for wireless sensor networks

Omar Cheikhrouhou, Anis Koubâa, Gianluca Dini, Hani Alzaid , Mohamed Abid

ABSTRACT

Secure group communication is a paradigm that primarily designates one-to-many communication security. The proposed works relevant to secure group communication have predominantly considered the whole network as being a single group managed by a central powerful node capable of supporting heavy communication, computation and storage cost. However, a typical Wireless Sensor Network (WSN) may contain several groups, and each one is maintained by a sensor node (the group controller) with constrained resources. Moreover, the previously proposed schemes require a multicast routing support to deliver the rekeying messages. Nevertheless, multicast routing can incur heavy storage and communication overheads in the case of a wireless sensor network. Due to these two major limitations, we have reckoned it necessary to propose a new secure group communication with a lightweight rekeying process. Our proposal overcomes the two limitations mentioned above, and can be applied to a homogeneous WSN with resource-constrained nodes with no need for a multicast routing support. Actually, the analysis and simulation results have clearly demonstrated that our scheme outperforms the previous well-known solutions.

Keywords:
Secure group communication (SGC)
Wireless sensor networks (WSNs)
Group key management
Simulation
Analysis

1. Introduction

Wireless sensor networks are generally deployed in a wide area for the purpose of monitoring some environmental parameters, such as lightweight, humidity, temperature, and pressure. In some applications, different security levels and services are often required for each type of information. For this purpose, the network would be divided into several groups, each of which is responsible for reporting certain specific data. Inside each single group, a one-to-many communication type might be needed, for example, when the group controller sends commands/queries to all group members. Moreover, there are some applications in which in-network processing is needed, for instance, to reduce the communication cost. As a matter of fact, in-network processing enables to send aggregated information instead of large streams of raw data. For this aim, nodes need to communicate among themselves in order to produce such a type of aggregated information.

In some critical applications, e.g. healthcare, group communication has to be protected. Generally, secure group communication could be achieved through the use of a group key shared among the group members. The group key can be managed in either a centralized manner, a distributed manner, or a contributed manner [1,2]. In the centralized approach, the group controller generates the group key and delivers it to all the group members. In the distributed approach, the group is divided into smaller subgroups each of which is managed locally. However, as each subgroup has its different key, this approach presents the limits of translating the encrypted data when forwarded from one subgroup to another. In other words, encrypted messages need to be decrypted whenever they reach another subgroup in order to make use of the transmitted data if in-network processing is available. Therefore, it may introduce a computation overhead. As for the contributed approach, each group member contributes in the computation of the group key which leads to a heavy complexity and communication cost.

Taking into account the limitations of the distributed and contributed approach, many schemes have opted to the centralized approach. The trivial solution to deliver the group key, in the centralized approach, is to send it by unicast to each group member. However, this solution requires an $O(n)$ communication cost complexity, where n is the number of group members. So, numerous schemes have been proposed for the sake of reducing the communication, computation and storage cost needed for the rekeying process [3–7].

However, the majority of the proposed schemes assume that the group controller is a powerful node such as a PC or a base station. Actually, they consider the whole network as being a single group managed by the base station. However, this presents a restrictive definition of a group and a network model. As a matter of fact, in some applications several groups managed by resource-constrained sensor nodes might be needed. In addition, the proposed schemes such as [3–6] suppose that the network has a multicast routing support to deliver the rekeying messages in a multicast way. However, multicast routing introduces a heavy overhead and communication cost for maintaining the multicast table and the multicast routes. Moreover, the rekeying messages used in the majority of previous schemes present some security vulnerabilities. For example, some schemes such as LKH [3], and TKH [5], are not protected against replay attacks. Hence, an attacker can replay a revealed group key in order to intercept the communication. Revoked nodes can also replay an old group key in order to rejoin the group. In addition, some schemes [3,5,6] do not consider rekeying messages authentication which makes them vulnerable to messages injection attacks.

As a matter of fact, the previous schemes' limitations, described above, have given us ground to conceive a new secure group communication scheme. The strengths of the proposed scheme are that it can be applied to a group with a resource-constrained group controller, does not require a multicast routing support and is robust against possible attacks.

1.1. Contribution

This paper proposes a secure group communication scheme that allows sensor nodes belonging to the same group to communicate securely. The proposed scheme is composed of two main components: the group membership management and the group key management. The group membership management component defines in a secure manner the group creation, the group join and the group leave processes. The group key management component presents a lightweight method to update the group key after each membership change for the sake of guaranteeing forward and backward secrecy properties. The proposed method is based on the construction of a logical neighbor tree that helps to share the task of rekeying messages distribution among the group members. Therefore, our scheme eliminates the necessity of a powerful group controller and so that can fit resource-constrained sensor nodes.

The idea of secure group communication with resource-constrained group controller was first addressed by Cheikhrouhou et al. in their paper RiSeG [8,9]. Yet, the proposed scheme presents an $O(n)$ latency in the rekeying process and, therefore, cannot endure large scale WSNs. Moreover, RiSeG requires synchronization between nodes in order to avoid replay attacks. However, synchronization is a hard task to be achieved in a WSN [10].

As an improvement, a Logical Neighbor Tree (LNT) is proposed in order to distribute the key update messages. Actually, the LNT scheme enables to reduce the key update latency from $O(n)$ to $O(log(n))$. In addition, LNT eliminates the requirement of node synchronization.

In summary, LNT is distinguished by the following features:

- The concept of group is application-based, i.e., groups reflect the application needs.
- The group controller could be a sensor node with constrained resources.
- The LNT scheme proposes the membership management and the group key management in an efficient and secure way.

1.2. Roadmap

The remainder of this paper is organized as follows: Section 2 presents relevant work to secure group communication in WSNs. Section 3 presents the network model, the adversarial model, requirements and assumptions used in our work. Section 4 gives an overview of the LNT scheme and its design principles. Sections 5–7 describe the LNT membership management, the logical neighbor tree formation and the LNT rekeying process, respectively. Section 8 presents a comparison between the LNT rekeying process and other well-known group key management schemes namely, LKH [3], TKH [5], and RiSeG [8]. Section 9 presents a security analysis of the LNT scheme. Finally, we conclude and give possible future work.

2. Related work

In the literature, there are a few works that detail a complete secure group communication schemes with its different components. The proposed SGC schemes in the recent decade are [11,12,8,9].

In [11], the authors proposed SLIMCAST: a secure level key infrastructure for multicast to protect data confidentiality via hop-by-hop re-encryption and mitigate the DoS-based flooding attack through an intrusion detection and deletion mechanism. The SLIMCAST protocol divides a group routing tree into levels and branches in a clustered way. Communications among nodes at each level of each branch of the group tree are protected by a level key in such a way that only the local level key is updated during a joining or a leaving process. The scheme presents a low communication overheads and power consumption and also is scalable. However, the performance is degraded (i.e., high power consumption) when membership changes are massive [2].

In [12], the authors proposed SeGCom a secure group communications mechanism for cluster tree wireless sensor networks. The scheme uses Blundo technique [13] in order to share a pairwise key between each pair of nodes. However, in order to broadcast the group controller identity, the scheme uses μTESLA, which requires synchronization between nodes. Moreover, the scheme did not explain how the authentication process is done and it presents an $O(n)$ communication overhead.

The RiSeG (Ring based Secure Group communication) [9,8] scheme has addressed the secure group communication problem with a constrained group controller. The idea of the scheme is to divide the group controller task among group members by constructing a logical ring architecture. This logical ring permits to deliver the rekeying messages as follows. The group controller sends the rekeying message to the next hop in the ring and then the message will be forwarded from a node to another until the message returns back to the group controller. The scheme reduces the communication, computation and storage cost at the group controller, but it introduces a big latency that cannot scale with a large group. Indeed, the latency of the rekeying process is $O(n)$.

The main concern of a secure group communication scheme is the group key management. Several group key management systems for WSNs have been proposed [3,4,14,5,6,15–19].

The Logical Key Hierarchy (LKH) [3] permits to reduce the number of rekeying message from $O(n)$ to $O(log(n))$. The idea of the LKH scheme is based on dividing the group into subgroups and then generating a key encryption key (*KEK*) for each subgroup. This *KEK* will then serve to securely deliver the group key to the appropriate subgroup. The group controller maintains a logical key hierarchy in which it associates a group member to each leaf node. The root of this hierarchy is the group key, the internal nodes represent the *KEKs*, and the leaves are the symmetric key shared with the corresponding group member. Each group member maintains the keys lying on the path from the leaf to the root. The *KEK* allows to replace several unicast rekeying messages by a single multicast one. This technique enables to reduce the number of rekeying messages from $O(n)$ to $O(log(n))$. However, the LKH scheme introduces additional computation and storage cost, especially at the group controller.

Many improvements [4–6,14] were designed to the LKH scheme. The OKD (One-way Key Derivation) [4] scheme further reduces the number of rekeying messages by means of a local computation of the group key, using a one-way hash function. In fact, in a join operation current group members compute the new key as $k' = H(k)$, where H is a one-way hash function and k is the current key. Therefore, the group controller needs to send the new key only to the new member, and thus, the number of rekeying message is reduced to a single message in the join operation. For the leave operation, the number of the rekeying messages remains $O(log(n))$.

The S2RP (Secure and Scalable Rekeying Protocol) [14] permits to further secure the rekeying process by authenticating the rekeying messages using a one-way-hash function as well as a key-chain. As LKH, the S2RP scheme constructs a logical key hierarchy, in which each internal node has a separate key-chain. The key-chain is extracted using a One-Way-Hash-Function (H), so that each element in the key-chain is the image of the next one under H. The chain keys are then revealed in the reverse order of their creation. Therefore, on receiving a key, the group members would check its authenticity by verifying that $k = H(k')$, where k' is the new key and k is the current key. However, this technique compromises the properties of backward security as a new joining member might reveal the previous keys by simply computing the hash of the newly-received key, repeatedly. This has led the authors to introduce an additional key called *join-key* (K_J). This key is renewed whenever a new node joins the group, and the group key is a mixture of the current key-chain at the root node in the tree and this K_J. In order to minimize the rekeying message, the new key K_J is locally renewed by computing $K'_J = h(K_J)$. The scheme performance is similar to that of the LKH. The difference is that in S2RP, the node must store, along with the current key of internal nodes, the key K_J as well as the key K_C (to authenticate command). The node has also to store more than one hash function (He, Hj, Hc). Moreover, the group controller has a heavy storage cost as it must store the key-chain (instead of a single key in the LKH scheme case) of each internal node. The CSET (Computation and Storage Efficient Tree) [6] scheme permits to improve the LKH efficiency by considering the available sensor nodes resources when constructing the logical tree. The TKH (Topological Key Hierarchy) [5] scheme allows to reduce the communication cost of the rekeying messages delivery by mapping the logical key tree to the physical topology. The idea is to construct a key tree that reflects the physical topology of the network. However, TKH does not face with group key authentication.

LARK is the first group key management system that supports a group topology defined by applications [15]. In LARK, sensor nodes are logically grouped according to application needs and not for network management purposes. The resulting group topology model is quite general and encompasses multiple groups, organized hierarchically, and possibly overlapping. Such a group model is particularly appropriate for Wireless Sensor and Actuator Networks (WSANs), a new paradigm that is asserting itself in the networked embedded system world [20]. LARK features a centralized group key management service and owes its efficiency to a novel integration of two well-known basic mechanisms: key chains [21], for efficient key authentication, and key graphs [3], for efficient key distribution. However, LARK assumes both a resource-rich group manager and multicast routing, as necessary.

In LEAP (Localized Encryption and Authentication Protocol)[16], the authors proposed a key management protocol for sensor networks that are designed to support in-network processing, while at the same time restricting the security impact of a node compromise to the immediate network neighborhood of the compromised node. LEAP endures the establishment of four types of keys for each sensor node – an individual key shared with the base station, a pairwise key shared with another sensor node, a cluster key shared with multiple neighboring nodes, and a global key shared by all the nodes in the network. For the update of the global key LEAP assumes the use of a routing protocol in which the nodes are organized into a spanning tree. However, this assumption limits the deployment of the scheme. Moreover, the scheme rests on the *μTESLA* scheme [22] for the broadcast authentication. The *μTESLA* assumes synchronization between nodes, which is a hard task to achieve in WSN [10].

Exclusion Basis System (EBS) proposed a combinatorial clustering that attempts to minimize the number of rekeying messages while minimizing the number of keys stored by each sensor node [17,23,24]. In order to achieve this goal, EBS organizes nodes into overlapping groups that are, however, completely unrelated to the application needs [17]. Furthermore, two EBS-based rekeying schemes require the knowledge of the positions of sensor nodes [23,19]. Implementing a location service in WSNs, especially a secure one, is quite a difficult task [15].

3. Network model and security requirements

This section is devoted to the presentation of the network model to which the proposed secure group communication scheme is applied as along with the adversarial model and requirements.

3.1. Network model

It is worth noting that a wireless sensor network maintained by a base station is considered in this study. As for the information within the network, it is routed using a routing protocol such as Ad hoc On Demand Distance Vector routing algorithm (AODV) [25] or Dynamic Source Routing algorithm (DSR) [26]. In addition, the following types of nodes have also been considered:

- The Base Station (BS): is responsible for securing the whole network. It maintains a table containing the group controller addresses corresponding to each group. The base station is secure and able to detect all compromised nodes. Detection of compromised nodes can be actually achieved by means of an Intrusion Detection System (IDS) such as [27–30]. The BS maintains a *blacklist* containing the identity of compromised nodes. These nodes will not be allowed to join any group in the future and are, therefore, excluded from the network.
- The Group Controller (GC): is a node responsible for maintaining the security inside its group. To note, no security property has been assumed for the GC.
- The End Device (ED): is a node which belongs to one or multiple groups. For each group, it maintains the group controller, the parent node and the child nodes' addresses.

We also assume that each group has a unique group identifier, which represents the sensory information corresponding to this group. These group identifiers are known to all nodes. This can be done by loading the group identifiers to nodes at the deployment phase.

3.2. Adversarial model

For the adversarial model, we assume that an attacker can eavesdrop, modify, replay, and inject messages. This is due to the wireless medium. The attacker can also compromise a node. So the following attacks can be launched in our WSNs:

- Eavesdropping: an attacker may eavesdrop communication inside a group. A solution to mitigate this attack is to encrypt communication.
- Impersonation attack: an attacker may impersonate a legal user to gain access to a group. A solution to mitigate this attack is to authenticate node and message' source.
- Replay attack: an attacker may replay an old message to gain access to a group or to disturb the operation of a group. A solution to mitigate this attack is to add a sequence number indicating the freshness of the message.
- Injecting false message: an attacker may inject false message to disturb the operation inside a group or to eavesdrop encrypted data.
- Compromise attack: an attacker may compromise a sensor node. After compromising the node an attacker may reveal secret key and all information stored on that sensor.

3.3. Requirements

In what follows, the requirements to be achieved by a secure group communication scheme are presented:

- Nodes belonging to the same group must communicate securely and their exchanged information must not be revealed to non-member nodes even if they belong to the same network.
- A node may belong to more than one group. However, it must store a per-group profile containing the GC address, the group key, the group identifier, etc.
- Compromised nodes must be ejected from the group as soon as they are detected.
- Multi-hop communication paradigm is employed between nodes in order to deliver encrypted messages. Note that intermediate nodes forward encrypted messages without knowing their content.
- Both backward and forward secrecy must be achieved. Backward secrecy means that a node joining the group must not reveal previous exchanged information. Forward secrecy means that a node leaving the group must not reveal future exchanged information.

Table 1
Notations.

Notations	Meaning
Ni	Sensor node with identity *Ni*
Gid	Group identity
$MAC(m,K)$	Message Authentication Code (MAC) computed over the message *m* using the key *k*. The message *m* refers to current message' fields except the MAC (ex; in join-request, *m* = *Ni*, *Gid*, $nonce_{Ni}$)
Nonce	Is a random number used once. It permits to ensure replay attack protection
$K_{Ni,\ Nj}$	Pairwise key shared between node *Ni* and node *Nj*
$\{m\}_K$	Message *m* encrypted with key *K*

Table 2
Computation cost parameters.

Notations	Meaning
C_{enc}	The computation cost needed to compute the encryption of the group key
C_{dec}	The computation cost needed to compute a decryption operation on the group key
C_{ekg}	The computation cost needed to generate an elliptic curve private/public keys
C_{sign}	The computation cost needed to generate a signature
C_{verif}	The computation cost needed to verify a signature
C_{kg}	The computation cost needed to generate a group key
C_{kc}	The computation cost needed to compute a pairwise key
C_{mac}	The computation cost needed to compute a message authentication code (MAC)

Table 3
Communication cost parameters.

Notations	Meaning
$\|m\|$	The size in bits of the message *m*
mjr	Join-request message
mlr	Leave-request message
mjk	Join-key message
mji	Join-inform message
mci	Group-creation-invite message
mcd	Group creation decision message i.e. Grp-creation-accept or Grp-creation-refuse message
mku	Key-update message
mnd	Neighbor discovery message
mcu	Child update message
mpu	Parent update message
mpr	Parent request message
e_{tx}	The energy consumed for the transmission of 1 bit
e_{rx}	The energy consumed for the reception of 1 bit

Table 4
Execution time parameters.

Parameter	Signification
T_{enc}	Time needed to encrypt the group key
T_{dec}	Time needed to decrypt the group key
T_{sign}	Time needed to generate a signature
T_{verif}	Time needed to verify a signature
T_{kg}	Time needed to generate a group key
T_{kc}	Time needed to compute the pairwise key
T_{mac}	Time needed to compute a MAC
T_{tx}	Time needed for the transmission of 1 bit
T_{rx}	Time needed for the reception of 1 bit
T_{nd}	Time needed by the neighbor discovery process

- Security parameters' maintenance such as the rekeying process must be lightweight and effective.

4. LNT overview and design principles

The idea of the LNT scheme is based on distributing the rekeying process task among group members. Indeed, as the group controller is a sensor node with constrained resources, it cannot support the delivery of rekeying messages to all group members, nor can it stand the heavy storage cost introduced by the key encryption keys of the LKH-based schemes. Therefore, we propose to construct a logical neighbor tree that helps deliver the rekeying messages inside the group.

The logical neighbor tree represents the neighborhood relationship among the group members. The root of the tree represents the group controller which is connected to its neighbor nodes, and each nodes of these neighbor nodes is connected to its neighbor nodes and so on until covering all group members. The whole tree is maintained by the group controller, and each group member maintains only the list of its child nodes. On receiving a key update message from its parent node, the current node forwards this key update message to its child nodes. Therefore, the key update message sent by the group controller will be forwarded from parent to child nodes until reaching all group members. In this way, the group controller should send key update messages only to its neighbors instead of sending it to all the group members.

The key update message contains the new group key that must be kept exclusively secret among the group members. Consequently, this message has to be protected against potential interception attacks. This is achieved by means of encryption. In addition, for the sake of preventing a malicious node from injecting false key update messages, the group controller computes a digital signature which is verified by the group members. The validity of the signature proves that the key update message is indeed sent by the group controller and not by a malicious node. As for the replay attack robustness, it is achieved by means of a sequence number shared among the group members.

To sum it up, LNT helps to eliminate the heavy storage cost introduced by the key encryption keys used in the LKH-based schemes and, simultaneously, provides both authentication and robustness against replay attacks of the rekeying messages.

5. LNT membership management

This section, is allocated to the presentation of the different membership processes of the LNT scheme, specifically, the group creation, the group join and the group leave processes. It is worth noting that these processes must be executed in a secure way so as to avoid any possible illegal access to the network (during the group creation and the group join processes) as well as the denial-of-service (DoS) attack against the network (e.g. by sending fake leave-requests during the group leave process).

Table 5
LNT parameters.

Notations	Meaning
d	Average number of the tree degree (corresponds to the average number of child node)
h	The tree height (corresponds to the number of levels, the root is at level 0)
n	Number of group members ($n = \left(\frac{1-d^{h+1}}{1-d}\right)$ for a balanced tree)
hop	Average number of hops between two group members
b	Average number of neighbor nodes belonging to the same group of a node
Pi	The selected parent of the joining node

After describing each process, we present the analytical performance study. The chosen performance metrics are the computation cost, the communication cost and the execution time.

The computation cost can be measured in terms of time, use of CPU or energy dissipation. In fact, these parameters are related and each one can be deduced from the other. For instance, the energy dissipation can be deduced from the time as follows: *Energy* = *Power* × *Time*, where *Power* represents the CPU power when it is in its active state and *Time* represents the computing time. In the present analysis, the term cost is used in its general form and we have not specified the unit (which can be second, Joule or number of CPU cycles). The computation cost of each membership process can be computed as the sum of the computation cost of the main operations executed during this membership process. The main operations required in the LNT scheme are presented in Table 2 and they are namely: the encryption/decryption operation, the signature generation/verification operation, the generation of a key and the MAC operation.

The main factor of the communication cost is the energy dissipation. The communication cost is computed using the same approach as TKH [5]. Actually, the communication cost in terms of energy dissipation is computed as the size of sent/received messages multiplied by the energy dissipated for the sent/reception of one bit. The different messages used in LNT scheme are presented in Table 3.

The execution time is the time needed for the execution of the overall process: time between the beginning of the membership operation and its end. The different parameters related to the execution time are explained Table 4.

The different notations used throughout this section are, respectively, explained in Tables 1–5.

5.1. Pre-deployment phase

As in [9], we propose to apply the key pre-distribution scheme proposed by Blundo et al. [13] in order to share a symmetric key between each pair of nodes. The network administrator chooses a t degree bi-variate polynomial over a finite field F_q: $f(x,y) = \sum_{i=0}^{i=t}\sum_{j=0}^{j=t} a_{i,j}x^iy^j$. The value of q is a prime number that is large enough to accommodate a cryptographic key. Then, the administrator loads in each node Ni the polynomial $f(x,Ni)$. The function f is symmetric. This means that, when two nodes Ni and Nj wish to share a pairwise key, each of them computes $K_{Ni,Nj} = f(Ni,Nj) = f(Nj,Ni)$.

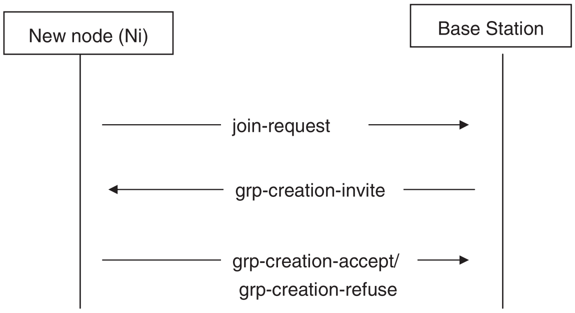
Moreover, to ensure rekeying message authentication, a digital signature scheme was used. The Elliptic Curve Digital Signature Algorithm (ECDSA) [31] was chosen as its feasibility in WSNs [32,33] was demonstrated, and that it is more suitable than other signature schemes (such as RSA) in the context of WSNs, due to the use of small key size (reduce the storage cost) [34–36]. Each node is preloaded with the domain parameters needed to compute and verify the ECDSA [31]. The domain parameters are the six-tuple $T = (p,a,b,G,r,h)$, where p is a prime number, a and b are two points from the primary field F_p $(a,b \in F_p)$ defining the curve, G a base point on the curve with order r and cofactor f.

5.2. Group creation process

5.2.1. Description

The group creation process is executed when a node requests to join a non-existent group. In this case, and after having successfully authenticated the node, the Base Station (BS) would invite it to be the Group Controller (GC) of the requested group. The group creation process is depicted in Fig. 1 and is executed according to the following steps:

1. The node sends a *join-request* message. This message contains the node identity Ni, the requested group identity Gid, and a random number used only once $nonce_{Ni}$. This message is protected by a Message Authentication Code (MAC) field computed over the message and using the pairwise key $K_{BS,Ni}$ shared between the node Ni and the BS.
2. Upon receiving this message, the BS verifies the validity of the node, i.e., that the node does not belong to the black list of compromised nodes. Then the BS checks the message authenticity by comparing the received MAC to the locally computed one using the pairwise key $K_{BS,Ni}$ shared with this node Ni. If the node's identity is valid, and the message is authentic and the requested group does not exist yet, the BS sends a *grp-creation-invite* message containing the node nonce $nonce_{Ni}$, and a new generated one $nonce_{BS}$. This message enables to authenticate the BS.
3. After authenticating the BS by verifying the MAC field, the node has to decide whether to be a group controller or not. If it does, it sends a *grp-creation-accept* message. If it does not, the node sends a *grp-creation-refuse* message. Note that, a node might refuse to become a GC, for instance, if it does not have enough resources to achieve the GC task. Both messages contain the received nonce $nonce_{BS}$ and are protected by a MAC. This permits to avoid any potential replay attack and guarantees the node authentication.
4. Upon receiving the node response the BS verifies the validity of the received message. In case the BS receives a *grp-creation-refuse* message, it ignores the node request and the group is not created. If the BS receives a *grp-creation-accept* message a new group is being



Ni→BS: join-request = Ni, Gid, nonce_{Ni}, MAC(m,$K_{BS,Ni}$)
BS→Ni: grp-creation-invite= Gid, nonce_{Ni}, nonce_{BS}, MAC(m,$K_{BS,Ni}$)
Ni→BS: grp-creation-accept= Ni, Gid, nonce_{BS}, MAC(m,$K_{BS,Ni}$)
or : grp-creation-refuse= Ni, Gid, nonce_{BS}, MAC(m,$K_{BS,Ni}$).

Fig. 1. Group creation.

created and the BS memorizes the group controller address of the newly created group *Gid*. Moreover, the GC and the BS agree on a specific sequence number *seqNbr*. This sequence number is incremented at each message sent and permits to avoid replay attacks, as will be explained later.

Noteworthy, the GC maintains two sequence numbers. A *seqNbr* shared with the BS and another *seqNbr* shared with the group members used for communication inside the group. The *seqNbr* shared with the BS is incremented for each communication with the BS (e.g. *join-inform* message) and the *seqNbr* shared among the group members is updated for any group-related communication (e.g. *key-update* message).

Moreover, in order to sign subsequent key-update messages the group controller needs a public/private key. For this purpose, the *GC* selects a random integer d_G in the interval $[1, r-1]$ and then computes $Q_G = d_G \times G$. The tuple (d_G, Q_G) respectively represents the GC's private and public keys.

5.2.2. Performance evaluation

The group creation process involves three categories of nodes, namely:

- The joining node (*Ni*): is the node that sends a *join-request* message to the BS and becomes a GC if it accepts.
- The base station (BS): interacts and authenticates the requesting node.
- The intermediate nodes (IN): are nodes located on the routing path between the joining node and the BS. Their task is to route messages exchanged between the BS and *Ni*.

Computation cost. The computation cost of the LNT scheme during the group creation process is calculated, in this section, at both the joining node (*Ni*) and the base station (*BS*). This calculation is as follows:

- *Ni*: Computation cost = $C_{kc} + 3C_{mac} + C_{ekg}$
- BS: Computation cost = $C_{kc} + 3C_{mac}$

The joining node, needs to compute the pairwise key shared with the base station using the Blundo technique (C_{kc}), three MACs ($3C_{mac}$) and an elliptic curve private/public keys (C_{ekg}). The BS computes also the pairwise key $K_{BS,Ni}$, which requires (C_{kc}) along with three MACs ($3C_{mac}$). Note that, *Ni* and BS have the same computation cost, this is due to the fact that each computed MAC must be recomputed at the receiver in order to be verified.

Communication cost. The communication cost of the LNT scheme group creation process is calculated, in this section, at the joining node, the base station, and the intermediate node. This calculation is as follows:

- *Ni*: $|mjr|e_{tx} + |mci|e_{rx} + |mcd|e_{tx}$
- BS: $|mjr|e_{rx} + |mci|e_{tx} + |mcd|e_{rx}$
- IN: $(|mjr| + |mci| + |mcd|)(e_{tx} + e_{rx})$

An intermediate node (*IN*) would forward each message exchanged between the node *Ni* and the BS, so that the communication cost is equivalent to the size of exchanged messages multiplied by $(e_{tx} + e_{rx})$.

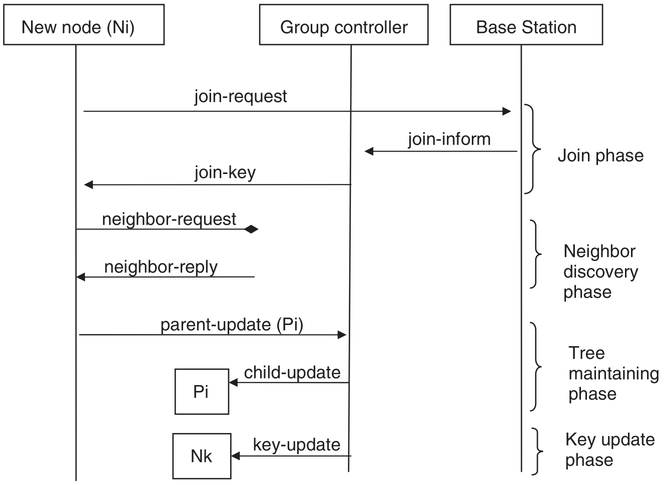
Execution time. The execution time of the group creation process is:

$Tgc = 2T_{kc} + 6T_{mac} + (|mjr| + |mci| + |mcd|).hop_{BS,Ni}.T_{tx}$, where $hop_{BS,Ni}$ is the number of hops between Node *Ni* and the BS.

5.3. Group join process

5.3.1. Description

Once the group is created, the BS informs the GC about each subsequent *join-request* after successfully authenticating the requesting node. The requesting node authentication is achieved through the BS as it holds a global overview of the compromised nodes. Consequently, if a node fails to successfully pass the authentication process, its request will be ignored.



---Join phase---

Ni→BS: join-request= Ni, Gid, $nonce_{Ni}$, MAC(m, $K_{BS,Ni}$)

BS→GC: join-inform= Ni, Gid, seqNbr, MAC(m, $K_{BS,GC}$)

GC→Ni: join-key= Gid, seqNbr, $\{GK'\}_{K_{GC,Ni}}$, MAC(m, $K_{GC,Ni}$)

---Tree maintain phase---

Ni→*: neighbor-discovery-request= Gid, hopCount, hop

Nj→Ni: neighbor-discovery-reply=Nj,physicalhop,logicalhop,childNbr (∀Nj neighbor of Ni)

Ni→GC: parent-update= Gid, Pi

GC→Pi: child-update= Gid, Ni, MAC(m, $K_{GC,Pi}$)

---Key update phase---

GC→Nk: key-update= Gid, seqNbr, $\{GK'\}_{GK}$,sign
(for all Nk in Child nodes list of GC)

Fig. 2. Group join.

The group join process is depicted in Fig. 2 and is executed as follows:

1. The node *Ni* sends a *join-request* message.
2. Upon receiving this message, the BS verifies the node authenticity. If the node is successfully authenticated, the BS informs the GC of the requested group about the join of the new node *Ni*. In order to prevent a compromised node already evicted from the group from replaying an old *join-inform* message, we protect this message with a *seqNbr*.
3. On receiving the *join-inform* message, the GC verifies its freshness based on the *seqNbr* before generating a new group key which it transmits to the node *Ni*.
4. Upon receiving the *join-key* message, the node *Ni* launches a neighbor discovery process. For this purpose, *Ni* broadcasts a *neighbor-discovery-request* message containing the group identity (*Gid*). Then, each neighbor node *Nj* belonging to the group *Gid* responds to *Ni* by sending a *neighbor-discovery-reply* message. The *neighbor-discovery-reply* message contains the number of child nodes attached to *Nj* and the number of hops from *Nj* to the GC (i.e how far the node *Nj* is from the GC).
5. After that, the newly joined node *Ni* chooses the appropriate parent from the list of responding neighbors. The node *Ni* then informs the GC about its selected parent by sending a *parent-update* message. The neighbor discovery and parent selection processes are more explained in Section 5.
6. When the GC receives a *parent-update* message from *Ni*, it first checks the validity of the node *Pi*. Then, the GC asks the selected parent node to add *Ni* to its child list. The GC also updates the local tree.

Notes:

- The join process must go through the BS which maintains the list of the compromised nodes. These nodes must be eliminated from the network.
- In case the selected parent node is invalid, the GC requests the node *Ni* to search for another parent.
- If a node has no neighbor member of the requested group, the GC takes the responsibility to send the *key-update* message to the newly joining node. So, the GC would add this node to its child in the logical tree. The neighbor discovery process could also be extended to more than one-hop neighbor. So far, if a node does not find a one-hop neighbor at first, it can look for a 2-hop neighbor, 3-hop neighbor, etc. until finding the nearest node that can be its parent. This extension of the neighbor discovery process minimizes the number of child node attached to the GC in case the group

members are not close to each other. The search of a *t*-hop neighbor ($t > 1$) can be implemented by the broadcast of a *neighbor-discovery-request* message containing a *hop* field initialized to *t* and decremented by each intermediate node before re-broadcasting it. This neighbor discovery message is transmitted from neighbor to neighbor until the value of *hop* reaches 0.

- The replay attack on the*join-request* message was not addressed as there is no interest for an attacker to replay this message (absence of risk). In fact, suppose an attacker intercepts an old *join-request* message, and then he replays it. Two cases are possible:
  1. Either the victim node is still within the group: in this case, the *GC* would ignore the request,
  2. or the victim node is leaving the group: in this case, even if the attacker succeeds in passing the routing protocol, the received message will be encrypted by the pairwise key shared between the *GC* and *Ni*. As a result, the attacker can neither reveal information of the group, nor send the message in the group.

5.3.2. Performance evaluation

The group join process involves four categories of node: The joining node (*Ni*), the base station (BS), the group controller (GC), and the *Ni* parent node (Pi). The GC is responsible for maintaining the group members, the group key and the LNT structure.

Computation cost. The computation cost of the LNT scheme during the group join process is calculated, in this section, for the four categories of nodes mentioned above; Ni, BS, GC, and Pi. This calculation is as follows:

- *Ni*: computation cost = $2C_{kc} + 2C_{mac} + C_{dec}$
- BS: computation cost = $2C_{kc} + 2C_{mac}$
- GC: computation cost = $3C_{kc} + 3C_{mac} + C_{kg} + C_{enc}$
- *Pi*: computation cost = $C_{kc} + C_{mac}$

The joining node, needs to compute two pairwise keys: the one shared with the base station and the one shared with the GC ($2C_{kc}$), two MACs ($2C_{mac}$), and a decryption (C_{dec}). The GC interacts first with BS to receive the join-inform message, and then with the joining node and sends to it the encrypted group key, and then with the parent of the joining node to maintain the tree, and finally with its children to send them the *key-update* message.

Communication cost. The communication cost of the LNT scheme group join process is calculated, in this section, for Ni, BS, GC, and Pi. This calculation is as follows:

- *Ni*: $|mjr|e_{tx} + |mjk|e_{rx} + |mnd|e_{tx} + b|mnd|e_{rx} + |mpu|e_{tx}$
- BS: $|mjr|e_{rx} + |mji|e_{tx}$
- GC: $|mji|e_{rx} + |mjk|e_{tx} + |mpu|e_{rx} + |mcu|e_{tx}$
- *Pi*: $|mnd|e_{rx} + |mnd|e_{tx} + |mcu|e_{rx}$

The GC receives a *join-inform* from the BS, sends a *join-key* to *Ni*, receives a *parent-update* message from the *Ni*, sends a *child-update* to Pi and then sends a *key-update* to its children.

Execution time. It refers to time occurring between the sending of the *join-request* by *Ni* and the receiving of the *child-update* message by its parent.

The execution time of the group join process is:

$Tgj = (T_{kc} + T_{mac} + |mjr|.hop_{BS,Ni}.\ T_{tx})_{Ni} + (T_{kc} + T_{mac} + T_{kc} + T_{mac} + |mji|.hop_{BS,GC}.\ T_{tx})_{BS} + (T_{kc} + T_{mac} + T_{kc} + T_{mac} + T_{kg} + T_{enc} + |mjk|.hop_{GC,\ Ni}.T_{tx})_{GC} + (T_{kc} + T_{mac} + T_{dec} + T_{nd} + |mpu|.hop_{GC,Ni}.\ T_{tx})_{Ni} + (T_{kc} + T_{mac} + |mcu|.hop_{GC,Pi}.T_{tx})_{GC}$

$Tgj = 7T_{kc} + 7T_{mac} + T_{kg} + T_{enc} + T_{dec} + T_{nd} + (|mjr|.hop_{BS,\ Ni} + |mji|.hop_{BS,GC} + |mjk|.hop_{GC,Ni} + |mpu|.hop_{GC,Ni} + |mcu|.hop_{GC,\ Pi}).T_{tx}$, where $hop_{X,Y}$ is the number of hops on the path between Node *X* and Node *Y*.

5.4. Group leave process

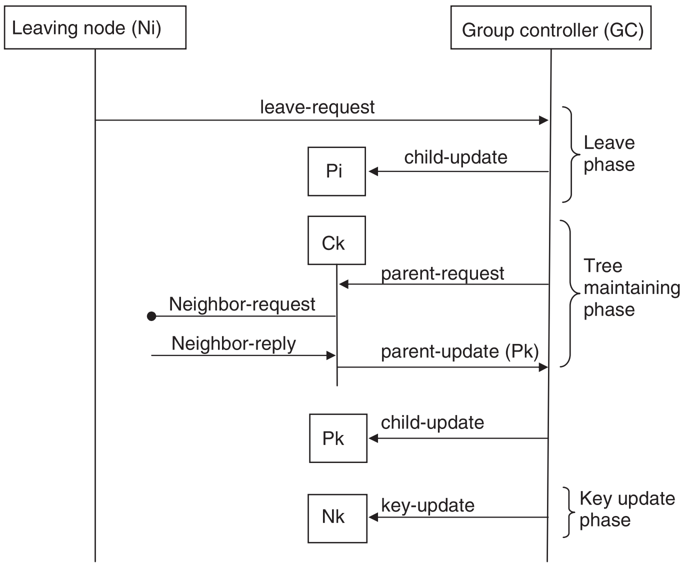
5.4.1. Description

A node can decide to leave the group, for instance, when its mission is over. In this case, the node explicitly leaves the group by sending a leave-request message. A node can also be forced to leave, for instance, if it is detected as compromised. In both cases, the GC must maintain the LNT tree and update the group key to achieve forward secrecy. Fig. 3 presents an explicit leave scenario.

1. The leaving node sends to the GC a *leave-request* message containing its identity, the group identity and protected by a sequence number and a MAC.
2. On receiving this message, the GC verifies the authenticity of the message. Then, the GC sends a *child-update* message to *Ni*'s parent in order to remove *Ni* from its child list. The *child-update* message is also protected by a MAC. This MAC prevents attackers from launching a DoS attack by sending fake *child-update* messages that cause the removal of group members.
3. In order to maintain the logical neighbor tree, for each *Ni*'s child node, the GC sends a *parent-request* message that invites the node to search for another parent.
4. The tree maintenance continues idem to the tree maintenance in the group join process: After selecting a new parent, each child node *Ck* sends a *parent-update* message to the GC, which in turn asks the selected parent to update its child node list. The GC updates also the local tree.
5. Finally, the GC launches the key-update process to renew the group key. The evicted node will not receive this new key as it is evicted from the logical tree.

Notes:

- The logical neighbor tree must be saved at the GC as it needs to know this tree structure in the case of a node compromise. In fact, suppose that the GC would like to revoke the node *Ni*, the GC must, in this case, eliminate the node *Ni* from the LNT. To this end, the GC would tell *Ni*'s parent to remove *Ni* from its child list and tell *Ni*'s child to find another parent. After that, the GC would update the group key, which would not reach the compromised node *Ni* as it is ejected from the tree.
- We assume that each sent message is acknowledged. So, after a number of attempts, if the sender does not succeed to receive an acknowledgment message, it will suppose that the link is no more available. Therefore, if a node fails or leaves the group without a notice, child nodes will detect that the communication with their



---Leave phase---

Ni→GC: leave-request = Ni, Gid, seqNbr, MAC(m, $K_{BS,\ Ni}$)

GC→Pi: child-update=remove, Gid, Ni, MAC(m, $K_{GC,Pi}$)

---Tree maintain phase---

GC→Ck: parent-request

(for all Ck in Child nodes list of Ni)

Ck→GC: parent-update=Pk

GC→Pk: child-update=add, Gid, Nk, MAC(m, $K_{GC,Pk}$)

---Key update phase---

GC→Nk: key-update= Gid, seqNbr, $\{GK'\}_{K_{GC,Nk}}$,sign

(for all Nk in Child nodes list of GC)

Fig. 3. Group leave.

parent is no longer available and so, will re-find another parent. Therefore, the leaving without a notice is similar to an explicit leaving.

- If the GC fails, one solution is that: the first child node that detect the GC failure will send a message to the BS informing it about GC failure. Then, the BS will invite this node to be the new GC. After that, the BS sends a broadcast message to inform group members about the address of the new GC. Then, every group member will send to this new GC a parent update message informing it about the address of its parent. Thanks to these parent update messages, the new GC can construct the LNT tree.
- With reference to step 1, in the case a node is suspected to be compromised and thus has to be evicted, it is the base station that sends the leave-request message to the group controller. The message is formatted as specified in step 1. Its authenticity is guaranteed by attaching a MAC computed by means of the key shared between the BS and the GC.
- It is worthwhile to notice that the logical tree is only used to disseminate rekeying messages. Logical tree management messages (e.g. child-update and parent-update messages) are instead routed by the chosen routing protocol (Section 3.1). Therefore, particular attention is necessary if the leaving/evicted node lies on the route from the GC to its parent and/or children (steps 2–3).

In the case of a leaving node this is not a problem at all. A leaving node leaves the group because its mission is over. However, it is genuine and therefore correctly cooperates towards the application and the system services (i.e, routing, key management, et cetera). Thus, the leaving node first routes child-update and parent-update messages, as needed, so contributing to establishing a new logical tree, and then actually leaves.

In the case of an evicted node things are more complicated. An evicted node is a node that the IDS has suspected as compromised. Thus we cannot rely on its cooperation. In particular, we cannot rely on its cooperation in routing child-update and parent-update messages. This means that the presence of a compromised node may cause a partition in the network and, consequently, in the logical tree. However, this form of DoS is a fundamental problem whose solution pertains to the routing service rather than the key management service. If we wish to avoid that the presence of a compromised node might cause a network partition then we have to adopt techniques such as redundant deployment, secure and multipath routing [37].

Of course, LNT gives its contribution to avoid DoS but at its layer. Actually, by guaranteeing authenticity and freshness of both tree and key management messages, LNT avoids that an adversary may cause any harm by transmitting fake ones. Furthermore, provided that the underlying routing service provides enough connectivity upon a node's leaving, LNT is able to establish a logical tree of genuine nodes that correctly forward key management messages upon rekeying.

5.4.2. Performance evaluation

The group leave process involves the following categories of node: the leaving node (*Ni*): is the node that sends a leave-request message to the GC, the group controller (GC), the intermediate node (IN), the *Ni*'s parent node (*Pi*), the *Ni*'s child nodes (*Ci*).

Computation cost. The computation cost of the LNT scheme during the group leave process is calculated, in this section, for three types of nodes: Ni, GC, and Pi. This calculation is as follows:

- *Ni*: computation cost = $C_{kc} + C_{mac}$
- GC: computation cost = $2C_{kc} + 2C_{mac}$
- *Pi*: computation cost = $C_{kc} + C_{mac}$

Communication cost. The communication cost of the LNT scheme group leave process is calculated, in this section, for three types of nodes: Ni, GC, and Pi. This calculation is as follows:

- *Ni*: $|mlr|e_{tx}$ (or 0 in case of a silently leaving)
- GC: $|mlr|e_{rx} + |mcu|e_{tx} + d|mpu|e_{rx}$
- *Pi*: $|mcu|e_{rx} + |mlr|e_{rx} + |mlr|e_{tx}$
- *Ci*: $|mpr|e_{rx} + |mnd|e_{tx} + b|mnd|e_{rx} + |mpu|e_{tx}$

A child node of the leaving node receives a parent-request message ($|mpr|e_{rx}$), send a neighbor-discovery message ($|mnd|e_{tx}$), receives b neighbor-reply ($b|mnd|e_{rx}$) and sends a parent-update message ($|mpu|e_{tx}$).

Execution time. The execution time of the group leave process is:

$Tgl = (T_{kc} + T_{mac} + |mlr|.hop_{GC,Ni}.T_{tx}) + (T_{kc} + T_{mac} + T_{kc} + T_{mac} + |mcu|.hop_{GC,Pi}.T_{tx}) + d.(|mpr|.hop_{GC,Ck}.T_{tx} + T_{nd} + |mpu|.hop_{GC,Ck}.T_{tx} + T_{kc} + T_{mac} + |mcu|.hop_{GC,Pk}.T_{tx})$.

$Tgl = (3 + d)T_{kc} + (3 + d)T_{mac} + (|mlr|.hop_{GC,Ni} + hop_{GC,Pi}. |mcu| + d.hop_{GC,Ck}.|mpr| + d.hop_{GC,Ck}.|mpu| + d.hop_{GC,Pk}.|mcu|).T_{tx}$.

6. LNT tree management

This section is devoted to an exemplified illustration of the LNT tree construction and update when a node joins or leaves the group. Consider the following physical network topology presented in Fig. 4. For simplicity, we have presented a network with one group, but which can be generalized to multiple groups with several group controllers.

The corresponding LNT tree maintained at the GC is presented in Fig. 5. This tree has been progressively created as a new node joins the group.

6.1. Tree update due to a join

As described in Section 5.3, after the join of a new node, the GC maintains the LNT tree based on the selected parent returned by the joining node.

The joining node sends a *neighbor-discovery-request* message to discover neighbors which are members of group Gid. The *neighbor-discovery-request* contains a parameter *hopCount* which specifies that the joining node is looking for neighbors that are at maximum *hopCount* far from the GC and a parameter *hop* that specifies how many times the *neighbor-discovery-request* will be forwarded. Each member of group *Gid* receiving this request responds by sending a *neighbor-discovery-reply* message. Only nodes that have a lower *hopCount* away from the GC respond. This guarantees that only nodes on the path from the joining node to the GC could be a parent of the joining node. If the node is not a member of the requested group, it will decrement the value of *hop*and forward the *neighbor-discovery-request*message if the value of *hop* is still greater than zero. This mechanism of forwarding the *neighbor-discovery-request*message permits to search for neighbors farther than one-hop.

The *neighbor-discovery-reply* message contains parameters that reflect the position of the node in the physical topology and the logical tree namely the *physicalhop*, the *logicalhop*, and the *childNbr*. The *physicalhop* indicates the distance in terms of hops between the GC and the node in the physical topology, the *logicalhop* represents the number of hops between the GC and the node in the logical tree, and the *childNbr* represents the number of nodes attached to the node. If the joining node does not receive any reply from its immediate neighbors, it will search for a two-hop neighbor. For this reason, it will increment the value of *hop* and decrement the value of *hopCount*. This process will be repeated until receiving a reply. On receiving *neighbor-discovery-reply* messages, the joining node selects a neighbor as a parent. The selection of the parent node is based on first the *physicalhop*, then the *logicalhop* and then the *childNbr*. Therefore, if the joining node receives replies from more than one neighbor with the same *physicalhop* then it selects the node with the smallest *logicalhop*. If more than one neighbor has the same *physicalhop* and *logicalhop*, the joining node selects the one with the small *childNbr*. Otherwise, if all the mentioned parameters are equal, then the choice can be based on the RSSI (Received Signal Strength Indication) or arbitrary. The *physicalhop* parameter promotes a tree with a minimum transmission cost. The *logicalhop* and *childNbr* parameters promote a balanced tree which speeds up the rekeying process. The joining node sends then a *parent-update* message to the GC. The latter informs the parent to add the joining node to its child list. The GC also updates the logical tree. The joining node broadcasts a neighbor reply message informing nodes that there is a new path to the GC with the indicated cost. Neighbor nodes that receive this message can then update their parents by sending a *parent-update* message to the GC.

Let us assume, for example, that node 23 wants to join the group as shown in Fig. 4. After its join, node 23 discovers its neighbors. For this reason, it broadcasts a *neighbor*-

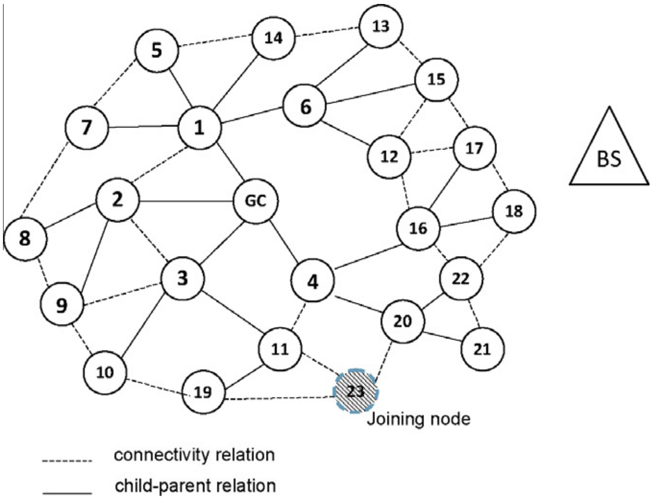


Fig. 4. Physical network topology.

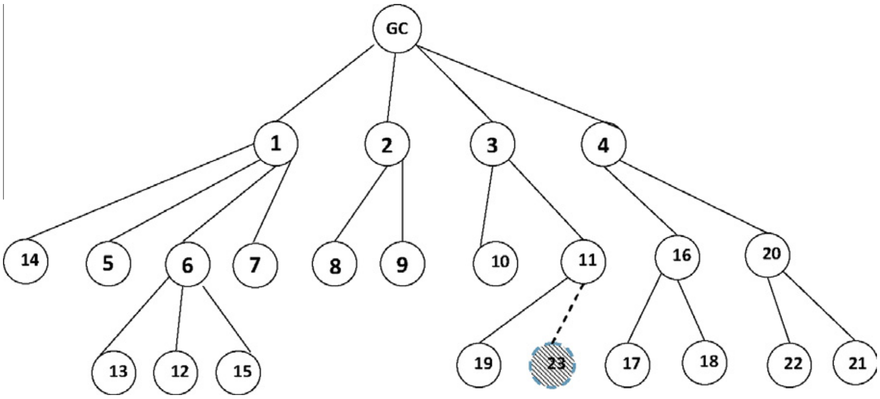


Fig. 5. LNT tree construction.

discovery-request message that contains the group identity *Gid*, with *hopCount* = 3 and *hop* initially equal to 1. Each node belonging to the group *Gid* and hearing the *neighbor-discovery-request* message replies with a *neighbor-discovery-reply* message. This message contains the node identity, the *physicalhop* (number of hops between the node and the GC in the physical topology), the *logicalhop* (number of hops between the node and the GC in the logical tree) and the *childNbr* (the number of child nodes attached to this node).

In our case, node 23 has three neighbors {19,11,20}, which would respond respectively with (19,3,3,0), (11,2,2,1), and (20,2,2,2). First node 23 selects node 11 and node 20 as they present the smallest *physicalhop* value. Then node 23 views the *logicalhop* value. As node 11 and node 20 have the same *logicalhop* value, the joining node (node 11) views the third parameter with is the *childNbr* value. As node 11 has the smallest *childNbr* (1), it would be selected by node 23 as the parent node. Node 23 then informs the group controller by sending a *parent-update* message containing the address of node 11. Upon receiving this message, the GC informs node 11 to add node 23 to its child list.

6.2. Tree update due to a leave

The LNT tree has two types of nodes: leaf nodes, which are nodes without any attached child node, and parent nodes, which are nodes with one or more children. Leaf nodes are those located at the bottom of the tree, while the parent nodes are intermediate nodes between a GC and leaf nodes.

The leaving of a leaf node requires simply the send of a child-update message to the parent node. Let us assume, for example, that node 15 in Fig. 4 wants to leave the group. In this case, the GC sends a *child-update* message to node 6, which is node 15's parent, to remove node 15 from its child node list. In this way, node 15 is no longer member of the group.

The leaving of a parent node is more complicated than the leaving of a leaf node as, for a parent node, its child nodes (descendant nodes) must be attached to another parent. Instead of node 15, let us assume that node 6 wants to leave the group. The update of the LNT tree can be launched in two ways. Either, node 6 informs child nodes {12,13} about its leave, or node 6 informs the GC about its leave, and the latter requests the child nodes to re-send

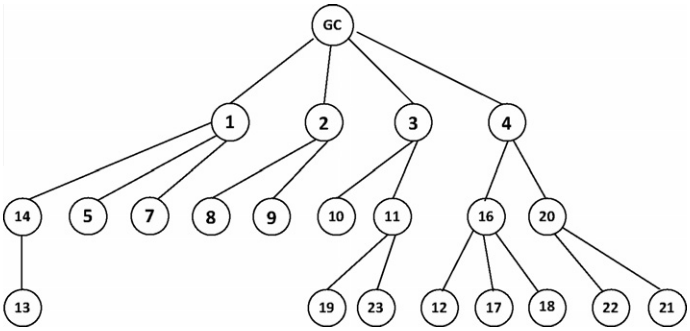


Fig. 6. LNT tree update.

an updated address of their parents. In both cases, each child node should re-find another parent using the neighbor discovery process and then inform the GC. Therefore, after the leaving of node 6, associated child nodes {12, 13} renew their parents as follows. Node 12 with neighbor members {16, 17} selects node 16 as parent, and node 13 with neighbor {14} selects node 14 as a parent.

The new LNT resulting from the join of node 23 and the leaving of node 15 and node 6 is presented in Fig. 6.

Note that, if node 6 leaves silently (e.g. due to energy exhaustion), child nodes will detect that the link to parent (node 6) is no more available, and so, they search for a new parent.

7. LNT rekeying process

The previously described tree will be used by the group controller in order to update the group key. More precisely, this tree is used in the distribution of the rekeying messages.

7.1. Description

After each membership change, the GC updates the group key. For this purpose, the GC generates a new group key and then delivers it to the group members.

To protect the key-update message from replay attack a sequence number was added. The signature avoids an attacker to inject a false key update on behalf of the GC. Moreover, to protect the group key from eavesdropping, the *GC* protects it by means of encryption.

In the case of a join operation, the GC uses the current group key to encrypt the new one, and then sends the *key-update* message to its child nodes. Upon receiving the key update message, a non-leaf node first verifies the signature and then forwards it, in its turn, to its child nodes. The key update process continues until reaching all leaf nodes in the tree. Therefore, in the case of a join operation, the key-update message has the following format: $mku = Gid, seqNbr, \{GK'\}_{GK}, sign$.

In a leave operation, the leaving node knows the current group key and, therefore, it is necessary to use pairwise keys for encryption so that the GC sends a separate key-update message to each child node by unicast. Upon receiving the key update message, a non-leaf node first verifies the signature and then decrypts the new group key and then re-encrypts it separately for each child node. Therefore, in the case of a leave operation, the key-update message has the following format: $mku = Gid, seqNbr, \{GK'\}_{PWK}, sign$, where *PWK* is the pairwise key shared between a parent and its child node.

The signature (*sign*) is computed over the *Gid*, *seqNbr* and *GK* fields so that these fields are authenticated and protected from modification.

The GC with (d_G, Q_G) as its private and public keys respectively, generates the signature using the ECDSA as follows:

1. Selects a random integer k from interval $[1; r-1]$;
2. Computes $kG = (x1; y1)$ and $s1 = x1 \; mod \; n$. If $s1 = 0$ go to step 1;
3. Computes $k^{-1} \; mod \; n$;
4. Computes $h(mku)$, where h is a hash function e.g. the Secure Hash Algorithm (SHA-1).
5. Computes $s2 = k^{-1}(h(mku) + d_G.s1) \; \text{mod} n$. If $s2 = 0$ go to step 1.

$sign = (s1; s2)$ is the GC signature of message *mku*.

A group member receiving this signature verifies it using the ECDSA as follows:

1. Verifies that $s1$ and $s2$ are integers in $[1; r-1]$;
2. Computes $h(m)$;
3. Computes $w = s2^{-1} \bmod n$;
4. Computes $u_1 = h(mku).w \bmod n$ and $u_2 = s1.w \bmod n$;
5. Computes $u_1 G + u_2 Q_G = (x1; y1)$;
6. Computes $v = x1 \bmod n$;

The group member accepts the signature if and only if $v = s1$.

7.2. Performance evaluation

In this section, we present the analytical performance of the proposed schemes during the rekeying process. The performance evaluation criteria are the computation cost, the communication cost, the storage cost and the duration. In what follows we present the analytical performance of the rekeying process of our proposed LNT scheme.

7.2.1. Computation cost

The computation cost depends on the membership operation and is as follows:

Case of join.
- Group controller: $C_{kg} + C_{enc} + C_{sign}$
- End device: $C_{dec} + C_{verif}$

Case of leave.
- Group controller: $C_{kg} + d(C_{kc} + C_{enc}) + C_{sign}$
- End device: $C_{kc} + C_{verif} + C_{dec} + d(C_{kc} + C_{enc})$

In case of join, the group controller needs to generate a new group key (C_{kg}), encrypts it using current group key (C_{enc}), signs the message (C_{sign}) and then sends it. An end device receiving this message verifies the signature (C_{verif}) and then decrypts the group key (C_{dec}).

Regarding the leave case, the group controller encrypts the new group key separately using pairwise keys shared with its child nodes ($d(C_{kc} + C_{enc})$). An end device needs to verify the signature (C_{verif}), decrypt the message (C_{dec}), and then forward it to its child nodes ($d(C_{kc} + C_{enc})$).

7.2.2. Communication cost

Every node sends d messages and receives 1 message (communication cost is $O(1)$). More specifically, single node communication cost is: $|mku| \cdot e_{rx} + d \cdot |mku| \cdot e_{tx} = |mku| \cdot (e_{rx} + d \cdot e_{tx})$. Please refer to Table 3 for parameters definition.

7.2.3. Storage cost

The storage cost is computed as the number of bytes that the sensor node (group controller or group member) has to store. Generally, this storage cost is introduced by the storage of different parameters and keys necessary to the operation of the LNT scheme. The proposed secure group communication scheme does not require much memory overhead. In fact, due to Blundo et al.'s key distribution technique, each sensor node has to store a polynomial function which occupies $(t + 1)logq$ storage space, where t stands for the degree of the polynomial and $log(q)$ represents the size of the keys [38]. Moreover, each member has to store the ECC domain parameters $T = (p, a, b, G, r, h)$ [31,39].

In addition, a group member has to store the group key, the address of the parent and child nodes, as well as the GC address and public key Q_G for each group it belongs to. The GC also stores the members' addresses that belong to its group and the pair of private/public key (d_G, Q_G). As for the base station, it has to store the GC address corresponding to each group as well as the *black list* containing the list of compromised nodes. The following equations summarize the storage cost at each entity, for a group of n nodes:

- The group controller has to store:
  - The neighbor tree = $n \cdot sizeof(ID)$
  - The Blundo polynomial share = $(t + 1) \cdot log(q)$, where t and q are defined in Table 6.
  - A Private/public key.
- The end device has to store:
  - The neighbors' addresses = $d \cdot sizeof(ID)$.
  - The Blundo polynomial share = $(t + 1) \cdot log(q)$.
  - The public key of the group controller.

Table 6
Blundo parameters.

Notations	Meaning
t	Is the degree of the polynomial (t reflects the scheme security: the scheme is t-collusion)
q	Is the large prime number, so large to accommodate a cryptographic key (q reflects the key size). NB: the size of q= key size = the size of the polynomial coefficients

7.2.4. Duration

The key update duration is the time needed to renew a group key and it corresponds to the time elapsed between the beginning of the key update process and its termination. The duration is composed of the processing time, the transmission time and the propagation time. The processing time includes the time for generating the key, the time for encrypting and deciphering the key, etc.

Case of join. Key update duration is $T_{kuj} = T_{kg} + T_{sign} + T_{enc} + d \cdot hop \cdot |mku| \cdot T_{tx} + (h - 1) \cdot (T_{dec} + T_{verif} + d \cdot hop \cdot |mku| \cdot T_{tx})$.

The *key-update* process begins with the generation of a group key (T_{kg}), then the GC encrypts it (T_{enc}), signs the message (T_{sign}) and sends it to its child nodes ($d \cdot hop \cdot |mku| \cdot T_{tx}$), where *hop* is the average number of physical hops between a parent and its child node. A group member receiving this *key-update* message, first decrypts the group key (T_{dec}), then verifies the signature (T_{verif}) and finally forwards the message to its d child nodes ($d \cdot hop \cdot |mku| \cdot T_{tx}$). Considering a tree with level h (the GC is at level 0 and non-leaf group members are at level 1 to level $h - 1$), the key-update message received by the last GC' child node needs $(h - 1) \cdot (T_{dec} + T_{verif} + d \cdot hop \cdot |mku| \cdot T_{tx})$ to reach the last child nodes in the logical tree. More precisely, $(h - 1)$ is the number of level in the tree containing non-leaf group members.

Case of leave. Key update duration is

$$\begin{aligned} T_{kul} &= T_{kg} + T_{sign} + d \cdot (T_{kc} + T_{enc}) + d \cdot hop \cdot |mku| \cdot T_{tx} \\ &\quad + (h - 1) \cdot (T_{kc} + T_{dec} + T_{verif} + d \cdot (T_{kc} + T_{enc}) \\ &\quad + d \cdot hop \cdot |mku| \cdot T_{tx}) \\ &= T_{kg} + T_{sign} + (h - 1) \cdot (T_{kc} + T_{dec} + T_{verif}) \\ &\quad + h \cdot d \cdot (T_{kc} + T_{enc} + hop \cdot |mku| \cdot T_{tx}) \end{aligned}$$

In case of a group leave, the encryption of the group key is done using pairwise keys so that a group member must re-encrypts the received group key separately to each d child nodes ($d \cdot (T_{kc} + T_{enc})$).

8. Performance comparison

In order to highlight the performance of the LNT scheme, we compared its performance against some well-known existing schemes in the literature. As the majority of works address only the group key management problem, comparison was made with some well-known group key management schemes such as LKH, TKH and RiSeG rekeying processes.

LKH [3] is a well-known and basic scheme on which several group rekeying schemes are based [40,4,41,14,15, 5,6]. TKH [5] is a group rekeying scheme that uses the concept of matching the physical topology to the logical key hierarchy. RiSeG [9,8] is a secure group communication that uses the concept of logical ring topology.

In what follows, we present the rekeying scheme and its performance evaluation.

8.1. Logical Key Hierarchy (LKH)

In the LKH group rekeying scheme, the group controller maintains a logical key tree. The root of the key tree is the group key and it is shared by all group members. The leaf nodes of the key tree are individual keys shared only between the individual group members and the group controller. The middle level keys are auxiliary key encryption keys used to facilitate the distribution of the group key [3].

In this scheme, each group member maintains all the keys on the path from its individual leaf node to the root of the key tree. As a result, when a user leaves or joins the group, in order to maintain forward and backward data confidentiality, all those keys have to be changed and redistributed.

To illustrate the LKH scheme, we consider in Fig. 7 a tree with height $h = 4$ and degree $d = 4$ (number of nodes is d^h). In case of a joining/leaving, the group controller renews all the keys on the path of the joining/leaving node to the root, and then, delivers them to the appropriate nodes. Thus, when a node joins/leaves the group, the GC needs to change h keys, including the group key (K_g).

The LKH scheme proposes three strategies to deliver these new keys as follows:

User-Oriented Rekeying. In user-oriented rekeying the group controller constructs a rekey message that contains precisely the new keys needed by the user (group member) and encrypts them using a key held by the user. As an illustration, we take the example of the joining (respectively the leaving) of Node $N1114$ presented in Fig. 7. The following keys have to be updated $\{kg, K1, K11, K111\}$. For this purpose, the group controller must send the messages presented in Fig. 8a (respectively Fig. 9a).

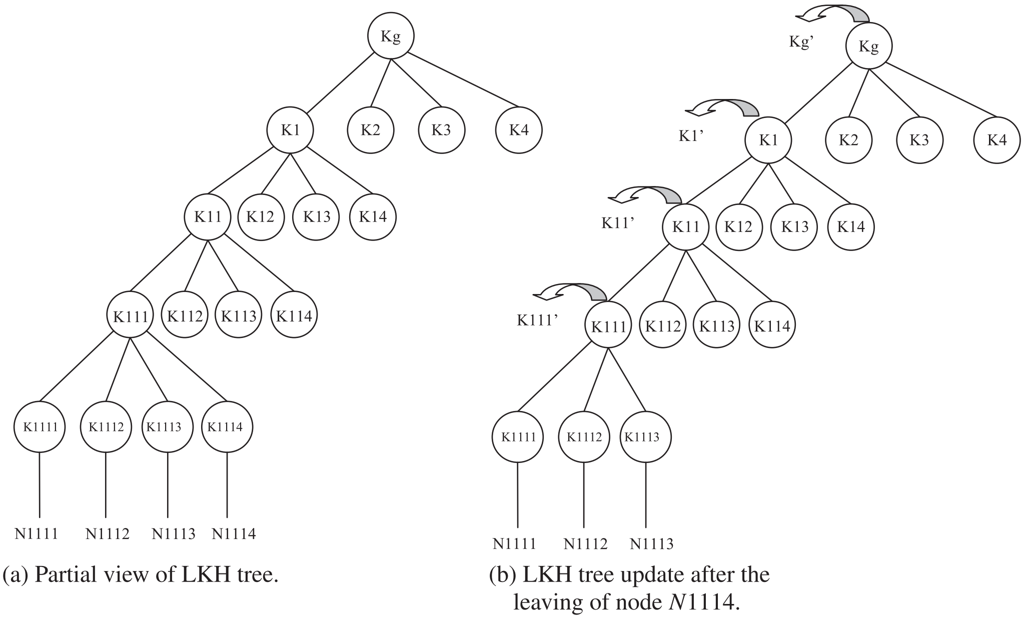
Key-Oriented Rekeying. In this approach, each new key is encrypted individually. Considering the joining (respectively the leaving) of Node $N1114$ presented in Fig. 7, the group controller must send the messages presented in Fig. 8b (respectively Fig. 9b).

Group-Oriented Rekeying. A single rekey message is constructed containing all new keys.The same encrypted keys computed in key oriented approach (Fig. 8) are broadcast to all group members in a one single message. This approach has a high communication overhead as all group members are obliged to receive a big key-update message even if it does not need all the contained keys. Therefore, this approach will not be considered in what follows.

The performance of the LKH scheme depends on whether approach is used and on the membership operation (join or leave).

8.1.1. Computation cost

The computation cost of the LKH scheme is presented differently for the case of join and leave. Note that in both cases the group controller generates h keys. However, the number of encryptions is different and it also depends on the rekeying approach.



(a) Partial view of LKH tree.

(b) LKH tree update after the leaving of node $N1114$.

Fig. 7. LKH tree.

GC→{N2111,…,N4444}:{Kg'}$_{Kg}$

GC→{N1211,…,N1444}:{Kg', K1'}$_{K1}$

GC→{N1121,…,N1144} :{Kg', K1', K11'}$_{K11}$

GC→{N1111,...,N1113}:{Kg', K1', K11', K111'}$_{K111}$

GC→{ N1114}:{Kg', K1', K11', K111'}$_{K1114}$

(a) user-oriented

GC→{N1111,…,N4444}-N1114:{Kg'}$_{Kg}$

GC→{N1111,…,N1444}-N1114:{K1'}$_{K1}$

GC→{N1111,…,N1144} -N1114:{K11'}$_{K11}$

GC→{N1111,...,N1113}-N1114:{K111'}$_{K111}$

GC→{ N1114}:{Kg', K1', K11', K111'}$_{K1114}$

(b) key-oriented

Fig. 8. LKH rekeying due to a join.

GC→{N2111,…,N2444}:{Kg'}$_{K2}$
GC→{N3111,…,N3444}:{Kg'}$_{K3}$
GC→{N4111,…,N4444}:{Kg'}$_{K4}$

GC→{N1211,…,N1244}:{Kg', K1'}$_{K12}$
GC→{N1311,…,N1344}:{Kg', K1'}$_{K13}$
GC→{N1411,…,N1444}:{Kg', K1'}$_{K14}$

GC→{N1121,…,N1124} :{Kg', K1', K11'}$_{K112}$
GC→{N1131,…,N1134} :{Kg', K1', K11'}$_{K113}$
GC→{N1141,…,N1144} :{Kg', K1', K11'}$_{K114}$

GC→{N1111}:{Kg', K1', K11', K111'}$_{K1111}$
GC→{ N1112}:{Kg', K1', K11', K111'}$_{K1112}$
GC→{ N1113}:{Kg', K1', K11', K111'}$_{K1113}$

(a) user-oriented

GC→{N2111,…,N2444}:{Kg'}$_{K2}$
GC→{N3111,…,N3444}:{Kg'}$_{K3}$
GC→{N4111,…,N4444}:{Kg'}$_{K4}$
GC→{N1111, …,N1444}-N1114:{Kg'}$_{K1'}$

GC→{N1211,…,N1244}:{K1'}$_{K12}$
GC→{N1311,…,N1344}:{K1'}$_{K13}$
GC→{N1411,…,N1444}:{K1'}$_{K14}$
GC→{N1111,…,N1144}-N1114:{K1'}$_{K11'}$

GC→{N1121,…,N1124} :{K11'}$_{K112}$
GC→{N1131,…,N1134} :{K11'}$_{K113}$
GC→{N1141,…,N1144} :{K11'}$_{K114}$
GC→{N1111, N1112, N1113}:{K11'}$_{K111'}$

GC→{N1111}:{K111'}$_{K1111}$
GC→{ N1112}:{K111'}$_{K1112}$
GC→{ N1113}:{K111'}$_{K1113}$

(b) key-oriented

Fig. 9. LKH rekeying due to a leave.

Case of join.

- Group controller: In the user-oriented rekeying, the number of encryptions is $1 + 2 + \ldots + h = h \cdot \frac{(h+1)}{2}$. So, the GC computation cost is = $h \cdot C_{kg} + h \cdot \frac{(h+1)}{2} \cdot C_{enc}$In the key-oriented rekeying, the number of encryptions is h. So, the GC computation cost is = $h \cdot C_{kg} + h \cdot C_{enc}$
- End device: It depends on its position in the key tree. The number of changed key could be $\{1, 2, \ldots, h\}$. So the average number of decryptions is $avg(1, \ldots, h)$ = $(h + 1)/2$. So, the ED computation cost is = $\frac{(h+1)}{2} \cdot C_{dec}$.

Case of leave.

- Group controller: In the user-oriented rekeying, the number of encryptions is $(d - 1)(1 + 2 + \ldots + h) = (d - 1) \cdot h \cdot \frac{(h+1)}{2}$. So, the GC computation cost is = $h \cdot C_{kg} + (d - 1) \cdot h \cdot \frac{(h+1)}{2} \cdot C_{enc}$In the key-oriented rekeying, the number of encryptions is $d \cdot h - 1$. So, the GC computation cost is = $h \cdot C_{kg} + (d \cdot h - 1) \cdot C_{enc}$

We note that, the key-oriented approach is more computationally-efficient than the user-oriented approach.

- End device: It depends on its position in the key tree. The number of changed key could be $\{1, 2, \ldots, h\}$. So the average number of decryption is $avg(1, \ldots, h)$ = $(h + 1)/2$. So, the ED computation cost is=$\frac{(h+1)}{2} \cdot C_{dec}$.

8.1.2. Communication cost

In LKH, the communication cost is $O(log(n))$. Moreover, we have to distinguish if the network has a multicast routing support or not. We define $|mku|$ as the size of a *key-update* message containing one key.

Case of join. The rekeying messages sent by the GC due to a join operation are presented in Fig. 8.

- Group controller: In the user-oriented rekeying, the GC needs to send a multicast message containing 1 key to $(d - 1) \cdot d^{h-1}$ nodes, …, a multicast message containing $h - 1$ keys to $(d - 1) \cdot d$ nodes, a multicast message containing h keys to $d - 1$ nodes. So, the GC communication cost in the user-oriented approach and using a multicast routing protocol is:

$$(1 + \ldots + h - 1 + h) \cdot |mku| \cdot e_{tx} = h \cdot \frac{h+1}{2} \cdot |mku| \cdot e_{tx}$$

If the network does not support multicast, each multicast message must be sent by unicast. So, the GC communication cost becomes:

$$\begin{aligned}&(1 \cdot (d - 1) \cdot d^{h-1} + \cdots + (h - 1) \cdot (d - 1) \cdot d\\&\quad + h \cdot (d - 1)) \cdot |mku| \cdot e_{tx}\\&\quad = (d - 1)(d^{h-1} + \cdots + (h - 1) \cdot d + h) \cdot |mku| \cdot e_{tx}\end{aligned}$$

In the key-oriented rekeying, all rekeying messages contains one key and the GC needs to send a message destined to $d^h - 1$ nodes, a message destined to $d^{h-1} - 1$ nodes, …, a message destined to $d - 1$ nodes. Using a multicast routing support, the GC communication cost in the key-oriented approach is:

$$h \cdot |mku| \cdot e_{tx}$$

Without a multicast routing support, the GC communication cost in the key-oriented approach is:

$$\begin{aligned}&(d^h + \cdots + d - h) \cdot |mku| \cdot e_{tx}\\ =\quad& d \cdot \frac{1-d^h}{1-d} - h\Bigg) \cdot |mku| \cdot e_{tx}\end{aligned}$$

Case of leave. The rekeying messages sent by the GC due to a leave operation are presented in Fig. 9.

- Group controller: In the user-oriented approach, the GC sends $(d - 1)$ messages each one contains one key and is destined to d^{h-1} nodes, $(d - 1)$ messages each one contains two keys and is destined to $d^{h\,-\,2}$ nodes, …, $(d - 1)$ messages each one contains $(h - 1)$ keys and is destined to d nodes and $(d - 1)$ messages each one contains h keys and is destined to 1 nodes. So, the communication cost of the GC during the rekeying process due to a leave operation, in the user-oriented approach and using multicast is:

$$\begin{aligned}&(1 \cdot (d-1) + \cdots + (h-1) \cdot (d-1)\\ &\quad + h \cdot (d-1)) \cdot |mku| \cdot e_{tx}\\ &\quad = (d-1) \cdot h \cdot \frac{h+1}{2} \cdot |mku| \cdot e_{tx}\end{aligned}$$

Without the multicast routing support, the GC communication cost is:

$$\begin{aligned}&(1 \cdot (d-1) \cdot d^{h-1} + \cdots + (h-1) \cdot (d-1) \cdot d + h \cdot (d\\ &\quad - 1)) \cdot |mku| \cdot e_{tx}\\ &\quad = (d-1)(d^{h-1} + \cdots + (h-1) \cdot d + h) \cdot |mku| \cdot e_{tx}\end{aligned}$$

In the key-oriented rekeying, all rekeying messages contain one key and the GC needs to send d messages each one is destined to d^{h-1} nodes (except one message is destined to $d^{h-1} - 1$), d messages each one is destined to d^{h-2} nodes (except one message is destined to $d^{h-2} - 1$), …, d messages each one is destined to d nodes (except one message is destined to $d - 1$), $d - 1$ messages each one is destined to 1 nodes. Using a multicast routing support, the GC communication cost in the key-oriented approach is:

$$(d \cdot h - 1) \cdot |mku| \cdot e_{tx}.$$

Without a multicast routing support, the GC communication cost in the key-oriented approach is:

$$\begin{aligned}&[d \cdot (d^{h-1} + \cdots + d + 1) - h] \cdot |mku| \cdot e_{tx}\\ =\quad& d \cdot \frac{1-d^h}{1-d} - h\Bigg) \cdot |mku| \cdot e_{tx}\end{aligned}$$

- End device: Every node has to forward the rekeying messages sent by the GC. The number of received messages depends on its position on the network. If we use broadcast to send messages, every node will receive and forward all messages sent by the GC.

8.1.3. Storage cost

- Group controller has to store: The logical key hierarchy tree, which contains=$\left[\frac{(d^{(h+1)}-1)}{(d-1)}\right]$ keys $\approx (\frac{d}{d-1}) \cdot n$ keys, where $n = d^h$. So, storage cost is $O(n)$.
  Moreover, the GC must store the identity of group members = $n \cdot |ID|$.
  Example: for a 128 bits key and $n = 1024$ and binary tree $(d = 2)$, we get storage = (2048 × 16 bytes = 32 Kbytes) which exceeds the TelosB RAM (equal to 10 Kbytes).
  Therefore, it can be concluded that LKH presents a storage cost problem.

- End device has to store: $h + 1$ keys.

LKH rekeying approach comparison: From Table 7, we can clearly note that the key-oriented approach is more efficient than the user-oriented approach in terms of computation and communication cost.

8.1.4. Duration

The key update duration for the LKH scheme is defined as the sum of the processing time and the transmission time. The processing time can be directly extracted from the Table 7 by simply replacing C_{op} by T_{op}, where *op* represents an operation such as encryption, signature, etc.

As for the transmission time, it contains the transmission time needed to send all key-update messages by the GC and the transmission time needed to forward a key-update message from the GC to a group member. This time is expressed as $hop \cdot |mku| \cdot T_{tx}$, where *hop* is the average number of hops between the GC and a group member.

Moreover, we consider only the key-oriented approach as it is the most efficient approach. The computation and communication time of the different version of the LKH scheme are summarized in Table 8.

Table 7
LKH rekeying approach comparison.

	Case of join		Case of leave	
	User-oriented	Key-oriented	User-oriented	Key-oriented
Computation cost	$h \cdot C_{kg} + h \cdot \frac{(h+1)}{2} \cdot C_{enc}$	$h \cdot C_{kg} + h \cdot C_{enc}$	$h \cdot C_{kg} + (d-1) \cdot h \cdot \frac{(h+1)}{2} \cdot C_{enc}$	$h \cdot C_{kg} +\ (d \cdot h - 1) \cdot C_{enc}$
Communication cost using multicast	$h \cdot \frac{h+1}{2} \cdot \lvert mku\rvert \cdot e_{tx}$	$h \cdot \lvert mku\rvert \cdot e_{tx}$	$(d-1) \cdot h \cdot \frac{h+1}{2} \cdot \lvert mku\rvert \cdot e_{tx}$	$(d \cdot h - 1) \cdot \lvert mku\rvert \cdot e_{tx}$
Communication cost using unicast	$(d-1)(d^{h-1} + \cdots$ $+ (h-1) \cdot d + h) \cdot \lvert mku\rvert \cdot e_{tx}$	$(d \cdot \frac{1-d^h}{1-d} - h) \cdot \lvert mku\rvert \cdot e_{tx}$	$(d-1)(d^{h-1} + \ldots + (h-1) \cdot d + h) \cdot \lvert mku\rvert \cdot e_{tx}$	$\left(d \cdot \frac{1-d^h}{1-d} - h\right) \cdot \lvert mku\rvert \cdot e_{tx}$

Table 8
LKH processing and transmission time at the GC.

	Case of join		Case of leave	
	LKH	LKH + Sign	LKH	LKH + Sign
Processing time	$h{\cdot}(T_{kg} + T_{enc})$	$h{\cdot}(T_{kg} + T_{enc} + T_{sign})$	$h{\cdot}T_{kg} + (d{\cdot}h - 1){\cdot}T_{enc}$	$h{\cdot}T_{kg} + (d{\cdot}h - 1){\cdot}(T_{enc} + T_{sign})$
	LKH	LKH + Multi	LKH	LKH + Multi
Transmission time	$\left(d \cdot \frac{1-d^h}{1-d} - h\right) \cdot \lvert mku\rvert \cdot T_{tx}$	$h{\cdot}\lvert mku\rvert{\cdot}T_{tx}$	$\left(d \cdot \frac{1-d^h}{1-d} - h\right) \cdot \lvert mku\rvert \cdot T_{tx}$	$(d{\cdot}h - 1){\cdot}\lvert mku\rvert{\cdot}T_{tx}$

Based on this table, resulting rekeying durations are as follows:

Case of join. The key update duration (latency) depends on whether the network has a multicast routing support or not.

- **Without multicast** support: In this case, we use unicast. So, a multicast message is replaced by several unicast ones.

$$\text{Duration} = h \cdot (T_{kg} + T_{enc}) + \left. d \cdot \frac{1-d^h}{1-d} - h \right) \cdot |mku| \cdot T_{tx} + hop \cdot |mku| \cdot T_{tx}$$

The quantity ($hop{\cdot}T_{tx}$) is the time needed for the forwarding of the last message by intermediate nodes, where *hop* represents the average number of hops between group controller and group members.

- **With multicast support:**The multicast routing support allows to reduce the number of sent messages as several unicast messages will be replaced by a single multicast one. However, the latency of the sending of a multicast message depends on the position of destination nodes in the physical topology. So the latency of a multicast message is the time between the sending of the message by the group controller and the receiving of this message by the latest destination.

$$\text{Duration} = h \cdot (T_{kg} + T_{enc}) + h \cdot |mku| \cdot T_{tx} + hop \cdot |mku| \cdot T_{tx}.$$

Case of leave. In the LKH scheme, after the leaving of a node the GC needs to send the messages presented in Fig. 9b. The key update duration contains the processing time (computation time), the transmitting time (communication time) and the propagation time. The key update duration (latency) depends on whether the network has a multicast routing support or not.

- **Without multicast support:**
  In this case, we use unicast. So, a multicast message is replaced by several unicast ones.

$$\text{Duration} = h \cdot T_{kg} + (d \cdot h - 1) \cdot T_{enc} + \left. d \cdot \frac{1-d^h}{1-d} - h \right) \cdot |mku| \cdot T_{tx} + hop \cdot |mku| \cdot T_{tx}$$

The quantity ($hop{\cdot}T_{tx}$) is the time needed for the forwarding of the last message by intermediate nodes, where *hop* represents the average number of hops between group controller and group members.

- **With multicast support:**The multicast routing support allows to reduce the number of sent messages as several unicast messages will be replaced by a single multicast one. However, the latency of a multicast message depends on the position of destination nodes in the physical topology. So the latency of a multicast message is the time between the sending of the message by the group controller and the receiving of this message by the latest destination.

$$\text{Duration} = h \cdot T_{kg} + (d \cdot h - 1) \cdot T_{enc} + (d \cdot h - 1) \cdot |mku| \cdot T_{tx} + hop \cdot |mku| \cdot T_{tx}.$$

8.2. Topological Key Hierarchy (TKH)

In the TKH scheme, the key tree is constructed in the following way. First we construct a routing tree (each node is attached to a parent node until reaching the sink node, which is the group controller). Then, for each neighbor of the sink we construct a Tree Key (TK) and for each set of neighbors to that node we construct a Sibling Key (SK). That is, the level of the key tree is always 4, independently from the number of group members, and each node holds 4 keys, which are a Group Key (GK), TK, SK, and an Individual key (IK). Fig. 10 explains the tree construction in the TKH scheme.

As shown in Fig. 11, the TKH tree can be modeled as an $\alpha\beta\gamma$ *-tree,* where the meaning of α, β, and γ are presented in Table 9. More precisely, α represents the number of neighbors to the sink (the group controller), and γ the number of neighbors to a parent node.

We will consider a tree with $\alpha = 5$, $\beta = 40$, $\gamma = 5$ in order to obtain a number of nodes $n = (\gamma\beta + 1)\alpha$.

$n = 1030 \approx 1024$ as similar with other simulations.

In case of a join, the authors propose to renew the keys locally using a one-way hash function. However, they do not present how to launch this key-update process in group members nor what messages need to be sent. Indeed, when a new node joins the group, the GC must inform group members in order to update their corresponding keys. So different messages must be sent according to the group member position in the hierarchy. Moreover, these messages must be authenticated (for example using a signature) otherwise an attacker can disturb a group operation by sending fake messages to group members that will renew keys uselessly.

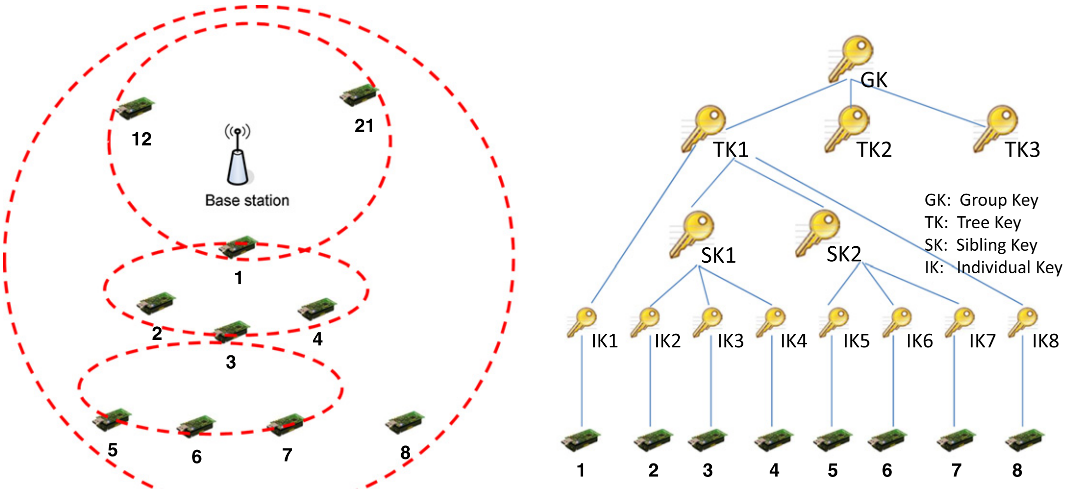


Fig. 10. (a) A sensor network tree topology example and (b) the corresponding TKH key tree structure.

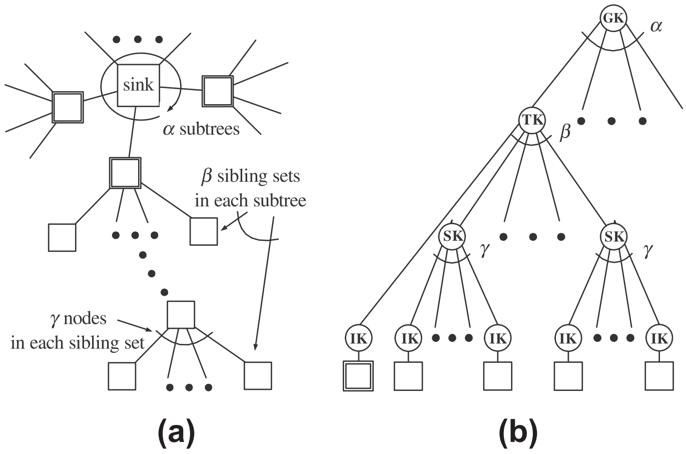


Fig. 11. (a) '$\alpha\beta\gamma$-tree' and (b) the corresponding TKH key tree structure [5].

Table 9
TKH parameters.

Notations	Meaning
α	Number of subtrees
β	Number of sibling sets per subtree
γ	Number of sibling nodes per sibling set
n	Number of nodes in the group. $n = \alpha{\cdot}(\beta\gamma + 1)$

As authors do not present details of group rekeying due to a join operation, we consider in what follows only the rekeying cost due to a leave operation.

When a node leaves, the GC needs to change 3 keys GK, TK, SK. According to the network model presented in Fig. 11, the following rekeying messages are sent by the GC.

- GK requires α multicast, each multicast for $\gamma{\cdot}\beta$ nodes ($\{GK\prime\}_{TK_i}$)
- TK requires
  - β multicast, each multicast for γ nodes ($\{TK_i'\}_{SK_j}$)
  - 1 unicast for the sub-router, which is also a neighbor node
- SK requires $\gamma - 1$ unicast messages $\left(\left\{SK_j'\right\}_{IK_l}\right)$

8.2.1. Computation cost

The computation cost of the TKH group key management scheme during the rekeying process due to a leave operation is as follows:

- Group controller:
  - Number of key generations = 3
  - Number of encryptions = $(\gamma - 1) + (1 + \beta) + \alpha = \alpha + \beta + \gamma$

With the above example parameters, the number of encryptions is: $5 + 40 + 5 = 50 > log(n)$.

So, the GC computation cost during a rekeying due to a leave operation is: $3{\cdot}C_{kg} + (\alpha + \beta + \gamma).C_{enc}$.

- End device:
  - Number of decryption (maximum) = 3

So, the end device computation cost is: $3 \cdot C_{dec}$Note that the computation cost of an end device is independent from the group size.

8.2.2. Communication cost

In TKH, the communication cost in terms of number of rekeying messages is $> O(log(n))$.

- Group controller:
  Using a multicast routing support, the GC communication cost is:

$$(\alpha + \beta + \gamma) \cdot |mku| \cdot e_{tx}$$

Without a multicast routing support, the GC communication cost is:

$$(\alpha \cdot \beta \cdot \gamma + \beta \cdot \gamma + \gamma) \cdot |mku| \cdot e_{tx}$$

- Every node (must forward the rekeying messages): This depends on its position on the network, as it must forward the rekeying messages sent by the GC. However, as there is a mapping between the TKH and the network topology, the communication overhead is reduced compared to LKH.

8.2.3. Storage cost

- The group controller has to store: The TKH tree, which contains the following keys: $GK + \alpha\ \ TK + \alpha\beta SK + nIK$. So the number of keys is $= (1 + \alpha + \alpha\beta + n) = 1 + n + \alpha (1 + \beta)$. The storage cost is $O(n)$. Example: for a 128 bits key and $n = 1024$ and $\gamma = 4$, We get storage = 20 Kbytes, which exceeds the TelosB RAM. TKH presents a storage cost problem.
- The end device has to store: 4 keys.

8.2.4. Duration

During the rekeying process due to a leave, the GC sends $\gamma - 1$ unicast + 1 neighbor unicast $+\beta$ multicast (each message is for γ nodes) + α multicast (each message is for $\beta\gamma$ nodes). Each message contains 1 encrypted key.

- **Without multicast** support

$$\text{Duration} = 3 \cdot T_{kg} + (\alpha + \beta + \gamma) \cdot T_{enc} + (\alpha \cdot \beta \cdot \gamma + \beta \cdot \gamma + \gamma) \cdot |mku| \cdot T_{tx} + hop \cdot |mku| \cdot T_{tx}$$

- **With multicast** support

$$\text{Duration} = 3 \cdot T_{kg} + (\alpha + \beta + \gamma) \cdot T_{enc} + (\gamma + \beta + \alpha) \cdot |mku| \cdot T_{tx} + hop \cdot |mku| \cdot T_{tx}$$

As an example: for $n = 1024(\alpha = 5,\ \beta = 40,\ \delta = 5)$ and $T_{tx} = 1$ ms and $hop = 3$ we get TKH duration = $(5 + 40 + 5 + 3) \times 1$ ms = 54 ms > LKH.

8.3. Ring based Secure Group communication (RiSeG)

RiSeG [8,9] is the first scheme that has addressed the secure group communication problem with a constrained group controller. The idea of the scheme is to divide the group controller task among group members by constructing a logical ring architecture. This logical ring permits to deliver rekeying messages as follows. The GC sends the rekeying message to next hop in the ring and then the message will be forwarded from node to node until returning back to the GC. The scheme reduces the communication, computation and storage cost at the group controller, however, it introduces a big duration that cannot scale with large group. In fact the duration of the rekeying process is $O(n)$.

8.3.1. Computation cost

In case of a join operation, the GC generates a group key, encrypts it, signs it and sends two *key-update* messages so that the GC performs $C_{kg} + C_{enc} + C_{sign}$. Upon receiving the *key-update* message a group member verifies the signature, decrypts the key and then forwards the message so that a group member performs $C_{verif} + C_{dec}$.

In case of a leave operation, the GC generates the group key, signs it and then sends two *key-update* messages containing group key encrypted using pair-wise key so that the GC performs $C_{kg} + C_{sign} + 2C_{kc} + 2C_{enc}$. Upon receiving a key update message a group member decrypts the key, verifies the signature, and then re-encrypts message and forwards it so that a group member performs $2C_{kc} + C_{dec} + C_{verif} + C_{enc}$.

Case of join.
- Group controller: $C_{kg} + C_{enc} + C_{sign}$
- End device: $C_{dec} + C_{verif}$

Case of leave.
- Group controller: $C_{kg} + C_{sign} + 2C_{kc} + 2C_{enc}$
- End device: $2C_{kc} + C_{dec} + C_{verif} + C_{enc}$

8.3.2. Communication cost

Regarding the key update process communication cost, in both cases (join and leave), the GC sends two *key-update* messages so that it consumes $2|mku| \times e_{tx}$ while a group member receives 1 *key-update* message and sends 1 *key-update* message so that it consumes $|mku|.e_{rx} + |mku|.e_{tx} = |mku|.\ (e_{rx} + e_{tx})$.

Hence we can note that the RiSeG rekeying process communication cost is $O(1)$.

8.3.3. Storage cost

- Group controller:
  - The ring topology = $n \cdot sizeof(ID)$
  - The Blundo polynomial share = $(t + 1) \cdot log(q)$
  - The ECC domain parameters $T = (p,a,b,G,r,h)$, and the pair of GC public/private key (Q_G,d_G)
- End device stores:
  - The next hop and previous hop (in the ring) addresses = $2 \cdot sizeof(ID)$
  - The Blundo polynomial share = $(t + 1) \cdot log(q)$
  - The ECC domain parameters $T = (p,a,b,G,r,h)$, and the GC public key Q_G

8.3.4. Duration

In the RiSeG scheme, and in case of a rekeying caused by a join operation, the same key update message is

Table 10
Storage and computation cost comparison.

Scheme	Storage cost	Computation cost (single node joining)	Computation cost (single node leaving)
LKH [3]	• GC:$\frac{(d^{(h+1)}-1)}{(d-1)}$ keys + $n\cdot\|ID\|$ $\approx \left(\frac{d}{d-1}\right)\cdot n$ keys + $n\cdot\|ID\|$ • ED: h keys	• GC:$h\cdot C_{kg} + h\cdot C_{enc}$ • ED:$\frac{(h+1)}{2}\cdot C_{dec}$	• GC:$h\cdot C_{kg} + (d\cdot h - 1)\cdot C_{enc}$ • ED:$\frac{(h+1)}{2}\cdot C_{dec}$
LKH [3]+Sign	• GC:$\left(\frac{d}{d-1}\right)\cdot n$ keys + $n\cdot\|ID\| + T + (d_G, Q_G)$ • ED:h keys + $T + Q_G$	• GC:$h\cdot(C_{kg} + C_{enc} + C_{sign})$ • ED:$\frac{(h+1)}{2}\cdot(C_{dec} + C_{verif})$	• GC:$h\cdot C_{kg} + (d\cdot h - 1)\cdot(C_{enc} + C_{sign})$ • ED:$\frac{(h+1)}{2}\cdot(C_{dec} + C_{verif})$
TKH [5]	• GC: $(1 + \alpha + \alpha\beta + n)$ keys + $n\cdot\|ID\|$ • ED: 4	N/A	• GC:$3\cdot C_{kg} + (\alpha + \beta + \gamma)\cdot C_{enc}$ • ED: $3\cdot C_{dec}$
TKH [5]+ Sign	• GC: $(1 + \alpha + \alpha\beta + n) + T + (d_G, Q_G)$ • ED:$4 + T + Q_G$	N/A	• GC: $3\cdot C_{kg} + (\alpha + \beta + \gamma)\cdot(C_{enc} + C_{sign})$ • ED: $3\cdot(C_{dec} + C_{verif})$
RiSeG [8,9]	• GC: $n\cdot\|ID\| + (t + 1)\cdot log(q) + T + (d_G, Q_G)$ • ED: $2\cdot\|ID\| + (t + 1)\cdot log(q) + T + Q_G$	• GC: $C_{kg} + C_{enc} + C_{sign}$ • ED: $C_{dec} + C_{verif}$	• GC: $C_{kg} + 2C_{kc} + 2C_{enc} + C_{sign}$ • ED: $2C_{kc} + C_{enc} + C_{dec} + C_{verif}$
LNT	• GC: $n\cdot\|ID\| + (t + 1)\cdot log(q)$ + Private/public key • ED: $d\cdot\|ID\| + (t + 1)\cdot log(q)$ + GC public key.	• GC: $C_{kg} + C_{enc} + C_{sign}$ • ED: $C_{dec} + C_{verif}$	• GC: $C_{kg} + d(C_{kc} + C_{enc}) + C_{sign}$ • ED: $C_{kc} + C_{verif} + C_{dec} + d(C_{kc} + C_{enc})$

N/A: Information not available.

Table 11
Communication cost comparison.

Scheme	Communication cost (single node joining)	Communication cost (single node leaving)
LKH [3]	• GC: $\left(d\cdot\frac{1-d^h}{1-d} - h\right)\cdot\|mku\|\cdot e_{tx}$	• GC: $\left(d\cdot\frac{1-d^h}{1-d} - h\right)\cdot\|mku\|\cdot e_{tx}$ • ED: $\frac{(h+1)}{2}\cdot\|mku\|\cdot e_{rx}$
LKH + Multicast	• GC: $h\cdot\|mku\|\cdot e_{tx}$	• GC: $(h\cdot d - 1)\cdot\|mku\|\cdot e_{tx} \approx h\cdot d\cdot\|mku\|\cdot e_{tx} = d\cdot log_d(n)\cdot\|mku\|\cdot e_{tx}$ • ED: $\frac{(h+1)}{2}\cdot\|mku\|\cdot e_{rx}$
TKH [5]	N/A	• GC: $(\alpha\cdot\beta\cdot\gamma + \beta\cdot\gamma + \gamma)\cdot\|mku\|\cdot e_{tx}$
TKH + Multicast	N/A	• GC: $(\alpha + \beta + \gamma)\cdot\|mku\|\cdot e_{tx}$
RiSeG [8,9]	• GC: $2\cdot\|mku\|\cdot e_{tx}$ • ED: $\|mku\|\cdot(e_{rx} + e_{tx})$	
LNT	• GC: $d\cdot\|mku\|\cdot e_{tx}$ • ED: $\|mku\|(e_{rx} + d\cdot e_{tx})$	

propagated from node to node. Therefore, a group member needs to decipher the group key, verifies the signature and then forwards the rekeying message ($T_{dec} + T_{verif} + hop\cdot|mku|\cdot T_{tx}$).

For the case of a rekeying caused by a leave operation, the group member must also re-encrypt the group key using pairwise key.

$$\begin{aligned}
\textit{Case of join}\cdot\ T_{kuj} &= T_{kg} + T_{enc} + T_{sign} + 2\cdot hop\cdot|mku|\cdot T_{tx} \\
&\quad + \frac{n}{2}\cdot(T_{dec} + T_{verif} + hop\cdot|mku|\cdot T_{tx}). \\
\textit{Case of leave}\cdot\ T_{kul} &= T_{kg} + T_{sign} + 2\cdot(T_{kc} + T_{enc} + 2\cdot hop\cdot|mku|\cdot T_{tx}) \\
&\quad + \frac{n}{2}\cdot(T_{kc} + T_{dec} + T_{verif} + T_{kc} + T_{enc} + hop\cdot|mku|\cdot T_{tx}).
\end{aligned}$$

8.3.4.1. Comparative table and analysis. In this section, we show a comparison between the LNT proposed scheme rekeying process and the LKH and TKH schemes.

As LKH and TKH rekeying schemes do not add authentication to rekeying messages, we consider adding a signature to these protocols. Therefore, we refer to "LKH + Sign" (respectively "TKH + sign") to represent the LKH (respectively TKH) scheme where messages are authenticated using a signature mechanism. In this way, schemes are compared in the same context. In the same way, we note "LKH + Multicast" (respectively "TKH + Multicast") to designate the LKH (respectively TKH) scheme with the multicast routing capabilities. Moreover, we consider the LKH scheme in its efficient way (i.e, the key-oriented approach).

Table 10 establishes a comparison between the studied schemes in terms of storage and computation cost.

Table 11 shows the communication cost of the studied schemes.

Table 12 summarizes the rekeying duration of the studied schemes.

Note that, first the proposed schemes suppose that the GC is a powerful node, while our proposed work assumes a homogeneous network. That is, the GC is a sensor node with constrained resources. Second, for the LKH-based schemes, it is true that they minimize the number of rekeying messages. However, when considering the delivery of these rekeying messages, they introduce an overhead that can exceed the one used to send unicast messages for each node (with $O(n)$ as communication cost).

In order to put in evidence the performance comparison between LNT and the schemes described in Section 8, we have plotted curves showing the performance of these schemes when varying the number of group members n.

For the storage cost, the size of keys is set to 128*bits*. This keys size is used for AES cryptographic algorithm. The identity of node is set to 16*bits*. As shown in Fig. 12,

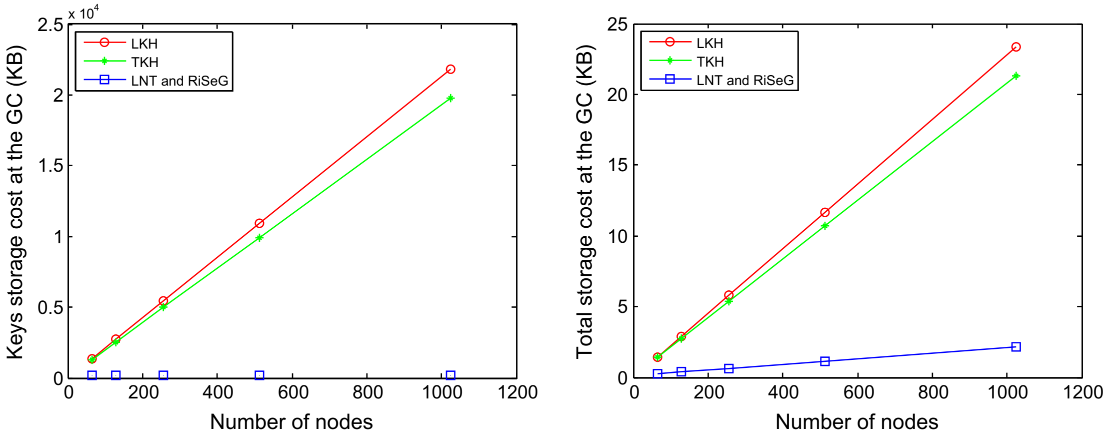
Table 12
Rekeying duration comparison.

Scheme	Rekeying duration
LKH [3] + Sign (unicast)	Case of join: $h \cdot (T_{kg} + T_{enc} + T_{sign}) + \left. \quad d \cdot \frac{1-d^h}{1-d} - h \right) \cdot \lvert mku \rvert \cdot T_{tx} + hop \cdot \lvert mku \rvert \cdot T_{tx}$ Case of leave: $h \cdot T_{kg} + (d \cdot h - 1) \cdot (T_{enc} + T_{sign}) + \left. \quad d \cdot \frac{1-d^h}{1-d} - h \right) \cdot \lvert mku \rvert \cdot T_{tx} + hop \cdot \lvert mku \rvert \cdot T_{tx}$
LKH [3]+Sign (multicast)	Case of join: $h \cdot (T_{kg} + T_{enc} + T_{sign}) + h \cdot \lvert mku \rvert \cdot T_{tx} + hop \cdot \lvert mku \rvert \cdot T_{tx}$ Case of leave: $h \cdot T_{kg} + (d \cdot h - 1) \cdot (T_{enc} + T_{sign}) + (d \cdot h - 1) \cdot \lvert mku \rvert \cdot T_{tx} + hop \cdot \lvert mku \rvert \cdot T_{tx}$
TKH [5]+Sign (unicast)	Case of leave: $3 \cdot T_{kg} + (\alpha + \beta + \gamma) \cdot (T_{enc} + T_{sign}) + (\alpha \cdot \beta \cdot \gamma + \beta \cdot \gamma + \gamma) \cdot \lvert mku \rvert \cdot T_{tx} + hop \cdot \lvert mku \rvert \cdot T_{tx}$
TKH [5]+Sign (multicast)	Case of leave: $3 \cdot T_{kg} + (\alpha + \beta + \gamma) \cdot (T_{enc} + T_{sign}) + (\gamma + \beta + \alpha) \cdot \lvert mku \rvert \cdot T_{tx} + hop \cdot \lvert mku \rvert \cdot T_{tx}$
RiSeG [8,9]	Case of join: $T_{kg} + T_{enc} + T_{sign} + 2 \cdot hop \cdot \lvert mku \rvert \cdot T_{tx} + \frac{n}{2} \cdot (T_{dec} + T_{verif} + hop \cdot \lvert mku \rvert \cdot T_{tx})$ Case of leave: $T_{kg} + T_{sign} + 2 \cdot (T_{kc} + T_{enc} + 2 \cdot hop \cdot \lvert mku \rvert \cdot T_{tx}) + \frac{n}{2} \cdot (T_{kc} + T_{dec} + T_{verif} + T_{kc} + T_{enc}$ $+ hop \cdot \lvert mku \rvert \cdot T_{tx})$
LNT	Case of join: $T_{kg} + T_{sign} + T_{enc} + d \cdot hop \cdot \lvert mku \rvert \cdot T_{tx} + (h-1) \cdot (T_{dec} + T_{verif} + d \cdot hop \cdot \lvert mku \rvert \cdot T_{tx})$ Case of leave: $T_{kg} + T_{sign} + (h-1) \cdot (T_{kc} + T_{dec} + T_{verif}) + h \cdot d \cdot (T_{kc} + T_{enc} + hop \cdot \lvert mku \rvert \cdot T_{tx})$

the RiSeG and LNT schemes represent the best storage cost. Indeed, in both RiSeG and LNT schemes, the GC has to store only the Blundo polynomial parameters and some other keys (the group key, the private/public keys, etc.) that are independent from the number of group members. However, in both LKH and TKH schemes the number of keys is proportional to the number of nodes in the group. With number of group members $n = 1024$, in the RiSeG and LNT, the GC storage cost is less than 1*Kbyte*, however, in the LKH scheme it reaches 21.3*Kbytes* and in TKH it reaches 19.2*Kbytes*.

Moreover, for the total storage cost, the GC needs to store not only the keys but also the group members identities (of size 2*bytes*). Therefore, with $n = 1024$, LKH requires about 23.3*Kbytes*, TKH requires 21.2*Kbytes*, RiSeG and LNT require 2.1*Kbytes* of storage space. If we suppose that these parameters are stored in the ROM memory of the sensor node, for TelosB motes, which have 48*Kbytes* of ROM, LKH and TKH consume about 50% of the available ROM, but, LNT consumes less than 5% of ROM. However, if these parameters are needed to be loaded in the RAM memory, for TelosB motes, which have 10*Kbytes* of RAM, LKH and TKH storage requirements exceeds the available memory and so they cannot be implemented in a TelosB sensor node.

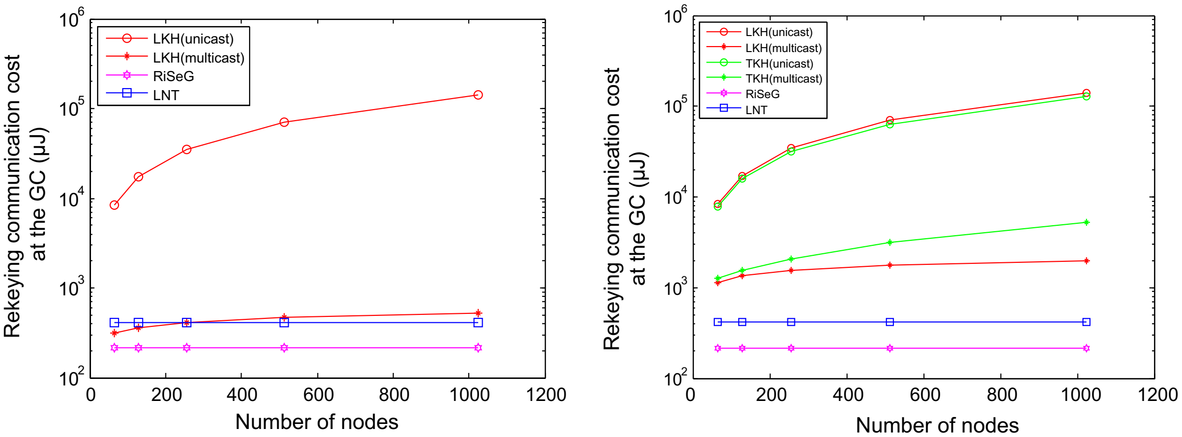
For the communication cost, the unit communication costs are set to $e_{tx} = 0.209$ μJ and $e_{rx} = 0.226$ μJ from the characteristics of the CC2420 transceiver used in the Xbow's MICA-Z and TelosB sensor nodes [42]. Moreover, the key update message size is set to 64 bytes ($\lvert mku \rvert = 512$ bits), which represents the implemented LNT key update message size. Fig. 13 shows that the communication cost at the GC in the LNT scheme is independent from the number of group members and is smaller than those of the LKH and TKH schemes. Moreover, for the LKH and TKH schemes, without using a multicast routing support the communication cost is very high (for $n = 1024$ it is 140 mJ for LKH and 127 mJ for TKH). We also note that,



(a) Keys storage cost at the GC

(b) Total storage cost at the GC

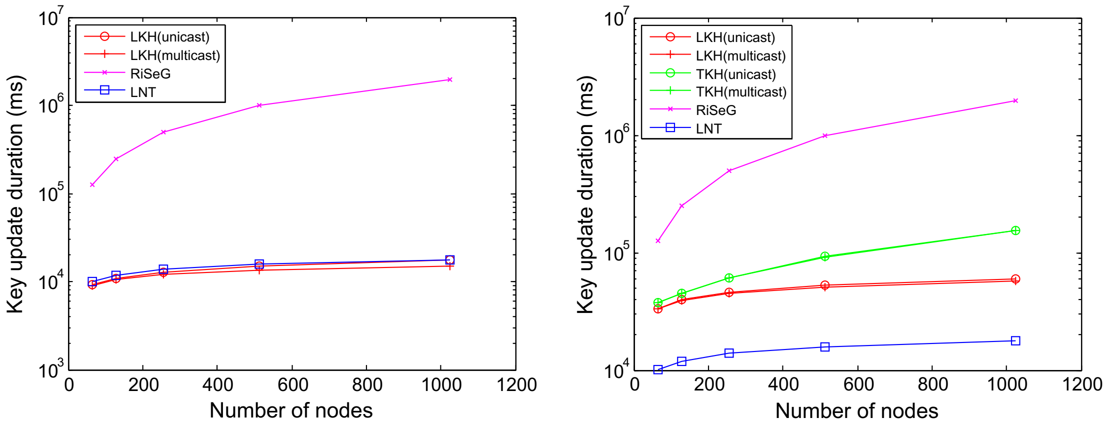
Fig. 12. Storage cost comparison.



(a) Rekeying due to a join

(b) Rekeying due to a leave

Fig. 13. Rekeying communication cost comparison.



(a) Rekeying due to a join

(b) Rekeying due to a leave

Fig. 14. Key update duration comparison.

Table 13
Rekeying duration parameters value.

Parameter	Value (ms)	Parameter	Value
T_{enc}	2.6	$\lvert mku \rvert$	512 bits
T_{dec}	3.2	e_{tx}	0.209 μJ
T_{sign}	3014	e_{rx}	0.226 μJ
T_{verif}	3839		
T_{kg}	0.12		
T_{kc}	0.908		
T_{mac}	0.261		
T_{tx}	0.04		
T_{nd}	3		

for LKH and TKH schemes, the rekeying communication cost resulting from a leave operation is higher than the rekeying resulting from a join operation (for $n = 1024$, LKH GC rekeying communication cost is 518 μJ in case of join and 1970 μJ in case of leave). However, for RiSeG and LNT the same communication is consumed by the group controller in both cases and it is about 50 μJ for RiSeG and 100 μJ for LNT.

Fig. 14 shows the variation of the key update duration when varying the number of nodes. From the TelosB datasheet [43], the transmit data rate is 250 kbps, so, T_{tx} is equal to 4 μs (1/250).

The curves presented in Fig. 14 are plotted based on the equation of Table 12 and using values presented in Table 13. These values are obtained from experimental test on TelosB motes.

From Fig. 14, we note that RiSeG has a big duration as it was assumed. Moreover, the key update duration due to a leave operation is bigger than the key update duration due to a join operation.

Fig. 14 shows that without multicast support LKH has a bigger key update duration than LNT. With multicast support, the rekeying duration in case of a leave operation is 57 s for LKH and 150 s for TKH for a group of 1024 nodes. However, LNT needs only 18 s.

Note that LNT outperforms other group communication schemes in the following way:

- Storage cost.
- Computation and communication cost at the GC.
- No multicast support is required.
- Alleviate the task of the GC.

9. Security analysis and discussion

This section is devoted to discuss the merits of the different cryptographic tools used in the proposed scheme and analyze its security. In the design of our scheme a nonce has been applied in order to prevent replay attacks, along with a Message Authentication Code (MAC) intended to avoid impersonation attacks, as well as a signature aiming at providing authentication of the rekeying messages. The proposed secure group communication scheme provides the following security services:

- Replay attack robustness: in the proposed scheme, intercepted messages cannot be replayed by an attacker as all sent messages are proved to be fresh through a nonce or a sequence number. In addition, attackers cannot modify the value of the nonce (or the seqNbr) as the message is protected by a MAC.
- Impersonation attack robustness: all sent messages are protected by a MAC computed over the identity of the sender node. This prevents attackers from impersonating legal nodes and, hence, prevents attackers from gaining access to a group during the group creation and group join processes or realizing a denial of service attack during the group leave process.
- Authentication of the rekeying messages: key-update messages carry a signature computed by the GC. This signature proves that the key is sent by the GC and, therefore, precludes an attacker from injecting a fake group key.
- Backward and forward secrecy: when a new node joins the group, the group controller generates a new key and delivers it to the group members. Therefore, the new node has no means to decrypt the previously exchanged messages. Moreover, when a node leaves the group, the group controller generates a new key. This key will be sent by unicast and, therefore, the leaving node will be unable to decrypt the future sent messages.
- Collusion attacks robustness: in our scheme, in the case of a comprise attack detection, the group controller generates a new key and sends it to the group members. As this new group key is independent from previously used keys, the attacker cannot guess the value of this group key according to the revealed cryptographic materials obtained from compromised nodes. Therefore, our scheme is robust against collusion attack. The generation of the group key can be done by means of a pseudo-random function, so keys will be independent.
- Mutual authentication: our scheme achieves mutual authentication since not only the base station authenticates the requesting node, but also the node authenticates the base station. The authentication of messages sent by the base station is critical. In fact, if we do not authenticate the grp-creation-invite message, an attacker can impersonate the base station by sending this message even when the group exists. This scenario will disturb the network operation as there will be a creation of multiple copies of the same group, each of which is composed of a single node.
- Remove of compromised nodes: by means of LNT we are able to logically evict compromised nodes from the system by rekeying, once they are detected by the IDS.

10. Conclusion

In this paper, we have proposed LNT: a Logical Neighbor Tree secure group communication scheme that can be applied to a homogeneous WSN network with a resource-constrained group controller. The scheme alleviates the group controller's task by constructing a logical neighbor tree that helps deliver the rekeying messages. Performance analysis has shown that our scheme outperforms the previously well-known ones in terms of computation, communication and storage costs. LNT scheme can be improved by replacing the ECC-based digital signature scheme by a more lightweight method of authentication such as the use of a key-chain [44].

Acknowledgments

This work has been partially supported by EU FP7 Network of Excellence CONET (Grant Agreement No. FP7-224053), and EU FP7 Integrated Project PLANET (Grant agreement No. FP7-257649).

The authors thank the anonymous reviewers for their comments that help improve the manuscript.

References

[1] S. Rafaeli, D. Hutchison, A survey of key management for secure group communication, ACM Comput. Surv. 35 (3) (2003) 309–329.

[2] S. Pitipatana, A. Nirwan, Security services in group communications over wireless infrastructure, mobile ad hoc, and wireless sensor networks, in: IEEE Wireless Communications, IEEE Computer Society, Washington, DC, USA, 2007, pp. 8–20.

[3] C.K. Wong, M. Gouda, S.S. Lam, Secure group communications using key graphs, IEEE/ACM Trans. Netw. 8 (1) (2000) 16–30.

[4] J.-C. Lin, F. Lai, H.-C. Lee, Efficient group key management protocol with one-way key derivation, 2005, pp. 336–343.

[5] J.-H. Son, J.-S. Lee, S.-W. Seo, Topological key hierarchy for energy–efficient group key management in wireless sensor networks, Wirel. Pers. Commun. 52 (2) (2010) 359–382.

[6] D.-H. Je, J.-S. Lee, Y. Park, S.-W. Seo, Computation-and-storage-efficient key tree management protocol for secure multicast communications, Comput. Commun. 33 (2) (2010) 136–148. <http://www.sciencedirect.com/science/article/B6TYP-4X1SBCM-1/2/15372974ea6466e9e5da3e6899c8f498>.

[7] G. Dini, F. Giurlanda, Neighbourhood monitoring for decentralised coordination in multi-agent systems: a case-study, in: Proceedings of the IEEE International Symposium on Computers and Communications (ISCC'11), Kerkyra (Corfù, Greece), June 28–July 1, 2011, pp. 1–3.

[8] O. Cheikhrouhou, A. Koubaa, G. Dini, M. Abid, RiSeG: a ring based secure group communication protocol for resource-constrained wireless sensor networks, in: Springer Journal on Personal and Ubiquitous Computing (Impact Factor: 1.554 in 2009), 2011. <http://www.springerlink.com/content/k386274714528804/>.

[9] O. Cheikhrouhou, A. Koubaa, O. Gaddour, G. Dini, M. Abid, Riseg a logical ring based secure group communication protocol for wireless sensor networks, in: The International Conference on Wireless and Ubiquitous Systems (ICWUS 2010), Sousse, Tunisia.

[10] S. Lasassmeh, J. Conrad, Time synchronization in wireless sensor networks: a survey, in: Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon), 2010, pp. 242–245.

[11] J.-H. Huang, J. Buckingham, R. Han, A level key infrastructure for secure and efficient group communication in wireless sensor network, in: Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks, IEEE Computer Society, Washington, DC, USA, 2005, pp. 249–260. <http://portal.acm.org/citation.cfm?id=1128018.1128494>.

[12] O. Gaddour, A. Koubaa, M. Abid, Segcom: a secure group communication mechanism in cluster-tree wireless sensor networks, in: First International Conference on Communications and Networking, 2009. ComNet 2009, 2009, pp. 1–7.

[13] C. Blundo, A. De Santis, U. Vaccaro, A. Herzberg, S. Kutten, M. Yong, Perfectly secure key distribution for dynamic conferences, Inform. Comput. 146 (1) (1998) 1–23.

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

[15] D. Gianluca, S.I. maria, Lark: a lightweight authenticated rekeying scheme for clustered wireless sensor networks, ACM Trans. Embed. Comput. Syst. 10 (4).

[16] S. Zhu, S. Setia, S. Jajodia, Leap+: efficient security mechanisms for large-scale distributed sensor networks, ACM Trans. Sen. Netw. 2 (4) (2006) 500–528.

[17] M. Eltoweissy, M.H. Heydari, L. Morales, I.H. Sudborough, Combinatorial optimization of group key management, J. Netw. Syst. Manage. 12 (2004) 33–50. <http://portal.acm.org/citation.cfm?id=969905.969914>.

[18] M. Eltoweissy, A. Wadaa, S. Olariu, L. Wilson, Group key management scheme for large-scale sensor networks, Ad Hoc Netw. 3 (2005) 668–688. <http://portal.acm.org/citation.cfm?id=640912.1640950>.

[19] M.F. Younis, K. Ghumman, M. Eltoweissy, Location-aware combinatorial key management scheme for clustered sensor networks, IEEE Trans. Parall. Distrib. Syst. 17 (2006) 865–882.

[20] I.F. Akyildiz, I.H. Kasimoglu, Wireless sensor and actor networks: research challenges, Ad Hoc Netw. 2 (4) (2004) 351–367. <http://www.sciencedirect.com/science/article/B7576-4CDWMMM-1/2/8c25ac6ebd5ab2e50ce172498c2add9a>.

[21] L. Lamport, Password authentication with insecure communication, Commun. ACM 24 (1981) 770–772. <http://doi.acm.org/10.1145/358790.358797>.

[22] A. Perrig, R. Canetti, J.D. Tygar, D. Song, The tesla broadcast authentication protocol, RSA CryptoBytes 5 (2002) 2–3.

[23] M. Eltoweissy, A. Wadaa, S. Olariu, L. Wilson, Group key management scheme for large-scale sensor networks, Ad Hoc Netw. 3 (2005) 668–688. <http://portal.acm.org/citation.cfm?id=1640912.1640950>.

[24] M. Moharrum, M. Eltoweissy, R. Mukkamala, Dynamic combinatorial key management scheme for sensor networks: research articles, Wirel. Commun. Mob. Comput. 6 (2006) 1017–1035. <http://portal.acm.org/citation.cfm?id=1182929.1182937>.

[25] C. Perkins, E. Royer, Ad-hoc on-demand distance vector routing, in: Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, 1997, pp. 90–100.

[26] D.B. Johnson, D.A. Maltz, Dynamic source routing in ad hoc wireless networks, in: Mobile Computing, Kluwer Academic Publishers, 1996, pp. 153–181.

[27] A.H. Farooqi, F.A. Khan, Intrusion detection systems for wireless sensor networks: a survey, Commun. Netw. (2009) 234–241. doi:10.1007/978-3-642-10844-0-29.

[28] B. Sun, L. Osborne, Y. Xiao, S. Guizani, Intrusion detection techniques in mobile ad hoc and wireless sensor networks, Wireless Commun., IEEE 14 (5) (2007) 56–63.

[29] L. Mostarda, A. Navarra, Distributed intrusion detection systems for enhancing security in mobile wireless sensor networks, Int. J. Distrib. Sen. Netw. 4 (2) (2008) 83–109.

[30] G. Li, J. He, Y. Fu, Group-based intrusion detection system in wireless sensor networks, Comput. Commun. 31 (18) (2008) 4324–4332.

[31] Certicom Research. Standards for efficient cryptography-Section 1: Elliptic curve cryptography, May 2009 <http://www.secg.org/download/aid-780/Section1-v2.pdf>.

[32] D.J. Malan, M. Welsh, M.D. Smith, A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography, in: First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004, 2004, pp. 71–80.

[33] A. Liu, P. Ning, Tinyecc: a configurable library for elliptic curve cryptography in wireless sensor networks, in: Proceedings of the 7th International Conference on Information Processing in Sensor Networks, IPSN '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 245–256. <http://dx.doi.org/10.1109/IPSN.2008.47>.

[34] H. Wang, B. Sheng, Q. Li, Elliptic curve cryptography& #45;based access control in sensor networks, Int. J. Secur. Netw. 1 (2006) 127–137. <http://dl.acm.org/citation.cfm?id=1359224.1359225>.

[35] R. Roman, C. Alcaraz, J. Lopez, A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes, Mob. Netw. Appl. 12 (2007) 231–244.

[36] S. Kalra, S.K. Sood, Elliptic curve cryptography: survey and its security applications, in: Proceedings of the International Conference on Advances in Computing and Artificial Intelligence, ACAI '11, ACM, New York, NY, USA, 2011, pp. 102–106. <http://doi.acm.org/10.1145/2007052.2007073>.

[37] E. Stavrou, A. Pitsillides, A survey on secure multipath routing protocols in wsns, Comput. Netw. 54 (2010) 2215–2238. <http://dx.doi.org/10.1016/j.comnet.2010.02.015>.

[38] D. Liu, P. Ning, R. Li, Establishing pairwise keys in distributed sensor networks, ACM Trans. Inform. Syst. Secur. (TISSEC) (2005) 41–77.

[39] Certicom Research, Standards for efficient cryptography-Section 2: Recommended elliptic curve domain parameters, January 2010 <http://www.secg.org/download/aid-780/Section2-v2.pdf>.

[40] R.D. Pietro, L.V. Mancini, S.E.Y.W. Law, P.J.M. Havinga, Lkhw: a directed diffusion-based secure multicast scheme for wireless sensor networks, in: ICPPW '03: Proceedings of the 32nd International Conference on Parallel Processing Workshops, IEEE Computer Society Press, 2003, pp. 397–406.

[41] A. Sherman, D. McGrew, Key establishment in large dynamic groups using one-way function trees, Softw. Eng., IEEE Trans. 29 (5) (2003) 444–458.

[42] Texas Instruments Inc., 2007, Single-Chip 2.4GHz IEEE 802.15.4 Compliant and ZigBee(TM) Ready RF Transceiver, 2010 <http://www-s.ti.com/sc/ds/cc2420.pdf>.

[43] Telosb datasheet: www.ece.osu.edu/bibyk/ee582/telosMote.pdf.

[44] H. Alzaid, D.-G. Park, J.M.G. Nieto, C. Boyd, E. Foo, A forward and backward secure key management in wireless sensor networks for pcs/scada, in: Proceedings of the 1st International ICST Conference on Sensor Systems and Software, Springer, Grand Hotel Duomo of Pisa, Pisa, 2009.