$ is a finite ordered set of vertices V', in which no element is repeated, that begins with v_0 and ends with v_k, and a set of edges E', in which each edge connects a vertex in V' to the following one.

Given a digraph $\mathcal{G}$, we say that exists a directed path from the vertex v_i to v_k if $\mathcal{P}(v_i, v_k)$ is contained in $\mathcal{G}$. That is, $\mathcal{P}(v_i, v_k) \subseteq \mathcal{G}$. Furthermore, if a directed path leads from v_i to v_k, then v_i is defined a *predecessor* of v_k, and v_k is defined a *successor* of v_i. Let us define $Pred(v_i)$ and

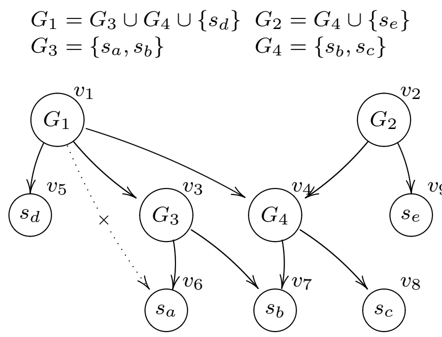

$G_1 = G_3 \cup G_4 \cup \{s_d\}$ $G_2 = G_4 \cup \{s_e\}$
$G_3 = \{s_a, s_b\}$ $G_4 = \{s_b, s_c\}$
v_1
G_1
v_2
G_2
v_5
s_d
×
v_3
G_3
v_4
G_4
v_9
s_e
v_6
s_a
v_7
s_b
v_8
s_c

Fig. 5. Example of a Group-Graph.

$Succ(v_i)$ the set of predecessors and successors of vertex v_i, respectively.

$$\begin{aligned} Pred(v_i) = &\ \{v_k \mid \exists \mathcal{P}(v_k, v_i) \ \subseteq \mathcal{G} \wedge v_k \neq v_i\} \\ Succ(v_i) = &\ \{v_k \mid \exists \mathcal{P}(v_i, v_k) \ \subseteq \mathcal{G} \wedge v_k \neq v_i\} \end{aligned} \tag{2}$$

Furthermore, we define the *extended set of predecessors* $Pred_E(v_i)$ the set of vertices that includes the v_i predecessors and the same vertex v_i. On the other hand, the extended set of successors $Succ_E(v_i)$ includes the v_i successors and the same vertex v_i.

$$\begin{aligned} Pred_E(v_i) = &\ \{v_i\} \cup Pred(v_i) \\ Succ_E(v_i) = &\ \{v_i\} \cup Succ(v_i) \end{aligned} \tag{3}$$

Finally, an *acyclic* digraph is a directed graph with no directed cycles. That is, for any vertex v_i there is no directed path that starts and ends on v_i.

5.2 Group Management

Let us suppose the sensor nodes are organised in groups. *KMS* manages membership of groups by means of a graph, called *Group-Graph.* The Group-Graph is an acyclic digraph $\mathcal{G} = (V, E)$ so that each vertex of V is associated with either a sensor node or a group. In the following we refer $\nu(s)$ the vertex associated with the sensor node s, and $\nu(G)$ the vertex associated with the group G. The set of edges E in the Group-Graph reflects the group topology existing in the system and it is defined as follows. A vertex associated with a sensor node has no successors, whereas the successors of a vertex associated with a group, i.e., $\nu(G)$, are the vertices associated with the sensor nodes belonging to G and the sub-groups of G.

$$\begin{array}{ll} Succ(\nu(s)) = \emptyset & \forall \nu(s) \in V \\ Succ(\nu(G)) = \{\nu(s) \mid \forall s \in G\} \ \cup \{\nu(G') \mid \forall G' \neq G, G' \subseteq G)\} & \forall \nu(G) \in V \end{array}$$

Finally, given a vertex v and its successor w, the set E contains the edge $\langle v, w\rangle$ only if there is no vertex z so that z is successor of v and w is successor of z.

$$\neg \exists z \in Succ(v) \mid w \in Succ(z) \Rightarrow \langle v, w\rangle \in E, \ \forall v \in V \ \wedge \forall w \in Succ(v) \tag{4}$$

It follows that given a group G and a sensor node s, there is an edge connecting the vertices $\nu(G)$ and $\nu(s)$ only if s belongs to G and there is no sub-group of G containing s. Furthermore, given the group G and its sub-group G', there is an edge connecting the vertices $\nu(G)$ and $\nu(G')$ only if there is no sub-group of G containing G'.

Let us consider the example of grouping shown in Figure 1. The vertices are associated with sensor nodes and groups so that we have the labelling shown in Figure 5. Furthermore, the set of edges E has to guarantee the following hierarchy:

$$\begin{aligned} Succ(v_1) = &\ \{v_3, v_4, v_5, v_6, v_7, v_8\} \\ Succ(v_2) = &\ \{v_4, v_7, v_8, v_9\} \\ Succ(v_3) = &\ \{v_6, v_7\} \\ Succ(v_4) = &\ \{v_7, v_8\} \\ Succ(v_j) = &\ \emptyset, \ j \in [5, 9] \end{aligned}$$

Let us consider the vertices v_1 and its successors v_5 and v_6. The set E contains the edge $\langle v_1, v_5\rangle$ because there is no vertex that is successor of v_1 and predecessor of v_5. On the other hand, there is the vertex v_3 that is successor of v_1 and predecessor of v_6 so that E does not contain the edge $\langle v_1, v_6\rangle$. By applying the Formula 4 to each vertex and its successors, the resulting set of edges E, shown in Figure 5, is the following:

$$\begin{aligned} E = \ & \{\langle v_1, v_3\rangle, \langle v_1, v_4\rangle, \langle v_1, v_5\rangle, \langle v_3, v_6\rangle, \langle v_3, v_7\rangle, \\ & \langle v_4, v_7\rangle, \langle v_4, v_8\rangle, \langle v_2, v_4\rangle, \langle v_2, v_9\rangle\} \end{aligned}$$

5.3 Forward security

In order to guarantee the forward security, *KMS* maintains a *Leaving Key-Graph*, that is a Group-Graph where each vertex v contains a key, referred to as the *leaving-key* $Key_L(v)$ and defined as follows. Each vertex $\nu(s)$ associated with the sensor node s contains the sensor-key K_s. For every vertex $\nu(G)$, *KMS* defines an Inverted Key-Chain $ICh_{L,\nu(G)}$, and the vertex $\nu(G)$ contains the current-key of that chain.

$$Key_L(v) = \begin{cases} K_s & \texttt{if } v = \nu(s) \\ curr(ICh_{L,v}) & \texttt{if } v = \nu(G) \end{cases}$$

Each sensor node s stores all the leaving-keys associated with $\nu(s)$ predecessors. Hence, the sensor node s stores the following key set $KeyRing_L(s)$:

$$KeyRing_L(s) = \{Key_L(v) \mid \forall v \in Pred(\nu(s))\}$$

It is worthwhile to notice that all the members of group G store the leaving-key contained in $\nu(G)$. Hence, this key acts as the group-key K_G and has to be renewed whenever a member leaves the group.

$$K_G = Key_L(\nu(G)) = curr(ICh_{L,\nu(G)}) \tag{5}$$

When the sensor node $\tilde{s}$ leaves a group G, *KMS* must prevent $\tilde{s}$ from accessing the future communications. So, the corresponding group-key K_G is considered compromised and has to be renewed. Furthermore, when $\tilde{s}$ leaves the group G, it also leaves all the sub-groups of G to which $\tilde{s}$ belongs (leave cascade). Hence, all the keys of these sub-groups are considered compromised and have to be renewed. In the following we present the protocol performed both at the *KMS* and at sensor nodes side that guarantees the forward security.
When the sensor node $\tilde{s}$ leaves the group G, *KMS* performs the following actions:

(1) *KMS* identifies the set of vertices associated with compromised keys, referred to as $V_{cmp}(\tilde{s}, G)$. $V_{cmp}(\tilde{s}, G)$ contains the vertices associated with sub-groups of G containing $\tilde{s}$.

$$\begin{aligned} V_{cmp}(\tilde{s}, G) = \ & \{\nu(G') \mid G' \subseteq G \wedge \tilde{s} \in G'\} \\ = \ & Pred(\nu(\tilde{s})) \cap \ Succ_E(\nu(G)) \end{aligned}$$

(2) *KMS* removes the edges connecting $\nu(\tilde{s})$ to its direct predecessors in $V_{cmp}(\tilde{s}, G)$. Then, if the set of $\nu(G)$ direct predecessors is not empty, *KMS* adds the edges connecting the vertex $\nu(\tilde{s})$ to each of $\nu(G)$ direct predecessors.

$$E = E - \{\langle v, \nu(\tilde{s})\rangle \mid \forall v \in V_{cmp}(\tilde{s}, G)\}$$
$$E = E \cup \{\langle v, \nu(\tilde{s})\rangle \mid \forall v \in Pred_D(\nu(G))\}$$

If the vertex $\nu(\tilde{s})$ has no predecessors, then it is removed from the graph.

$$Pred(\nu(\tilde{s})) = \emptyset \Rightarrow V = V - \{\nu(\tilde{s})\}$$

(3) For each vertex v in $V_{cmp}(\tilde{s}, G)$, *KMS* updates the corresponding leaving-key $Key_L(v)$ with the next-key of the chain $ICh_{L,v}$.

$$\forall v \in V_{cmp}(\tilde{s}, G), Key_L(v) = next(ICh_{L,v})$$

(4) For each vertex v in $V_{cmp}(\tilde{s}, G)$ and for each direct successor w of v, *KMS* builds the rekeying message $M_L(v, w)$ as follows.

$$M_L(v, w) : v, w, E_{Key_L(w)}(Key_L(v)) \quad \forall v \in V_{cmp}(\tilde{s}, G), \forall w \in Pred_D(v),$$

where $E_K(m)$ the encryption of message m by using the key K. Notice that by construction the vertex v is associated with a group, whereas the vertex w may be associated with either a group or a sensor node.

(5) *KMS* broadcasts the rekeying messages in a bottom-up order. More in detail, *KMS* broadcasts a message $M_L(v, w)$ only if it has already broadcast the messages $M_L(v', w'), \forall v' \in Succ(v) \cap V_{cmp}(\tilde{s}, G) \wedge \forall w' \in Pred_D(v')$. It is worthwhile to notice that after broadcasting the message $M_L(v, w)$ the key $Key_L(v)$ is the last distributed key of the chain $ICh_{L,v}$ or, in other words, the new current-key $curr(ICh_{L,v})$.

With reference to Figure 5, let us suppose that the sensor node s_a leaves the group G_1, and consequently the group G_3. Hence, the set $V_{cmp}(s_a, G_1)$ contains v_1 and v_3 (Step 1). Then, *KMS* removes the edge $\langle v_3, v_6 \rangle$ connecting the vertices associated with G_3 and s_a. Since the vertex v_6 has no predecessor, v_6 is removed from the graph (Step 2). Then, for each vertex in $V_{cmp}(s_a, G_1)$, *KMS* chooses the next-key of the corresponding Inverted Key-Chain (Step 3). Hence, the leaving-key of vertex v is defined as follows:

$$Key_L(v) = \begin{cases} K_s & \texttt{if } v = \nu(s) \\ curr(ICh_{L,v}) & \texttt{if } v = \nu(G') \wedge v \notin V_{cmp}(\tilde{s}, G) \\ next(ICh_{L,v}) & \texttt{if } v = \nu(G') \wedge v \in V_{cmp}(\tilde{s}, G) \end{cases}$$

Finally, *KMS* broadcasts the following messages in the following order (Step 4 and 5):

$$\begin{array}{l} M_L(v_1, v_5):\ v_1, v_5, E_{K_{s_d}}(next(ICh_{L,v_1})) \\ M_L(v_3, v_7):\ v_3, v_7, E_{K_{s_b}}(next(ICh_{L,v_3})) \\ M_L(v_1, v_3):\ v_1, v_3, E_{next(ICh_{L,v_3})}(next(ICh_{L,v_1})) \\ M_L(v_1, v_4):\ v_1, v_4, E_{curr(ICh_{L,v_4})}(next(ICh_{L,v_1})) \end{array}$$

Let us consider the sensor node s that receives the message $M_L(v, w)$ broadcast by *KMS* at Step 5. Firstly, s verifies whether the rekeying message refers to a group which s belongs to. Then, s verifies whether it is able to decrypt the renewed key contained in the message. That is, s verifies whether the set of its predecessors $Pred(\nu(s))$ contains the vertex v, and whether the vertex w is associated with either itself (i.e. $w = \nu(s)$), or with a group which s belongs to (i.e., $w \in Pred(\nu(s))$). In case both the conditions are verified, the sensor node s decrypts the received message, otherwise it discards the message.

In case one of the above conditions is not verified, the sensor node s discards the received message. Otherwise, s decrypts the key contained in the message by using $K_{dec}(w)$ defined as follows:

$$K_{dec}(w) = \begin{cases} K_s & \texttt{if } w = \nu(s) \\ curr(ICh_{L,w}) & \texttt{if } w \in Pred(\nu(s)) \\ \perp & \texttt{otherwise} \end{cases}$$

It is worthwhile to notice that if the vertex w belongs to $V_{cmp}(\tilde{s}, G)$, the sensor node s has already received the renewed key of vertex w (Step 5). Let us define K the value of the decrypted key, then the sensor node s verifies the key authenticity as described in Section 4.2. In fact, the leaving-key $Key_L(v)$ contained in $KeyRing_L(s)$ corresponds to the current-key of chain $ICh_{L,v}$. So the sensor node s has to check whether the hash value of K matches the value of $Key_L(v)$. That is, $H(K) = Key_L(v)$. If this is the case, s updates the key ring with the new value so that $Key_L(v) = K$.

Let us suppose that the vertex v is associated with the group G'. Since the key of group G' corresponds to the leaving-key of vertex $\nu(G')$ (Equation 5), then the value of the key $K_{G'}$ is automatically updated.

5.4 Backward security

In order to guarantee backward security, LARK maintains a Joining Key-Graph, that is a Group-Graph such that each vertex v contains the *trigger-key* $Key_T(v)$ and the *joining-key* $Key_J(v)$ defined as follows. For each vertex associated with the sensor node s, the trigger-key and the joining-key correspond to the sensor-key K_s. For each vertex v associated with a group, *KMS* defines two chains of keys: an Inverted Key-Chain $ICh_{T,v}$ and a Key-Chain $Ch_{J,v}$. Therefore, the keys $Key_T(v)$ and $Key_J(v)$ are the current-keys of the chains $ICh_{T,v}$ and $Ch_{J,v}$, respectively.

$$Key_T(v) = \begin{cases} K_s & \texttt{if } v = \nu(s) \\ curr(ICh_{T,v}) & \texttt{if } v = \nu(G) \end{cases}$$

$$Key_J(v) = \begin{cases} K_s & \texttt{if } v = \nu(s) \\ curr(Ch_{J,v}) & \texttt{if } v = \nu(G) \end{cases}$$

Each sensor node s stores all the trigger-keys and the joining-keys associated with $\nu(s)$ predecessors. Hence, the sensor node s stores the following key sets $KeyRing_T(s)$ and $KeyRing_J(s)$:

$$\begin{array}{l} KeyRing_T(s) = \{Key_T(v) \mid \forall v \in Pred(\nu(s))\} \\ KeyRing_J(s) = \{Key_J(v) \mid \forall v \in Pred(\nu(s))\} \end{array}$$

In particular, all the members of group G store the joining-key $Key_J(\nu(G))$ associated with $\nu(G)$. Hence, this key acts as the group-key K_G and has to be renewed whenever a member joins the group.

$$K_G = Key_J(\nu(G)) = curr(Ch_{J,\nu(G)}) \tag{6}$$

In order to guarantee the backward security, when a sensor node $\tilde{s}$ joins the group G, KMS must prevent $\tilde{s}$ from accessing the previous communication. So, the group-key K_G is considered compromised and has to be renewed. Furthermore, when $\tilde{s}$ joins the group G, it also joins all the groups containing G (join cascade). Hence, all the keys of groups containing G are considered compromised and have to be renewed.

Let us suppose that a sensor node $\tilde{s}$ joins the group G. Then, KMS performs the following actions to guarantee the backward security.

(1) In case the set V contains no vertex associated with $\tilde{s}$, then KMS adds the new vertex $\nu(\tilde{s})$. In this case, we have that $Pred(\nu(\tilde{s})) = \emptyset$.

$$V = V \cup \{\nu(\tilde{s})\}$$

(2) KMS identifies the set of vertices associated with compromised keys, referred to as $V_{cmp}(\tilde{s}, G)$. $V_{cmp}(\tilde{s}, G)$ contains the vertices associated with groups containing G that do not already include $\tilde{s}$.

$$\begin{aligned} V_{cmp}(\tilde{s}, G) &= \{\nu(G') \mid G \subseteq G' \wedge \tilde{s} \notin G'\} \\ &= Pred_E(\nu(G)) - Pred(\nu(\tilde{s})) \end{aligned}$$

(3) KMS removes the edges connecting the vertex $\nu(\tilde{s})$ and each vertex in $Pred(\nu(G)) \cap Pred(\nu(\tilde{s}))$. Then, KMS adds an edge connecting the vertex $\nu(G)$ and $\nu(\tilde{s})$.

$$\begin{aligned} E &= E - \{\langle v, \nu(\tilde{s})\rangle \mid \forall v \in Pred_E(\nu(\tilde{s})) \cap Pred(\nu(G))\} \\ E &= E \cup \{\langle \nu(G), \nu(\tilde{s})\rangle\} \end{aligned}$$

(4) For each vertex v in $V_{cmp}(\tilde{s}, G)$, KMS updates the corresponding trigger-key and the joining-key with the next-key of the chains $ICh_{T,v}$ and $Ch_{J,v}$.

$$\forall v \in V_{cmp}(\tilde{s}, G), \begin{cases} Key_T(v) = next(ICh_{T,v}) \\ Key_J(v) = next(Ch_{J,v}) \end{cases}$$

(5) For each vertex v in $V_{cmp}(\tilde{s}, G)$, KMS broadcasts the rekeying message $M_T(v)$ containing the key $Key_T(v)$ so that the sensor nodes can locally calculate the next joining-key of vertex v, after verifying the $Key_T(v)$ authenticity.

$$M_T(v) : v, Key_T(v) \quad \forall v \in V_{cmp}(\tilde{s}, G)$$

(6) Finally KMS updates the key sets of the joining member $\tilde{s}$. Let us consider, for example, the key set $KeyRing_T(\tilde{s})$. Then, for each vertex v in $V_{cmp}(\tilde{s}, G)$ KMS unicasts $\tilde{s}$ the message

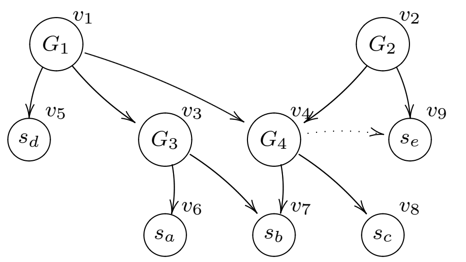


Fig. 6. Group-Graph in case of joining.

$M^\star_T(v, \nu(\tilde{s}))$ containing the trigger-key $Key_T(v)$ and the message $M^\star_J(v, \nu(\tilde{s}))$ containing the joining-key $Key_J(v)$, both encrypted by means of the sensor-key $K_{\tilde{s}}$.

$$\begin{array}{ll} M^\star_T(v, \nu(\tilde{s})) : v, E^h_{K_{\tilde{s}}}(Key_T(v)) & \forall v \in V_{cmp}(\tilde{s}, G) \\ M^\star_J(v, \nu(\tilde{s})) : v, E^h_{K_{\tilde{s}}}(Key_J(v)) & \forall v \in V_{cmp}(\tilde{s}, G) \end{array}$$

where $E^h_K(m)$ is the encryption of the message m concatenated with its hash value by means of the key K. That is, $E^h_K(m) = E_K(m \| h(m))$.

With reference to Figure 6, let us suppose that the sensor node s_e joins the group G_4, and consequently the group G_1. The sensor node s_e belongs to G_2 so that its corresponding vertex v_9 is already included in V (Step 1). It follows that $Pred(\nu(s_e)) = Pred(v_9) = \{v_2\}$.

KMS defines the set $V_{cmp}(s_e, G_4)$ containing v_1 and v_4 (Step 2). Since G_4 is sub-group of G_2, KMS removes the edge $\langle v_2, v_9\rangle$ and adds the edge $\langle v_4, v_9\rangle$ (Step 3). For each vertex v in $V_{cmp}(s_e, G_4)$, KMS chooses the next-key of chains $ICh_{T,v}$ (Step 4). Then, at Step 5 KMS broadcasts the following messages:

$$\begin{array}{l} M_T(v_1):\ v_1, next(ICh_{T,v_1}) \\ M_T(v_4):\ v_4, next(ICh_{T,v_4}) \end{array}$$

Finally, KMS unicasts s_e the following messages (Step 6):

$$\begin{array}{rl} M^\star_J(v_1, v_9): & v_1, v_9, E^h_{K_{\tilde{s}}}(next(Ch_{J,v_1})) \\ M^\star_T(v_1, v_9): & v_1, v_9, E^h_{K_{\tilde{s}}}(next(ICh_{T,v_1})) \\ M^\star_J(v_4, v_9): & v_1, v_9, E^h_{K_{\tilde{s}}}(next(Ch_{J,v_4})) \\ M^\star_T(v_4, v_9): & v_4, v_9, E^h_{K_{\tilde{s}}}(next(ICh_{T,v_1})) \end{array}$$

In order to reduce the communication overhead KMS can unicast more keys in one message. Such implementation depends on the structure of the CSMA protocol. However, KMS has to take into account that using longer frame could increase the probability of frame loss as a result of collisions.

A sensor node behaves differently according to whether it is the joining node or it is already member of the network.

Let us consider a sensor node $s \neq \tilde{s}$ that receives the message $M_T(v)$ broadcast by KMS (Step 5). Firstly, s verifies whether the rekeying message refers to a group which s belongs to. That is, s verifies whether the set of s predecessors $Pred(\nu(s))$ contains the vertex v. If this is the case, the sensor node s has to verify the key authenticity of the trigger-key K contained in $M_T(v)$. Since the trigger-key belongs to an Inverted Key-Chain (Section 4.2), the sensor node s checks whether the hash value of the received key K matches the value of $Key_T(v)$. That is, $H(K) = Key_T(v)$. If it is the case, s updates the key set $KeyRing_T(s)$ with the new value so that $Key_T(v) = K$. Then, the sensor node s renews the joining-key of vertex v by applying the hash function to the key $Key_J(v)$ stored in $KeyRing_J(s)$ (Section 4.1). Let us suppose the vertex v is associated with group G'. Since the key $K_{G'}$ corresponds to the joining-key of vertex v (Equation 6), at the end of the protocol the sensor node s updates the value of $K_{G'}$.

Let us consider the joining node $\tilde{s}$ receiving a rekeying message $M^\star_J(v, \nu(\tilde{s}))$ (Step 6). The joining sensor node decrypts the message with its sensor-key and separates the recovered key K from the recovered hash H. Then, $\tilde{s}$ computes the hash function on K and compares it with H. If these quantities are equal, K is accepted as being authentic. The encryption protects the appended hash so that it is unfeasible for an attacker without $K_{\tilde{s}}$ to alter the message without disrupting the correspondence between K and its hash value [Menezes et al. 1996]. Then, $\tilde{s}$ inserts the key K in the corresponding key set on the basis of the received message. In fact, in case of $M^{\star}_{J}(v, \nu(\tilde{s}))$, the sensor node $\tilde{s}$ updates the key $Key_J(v)$ belonging to $KeyRing_J(s)$.

5.5 Forward and backward security

In order to guarantee both the backward and forward security, LARK hinges on the Leaving Key-Graph, described in Section 5.3, and the Joining Key-Graph, described in Section 5.4.

In this case, each sensor node s stores the following key sets:

$$\begin{aligned} KeyRing_L(s) =&\ \{Key_L(v) \mid \forall v \in Pred(\nu(s))\} \\ KeyRing_T(s) =&\ \{Key_T(v) \mid \forall v \in Pred(\nu(s))\} \\ KeyRing_J(s) =&\ \{Key_J(v) \mid \forall v \in Pred(\nu(s))\} \end{aligned}$$

The key of group G is given by mixing the leaving-key of the vertex $\nu(G)$, and the joining-key of $\nu(G)$. Hence, given μ the mixing function, we have the following equation:

$$\begin{aligned} K_G =&\ \mu(Key_L(\nu(G)), Key_J(\nu(G)) \\ =&\ \mu(curr(ICh_{L,\nu(G)}), curr(Ch_{J,\nu(G)})) \end{aligned} \tag{7}$$

When a sensor node $\tilde{s}$ leaves the group G, and thus all the sub-groups of G, KMS and the sensor nodes perform the algorithm described in Section 5.3. In particular, KMS renews the leaving-key of vertex $\nu(G)$, and securely distributed it to all G members but the leaving one. Since the leaving member is not able to calculate the renewed leaving-key, and thus the renewed group-key (Equation 7), the protocol guarantees the forward security.

When a sensor node $\tilde{s}$ joins the group G, and thus all the groups containing G, KMS and the sensor nodes perform the algorithm described in Section 5.4. In particular, KMS broadcasts the the trigger-key of vertex $\nu(G)$ so that each sensor node locally renews the corresponding joining-key. Then, KMS unicasts $\tilde{s}$ the renewed joining-key. Since the joining member is not able to calculate the previous joining-key, and thus the previous group-key (Equation 7), the protocol guarantees the backward security.

5.6 Initialization and reconfiguration

In the initialization phase, KMS initialises sensor nodes via off-line methods. KMS assigns an identifier and a sensor-key to each member. The sensor-key for s is generated as follows: $K_s = f(MK, s)$, where s is the node identifier, 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 and computes K_s whenever it needs to communicate with s.

In order to guarantee both the forward and backward security, for each vertex v associated with a group, KMS computes two Inverted Key-Chains, $ICh_{L,v}$ and $ICh_{T,v}$, and a Key-Chain, $Ch_{J,v}$. During the initialization phase, each sensor node s securely receives the chain-heads of vertices that are predecessors of $\nu(s)$. More in detail, the set $KeyRing_L(s)$ initially contains the chain-heads of $ICh_{L,v}$ for each vertex v belonging to $Pred(\nu(s))$. Equivalently, the sets $KeyRing_T(s)$ and $KeyRing_J(s)$ contain the chain-heads of $ICh_{T,v}$ and $Ch_{J,v}$ respectively for each vertex v belonging to $Pred(\nu(s))$. Such initialization takes places via off-line methods or through the network. In the latter case, KMS has to guarantee the key confidentiality and authenticity by unicasting s the following messages $M^{\star}_{L}(v, \nu(\tilde{s}))$, $M^{\star}_{T}(v, \nu(\tilde{s}))$, and $M^{\star}_{J}(v, \nu(\tilde{s}))$ for each vertex in $Pred(\nu(s))$.

The chains have a limited length so that when all keys have been distributed, the chain has run out, and KMS has to reconfigure it as follows. Let us suppose that KMS has distributed all the leaving-keys of the Inverted Key-Chain associated with vertex v. KMS builds a new key chain $ICh_{T,v}$ as specified in Section 4. Then, for each sensor node whose corresponding vertex is a successor of v KMS has to unicasts $M^{\star}_{T}(v, \nu(\tilde{s}))$ containing the chain-head $head(ICh_{T,v})$.

The length of the chains could depend on the vertex with which it is associated. More in detail, the chains of vertices associated with groups whose membership is highly dynamic are longer than the chains of vertices associated with static groups that are periodically rekeyed. Furthermore, let us consider a group G and its sub-group G'. The members of group G' are also members of G and

Table I. Rekeying messages

Message	***Destination***	***Keys***
$M_L(v, w)$	BRDcst	$E_{Key_L(w)}(Key_L(v))$
$M_L^\star(v, \nu(s))$	UNIcst	$E^h_{K_s}(Key_L(v))$
$M_T(v)$	BRDcst	$Key_T(v)$
$M_T^\star(v, \nu(\tilde{s}))$	UNIcst	$E^h_{K_s}(Key_T(v))$
$M_J^\star(v, \nu(\tilde{s}))$	UNIcst	$E^h_{K_s}(Key_J(v))$

when a sensor node joins the group G' it also joins the group G. When a sensor node leaves the group G', it usually leaves also the group G. For this reason, the chains associated with a vertex v are consumed less quickly than the ones associated with v predecessors.

The size of messages broadcast by *KMS* limits the length of the chains and the number of groups that are included in the system. As shown in Table I, the rekeying messages contain the indexes of the corresponding vertices. Hence, these indexes limit the number of vertices so that the graph is composed of at most $2^{\|v\|}$ vertices, where $\|.\|$ is the bit-length of the vertex identifier. Furthermore, the messages contain the indexes corresponding to the position of the key in the chains. These indexes are used for synchronisation among sensor nodes. Consequently, these indexes limit the length of the corresponding key-chains.

It is worthwhile to notice that messages $M_L(v, w)$ and $M_T(v)$ contain only the key without appending a digital signature for guaranteeing authenticity.

6. ANALYSIS AND EVALUATION

In this section we present a formal security analysis of LARK and a performance evaluation based on the protocol prototype. In particular, the prototype has been implemented on very simple devices of TmoteSky class [Moteiv]. Thus, we show that the protocol is suitable for devices with low computational and communication capabilities.

6.1 Security Analysis

In this section, we formally argue that LARK guarantees both the forward and the backward security. Let us suppose that at time $t^\star$ the membership of a group G changes so that a sensor node leaves the group, or a new sensor node joins the group.

Let us define G_{bfr} and G_{aft} the set of sensor nodes holding the key of group G before and after $t^\star$ respectively. It follows that:

(1) in case the sensor node $\tilde{s}$ leaves the group G at time $t^\star$, we have to prove that G_{aft} includes every user of G_{bfr}, but the leaving one (*forward security*).

$$G_{aft} = G_{bfr} - \{\tilde{s}\} \tag{8}$$

(2) in case the new sensor node $\tilde{s}$ joins the group G at time $t^\star$, we have to prove that G_{aft} includes every user of G_{bfr} and the joining one (*backward security*).

$$G_{aft} = G_{bfr} \cup \{\tilde{s}\} \tag{9}$$

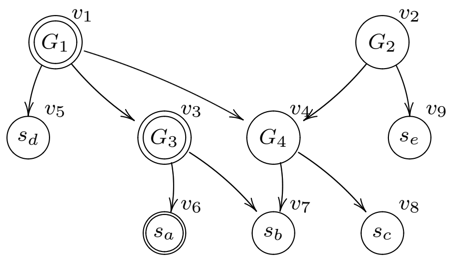
6.1.1 *Forward security.* In order to prove forward security, Equation 8 can be rewritten as follows:

$$s \in G_{aft} \Leftrightarrow (s \in G_{bfr}) \wedge (s \neq \tilde{s}) \tag{10}$$

Let $V_{enc}(\tilde{s}, G)$ be the set of vertices not included in $V_{cmp}(\tilde{s}, G)$ that have a direct predecessor in $V_{cmp}(\tilde{s}, G)$. Intuitively, $V_{enc}(\tilde{s}, G)$ contains the vertices whose key is not compromised and is used to encrypt a rekeying message. Figure 7 shows an example of $V_{enc}(\tilde{s}, G)$ when the sensor node s_a leaves the group G_1.

$$V_{enc}(\tilde{s}, G) = \{w \mid w \notin V_{cmp}(\tilde{s}, G) \wedge \exists v \in V_{cmp}(\tilde{s}, G) \cap Pred_D(w)\} \tag{11}$$

The proof will be articulated in two parts. In Proposition 6.1 we prove that a sensor node s belongs to G_{aft} if and only if the vertex of node s, or the vertex of one of its predecessors, belongs



$$V_{cmp}(s_a, G_1) = \{v_1, v_3\}$$
$$V_{enc}(s_a, G_1) = \{v_4, v_5, v_7\}$$

Fig. 7. Example: s_a leaves the group G_1.

to $V_{enc}(\tilde{s}, G)$.

$$s \in G_{aft} \Leftrightarrow (Pred_E(\nu(s)) \cap V_{enc}(\tilde{s}, G) \neq \emptyset) \tag{12}$$

In Proposition 6.2 we prove that the vertex of node s, or the vertex of one of its predecessors, belongs to $V_{enc}(\tilde{s}, G)$ if and only if s belongs to G_{bfr} and $s \neq \tilde{s}$.

$$(Pred_E(\nu(s)) \cap V_{enc}(\tilde{s}, G) \neq \emptyset) \Leftrightarrow (s \in G_{bfr}) \wedge (s \neq \tilde{s}) \tag{13}$$

By combining Equation 12 and 13, it follows that a sensor node belongs to G_{aft} if and only if it belongs to G_{bfr} and $s \neq \tilde{s}$ that is exactly what we would like to prove (Equation 10).

PROPOSITION 6.1. *The sensor node s belongs to G_{aft} if and only if the set of vertices $V_{enc}(\tilde{s}, G)$ contains the vertex of node s or the vertex of one of its predecessors (Equation 12).*

PROOF. In general, a sensor node belongs to the group G if and only if the node holds the group key K_G. According to Equations 5 and 7, a sensor node s can calculate K_G if and only if it can access the corresponding leaving-key $Key_L(\nu(G))$. Therefore, the sensor node s belongs to G_{aft} if and only if s can access the renewed leaving-key of vertex $\nu(G)$. Given the one-way properties of the Inverted Key-Chain $ICh_{L,\nu(G)}$, a sensor node cannot compute the next key even if it has access to the previous ones (Section 4.3). It follows that s belongs to G_{aft} if and only if it is able to decrypt at least one of the rekeying messages that *KMS* broadcasts at Step 5 (Section 5.3).

Let us consider the rekeying message $M_L(v, w)$ containing the renewed leaving-key of vertex v encrypted by means of $Key_L(w)$. It is worthwhile to notice that *KMS* broadcasts the rekeying message in such an order that when *KMS* broacasts the rekeying message of vertex v, it has already broadcast the rekeying messages of the compromised successors of v, i.e., those successors belonging to $V_{cmp}(\tilde{s}, G)$. So, if the sensor node s has access to the renewed leaving-key of vertex $v \in V_{cmp}(\tilde{s}, G)$, $v \neq \nu(G)$, then s can recursively have access to $Key_L(\nu(G))$. So, given $v \in V_{cmp}(\tilde{s}, G)$, the sensor node s can decrypt $M_L(v, w)$ and thus have access to the renewed key of vertex v if and only if 1) $Key_L(w)$ is not compromised, and 2) s stores the key $Key_L(w)$. If $Key_L(w)$ is not compromised, then w does not belong to $V_{cmp}(\tilde{s}, G)$, w is a direct successor of v, and v is compromised. Thus, the vertex w belongs to $V_{enc}(\tilde{s}, G)$. If s stores the key $Key_L(w)$, then either $Key_L(w)$ corresponds to K_s or belongs to $KeyRing_L(s)$. In other words, the vertex w is either associated with the sensor node s or is included in $Pred(\nu(s))$. So, the vertex w is contained in $V_{enc}(\tilde{s}, G) \cap Pred_E(\nu(s))$. It follows that s can have access to the key of the compromised vertex v if and only if the intersection between $V_{enc}(\tilde{s}, G)$ and $Pred_E(\nu(s))$ is not empty. $\square$

PROPOSITION 6.2. *The set of vertices $V_{enc}(\tilde{s}, G)$ contains the vertex $\nu(s)$ or one of $\nu(s)$ predecessors if and only if the sensor node s belongs to G and $s \neq \tilde{s}$ (Equation 13).*

PROOF. In the first part, we prove that:

$$(s \in G_{bfr}) \wedge (s \neq \tilde{s}) \Rightarrow (Pred_E(\nu(s)) \cap V_{enc}(\tilde{s}, G) \neq \emptyset) \tag{14}$$

Let us consider a node s belonging to G_{bfr} so that $s \neq \tilde{s}$. The node s is included in a sub-group of G_{bfr} containing $\tilde{s}$, or in a sub-group of G_{bfr} that does not contain $\tilde{s}$. Consider a sub-group

G' of G_{bfr}, $G' \subseteq G_{bfr}$ with s and $\tilde{s}$ in G' and $s \neq \tilde{s}$. Then, 1) there is an edge connecting the vertices $\nu(s)$ and $\nu(G')$, and 2) and $\nu(G')$ is included in $V_{cmp}(\tilde{s}, G)$. By definition $V_{cmp}(\tilde{s}, G)$ contains only vertices associated with groups, that is, $\nu(\tilde{s}) \notin V_{cmp}(\tilde{s}, G)$. Hence, $\nu(s)$ is included in $V_{enc}(\tilde{s}, G)$. Since $Pred_E(\nu(s))$ contains $\nu(s)$, it follows that $\nu(s) \in Pred_E(\nu(s)) \cap V_{enc}(\tilde{s}, G)$, and thus Equation 15.

$$(s \in G' \wedge G' \subseteq G_{bfr} \wedge \tilde{s} \notin G') \Rightarrow Pred_E(\nu(s)) \cap V_{enc}(\tilde{s}, G_{bfr}) \neq \emptyset \tag{15}$$

Consider now $G' \subseteq G_{bfr}$ with $\tilde{s} \notin G'$, $s \in G'$ and $s \neq \tilde{s}$. Then, 1) the set $Pred_E(\nu(s))$ contains $\nu(G')$ and its predecessors, and 2) $\nu(G')$ is not included in $V_{cmp}(\tilde{s}, G)$, whereas $\nu(G)$ is. Since G' is sub-group of G_{bfr}, we have that the vertex $\nu(G')$ or one of its predecessors is not compromised and has a direct predecessor in $V_{cmp}(\tilde{s}, G)$. It follows that there is a vertex $\overline{w}$ so that $Pred(\nu(s)) \cap V_{enc}(\tilde{s}, G) \neq \emptyset$ and thus Equation 16.

$$(s \in G' \wedge G \subseteq G' \wedge \tilde{s} \in G') \Rightarrow Pred_E(\nu(s)) \cap V_{enc}(\tilde{s}, G) \neq \emptyset \tag{16}$$

So, by combining the Equations 15 and 16 we have Equation 14.

In this second part of the proof, we prove that if the intersection between $Pred_E(\nu(s))$ and $V_{enc}(\tilde{s}, G)$ is not empty, then the sensor node s belongs to $\tilde{s}$ and $s \neq \tilde{s}$. That is,

$$(Pred_E(\nu(s)) \cap V_{enc}(\tilde{s}, G) \neq \emptyset) \Rightarrow (s \in G) \wedge (s \neq \tilde{s}) \tag{17}$$

The proof is by contradiction. By absurd let us suppose that $s \notin G$ or $s = \tilde{s}$. Let us consider a sensor node s so that s does not belong to G_{bfr}. Then, we have that $\nu(s)$ is not included in $Succ_E(\nu(G))$, or in other words that $Pred_E(\nu(s)) \cap Succ_E(\nu(G)) = \emptyset$. Since $V_{enc}(\tilde{s}, G) \subseteq Succ_E(\nu(G))$, it follows that $Pred_E(\nu(s)) \cap V_{enc}(\tilde{s}, G) = \emptyset$. Given $s = \tilde{s}$, at step 2 (Section 5.3), *KMS* removes every edge connecting the vertex $\nu(s)$ with the vertices belonging to $V_{cmp}(\tilde{s}, G)$. It follows that neither $\nu(s)$ nor one of $\nu(s)$ predecessors belong to $Succ_E(\nu(G))$. That is, $Pred_E(\nu(s)) \cap Succ_E(\nu(G)) = \emptyset$. Since $V_{enc}(\tilde{s}, G)$ is a subset of $Succ_E(\nu(G))$, it follows that $Pred_E(\nu(s)) \cap V_{enc}(\tilde{s}, G) = \emptyset$. Then, the assumptions $s \notin G$ or $s = \tilde{s}$ lead to an absurd and thus we have the Equation 17. □

6.1.2 *Backward security.* In order to prove the backward security, Equation 8 can be rewritten as follows:

(1) The set G_{aft} includes every sensor node in G and the joining one $\tilde{s}$.

$$s \in G_{aft} \Leftrightarrow (s \in G_{bfr}) \vee (s = \tilde{s}) \tag{18}$$

(2) The joining sensor node $\tilde{s}$ has not access to the group-key held by nodes in G_{bfr}.

According to Equation 6 and 7, a sensor node s belongs to G_{aft} if and only if it can have access to the renewed joining-key of vertex $\nu(G)$. The one-way properties of the Key-Chain $Ch_{J,\nu(G)}$ guarantees the a sensor node s can obtain the next joining-key $next(Ch_{J,\nu(G)})$ only if it already stores the current-key $curr(Ch_{J,\nu(G)})$, or if it receives the next-key through a predefined authenticated channel, as described in Section 4.3.

A sensor nodes s belonging to G_{bfr} already stores the current joining-key $curr(Ch_{J,\nu(G)})$ and the current trigger-key $curr(ICh_{T,\nu(G)})$. So, when the sensor node s receives the rekeying message $M_T(\nu(G))$ containing the renewed trigger-key, s can verify the key authenticity on the basis of the properties of the Inverted Key-Chains (Section 4.2). Then, s can also locally calculate the next joining-key by simply applying the hash function as described in Section 4.1. Thus, if a sensor node s belongs to G_{bfr}, it also belongs to G_{aft}.

On the contrary, the joining member $\tilde{s}$ receives the message $M_J^{\star}(\nu(G), \nu(\tilde{s}))$ containing the renewed joining-key of vertex $\nu(G)$ encrypted with its own sensor-key. Thus, the sensor node $\tilde{s}$ belongs to G_{aft}. Furthermore, given the one-way properties of the Key-Chain $Ch_{J,\nu(G)}$, $\tilde{s}$ is not able to compute the previous keys (Section 4.1) and thus have access to the previous communication.

Table II. Computational cost to process a rekeying message (in milliseconds) using SkipJack (a) or RC5 (b)

(a)

Operation	M_L	$M_L^\star$	M_T	$M_T^\star$	$M_J^\star$
SHA-1	3.91	3.91	2×3.91	3.91	3.91
SkipJack	1.46	1.93	-	1.93	1.93
Preproc.	1.22	1.22	1.22	-	1.22
Total	7.24	7.47	9.34	5.88	7.47

(b)

Operation	M_L	$M_L^\star$	M_T	$M_T^\star$	$M_J^\star$
SHA-1	3.91	3.91	2×3.91	3.91	3.91
RC5	3.85	5.12	-	5.12	5.12
Preproc.	6.45	6.45	6.45	-	6.45
Total	14.81	15.76	14.47	9.60	15.76

6.2 Performance evaluation

In this subsection we analyse storage, communication, and computing overhead of LARK. In particular, we show the protocol is adequate for resource-constrained devices, such as TmoteSky sensor nodes.

6.2.1 *Prototype.* TmoteSky sensor nodes are device powered with two AA batteries and equipped with a 16-bit 8MHz MSP430 micro-controller, 48 Kbytes of ROM, 10 Kbytes of RAM, and IEEE 802.15.4 radio interface. They run the TinyOS operating system [Hill et al. 2000]. Our implementation uses SkipJack [National Institute of Standards and Technology 1998] or RC5 [L. 1994] as symmetric cipher, and SHA-1 [National Institute of Standards and Technology 1995] as hash function. We borrowed the TinySec implementation of these algorithms [Karlof et al. 2004]. However, as TinySec did not support Tmote Sky, we had to port the machine-dependent parts of such implementations onto our platform. The structure of messages of the prototype is based on the packet format of TinyOS. The header of the message contains the destination address, the appropriate handler function to extract and interpret the message on the receiver, and the length of the payload. The payload contains the fields `KData` and `KIndex`. In particular, `KData` contains the cryptographic key that *KMS* unicasts to a specific-sensor node or broadcasts to the group members. The field `KIndex` contains the identifier of the keys contained in the rekeying message. The key of a group G is identified by its vertex $\nu(G)$ and its position in the corresponding key-chain, whereas the sensor-key is identified by the sensor identifier. Let us consider the message $M_L(v, w)$ containing the key $Key_L(v)$ encrypted by using the key $Key_L(w)$. Hence, `KIndex` contains the index v and w, and the position of keys $Key_L(v)$ and $Key_L(w)$ in the key chains $ICh_{L,v}$ and $ICh_{L,w}$ respectively. In case of message $M_T(v)$, `KIndex` contains the index v and the position of key $Key_T(v)$ in the key chain $ICh_{T,v}$. In case of unicast messages, such as $M_J^\star(v, \nu(\tilde{s}))$, `KIndex` contains the indexes v and $\nu(\tilde{s})$, and the position of $Key_J(v)$ in the key chain $Ch_{J,v}$. Finally, the rekeying message contains Cyclic Redundancy Code (CRC) that is used by receivers to detect transmission errors.

6.2.2 *Computing and storage overhead.* In this section we discuss computing and storage costs of LARK for both sensor nodes and Key Management Service *KMS*. As to sensor nodes, we consider both the computing and storage overhead due to their reduced resources. In contrast, as to *KMS*, we only consider the storage overhead.

Table II reports the total amount of time (in milliseconds) employed by each sensor node to process a rekeying message. The table highlights the three main operations contributing to the computation overhead that are: i) decrypting the message by means of a symmetric cypher, ii) verifying the key authenticity by means of an hash function, and iii) refreshing the corresponding group-key. In particular, from the implementation point of view refreshing a key consists in pre-processing the key as required by cypher. More in detail, upon receiving a message containing a leaving-key, e.g., M_L or $M_L^\star$, the sensor node decrypts the message and calculates the hash value to verify the authenticity. Then it calculates and pre-processes the corresponding group-key. When a member receives a message M_T containing the trigger-key, it verifies the key authenticity by applying the hash function. Then it renews the join-key by means of the hash function and pre-processes the corresponding group-key. When a sensor node receives a message $M_T^\star$, it has only to decrypt the messages and to verify the key authenticity without renewing the other keys. When a sensor node receives a message $M_J^\star$, it decrypts the message and verifies its authenticity. Then, it updates its stored join-key and pre-processes the corresponding group-key.

As shown in the table, the proposed solution requires from 5.88 ms to 9.34 ms to process a rekeying message by using Skipjack as symmetric cypher (Table II.a), and from 9.80 ms to 15.56 ms by using RC5 (Table II.b). In particular, to verify the authenticity of a key each sensor node applies the hash-function SHA-1 that requires 3.91 ms. This is an improvement with respect to other solutions such as [Waldvogel et al. 1999], that uses a digital signature for key authentication.

As to storage overhead in sensor nodes, we consider two aspects, namely the storage necessary for cryptographic keys and that for the code. Although memory space is a very scarce resource for the current generation of sensor nodes, storage of keys is not a problem in our scheme. A sensor node s needs to store the sensor-key K_s, and the key ring. With reference to Section 5.5, a sensor node could avoid storing the group-keys and could compute them from the corresponding joining-keys with leaving-keys as needed. However, so doing, it would each time incur in the cost of pre-processing the group-key as required by the cipher. Let us suppose the sensor node belongs to 10 groups, i.e. in case of a WSN composed of almost 1024 sensor nodes and LKH grouping. Hence, every sensor node has to store at most 31 keys in order to guarantee both forward and backward security. Assuming a key size of 10 bytes, keys totally require 310 bytes of storage, that is 3.1% of data memory. So, the proposed protocol is even suitable for devices with such strict memory requirement.

As to the storage necessary for the code, the memory footprint of SHA-1 raises some concerns. This is a general problem for hash functions: their memory footprint tends to be so large that little space remains in memory for application code [Roman et al. 2007]. Our implementation of SHA-1 has quite a considerable memory footprint, about 23 KB, as we have spent no particular effort in optimising its code size given our primary interest being to show that LARK has an acceptable overhead for sensor nodes. However, optimised implementations of SHA-1 do exist. A notable example is the one in TinyECC whose footprint for Tmote Sky is about 2.4 KB [Liu et al. 2007]. This implementation is for TinyOS, and therefore can be ported into LARK in a straightforward way.

Another possibility is to completely change the class of one-way hash functions. Instead of using a customised hash function, such as SHA-1, one could use a hash function based on block ciphers [Menezes et al. 1996]. While customised hash functions are specifically designed for hashing, with speed in mind and completely independent of other components, the hash functions based on block ciphers are instead conceived to exploit a block cipher component that may be already present in the system. So doing, it is possible to provide the hashing functionality at little additional cost. We have made an experiment in this direction by implementing the Davies-Mayer hash [Menezes et al. 1996], using the AES block cipher with 128 bits key [National Institute of Standards and technology 2001]. We have exploited the hardware implementation of AES provided by the CC2420 [Chipcon 2004], the Chipcon transceiver that implements 802.15.4 on Tmote Sky sensor nodes. We have used the standalone encryption mode. The memory footprint of the resulting implementation of the Davies-Mayer hash, including the commands to drive the standalone encryption mode, amounts to about 500 bytes which is 4 times smaller than the TinyECC optimised version of SHA-1. Furthermore the Davies-Mayer scheme reduces the hash computation overhead from 3.91 ms (Table II) to about 15 μs.

The above results regarding Davies-Mayer hash function are encouraging. However, attention must be paid when dealing with hash functions based on block ciphers. Security of such functions assumes certain ideal properties of underlying block ciphers. However, in practice real block ciphers do not possess the same properties as random functions, e.g., they are invertible, and properties adequate for block cipher may not guarantee a good hash function. In summary, while

Table III. Communication overhead per packet (in bytes)

Field	M_L	$M_L^\star$	M_T	$M_T^\star$	$M_J^\star$
Header+CRC	13	13	13	13	13
`KData`	10	20	10	20	20
`KIndex`	6	5	2	5	5
Total	29	38	25	38	38

necessary conditions are known, it is still unclear what properties of a block cipher are sufficient to construct a hash function [Menezes et al. 1996].

In an aside, it has not been possible replace to Skipjack by an hardware implementation of AES because, in the standalone mode, the CC2420 does not provide the decryption operation. This limitation can be overtaken by using CC2420 in the inline mode but this requires to integrate LARK with the IEEE 802.15.4 security sub-layer. Future work will be in this direction. A point to note is that the successor of CC2420, Chipcon CC2430, does offer AES decryption.

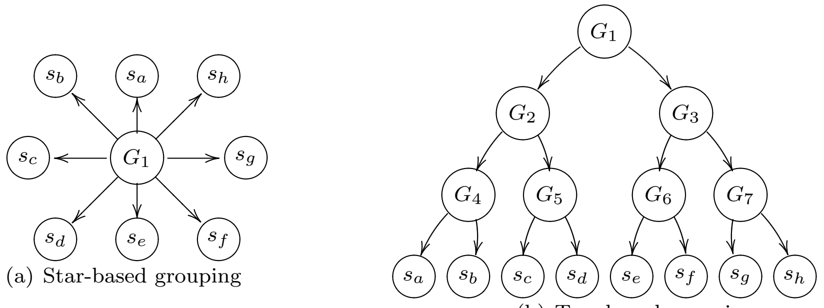
We would like to finally remark that identifying, evaluating and selecting the most appropriate hash functions and block ciphers for sensor nodes is a very important issue that is however orthogonal to the scope of the paper. There is a large amount of recent, high-quality research on this issue that LARK can exploit. Relevant examples are [Law et al. 2006; Roman et al. 2007]. However, the implementation of LARK based on SHA-1 and Skipjack presented in the paper has the aim at, and the indubitable merit of, showing that LARK is indeed feasible on resource-constrained devices such as Tmote Sky sensor nodes.

As to storage overhead at *KMS*, it stores one master key *MK* to generate n sensor-keys. *KMS* computes the sensor-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. *KMS* needs to store a key-chain for each vertex. To avoid storing the entire key-chain, we can exploit the optimisation algorithm by Coppersmith and Jakobsson [Coppersmith and Jakobsson 2002] to trade storage and computation cost. The algorithm requires $\log_2(l)$ memory cells to store a chain composed of l keys. Thus, if we assume for simplicity that every node vertex is associated with a fixed-length chain of l keys, *KMS* has to store $\log_2 l$ keys for each chain. With reference to the previous example, *KMS* has to store only 8 keys for each chain of 256 keys and thus it requires about 240 Kbytes to store all the keys in order to guarantee forward and backward security.

6.2.3 *Communication overhead.* In this section, we analyse communication overhead of LARK from two points of view: the size (in bytes) of rekeying messages and the number of messages. As mentioned in Section 5.6, the field `KData` contains the key, whereas the field `KIndex` contains the indexes of the corresponding vertices. In case of a vertex associated with a group, the field `KIndex` contains also the position of the key in the chains.

As shown in Table III, the field `KData` of unicast messages, i.e., $M_L^\star$, contains the key concatenated with its hash value. In contrast, the field `KData` of broadcast messages, such as M_L or M_T, contains only the key without a digital signature or a MDC to guarantee the authenticity. Assuming a key size of 10 bytes, then `KData` is 20 bytes-long in unicast messages, and 10 bytes-long in broadcast messages. This reduces the communication overhead of LARK. In case of a network composed of n members and n_g groups, the field `KIndex` contains the vertices identifiers that are $\log_2(n + n_g)$ bit-long. In case of a key associated with a group, `KIndex` contains the position of the key in the chain, that is $\log_2(l)$ bit-long where l is the chain length. So, the field `KIndex` of a message M_L contains the identifiers of two keys associated with groups and thus it is $2 \times (\log_2(n + n_g) + \log_2(l))$ bit-long. On the other side, `KIndex` of a message $M_L^\star$ contains the identifier of a key associated with a group and a sensor-key. Thus, `KIndex` is $2\log_2(n+n_g)+\log_2(l)$ bit-long. As shown in Table III, in case of n=1024, n_g=256, and l=256 the packet size increases only with 2-6 bytes. In particular, the total amount of bytes in a broadcast message is less than 30 bytes.

Finally, a fresh Initialization Vector (IV) is sent in the packets with encrypted data. In order to reduce the communication overhead introduced by IV, we reuse some of the fields in the packet header [Karlof et al. 2004]. In particular, IV is given by the concatenation of some fields of the



(a) Star-based grouping

(b) Tree-based grouping

Fig. 8. Examples of grouping.

header and KIndex.

In order to evaluate the number of messages we consider two topologies of grouping: the *star*-based and the *tree*-based grouping. In the former case, there is a single group containing all sensor nodes and the resulting key-graph is composed of n vertices containing the individual keys of members and connected to one vertex containing the group-key. Hence, each sensor node stores its sensor-key and the group-key shared by all the n members of the network(see Figure 8.a). In this case, LARK requires $n-1$ messages in case of a sensor node leaves the network to guarantee the forward security. On the other hand, when a new sensor node joins the network LARK requires two rekeying messages containing the trigger-key and the joining-key respectively.

In the tree-based grouping the key-graph is a tree composed of n leaves (Figure 8.b). Each leaf associated with a sensor node and the internal nodes are associated with the the groups. In case of balanced m-ary tree, each sensor node belongs to h where $h = \lceil \log_m(n) \rceil$. Hence, when a sensor node $\tilde{s}$ leaves, LARK requires $h \times m - 1$ rekeying messages. When a sensor node joins a group, LARK broadcasts at most h messages conveying the trigger-keys. Furthermore, *KMS* needs to initialise the joining member either off-line methods or by unicasting the key rings to the joining node. It follows that our approach requires the number of messages that is a logarithmic function of the network size.

In conclusion, the communication overhead strictly depends on the grouping so that the number of messages ranges from $O(n)$ to $O(\log n)$. In particular, the tree-based grouping is proposed by LKH to increase scalability in large-scale scenarios by reducing the communication overhead while guaranteeing forward/backward security [Waldvogel et al. 1999; Wong et al. 2000]. Whenever a sensor node leaves or joins the group communication LKH guarantees a number of messages that is a logarithmic function of the network size. In such a scenario our approach guarantees the same results achieved by LKH in terms of communication overhead.

6.3 Security evaluation

In this section we discuss LARK security with respect to security of hash functions (Section 6.3.1) and collusion attacks (Section 6.3.2).

6.3.1 *On hash functions.* LARK relies on the unique properties of the cryptographic one-way function. However, hash functions tend to be broken after some years. For instance, the complexity of a recent attack against the collision resistance property of SHA-1 is 2^{63} [Wang et al. 2005]. It follows that the strength of SHA-1 against collision attacks is weaker than ideal, namely 2^{80} [Menezes et al. 1996]. For this reason, SHA-1 is considered somewhat flawed [Roman et al. 2007]. Similar considerations hold for MD5, too. As sensor networks are supposed to remain operational for long periods, even several years, the possibility that a hash function may be broken meanwhile becomes a problem.

From a purely practical point of view one can use other hash functions, namely RIPEMD-160 [Dobbertin et al. 1996] or SHA-256, with no known flaws and that use the same basic operations as SHA-1. It is also important to note that there is a research effort for creating a new hash function standard which is resilient against known SHA-1 attacks [National Institute of Standards and Technology 2005]. Furthermore, it must also be noted that most of attacks are against collision resistance whereas the relevant property for LARK is resistance against preimage attacks. SHA-1 is still considered secure against this kind of attacks.

Notwithstanding these considerations, a good approach would be to consider a mechanism that allows us to revoke a broken hash algorithm and replace it with a stronger one. Of course, this is not a problem peculiar to LARK but it is an instance of the more general problem of dynamically and remotely reconfiguring (or reprogramming) a networked embedded system in order to make it adaptable to the changed operating conditions. Unfortunately reconfigurability conflicts against security as it introduces new sources of vulnerabilities. Actually, the drive to provide reconfigurability requires the ability to remotely download software after sensor nodes have been deployed [Ravi et al. 2004]. However, downloading malicious software (including viruses, worms, and Trojan horses) is by far the instrument of choice in launching security logical attacks. The magnitude of this problem will only worsen with the rapid increase in the software content of embedded systems.

We have faced with this problem by defining a security architecture that supports secure communication and secure reconfiguration in WSN. In the rest of this section we provide a brief overview of the architecture and give some intuitions about how the services it offers can be used to achieve secure reconfiguration. For a more detailed description readers can refer to [Dini et al. 2008; Dini and Savino 2010].

The security architecture integrates, enriches, and extends a component-based middleware called RUNES [Costa et al. 2005]. The resulting architecture is general because it has been designed from the ground up to be implementable on a wide range of devices, comprising low-end sensor nodes, and the abstractions it provides can be used to build applications and higher-level services. Furthermore, it is flexible as it accommodates different implementations of the security services according to the specific application requirements and constraints.

The RUNES middleware hinges on basic runtime units of encapsulation and deployment, called *components*. Components provide services to other components through one or more well-defined interfaces and can have dependencies on other components. This enables the implementation and deployment of different versions of the same component, each tailored to a specific device or operating condition. Components can be dynamically added and removed. This makes it possible to dynamically reconfigure applications according to the changing operational conditions. Reconfiguration consists in downloading a new component from a possibly remote source, instantiating and/or removing components at runtime, and dynamically changing the components interconnections.

On this component-based layer, we have defined a suite of security services. Those relevant for this discussion are the Authenticated Loader and the Negotiator. The *Authenticated Loader* remotely downloads components guaranteeing that they come from trusted sources. Newly downloaded components can be then instantiated and bound, whereas resident components may be unbound and discarded as necessary. Of course, this reconfiguration activity has to be synchronised with applications. The *Negotiator* fulfils this task.

In this component-model, LARK has been implemented as a *component framework*, i.e., a set of inter-dependent components. One of these components implements the one-way hash function. It follows that revoking and replacing a broken hash functions consists in downloading a new component implementing a stronger hash function, unbinding and discarding the component implementing the broken one, and, finally, instantiating and binding the newly downloaded one to LARK.

One complication stems from the fact that authenticated downloading of software components in WSN must be efficient in terms of communication, storage and computation in order to be sustainable on sensor nodes. For this reason the Authenticated Loader uses the *authenticated hash chain*, a scheme using a one-way hash function and based on the same principle as Inverted Key-Chains. Other secure code distribution schemes use this scheme or its variations [Deng et al. 2006; Dutta et al. 2006]. However, if the hash function is broken, authenticated downloading would be not secure anymore. This problem can be faced in two ways. The hash function is replaced as soon as the early signs of weakness appear. Alternatively, the Authenticated Loader component is implemented in two ways: an efficient way using the authenticated key chain scheme, and an inefficient way not using that scheme. A sensor node is equipped with both implementations. The efficient implementation will be used to download application components, whereas the inefficient implementation will be used to replace components implementing a broken hash function. Upon downloading, the Negotiator negotiates which implementation of the Authenticated Downloading has to be used.

6.3.2 *On collusion attacks.* In a collusion attack an adversary attacks individual nodes with the objective of incrementally aggregating keys to a level that makes it possible to violate protected traffic in the network. For simplicity, but without loss of generality, let us consider an adversary that succeeds in compromising nodes belonging to group G, say node s_a and s_b. Let us assume that at some point the IDS detects the compromised nodes which are then removed from the group by distributing a new group key. However, when a node is compromised all the keys it has received during its membership in the group fall into the adversary's hands and the adversary may attempt to exploit keys obtained from compromised nodes to violate the backward and forward security requirements. Forward security is violated if the adversary succeeds in somehow combining the keys obtained from compromised nodes, e.g., s_a and s_b, in such a way to obtain the new group G key. This possibility mainly depends on the group re-keying scheme and should be avoided altogether because, otherwise, the adversary can break the scheme and capture the network. As to backward security, if node s_b joins the group G after s_a (possibly even after its leaving), and the adversary succeeds in passing node s_b the keys obtained from node s_a, then node s_b becomes able to access all the traffic protected by using the "old" keys. As this traffic dates back to prior node s_b's joining, then backward security is violated. It follows that backward security violation is inevitable in a group re-keying scheme. It is the specific security-performance trade-off the scheme implements that influences the portion of past traffic that is affected by a collusion attack.

From a purely practical point of view, we initially observe that, in general, in order to compromise the group key of a given group the adversary has to attack individual members of that group. As in LARK grouping follows application requirements, physically close nodes may not be members of the same given application group. Therefore, just compromising neighbouring nodes picked at random might be not effective, and the adversary would instead need precise information about the grouping topology. Furthermore, for an adversary it is preferable to compromise physically close nodes or, otherwise, the chances of being promptly detected would increase [Younis et al. 2005].

In the literature, concerns from collusion attacks are mainly about forward security in order to avoid that evicted members can work together and share their individual piece of information to *regain* access to the group key [Rafaeli and Hutchison 2003]. LARK has the merit to guarantee the forward security in presence of collusion attacks. In fact, the use of inverted key chains at the group level guarantees that it is not practically possible to derive the next leaving keys from previous ones, no matter how many of them have been compromised. Other approaches, such as SHELL [Younis et al. 2006] and LiSP [Park and Shin 2004], do not guarantee forward security in presence of colluding nodes. In fact, SHELL is based on a combinatorial approach where keys are reused in multiple nodes and only key combinations are unique [Younis et al. 2006]. It follows that collusion of a few nodes can reveal all the keys employed in the network to the adversary so causing forward security to be completely broken and the consequent capture of the entire network. Optimal assignment of keys to prevent network capturing is a classical resource allocation problem that is NP-hard. Thus, SHELL mitigates, but not eliminates, collusion using allocation heuristics that reduce the probability of capturing the entire network but require the knowledge of nodes' locations in computing keys. In LiSP, a collusion attack makes it possible to compromise "future" keys of certain clusters (see Section 7.2 for more details). It follows that forward security can be temporarily violated in portions of the network.

As to backward security, LARK fully guarantees it with respect to isolated, uncoordinated attacks whereas, in the case of a collusion attack, LARK exposes a segment of past traffic. More in detail, in case an adversary compromises node $s_a \in G$ and later node s_b belonging to the same group, then node s_b can access the traffic of group G that extends in the past to the moment node

s_a was compromised. In other re-keying schemes, a backward security violation has a *minimum* impact. For instance, Key Graphs generates fresh keys for each joining and leaving event, use key graphs for key distribution and conventional digital signatures for key authentication [Wong et al. 2000]. Thus, upon receiving the keys of node s_a, the colluding node s_b becomes able to access the traffic corresponding to the period in which the compromised node s_a was member of G. Key Graphs achieves this result as it uses digital signatures for key authentication which are are possible but not advisable on sensor nodes (see Section 7.1). Using a more lightweight key authentication mechanism determines a different impact of a backward security violation. For instance, LiSP uses only inverted key chains for key authentication, and, practically, does not guarantee backward security. Actually, if the adversary obtains a group key TK_t—a temporal key in their parlance— from a node then he becomes able to compute *all* the previous group keys $TK_i, i \leq t$ and thus access all the related past traffic.

LARK positions itself in between these two extremes. In fact, LARK uses the key graph mechanism for efficient large-scale key distribution but, differently to Key Graphs, uses an inverted key chain (leaving keys) instead of digital signatures for key authentication, and, differently to LiSP, uses also a direct key chain (joining keys) to guarantee backward security to the extent mentioned above. If LARK's trade-off is not acceptable, one has to resort to other schemes that sacrifice performance at the benefit of increased security. For instance LEAP+ has a minimum exposure to a backward security violation without using digital signatures for authentication. However, it uses one μTesla authenticated broadcast to revoke a node and another confidential broadcast to distribute a fresh key, for a total of three broadcasts, and operates under the strong μTesla assumptions on time synchronisation and maximum round-trip time [Zhu et al. 2006].

However, we believe that the performance penalties deriving from reducing the impact of a backward security violation down to the minimum—i.e., that of LEAP+ or Key Graphs—are not worth given the particular application context for which LARK has been conceived. Backward security is generally an important requirement that is crucial in applications and services based on secure contents distribution such as secure radio, video broadcast, pay-Tv [Caronni et al. 1999; Sherman and McGrew 2003; Wong et al. 2000]. Intuitively, backward security prevents a service subscriber to access contents broadcast prior to its subscription. However, we believe that backward security is not so crucial for the radically different application context for which LARK has been conceived, namely WSANs (Section 2). In WSANs, where applications deal with monitoring and control [Årzén et al. 2007; Bicchi et al. 2008], integrity is a top priority [Cárdenas et al. 2009]. Thus in WSANs forward security, and in particular the ability to force a compromised node to leave, acquires more relevance. Of course, this does not mean that backward security is not important at all in WSANs. However, the effect of a backward security violation on integrity would be not critical. Actually, the adversary could attempt to inject messages by means of "past" keys but they will be discarded by the monitoring and control algorithm.

7. RELATED WORK

Several group key management systems for WSNs have been proposed [Choudhary et al. 2007; Eltoweissy et al. 2004; Eltoweissy et al. 2005; Park and Shin 2004; Son et al. 2007; Wang and Ramamurthy 2007; Younis et al. 2006; Zhu et al. 2006]. Some take a centralised approach as LARK [Eltoweissy et al. 2004; Eltoweissy et al. 2005; Younis et al. 2006; Perrig et al. 2001; Park and Shin 2004; Younis et al. 2006; Zhu et al. 2006], whereas others a distributed one [Choudhary et al. 2007; Zhu et al. 2006]. Distributed versus centralised key group management is an open debate. In a centralised scheme, the key distribution centre constitutes a single-point-of-failure and may cause a performance bottleneck. Of course, this is not the case in distributed schemes. However, distributed schemes typically establish pair-wise keys first, then group keys with geographical neighbours, and, finally, a network-wide key. This bottom-up approach to key establishment is conducive to increase the computation and communication costs with respect to the centralised one. Actually, in a centralised scheme, the new keying material is generated by the key distribution centre so minimising the computation overhead for sensor nodes and the communication overhead is limited to the one-time delivery cost of new keys.

Similarly to LARK, the *scalability problem* of group key management for large scale WSNs with joins and leaves has been addressed by all these schemes. Both these schemes and LARK make use of hierarchy to solve the scalability problem. However, the similarity ends here. LARK and these schemes differ from both the approach and the architectural point of view.

First of all, approaches and objectives are very different. The other schemes use hierarchy, i.e., clustering, as a network topology control technique [Choudhary et al. 2007; Park and Shin 2004; Zhu et al. 2006]. They group neighbouring sensor nodes and hierarchically merge groups to establish a network-wide shared key [Choudhary et al. 2007; Zhu et al. 2006]. This features a bottom-up approach in which the application level "inherits" the group topology fixed by the network management level. As discussed in Section 2, this is adequate for mainstream WSNs but is not for the emerging WSANs. In contrast, LARK takes a radically different, top-down, approach and uses hierarchy as a means to specify application-defined secure groups. This means that a particular group topology is defined according to the applicative needs and that the group key management system has to manage group keys according to it.

Furthermore, system architectures are very different too. To fix ideas, let us consider a tree hierarchy with a single root. In LARK, the hierarchy is *logical* and consists of keys, with sensor node keys at the leaves, the group key at the root, and sub-group keys elsewhere. There is only one key server *KMS* (although it can be implemented in a distributed way, if necessary), there are no security agents, e.g., cluster-heads, and each sensor node is given multiple keys (the sensor key, the group key, and some sub-group keys). LARK requires neither an *a priori* secure service nor the knowledge of the sensor nodes locations or the network topology.

In the other schemes, the hierarchy is *physical* and consists of sensor nodes at the leaves and with multiple levels of security agents above. For each tree node, the tree node (an agent) and its children (sensor nodes or lower level agents) form a subgroup (cluster) and share a subgroup key. Depending on the scheme there may be a globally shared group key or not. For instance, in LiSP there is not such a key and thus join/leave in a subgroup does not affect the other subgroups [Park and Shin 2004]. Furthermore, certain schemes assumes the existence of very strong secure services or make strong assumption upon sensor deployment and topology. For instance, LEAP+ [Zhu et al. 2006] resorts to an authenticated broadcast primitive such as μTesla [Perrig et al. 2001]. That is, LEAP+ requires that at least a basic form of secure group communication is already solved by other means. Wang's et al.'s scheme uses an applicative notion of groups but it resorts to a variation of μTesla too [Wang and Ramamurthy 2007], Exclusion Basis System (EBS) proposes a combinatorial clustering that attempts to minimise the number of rekeying messages while minimising the number of keys stored by each sensor node [Eltoweissy et al. 2004; Eltoweissy et al. 2005; Younis et al. 2006]. In order to achieve this goal, EBS organises nodes into overlapping groups that are however completely unrelated to the application needs [Eltoweissy et al. 2004]. Furthermore, two EBS-based rekeying schemes require the knowledge of the positions of sensor nodes [Eltoweissy et al. 2005; Younis et al. 2006]. Implementing a location service in WSNs, especially a secure one, is quite a difficult task.

In WSNs, hierarchies of keys are used by the Topological Key Hierararchy (TKH) scheme [Son et al. 2007], and the Tree-based Group Polynomial Key Distribution (TGPKD) scheme [Choudhary et al. 2007]. TKH extends LKH by constructing the key tree hierarchy using topological information on the sensor network. Basically, TKH formalises the intuition of placing physically adjacent sensor nodes as close as possible in LKH so as to reduce communication overhead. However, TKH does not face with key authentication. TGPKD does not even address node revocation [Choudhary et al. 2007].

Key Graphs [Wong et al. 2000] and LiSP [Park and Shin 2004] are two systems that display certain similarities with LARK. Key Graphs has the same architecture as LARK and similarly uses graphs of keys to specify secure groups. Key Graphs achieves a rekeying communication overhead comparable to LARK's. However, in contrast to LARK, Key Graphs uses digital signatures to prove key authenticity which are not suitable for WSNs. As to LiSP, it proposes a periodic group rekeying and key recovery scheme that uses an inverted key chain for key authentication. Therefore, LARK and LiSP achieve a comparable performance when they authenticate newly received keys. However, the suggested LiSP's rekeying scheme incurs in a high rekeying communication overhead. Furthermore, LiSP raises security concerns. In the next sections we make a more detailed comparison between LARK and Key Graphs and LiSP, respectively.

7.1 Comparison with Key Graphs

LARK takes inspiration from Key Graphs which is perhaps the scheme it is most natural compare LARK to. Both LARK and Key Graphs use graphs of keys to specify secure groups. However, Key Graphs restricts itself to two special classes of key graphs, namely a star Key Graphs and a tree key graph. Section 6.2.3 shows that these classes represent extreme cases from the standpoint of communication overhead. In contrast, LARK pushes the concept of key graphs further and uses it as a means to specify secure groups as dictated by applicative requirements. General classes of key graphs can be specified in LARK. The actual communication overhead will depend on specific Key Graphs topology but it will be anyway constrained within $O(n)$ and $O(\log n)$.

Another difference between Key Graphs and LARK is the network environment for which each scheme has been proposed. While LARK has been proposed for a WSN, Key Graphs has been conceived for a conventional desktop/server computing network. The main impact of the network environment is in the rekeying strategy. Key Graphs defines three different rekeying strategies: user-oriented, key-oriented, and group-oriented. Details apart, in the key-oriented strategy a rekeying message carries just a new key, whereas in the other strategies a rekeying message may carry several keys. In a conventional network environment, a message is so large as to contain one or more keys. Therefore, user-/group-oriented strategies can be employed. In contrast, in a WSN, a network message is generally smaller (40 bytes in TinyOS) and thus can contain only a single key. For this reason, we have used a key-oriented rekeying strategy. Using this strategy, given a certain key graph topology, LARK and Key Graphs use the same number of rekeying messages when this strategy is used. However, the message size and the computing overhead to authenticate the key contained in a rekeying message is much greater in Key Graphs than LARK.

Yet another difference between Key Graphs and LARK is the key authentication mechanism employed. Key Graphs uses conventional digital signatures, namely RSA [Rivest et al. 1978], for authentication purposes. In LARK the use of digital signatures is possible but not advisable. With reference to [Piotrowski et al. 2006], the estimated time and power consumption for RSA-1024 signature verification on Tmote Sky are 0.22 s and 2.70 mJ, respectively, whereas for ECC-160 they are 1.02 s and 12.41 mJ. The key authentication mechanism based on inverted key-chains employed by LARK requires the computation of a hash function to verify the authenticity of a key. On a Tmote Sky, SHA-1 consumes around 0.814 μJ/byte [Mišić 2008]. Thus, in the case of LARK 128-bit keys, the time and power consumption for SHA-1 are about 3.91 ms (Table II) and 13 μJ, respectively. It follows that LARK's key authentication based on key chains is about $1 \div 2$ orders of magnitude faster than RSA-1024 and ECC-160 and consumes $2 \div 3$ orders of magnitude less energy. In addition, a digital signature constitutes an additional payload of a rekeying message. In the case of RSA-1024 bit, the digital signature is 128 bytes, whereas in the case of ECC-160, the digital signature is 40 bytes. CC2420 requires 231.42 μJ to receive an RSA-1024 digital signature and 73.32 μJ to receive an ECC-160 signature. All of this has a strong negative impact on both the network lifetime and the maximum achievable duty-cycle [Piotrowski et al. 2006].

7.2 Comparison with LiSP

Differently from LARK, LiSP uses a physical hierarchy. LiSP organises the network in clusters (groups in their parlance), each one managed by a capable node called group-head. A group-head manages a group-key to ensure backward and forward security. The group-head refreshes the group key periodically by broadcasting in the group the new key encrypted by means of the current one. The authenticity of the new group key is proven by means of an inverted key chain mechanism. However, when a sensor node leaves, the group-head unicasts the new group key to every sensor node in the group encrypted by means of the node key.

Upon periodically refreshing a group key, LiSP and LARK use the same number of rekeying messages, namely one broadcast, and have the same key authentication overhead, namely one hash function computation. However, LiSP requires $O(n)$ to rekey a group after a leave. A possibility to manage this overhead is to make small groups. However, this increases the number of groups and group heads and thus the overall scalability and cost of the system. Another possibility is to let LiSP use LARK to improve management of leaves. Every group-head could define a tree key graph where the LiSP inverted key chain is associated with the graph root. So doing, the rekeying overhead could be reduced to $O(\log n)$. A reduced overhead allows larger group size, a reduced number of groups and group-heads, and thus increased scalability and reduced costs.

In LiSP, group-heads manage their groups independently. There is no global network key that is shared across groups. Therefore, the join/leave in a group does not affect another one. However, group-heads share the same key chain in order to support inter-group communication. These design choices are justified by scalability purposes but raise security concerns.

First of all, at a given time, two groups may have different group keys. If the groups have experimented different numbers of leaves, their group-heads have picked the current group keys from different positions in the chain. Therefore, an adversary can exploit this fact to compromise a sensor node in a group, steal its group key, and reuse that key in an other group that is "behind", i.e., picks keys from positions in the key chain that are closer to the chain-head. Furthermore, if a group-head is compromised, LiSP is completely compromised because the group-head knows the key chain from which all the group keys are picked. This implies that group-heads are trusted entities that have to be protected by means strong security measures. This, together with the scalability limitations, makes the system even more expensive.

In contrast, LARK maintains a global notion of group key that is refreshed whenever any sensor node is compromised. It follows that group key management provided by LARK is more secure than LiSP. If scalability concerns raise, *KMS* can be replicated [Reiter 1996a; 1996b]. In this case, several trusted entities are needed but the trusted computing base in LiSP would be much larger than LARK as LiSP needs as many trusted entities as group-heads.

8. CONCLUSIONS

We have presented LARK an *heterogeneous* and *dynamic* group rekeying scheme [Eltoweissy et al. 2006]. LARK is heterogeneous because, being centralised, nodes of the network are assigned different roles. A key management server has the task of revoking the current key, and generating and distributing a new one, whereas sensor nodes have all the same task of verifying the new key authenticity. LARK is also dynamic as it advocates key rekeying, i.e. key revocation and redistribution, as a mean to achieve resilience to attacks in long-lived networks.

LARK supports quite a general group model where groups can be hierarchical and partially overlapping. LARK receives a a group topology that reflects the application needs and manages rekeying at single-group level. We have formally shown that LARK guarantees backward and forward security. Performance evaluation shows that LARK is scalable and efficient and thus attractive for large scale, highly dynamic WSNs comprising even low-end devices such as TMotes Sky sensor nodes. Security, scalability and efficiency of LARK hinge around the integration of key graphs and key chains. The former makes it possible to address communication scalability by reducing the number of rekeying messages whereas the latter makes key authenticity verification efficient by requiring only hash functions and symmetric ciphers.

ACKNOWLEDGMENTS

Authors also wish to thank anonymous referees whose valuable comments have made it possible to greatly improve the paper.

REFERENCES

Akyildiz, I. F. and Kasimoglu, I. H. 2004. Wireless sensor and actor network: Research challenges. *Ad Hoc Networks 2,* 4 (October), 351–367.

Anderson, R. J. 2008. *Security Engineering: A Guide to Building Dependable Distributd Systems, 2nd Edition.* Wiley Publishing Inc., Indianapolis, Indiana.

Årzén, K. H., Bicchi, A., Dini, G., Hailes, S., Johansson, K. H., Lygeros, J., and Tzes, A. 2007. A component-based approach to the design of networked control systems. *European Journal of Control 13,* 2-3, 261–279.

Bicchi, A., Danesi, A., Dini, G., La Porta, I., Pallottino, L., Savino, I. M., and Schiavi, R. 2008. A safe and secure component-based platform for heterogeneous multi-robot systems. *IEEE Robotics and Automation Magazine 15,* 1 (March), 62–70.

Cárdenas, A. A., Roosta, T., and Sastry, S. 2009. Rethinking security properties, threat models, and the design space in sensor networks: A case study in SCADA systems. *Ad Hoc Networks 7,* 8 (November), 1434–1447.

Caronni, G., Waldvogel, M., Sun, D., Weiler, N., and Plattner, B. 1999. The VersaKey framework: Versatile group key management. *IEEE Journal on Selected Areas in Communications 17,* 9 (Sept.), 1614–1631.

Chan, H., Gligor, V. D., Perrig, A., and Muralidharan, G. 2005. On the distribution and revocation of cryptographic keys in sensor networks. *IEEE Transactions on Dependable and Secure Computing 2,* 3 (July-September), 233–247.

Chipcon. 2004. CC2420 Datasheet. `http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf`.

Choudhary, D., Anshul, D., Roy, S., and Thejaswi, C. 2007. Computationally and resource efficient group key agreement for ad hoc sensor networks. In *Proceedings of the 2nd IEEE International Conference on Communication Systems Software and Middleware (COMSWARE 2007)*. IEEE, Bangalore, India, 1–10.

Cole, E. 2009. *Network Security Bible, 2nd Edition*. Wiley Publishing Inc., Indianapolis, Indiana.

Coppersmith, D. and Jakobsson, M. 2002. Almost optimal hash sequence traversal. In *Financial Cryptography*. Lecture Notes in Computer Science, Vol. 2357. Springer-Verlag, Southampton, Bermuda, 102–119.

Costa, P., Coulson, G., Mascolo, C., Picco, G. P., and Zachariadis, S. 2005. The RUNES middleware: a reconfigurable component-based approach to networked embedded systems. In *Proc. of the 16th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'05)*. Vol. 2. IEEE, Berlin, Germany, 806–810.

Deng, J., Han, R., and Mishra, S. 2006. Secure code distribution in dynamically programmable wireless sensor networks. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN'06)*. ACM, Nashville, Tennessee (USA), 292–300.

Dermibas, M. 2005. Wireless sensor networks for monitoring of large public buildings. Tech. rep., University at Buffalo (`www.cse.buffalo.edu/tech-reports/2005-26.pdf`).

Dini, G., Porta, I. S. L., Savino, I. M., Hailes, S., Gold, R., and Ahmed, M. 2008. Secure reconfiguration in networked embedded systems with the runes approach. In *Demo at the 5th European Conference on Wireless Sensor Networks (EWSN'08)*. Lecture Notes in Computer Science, Vol. 4913. Springer-Verlag, Bologna, 1–2.

Dini, G. and Savino, I. 2006. S^2RP: a secure and scalable rekeying protocol for wireless sensor network. In *Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 06)*. IEEE, Vancouver, Canada, 457–466.

Dini, G. and Savino, I. M. 2010. A Security Architecture for Reconfigurable Networked Embedded Systems. *International Journal of Wireless Information Networks 17,* 1 (June), 11–25.

Dobbertin, H., Bosselaers, H., and Preenel, B. 1996. Ripemd-160, a strengthened version of ripemd. In *Proceedings of the 3rd International Workshop on Fast Software Encryption (FSE'96)*. Lecture Notes in Computer Science, Vol. 1039. Springer-Verlag, Cambridge, UK, 71–82.

Dutta, P. K., Hui, J. W., Chu, D. C., and Culler, D. E. 2006. Securing the deluge network programming system. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*. ACM, Nashville, Tennessee (USA), 326–333.

Eltoweissy, M., Heydari, M. H., Morales, L., and Sudborough, I. H. 2004. Combinatorial optimization of group management. *Journal of Network and Systems Management 12,* 1 (March), 33–50.

Eltoweissy, M., Moharrun, M., and Mukkala, R. 2006. Dynamic key management in sensor networks. *IEEE Communication Magazine 44,* 4 (April), 122–130.

Eltoweissy, M., Wadaa, A., Olariu, S., and Wilson, L. 2005. Group key management for large-scale sensor networks. *Ad-Hoc Nwtworks 3,* 5 (September), 668–688.

Eschenauer, L. and Gligor, V. D. 2003. A key-management scheme for distributed sensor networks. In *Proceedings of 10th ACM Conference on Computer and Communication Security (CCS'03)*. ACM, Washington D.C., USA, 41–57.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. E., and K., P. 2000. System Architecture Directions for Networked Sensors. In *Proceedings of the 9th Symposium on Architectural Support to Programming Languages and Operating Systems (ASPLOS'00)*. ACM, Cambridge, Massachusetts, United States, 93–104.

Intanagonwiwat, C., Govindan, R., and Estrin, D. 2000. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of ACM Mobile Computing and Networking (Mobicom'00)*. ACM, Boston, MA, USA, 56–67.

Karlof, C., N., S., and Wagner, D. 2004. Tinysec: A link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*. ACM, Baltimore, MD, United States, 162–175.

L., R. R. 1994. The RC5 encryption algorithm. In *Proceedings of the 2nd International Workshop on fast Software Encryption*, B. Preenel, Ed. Vol. LNCS 1008. Springer-Verlag, Leuven, Belgium, 86–96.

Lamport, L. 1981. Password authentication with insecure communication. *Communications of the ACM 24,* 11 (November), 770–772.

Law, Y. W., Doumen, J., and Hartel, P. 2006. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks 2,* 1 (February), 65–93.

Liu, A., P.Kampanakis, and Ning, P. 2007. Tinyecc: elliptic curve cryptography for sensor networks (version 0.3). `http:discovery.csc.nvsu.edu/software/TinyECC/`.

Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J. 2002. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*. ACM, Atlanta, Georgia (USA), 88–97.

Menezes, A. J., van Oorschot, P. C., and A., V. S. 1996. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida.

Mišić, J. 2008. Traffic and energy consumption of an IEEE 802.15.4 network in the presence of authenticated, ECC Diffie-Hellman ephemeral key exchange. *Computer Networks 52,* 11, 2227–2236.

Moteiv. Tmote sky. `http://www.moteiv.com/`.

Mottola, L. and Picco, G. P. 2006a. "Logical Neighborhood: A Programming Abstraction for Wireless Sensor Networks". In *Proceedings of the 2nd International Conference on Distributed Computing in Sensor Systems (DCOSS'06)*. Lecture Notes in Computer Science, Vol. 4026. Springer-Verlag, San Francisco, CA, USA, 150–168.

Mottola, L. and Picco, G. P. 2006b. Programming wireless sensor networks with logical neighborhoods. In *Proceedings of the 1st International Conference in Integrated Internet Ad Hoc and Sensor Networks, (InterSense 2006)*. IEEE, Nice, France, 1–6.

National Institute of Standards and Technology. 1995. *FIPS PUB 180-1: Secure Hash Standard*. National Institute of Standards and Technology.

National Institute of Standards and Technology. 1998. *SKIPJACK and KEA Algorithm Specifications*. National Institute of Standards and Technology.

National Institute of Standards and technology. 2001. *NIST FIPS PUB 197 Specification for the Advanced Encryption standard (AES)*. National Institute of Standards and technology.

National Institute of Standards and Technology. 2005. *Plan for new cryptographic hash functions*. National Institute of Standards and Technology.

Park, T. and Shin, K. 2004. LiSP: A lightweight security protocol for wireless sensor networks. *ACM Transactions on Embedded Computing Systems 3,* 3 (August), 634 – 660.

Perrig, A., Szewczyk, R., Wen, V., Culler, D., and Tygar, J. D. 2001. SPINS: Security suite for sensor networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM'01)*. ACM Press, New York, 189–199.

Petriu, E., Georganas, N., Petriu, D., Makrakis, D., and Groza, V. 2000. Sensor-based information appliances. *IEEE Instrumentation and Measurement Magazine 3,* 4 (December), 31–35.

Piotrowski, K., Langendörfer, P., and Peter, S. 2006. How public key cryptography influences wireless sensor node lifetime. In *Proceedings of the 4th ACM Workshop on Security of ad hoc and Sensor Networks, (SASN 2006)*. ACM, Alexandria, VA, USA, 169–176.

Polastre, J., Szewczyk, R., and Culler, D. 2005. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*. ACM, Los Angeles, California, USA, Poster Session.

Rafaeli, S. and Hutchison, D. 2003. A Survey of Key Management for Secure Group Communication. *ACM Computing Surveys 35,* 3 (September), 309–329.

Ravi, S., Raghunathan, A., Kocher, P., and S., H. 2004. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems 3,* 3 (August), 461–491.

Reiter, M. K. 1996a. Distributing trust with the rampart toolkit. *CACM: Communications of the ACM 39,* 4, 71–74.

Reiter, M. K. 1996b. A secure group membership protocol. *IEEE Transactions on Software Engineering 22,* 1 (Jan.), 31–42.

Rivest, R. 1992. The MD5 message-digest algorithm. Internet Request for Comment RFC 1321, Internet Engineering Task Force. April.

Rivest, R., Shamir, A., and Adleman, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21,* 2 (February), 120.

Roman, R., Alcaraz, C., and Lopez, J. 2007. A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. *Mobile Network & Applications 12,* 4 (August), 231–244.

Roman, R., Zhou, J., and Lopez, J. 2006. Applying intrusion detection systems to wireless sensor networks. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC'06)*. IEEE, Las Vegas, USA, 640–644.

Sherman, A. T. and McGrew, D. A. 2003. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering 29,* 5 (May), 444–458.

Sinopoli, B., Sharp, C., Schenato, L., Schaffert, S., and Sastry, S. S. 2003. Distributed Control Applications Within Sensor Networks. *Proceedings of the IEEE 91,* 8 (August), 1235–1246.

Son, J., Lee, J., and Seo, S. 2007. Energy efficient group key management scheme for wireless sensor networks. In *Proceedings of the 2nd IEEE International Conference on Communication Systems Software and Middleware (COMSWARE 2007)*. IEEE, Bangalore, India, 1–9.

WALDVOGEL, M., CARONNI, G., SUN, D., WEILER, N., AND PLATTNER, B. 1999. The versakey framework: Versatile group key management. *IEEE Journal on Selected Areas of Communications (Special Issue on Middleware) 17,* 9 (August), 1614–1631.

WALLNER, D. M., HARDER, E. G., AND AGEE, R. C. 1999. Key management for multicast: issues and architecture. RFC 2627, IETF.

WANG, X., YAO, A., AND YAO, F. 2005. New collisions search sha-1. In *Rump session of the 25th Annual International Cryptology Conference (CRYPTO 2005).* Springer-Verlag, Santa Barbara, CA, USA.

WANG, Y. AND RAMAMURTHY, B. 2007. Group rekeying schemes for secure group communication in wireless sensor networks. In *IEEE International Conference on Communications (ICC '07).* IEEE, Glasgow, Scotland, 3419–3424.

WANG, Y., WANG, X., XIE, B., WANG, D., AND AGRAWAL, D. P. 2008. Intrusion detection in homogeneous and heterogeneous wireless sensor networks. *IEEE Transactions on Mobile Computing 7,* 6 (June), 698–711.

WONG, C. K., GOUDA, M., AND LAM, S. S. 2000. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking 8,* 1 (February), 16–30.

YOUNIS, M., GHUMMAN, K., AND ELTOWEISSY, M. 2006. Location-aware combinatorial key management scheme for clustered sensor networks. *IEEE Transactions on Parallel and Distributed Systems 17,* 8 (August), 865–882.

YOUNIS, M. F., GHUMMAN, K., AND ELTOWEISSY, M. 2005. Key management in wireless ad hoc networks: collusion analysis and prevention. In *Proceedings of the 24th IEEE International Performance Computing and Communications Conference.* IEEE, Phoenix, Arizona, USA, 199–203.

YOUNIS, O., KRUNZ, M., AND RAMASUBRAMANIAN, S. 2006. Node clustering in wireless sensor networks: Recent developments and deployment challenges. *IEEE Network 20,* 3 (May/June), 20–25.

ZHANG, Q., YU, T., AND NING, P. 2008. A Framework for Identifying Compromised Nodes in Wireless Sensor Networks. *ACM Transactions on Information and System Security 11,* 3 (March), 1–37.

ZHOU, X., RAMAMURTHY, B., AND MAGLIVERAS, S. 2005. *Secure group communication over data networks.* Springer, New York, NY, USA.

ZHU, S., SETIA, S., AND JAJODIA, S. 2006. LEAP+: Efficient Security Mechanims for Large-Scale Distributed Sensor Networks. *ACM Transactions on Sensor Networks 2,* 4 (November), 500–528.

Received XXX; YYY; accepted ZZZ