LKHW : a directed diffusion-based secure multicast scheme for wireless sensor networks

Published in:
32nd International Conference on Parallel Processing Workshops (Proceedings ICPP 2003 Workshops, Kaohsiung, Taiwan, October 6-9, 2003)

DOI:
10.1109/ICPPW.2003.1240395

Published: 01/01/2003

Document Version
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 15. Feb. 2018
Abstract

In this paper, we present a mechanism for securing group communications in Wireless Sensor Networks (WSN). First, we derive an extension of Logical Key Hierarchy (LKH). Then we merge the extension with directed diffusion. The resulting protocol, LKHW, combines the advantages of both LKH and directed diffusion: robustness in routing, and security from the tried and tested concepts of secure multicast. In particular, LKHW enforces both backward and forward secrecy, while incurring an energy cost that scales roughly logarithmically with the group size. This is the first security protocol that leverages directed diffusion, and we show how directed diffusion can be extended to incorporate security in an efficient manner.

1. Introduction

Imagine sensor networks being deployed in public areas to detect SARS viruses. The sensors spend most of their time in a dormant state and only report their measurements when requested. Health officers gather readings from the network by posting requests such as “start sending me your readings five times a minute if you are currently detecting the virus”. Assuming the network is heterogeneous (in that apart from SARS virus sensors, there are other types of sensors in the network), there are generally two ways of executing this request: (1) remember the IDs/names of the virus sensors and send the request explicitly to the sensors using an ID-based routing protocol (like AODV [14] or DSR [11]), or (2) never remember any ID, and instead, flood the network with the request.

Method (1) is most effective when the number of SARS sensors is fixed and known beforehand, and sensors are static (so that a new route does not need to be set up every time a send is requested). However these conditions may not be practical, because these sensors exist in large quantity; the bookkeeping overhead would be unmanageable if at the same time we allow arbitrary addition (or removal) of sensors to (or from) the network. On the other hand, if all the SARS sensors share a common ID/address, ID-based routing does not work. We would not go as far as suggesting setting up directory services for sensor lookup, as the problem of keeping these directory services up-to-date and the problem of locating these directory services only elevate the original problem to a higher level.

Method (2) may seem inefficient, however with the help of directed diffusion [10] that facilitates attribute-based naming and in-network processing, significant traffic reduction is possible (Heidemann et al. report a reduction of 42% [7]). In a nutshell, the energy expended in flooding the network with the request, is (more than) compensated by savings obtained from the exploitation of local interaction and caching. Furthermore, with cached information, after the initial flood, further flooding is often not necessary.

All is well until the health officers require that the communication between the requesting device (i.e. sink) and the SARS sensors (i.e. sources) to be secure. In other words, directed diffusion does not cater for secure group communication. By secure we mean just three aspects: (1) data confidentiality, (2) data integrity, and (3) data authentication. And by group communication, we mean a secure communication channel shared by the sources and the sink – the naïve approach of using pairwise secure channels between each individual source and the sink rules out in-network processing, which is essential for the success of directed diffusion. However our communication model is specific for WSN in that messages flow in a certain direction: interest (or query) messages from sinks to sources, and data messages from sources to sinks. We do not cater for general n-to-n communication, which is not common for WSN.

*This work is partially supported by the EU under the IST-2001-34734 EYES project.
In this paper, we propose LKHW (Logical Key Hierarchy for Wireless sensor networks), a secure group communication scheme based on directed diffusion and LKH. Our contribution is two-fold:

1. **Integration of security and routing**: Our scheme integrates security and routing in a single framework, by leveraging secure multicast techniques and the tried and tested concepts of directed diffusion. Such integration allows our security protocols to be optimized for directed diffusion in terms of energy efficiency, reducing the overhead that would otherwise be necessitated by a routing-unaware alternative.

2. **Efficiency**: We present a performance evaluation model that, unlike the conventional evaluation model for secure multicast schemes, is centered around energy consumption and that takes into account the dynamic nature of the topology of WSN. And with the model, we show that the energy efficiency of LKHW scales roughly logarithmically with the group size.

This paper is organized as follows. Section 2 introduces directed diffusion. Section 3 contains the essentials of LKHW. It starts by introducing LKH, then proceeds to discuss the initialization aspects of LKHW, and the central problems of user-leave and user-join operations. We back up our theories with performance evaluation in Section 4, while we discuss related work in Section 5. Finally, Section 6 gives the conclusion and some ideas for future work. For a more thorough treatment of the subjects in this paper, the reader is referred to our technical report [17].

**2. Overview of Directed Diffusion**

**Basic Terminology** Picture the classic directed diffusion scenario in which a WSN is deployed in a wilderness refuge to track animals [7, 10]. A tracking request represents an interest. The node that broadcasts the initial exploratory interest is the sink, i.e. the final destination of the requested data. In directed diffusion, the adjective “exploratory” indicates unoptimized flow, and that the flow will cease if it is not reinforced. Interest and data are named using attribute-operation-value tuples. (i.e. attr1 op1 val1; attr2 op2 val2...). For example, an interest I might look like “class IS interest; x GT 0”, where GT is a formal operator meaning “greater than”; a data D might look like “class IS data; x IS 1”, where IS is an actual operator meaning “equals”. Obviously this particular data D matches this particular interest I because D’s value ‘1’ is greater than I’s value ‘0’ as required by I’s formal operator. Other formal operators are described by Intanagonwiwat et al. [10]. Every node tries to match every data message it receives with every interest in its interest cache, and if a match is found, the data message will be sent to whoever originated the matching interest. It is important to note that an interest does not contain any attribute-operation-value tuple that describes the sink that originates it, nor does a data describe the source that produces it. An “interest about interests” is just a nested interest, e.g. an interest about interests in four-legged animals. The restriction is that interests cannot be further nested, i.e. there is no “interest about interest about interests” and so on. The concept of “interest about interests” is used extensively in LKHW, as shall be seen later.

Directed diffusion consists of three phases:

1. **Interest diffusion**: As the interest is diffused across the network, every sensor that receives the interest remembers the neighbour(s) from which the interest came, using their interest cache (which are essential for suppressing duplicate messages and preventing loops), before rebroadcasting the interest to all their neighbours. Moreover, every sensor node is task-aware and any node that matches the traveling interest will apart from forwarding the interest, reply with the relevant data and thus become a source. The traveling interest sets up gradients along the paths it has taken. A direction is downstream if it is from a source to the sink, and upstream if otherwise.

2. **Exploratory reply**: The first replies from the sources are exploratory. These replies, containing data, flow downstream along the gradients set up by the interest. Along the gradients, each node caches the data message in their data cache (which, similar to the interest cache, is used to suppress duplicate messages and to prevent loops). The fact that movements through the gradients are actually merely time progressions of data-interest matching events along the communication links cannot be stressed enough.

3. **Gradient reinforcement**: On receiving the exploratory data messages, the sink stores them in its data cache, and according to some system-defined parameters (e.g. latency), the sink reinforces its neighbours which satisfy these parameters (e.g. neighbours which delivered the exploratory data messages with the lowest latency). These neighbours in turn reinforce their upper-stream neighbours, according to the same principle. The effect of reinforcement is system-defined, e.g. possibly bumping up the data output rate of the sources connected to the reinforced gradients, and hence the data transfer rate along the reinforced gradients. Note that the meaning of “x reinforces y” is equivalent to the meaning of “x reinforces the downstream gradient from y to x”. Future data messages from the sources only travel down these reinforced gradients, and with priority given to the more heavily rein-
forced gradients. That said, gradients can also be negatively reinforced for increasing performance, for load-balancing, or for gradient maintenance in response to topological changes, node failures, environmental effects etc.

All in all, the directed diffusion model provides the basic primitives for data communications in WSN. It uses caches for data-interest matching, to suppress duplicate messages and to prevent loops. It uses data aggregation to optimize bandwidth usage. As a result of performing only local interaction, nodes require little local storage and the resulting network is capable of self-repairing. However this also implies a trade-off for robustness and scalability with energy efficiency.

3. The LKHW Model

After an overview of directed diffusion, we describe LKHW in this section. There are two basic aspects to a tree-based multicast model like LKHW: (1) the key tree structure, and (2) the re-keying scheme. As the name implies, LKHW adopts the key tree structure of Logical Key Hierarchy (LKH) [22]. The re-keying scheme of LKHW is based on Wong et al’s group-oriented re-keying scheme [23] and an improvement of ours (described in Section 3.3.1). In Section 3.1, we describe the key tree structure. We then proceed to describe our strategy for group initialization in Section 3.2. Group dynamics and the associated aspects of re-keying are detailed in Section 3.3.

3.1. Key Tree

<table>
<thead>
<tr>
<th>n</th>
<th>Number of users in a multicast group</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Degree or a-riety of a LKH tree</td>
</tr>
<tr>
<td>h</td>
<td>Height of a LKH tree, i.e. $\lceil \log_a n \rceil$</td>
</tr>
<tr>
<td>$\mathcal{L}$</td>
<td>The set of hierarchical levels of a LKH tree, i.e. $\mathcal{L} = {0, \ldots, h - 1}$</td>
</tr>
<tr>
<td>$K'$</td>
<td>Refreshed version of a key K</td>
</tr>
<tr>
<td>$E_K(M)$</td>
<td>Encryption function that takes key K and plaintext M</td>
</tr>
<tr>
<td>$MAC_K(M)$</td>
<td>Message Authentication Code (MAC) function that takes key K and plaintext M</td>
</tr>
</tbody>
</table>

Table 1. Notation.

In LKH, (symmetric) keys, e.g. $K_0, \ldots, K_{15}$, $P_0, \ldots, P_{15}$ in Figure 1, are logically distributed in a tree rooted at the key distribution center (KDC). The leaves of the tree correspond to the users, e.g. $u_0, \ldots, u_{15}$ in Figure 1. By ‘user’, we mean a user process on a sensor node. Every user stores all the keys on its key path, i.e. the path from the leaf node (corresponding to the user) up to the root. These keys comprise the user’s key set,

$$\mathcal{K}_i = \{ K_j \mid j = S_a(l) + \left\lfloor \frac{i}{a^{n-l}} \right\rfloor, \forall l \in \mathcal{L} \}$$

where $i$ is the index of user $u_i$; keys of the form $K_j$ are as illustrated in Figure 1; $a$, $n$, $\mathcal{L}$ are as defined in Table 1; and $S_a(l)$ is the sum of the first $l$ terms of a geometric progression with ratio $a$.

For an example using Equation 1, $u_0$ stores key $P_0$, $K_7$, $K_3$, $K_1$, $K_0$, where in particular $P_0$ is the unique individual key $u_0$ shares with the KDC; $K_j$ is the group key that is shared by all users in the group and is used to encrypt all group communication traffic; and $K_7$, $K_3$, $K_1$ are so-called key encryption keys which serve the sole purpose of encrypting new keys during re-keying (cf Section 3.3). The KDC maintains the structure of the key tree and stores all the keys in the key tree.

Note that the tree illustrated here is for simplicity binary and balanced, but in reality key trees need not be. Therefore in general, if we use the notation in Table 1, then the total number of stored keys per user is $h + 1$, and the total number of stored keys by the KDC is $\frac{a^{h+1} - 1}{a - 1}$, or $\frac{an - 1}{a - 1}$ if the tree is balanced.

The main reason for using such a key tree compared with more traditional structure-less approaches such as Blundo et al’s [3] is that re-keying can be more efficiently executed. Re-keying is the operation that refreshes a subset of the keys in the key tree, when one or more users join or leave the group, in such a way that ensures added users are unable to decrypt past traffic, while evicted users are not able to decrypt future traffic – or in other words, ensures backward secrecy and respectively forward secrecy. Up to this point, only the basic LKH model has been described. Details of LKHW-specific group initialization and re-keying follow.

3.2. Group Initialization

Continuing with our previous example of tracking animals, now suppose that the sensor results have to be protected from potential poachers, confidential and authenticated group communication has to be established among the sources and the sink. To achieve this efficiently, we apply LKHW: directed diffusion sources play the role of multicast group members, whereas the sink plays the role of the KDC.

Protocol Before the normal directed diffusion process can begin, a secure group has to be established first, with the group initialization process. The rationale is that the confidentiality of the query a node posts to the other nodes is just as important as the confidentiality of the data supplied
by the nodes. From a system point of view, the protocol is:

1. $S \rightarrow *$: interest about interests to join
2. $u_i \rightarrow S$: interests to join
3. $S \rightarrow u_i$: data for joining
4. $S \rightarrow u_i$: encrypted normal interest, i.e. secure interest
5. $u_i \rightarrow S$: encrypted normal data, i.e. secure data

where $i = 0, \ldots, n - 1$. Since a source in the directed diffusion sense can become a sink in the group initialization phase by emitting an “interest to join”, we are careful to not refer $u_i$ ($i = 0, \ldots, n - 1$) as sources, but as users/members, and $S$ as the KDC in the discussion below to avoid confusion. A formal specification of the protocol can be found in our technical report [17], while the protocol is described in detail below:

1. **Interest about interests to join**: By sending out an exploratory “interest about interests to join” at step 1, $S$ finds out which among the nodes that are capable of the task(s) specified, are ‘interested’ in joining its secure group. This “interest about interests” would be cached in the nodes that received it, and would match any future interest expressed by whichever nodes that want to join the group. This “interest about interests to join”, like normal interests, creates an initial, exploratory set of gradients that diffuse across the network.

2. **Interests to join**: ‘Interested’ nodes reply with an interest to join, by declaring the task(s) they are capable of, the ID of the group they are joining, their key ID, a nonce and the MAC of the entire message (Figure 2b). These “interests to join” travel down the gradients created by the the previous “interest about interests to join”. The interesting observation here is that by originating interests, these ‘interested’ nodes have actually become both sources and sinks – they are sources for “interest about interests to join” but sinks to which $S$ is going to dispatch their respective “data for joining” (i.e. keying material that includes a member’s assigned index and key set). Similarly, $S$ is simultaneously a sink for “interest to join” and a source for “data for joining”.

3. **Data for joining**: After collecting enough requests or after a timeout, $S$ proceeds to supply $u_i$ ($i = 0, \ldots, n - 1$) with their assigned index (i.e. the $i$ in $u_i$) and key set. Note that prior to this stage $u_i$ does not know it is user $i$; $u_i$ is just a name by which we call a node consistently. From this point onward, $S$ can start dispatching normal interests encrypted with the group key $K_{0}$, called secure interests (Figure 2d); and $u_i$ ($i = 0, \ldots, n - 1$) can reply with data, encrypted too with the same group key (Figure 2e), called secure data. Consequently, in-networking processing and data aggregation can start to take place, securely.

**Required Extension of Directed Diffusion** It is important to note that secure interests and secure data are not encrypted as a whole. Instead, only task-specific values as depicted in Figure 2 are encrypted. This is to allow data-interest matching to be carried out on non-task-specific fields so that non-member nodes know how to forward secure data. Specifically, when a node receives a secure data:

- If the node sees that it is not in the group specified by the groupID of the data, the node would perform the usual matching algorithm on the groupID and the task; ignoring the encrypted task-specific tuples.
- Otherwise, if the node sees that it is in the the group specified by the groupID, the node will decrypt the task-specific tuples, and perform the usual matching algorithm on the groupID, the task and the task-specific tuples.

As a result of this extension, members of group groupID would potentially receive data that do not match their interest. However this is just an efficiency issue, the security of the data is not compromised.

![Figure 1. Logical key hierarchy ($a = 2, n = 16$).](image-url)
3.3. Group Dynamics

In this section, we describe the protocols for leave and join operations, starting with the leave operation.

3.3.1 Leave

When a source leaves the group, it can either be due to voluntary leave or forced eviction. Voluntary leave maybe the result of load control, or the leaving node’s self-awareness that it is exiting the region of sensing interest, or that it is in the process of being compromised. On the other hand, forced eviction maybe a result of intrusion detection that decides the node in question is no longer trustable. Or it may just be that the sink is no longer interested in the readings of the particular node.

Example Regardless of the reason, after a node, say \( u_v \) has left the group, to ensure forward secrecy all the keys on the path \( N_v \)-root need to be refreshed, where \( N_v \) is the key tree node from which \( u_v \) is detached. Evicting \( u_v = u_12 \) in Figure 1 implies that all the keys in key set \( K_{12} \), i.e. \( K_0 \), \( K_2 \), \( K_6 \) and \( K_{13} \) have to be refreshed. Our scheme is to refresh \( K_0 \), \( K_2 \), \( K_6 \) according to Equation 2:

\[
K_0' = H(K_2') = H^{(2)}(K_6') = H^{(3)}(K_{13}')
\]

We also need to forward-securely generate \( K_{13}' \), We call the set of \( K_0', K_2', K_6' \) and \( K_{13}' \) the refreshed key set, denoted \( \hat{K}_{2} = K_{12} \). The next step is to deliver these refreshed keys to the appropriate members other than \( u_{12} \) securely and efficiently. In LKHW,

- \( u_0, \ldots, u_7 \) would receive \( E_{K_0'}(K_0') \);
- \( u_8, \ldots, u_{11} \) would receive \( E_{K_2'}(K_2') \), and compute \( K_0' \) according to Equation 2;
- \( u_{14}, u_{15} \) would receive \( E_{K_{14}'}(K_6') \), and compute \( K_0', K_2' \) according to Equation 2;
- \( u_{13} \) would receive \( E_{P_{13}'}(K_{13}') \), and compute \( K_0', K_2', K_6' \) according to Equation 2.

Note that according to Figure 1, \( K_0 \) is at level 0, \( K_2 \) is at level 1 and so on, that is, only one key is refreshed at a level. If we denote \( \mathcal{K}_l \) as the transmitted key set at level \( l \) on the eviction of node \( u_v \), then \( \mathcal{K}_0 = \{ E_{K_0}(K_0') \} \), \( \mathcal{K}_1 = \{ E_{K_2}(K_2') \} \), \( \mathcal{K}_2 = \{ E_{K_6}(K_6') \} \), \( \mathcal{K}_3 = \{ E_{P_{13}'}(K_{13}') \} \).

Similarly we define \( \mathcal{K}_l \) as the recipient set at level \( l \) on the eviction of node \( u_v \), i.e. the set of members who receive \( \mathcal{K}_l \). So \( \mathcal{K}_0 = \{ u_0, \ldots, u_7 \} \), \( \mathcal{K}_1 = \{ u_8, \ldots, u_{11} \} \), \( \mathcal{K}_2 = \{ u_{14}, u_{15} \} \), \( \mathcal{K}_3 = \{ u_{13} \} \). Mathematical formalization of these concepts can be found in our technical report [17].

Protocol The only difference between voluntary leave and forced eviction is that for voluntary leave, the node, say \( u_v \) starts by flowing down a leave message to the sink. From then on, both voluntary and forced eviction are the

---

Figure 2. Message formats for group initialization where encrypted entries are marked as `<encrypted>`.

As mentioned, since \( S \) is both a sink and a source, the keying materials \( S \) dispatches, double as data and reinforcements. To be precise, \( S \) as a sink decides which neighbour to send the “data for joining” to, i.e. decides on which neighbour to reinforce based on some system-defined parameters. These reinforcements limit the direction/paths of transmissions, and hence increase efficiency. They also provide another advantage that will become clear in Section 3.3.2.

The catch is that in the original directed diffusion model, reinforcements are actually refined versions of the initial interest. In our case, “data for joining” do not qualify as reinforcements per se, but re-sending “interest about interests” as reinforcements is expensive. Therefore LKHW has to add on top of directed diffusion an extra layer of logic that treating “data for joining” as reinforcements. This explains the reinforce tuple in Figure 2c.
same. From a system point of view, the protocol is:

1. \( S \rightarrow * \) : interest about interests to re-key
2. \( \forall u \rightarrow S \) : interests to re-key
3. \( S \rightarrow \forall l^i : \mathcal{P}^i, \forall l \in \mathcal{L} \)

The following is a step-by-step description of the protocol:

1. **Interest about interests to re-key**: \( S \) broadcasts an “interest about interests to re-key”, the format of which is depicted in Figure 3a, where \( \text{evictIndex} \) specifies the index of the evicted node. This interest diffuses across the network, establishing a new set of gradients.

2. **Interests to re-key**: All members except \( u_v \), upon receiving the “interest about interests”, set off a timer with a random time-out value \( \rho_i \Delta \), where \( \rho_i \) is the time-out ratio, \( 0 \leq \rho_i \leq 1 \) and \( \Delta \) is the maximum time-out value. The rationale of using a random time-out value is basically to facilitate data aggregation but the details are explained later. Denote \( l_i \) as the \( \text{level} \) at which node \( u_i \) has to re-key, i.e. the re-key level of \( u_i \). Then for every member \( u_i \), after the time-out \( \rho_i \Delta \) has elapsed, \( u_i \) replies with an interest that indicates the transmitted key set \( \mathcal{P}^i \), it should receive. See Figure 3b for the message format and notice the use of the \( \text{level} \) tuple, which specifies \( l_i \) and essentially \( \mathcal{P}^i \). Notice further that there can be more than one \( \text{level} \) tuple, to cater for data aggregation. These interests travel down the gradients that have previously been established by \( S \)’s “interest about interests”.

Now we justify the use of random time-outs with Proposition 1:

**Proposition 1.** Given a member \( u_i \), which has \( N \leq n-2 \) upstream neighbours, and given that the \( N \) neighbours are also group members, assuming the the link latency is negligible, the probability that there exists at least one \( u_j \) among the \( N \) neighbours such that \( l_j \neq l_i \) and \( \rho_j < \rho_i \) is given by

\[
\frac{N}{N+1} \left[ 1 - \sum_{i=0}^{L_{\text{max}}} \left( (a-1)\frac{a^{h-l_i-1}}{N+1} \right) \left( \frac{n-1}{N+1} \right) \right]
\]

where \( L_{\text{max}} = \left\lfloor h - \log_a \left( \frac{a(N+1)}{a-1} \right) \right\rfloor \).

This is the probability that node \( u_j \) is able to aggregate at least one re-key level from one of its upstream neighbours (see proof in Appendix A). To see how significant the probability is, let us use a binary key tree \( (a = 2) \), and suppose there are 16 members in the group before eviction \( (n = 16, h = 4) \). Suppose for the particular node \( u_i \) we are considering, \( u_i \) has 3 neighbours \( (N = 3) \), then according to Proposition 1, \( u_i \) has a probability of 71% for performing at least one aggregation.

To further illustrate the details, we borrow the help of a sample topology depicted in Figure 4, where node \( u_{12} \) is in the process of being evicted. Pick a random node, say \( u_7 \) in Figure 4. According to Figure 1, \( l_7 = l_0 = 0 \), and \( l_8 = 1 \). When \( u_8 \) receives \( u_7 \)’s “interest to re-key”, \( u_8 \) will cache and re-broadcast \( u_7 \)’s interest because \( l_7 \neq l_8 \). Similarly when \( u_0 \) receives \( u_7 \)’s and \( u_8 \)’s interest, \( u_0 \) will cache both interests, but only re-broadcast \( u_8 \)’s interest because \( l_0 = l_7 \neq l_8 \). Eventually, \( S \) receives \( l_0 = 0 \) and \( l_8 = 1 \) from \( u_0 \) (ignoring the \( \text{level} \) of other nodes than \( u_7, u_8, u_0 \) for this example).

Since \( u_8 \) needs to send \( u_7 \)’s and its own interest, and similarly, \( u_0 \) needs to send \( u_8 \)’s and its own interest, both \( u_8 \) and \( u_0 \) should ideally aggregate the interests they have to send, before sending them. Proposition 1 suggests that this will happen with a high probability.

3. **Data for re-keying**: Continuing with the example, \( S \) then knows \( \mathcal{P}^0_{12} \) and \( \mathcal{P}^1_{12} \) are needed and would deliver them upstream in the direction of \( u_0, u_0 \), remembering that \( \mathcal{P}^0_{12} \) and \( \mathcal{P}^1_{12} \) are needed, would likewise deliver the key sets upstream. Finally \( u_8 \) would send only \( \mathcal{P}^0_{12} \) upstream since it is only \( u_7 \) who needs it (but of course \( u_0 \) does not know it is \( u_7 \) who needs it – \( u_8 \) only knows some node needs it). The importance of the data cache cannot be emphasized more here. For example, if \( u_0 \) has sent and got \( \mathcal{P}^0_{12} \) before \( u_7 \)’s “interest to re-key” arrives, \( u_0 \) can easily retrieve \( \mathcal{P}^0_{12} \) from its data cache and hand it over to \( u_7 \) without going through \( S \) again.

![Figure 3. Message formats for leaving.](image_url)

**3.3.2 Join**

For a system to be scalable in general and for sensing tasks that require dynamic adjustment of sensing resolution in particular, the capability to add computing power dynamically to the network is essential. The join operation of LKHW makes this possible.
 Protocol  As it happens in many secure multicast schemes, the join protocol and the leave protocol are asymmetrical with the join protocol being more efficient because the joining node does not know any existing key of the system. Similar to a leave operation, a join operation can either be initiated by $S$ or by whichever node $u_v$ that wants to join. Suppose $S$ requires a higher sensing resolution, $S$ can start sending “interest for interests to join” again. Moreover, recall that in Section 3.2, $S$’s “interest about interests to join” is cached in the network, so for any $u_v$ which sends out its “interest to join”, its “interest to join” would syntactically match the cached “interest about interests to join” — with a catch: The format of Figure 2b specifies the ID of the group a node wants to join. As it is unrealistic for a node to find out the group ID beforehand, it should just omit the groupID tuple in its “interest to join” and the protocol will still work. From a system point of view, the protocol is:

1. $u_v \rightarrow \ast$: interest to join
2. $S \rightarrow u_v$: $\mathcal{K}$
3. $S \rightarrow u_i$: seed, $\forall i \in \{0, \ldots, n-1\} \setminus \{v\}$

where “seed” is the key regeneration seed. Detailed explanation follows:

1. **Interest to join**: Section 3.2 mentioned the dual role of “data for joining”, i.e. as data cum reinforcements. We will clarify the advantage of doing so here. First, it is clear that when $u_v$ broadcasts its “interest to join”, in the directed diffusion model, the interest has to diffuse across the network. Now the good news is that since $S$ has planted the seed of reinforced gradients in the group initialization phase, $u_v$’s “interest to join” can simply take advantage of the reinforced gradients, i.e. any node that receives the “interest to join” would flow it down the reinforced gradients if it happens to be on one or more of the reinforced gradients.

See for example in Figure 4, the possible reinforced gradients that take $u_v = u_{12}$’s “interest to join” to $S$. In this case, $u_{12}$’s neighbors $u_{13}, u_5, u_9$ send $u_{12}$’s “interest to join” only along the reinforced gradients.

2. **Data for joining**: Once $S$ receives $u_v$’s message, $S$ would, as in the group initialization phase, dispatch $\mathcal{K}$ (among other things) to $u_v$ as data cum reinforcement.

3. **Seed diffusion**: The key regeneration seed is used by existing members to refresh their respective key sets. The seed does not need to be encrypted but must be authenticated with the group key. Every member $u_i$ ($i \neq v$) that receives the seed would derive its new key set $\mathcal{K}_i'$ from its existing key set $\mathcal{K}_i$ as follows:

$$\mathcal{K}_i' = \{K'|K' = MAC_K(\text{seed}), \forall K \in \mathcal{K}_i\} \tag{3}$$

The forward security of the seed can be ensured by Bellare and Yee’s construction [2].

4. Performance Evaluation

In this section, we perform a theoretical evaluation of the performance of LKHW. The performance criterion for WSN is energy efficiency instead of throughput. We note that while it is a convention to evaluate computational as well as communication cost as benchmarks for secure multicast schemes, for WSN however communication cost dominates computational cost by typically three orders of magnitude [6], therefore we only consider communication cost. For large messages, energy consumption is proportional to the message length, however for small messages, the transmission overhead and hence the number of messages has a more significant effect on the energy cost. Therefore we will take both message length and number of messages into account. Moreover, the energy for reception is not insignificant, at least for the transceiver we are using, RF Monolithics TR1001 [19], it can be as high as 40% of the energy required for transmission. We denote the ratio of reception power to transmission power as $r$ for the discussion below. We will discuss the leave and join operation separately.

**4.1. Leave**

Let us consider the sink $S$ and the sources $u_i$ ($i = 0, \ldots, n-1$, $i \neq v$, $v$ is the index of the evicted node) separately. In LKHW, for $S$, the number and the total length of
messages sent, and hence energy cost is network topology-dependent. For example, in Figure 4, S would potentially receive requests for the transmitted key set at level 0, $\mathcal{F}_u^0$, from all its neighbours, i.e. $u_0$, $u_1$, $u_2$ and $x_0$. Directed diffusion dictates that S would send, instead of broadcast, $\mathcal{F}_u^0$ as data replies to $u_0$, $u_1$, $u_2$ and $x_0$ individually. So S would send $\mathcal{F}_u^0$ four times. On the other hand, requests for $\mathcal{F}_u^3$ would potentially only come from $u_1$ and $u_2$, and hence S would send $\mathcal{F}_u^3$ two times. Intuitively, we can certainly do better than what directed diffusion allows us to. Observe that instead of unicasting to each requesting neighbour, we might be able to save energy by broadcasting all the key sets $\mathcal{F}_u^l$, $l = 0, \ldots, h-1$ at once because each key set $\mathcal{F}_u^l$ would have to be sent at least once anyway. To formalize our argument, suppose S receives “interests to re-key” from $N'$ out of a total of $N$ neighbours. Using unicasts, the energy cost for S to dispatch the requested key sets is

$$E_{unicasts} = (1 + r) \sum_{i=1}^{N'} C_i E_{ks} + (1 + r) N' E_o$$  \hspace{1cm} (4)

where $C_i$ is the number of key sets requested by neighbour $i$, $i = 1, \ldots, N'$; $E_{ks}$ is the energy associated with sending a key set; and $E_o$ is the energy associated with the overhead of a single “data for re-keying" message. Notice that apart from considering the energy cost for S, we also consider the energy cost incurred on S’s neighbours for listening, hence the terms containing $r$. The energy cost for broadcasting all key sets at once is

$$E_{broadcast} = (1 + Nr) h E_{ks} + (1 + Nr) E_o$$  \hspace{1cm} (5)

The terms $N'$, $N$, $\sum_{i=1}^{N'} C_i$ are entirely topology-dependent, hence there is no way of determining whether $E_{unicasts}$ is larger or smaller than $E_{broadcast}$ without considering the topology. Now, we can make S adopt this adaptive policy: after collecting enough “interests to re-key”, compute $E_{unicasts}$ and $E_{broadcast}$, if $E_{unicasts} > E_{broadcast}$, then aggregate and unicast the requested key sets to each requesting neighbour, otherwise broadcasts all key sets at once.

In practice, we expect $N' \approx N$, and the term $(1 + r) \sum_{i=1}^{N'} C_i$ often be larger than $(1 + Nr) h$, in which case, S will choose to broadcast. Since $N$ is related to the network density which does not vary much for the same type of applications, intuitively then, $E_{broadcast}$ increases with the number of levels $h$, which is logarithmic to the group size. Therefore $E_{broadcast}$ scales logarithmically with the group size.

This adaptive policy does not break directed diffusion, because it needs only apply to S. In the event that S decides to use broadcast instead of unicasts, the neighbours of S which have not submitted any request for key sets but have received S’s broadcast, would just drop the broadcast data, since there is no matching interest.

Now we consider the sources. The upper bound of the energy cost for a source $u_i$ applies when the $N$ upstream neighbours of $u_i$ time out later than $u_i$, and the re-key levels received from these $N$ neighbours span the set $\mathcal{Z}$. The upper-bound energy cost is therefore:

$$E_{upper} = r \sum_{i=1}^{N} (C_i E_l + E_o) + (E_l + E_o) + (h - 1)(E_l + E_o)$$

$$= (h + r \sum_{i=1}^{N} C_i) E_l + (h + Nr) E_o$$

$$\approx (h + Nr) E_0$$  \hspace{1cm} (6)

where $C_i$ is the number of re-key levels sent by neighbour $i$, $i = 1, \ldots, N$; $E_l$ is the energy required for sending a re-key value; and $E_o$ is the energy associated with the overhead of a single “interest to re-key” message. Note that the number of bits for representing a re-key level is potentially very small compared with the the overhead of an “interest to re-key” message (Figure 3) which is taken into account by $E_o$, or in other words $E_l \ll E_o$. $N$ is related to the network density which does not vary much for the same type of applications. Using the same logic as before, $E_{upper}$ scales logarithmically with the group size. On the other hand, the lower bound applies when $u_i$ receives no request from its neighbours, and needs only to send its own re-key level:

$$E_{lower} = (E_l + E_o) \approx E_0$$  \hspace{1cm} (7)

4.2 Join

The case of join is much simpler because it simply involves dispatching the relevant key set to the new member, and flooding the network with the key regeneration seed. The transmission energy cost for $S$ alone is independent of the topology:

$$E = E_{ks} + E_{seed} + E_{o(unicast)} + E_{o(broadcast)}$$  \hspace{1cm} (8)

where $E_{seed}$ is obviously the energy for broadcasting the seed; $E_{o(unicast)}$ and $E_{o(broadcast)}$ are associated with the overhead of a single unicast and a single broadcast respectively. For the new joining member, the cost involves dispatching its “interest to join” and receiving the allocated key set. The existence of reinforced gradients ensures that the energy cost for propagating the “interest to join” across the network to the sink is low. For existing members, the cost primarily involves receiving the seed.
5. Related Work

LKHW is, as far as we know, the first secure group communication scheme to reap the benefits of directed diffusion. We have already reviewed directed diffusion [7, 10] in Section 2 in details. In the following, we will compare LKHW with other group communication schemes from two perspectives, the distributed approaches and the hierarchical approaches.

Distributed Approaches  Also called conference key distribution schemes, the approaches in this category try to solve the key agreement problem, i.e. the problem of deriving a secure common key among $n$ users. Ingermasson et al. [9] are the first to extend the Diffie-Hellman (DH) problem to groups. Burmester and Desmedt [4] correct Ingermasson et al.‘s security flaw by using cyclic instead of symmetric functions. Just and Vaudenay [12] patch the key authentication flaw of the Burmester-Desmedt scheme and generalize it. Steiner et al. [20, 21] begin to consider dynamic peer groups instead of a fixed number $n$ of users, and introduce CLIQUES, a suite of contributory key agreement protocols which requires less communication than the Burmester-Desmedt scheme does. Ateniese et al. [1] provide CLIQUES with authentication properties. The problem with these conference key distribution schemes is that they require a lot of exponentiation computations that are prohibitively expensive for sensors [6, 8], and re-keying is inefficient. Blundo et al. [3] did the pioneering investigative work on the storage requirements of $k$-secure $t$-conference key distribution scheme. The proposed scheme is information-theoretically secure, and does not require exponentiation, but requires $O(n^t)$ amount of keying material per node (where $n$ is the total number of users), which is impractically large. The conclusion is that both DH-based and information-theoretically secure schemes have their share of scalability problems.

Hierarchical Approaches  By imposing a hierarchical structure – binary tree being the mainstream – on dynamic peer groups, hierarchical approaches have been able to achieve better scalability than the distributed approaches. There is the pioneering work by Wallner et al. [22] who propose the logical key hierarchy (LKH) model. The LKH model is scalable because the total number of keys in the system is linearly proportional to the number of users, and both the number of keys per user and the number of messages required to manage group dynamics, are logarithmic in the number of users. Wong et al. [23] investigate and compare three re-keying paradigms, i.e. key-oriented, user-oriented and group-oriented re-keying. The re-keying paradigm of LKHW is an improved form of group-oriented re-keying. McGrew et al. [13] introduce the one-way function tree (OFT), where the KDC needs only dispatch $\lceil \log_2 n \rceil$ keys during re-keying, the same number as LKHW’s, instead of the $2\lceil \log_2 n \rceil$ required by the basic LKH model. Canetti et al. [5] replace McGrew et al.’s non-standard cryptographic primitive with pseudorandom generator. The EHBT scheme proposed by Rafaeli et al. [18] uses a non-standard cryptographic primitive similar to OFT’s, i.e. one-way function of the form $h(\text{key} \oplus \text{index})$ for key refreshment. In all the schemes mentioned so far, affected group members need to receive refreshed keys from the KDC during user-join events. Members in Perrig et al.’s ELK [15] however avoid that overhead by locally and periodically re-generating their keys. Di Pietro et al. [16] provide further improvement by exploiting pseudorandom function and the “level-awareness” of a node in a scheme called LKH++.

LKHW requires the same number of keys for the KDC, i.e. $2n - 1$; and the same number of keys for a member, i.e. $h + 1$, as OFT, EHBT, ELK and LKH++ do. LKHW’s handling of user-join events might seem inefficient since a seed is flooded across the network, but due to the multihop nature of WSN (where every node behaves as a router) and the data caching property of directed diffusion, this method is in fact not only efficient but also robust.

6. Conclusion and Future Work

We have presented a secure group communication scheme that is optimized for directed diffusion. The scheme is independent of the underlying key management architecture. We have given the details of handling group dynamics. In terms of efficiency, the re-keying overhead in terms of energy cannot be concretely quantified without considering the topology, but it is found to be approximately logarithmic to the group size. In fact, the conventional evaluation methodology is no longer apt in the WSN context. In view of this, we have presented a new evaluation methodology that is entirely based on energy efficiency. As part of our future work, we would address multiple join and multiple leave operations, as well as the efficient re-balancing of the LKH tree in the directed diffusion context.

References


Appendix A. Proof of Proposition 1

Denote $X$ as the event that $u_i$ can perform at least one aggregation, i.e. that there exists at least one $u_j$ among the $N$ neighbours such that $l_j \neq l_i$ and $\rho_j < \rho_i$. Further denote $A$ as the event that all $N$ neighbours have the same re-key level as $u_i$’s; and $B$ as the event that all $N$ neighbours have a time-out greater or equal to $u_i$’s. Then,
\[
Pr[X] = 1 - Pr[A] - Pr[B] + Pr[AB]
\]
\[
= 1 - Pr[A] - Pr[B] + Pr[A]Pr[B]
\]
(9)
since events $A$ and $B$ are mutually independent. Recall that $\rho_i$ is as defined in Section 3.3.1, the time-out ratio of $u_i$. We first derive $Pr[B]$, assuming $\rho_i$ is uniformly distributed between 0 and 1:
\[
Pr[B] = \int_{0}^{1} (1 - \rho_i)^N d\rho_i = \frac{1}{N+1}
\]
(10)
reducing (9) to the simpler form of (11).
\[
Pr[X] = \frac{N}{N+1}(1 - Pr[A])
\]
(11)
As for $Pr[A]$, we observe that the number of nodes, $n'$, having the same re-key level $l$ to be:
\[
n'(l) = (a-1)a^{h-l-1}
\]
expressed as a function of $l$. For example, in Figure 1, the nodes sharing the re-key level of $l = 1$ during the eviction of $u_{12}$ are $u_8, \ldots, u_{11}$, or equivalently $n' = 4$. Observe that $n'$ decreases as $l$ increases, so there exists a maximum value of $l$, $L_{max}$ such that $n'(L_{max}) = N + 1$, or
\[
L_{max} = h - \log_a \left[ \frac{a(N+1)}{a-1} \right]
\]
(12)
Constraining $L_{max}$ to integer values, we have
\[
L_{max} = \left\lfloor h - \log_a \left[ \frac{a(N+1)}{a-1} \right] \right\rfloor
\]
(12)
When $l > L_{max}$, at least one of the $N$ neighbours of $u_i$ will have a different re-key level, a condition precluding event $A$. Therefore,
\[
Pr[A] = \sum_{l=0}^{L_{max}} \binom{N-1}{n-1} \frac{(N+1)}{N+1}
\]
(13)
where $n$ is the total number of nodes in the group before eviction. Substituting $Pr[A]$ in (11) with (13), we get Proposition 1.