LKHW: A Directed Diffusion-Based Secure Multicast Scheme for Wireless Sensor Networks

Roberto Di Pietro Luigi V. Mancini
Università di Roma "La Sapienza",
Dip.to di Informatica,
Via Salaria 113, 00198-Roma, Italy.
{dipietro, mancini}@dsi.uniroma1.it

Yee Wei Law Sandro Etalle Paul Havinga *
Department of Electrical Engineering,
Mathematics and Computer Science,
University of Twente, PO Box 217,
7500 AE Enschede, The Netherlands.
{ywlaw, etalle, havinga}@cs.utwente.nl

July 10, 2003

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 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 5 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 [28] or DSR [22]), 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 (i.e. not mobile, so that a new route does not need to be set up everytime 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* [21] that facilitates *attribute-based naming* and *in-network processing*, significant traffic reduction is possible (Heidemann et al. report a reduction of 42% [17]). 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 es-

*This work is partially supported by the EU under the IST-2001-34734 EYES project.

sential 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.

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 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 and security analysis in Section 5, while we discuss related work in Section 6. Finally, Section 7 gives the conclusion and some ideas for future work.

2 Overview of Directed Diffusion

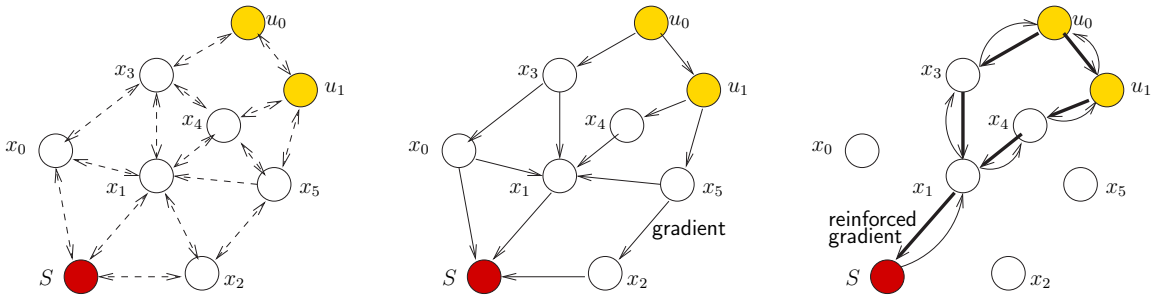
Basic Terminology Picture the classic directed diffusion scenario in which a WSN is deployed in a wilderness refuge to track animals [17, 21]. A tracking request represents an *interest*. The node that broadcasts the initial *exploratory* interest is the *sink*, i.e. the final destination of the requested 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 particlar 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. [21]. 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 re-broadcasting 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 matching events, i.e. matching of data with interest, 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.



(a) Interest diffusion (b) Exploratory reply (c) Gradient reinforcement

Figure 1: The three phases of directed diffusion.

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 reinforced gradients. That said, gradients can also be *negatively reinforce* for increasing performance, for load-balancing, or for gradient maintenance in response to topological changes, node failures, environmental effects etc.

The concepts surrounding gradients are vital to the understanding of the ensuing discussion, and hence would be illustrated in detail with an example (cf Figure 1). In the following, we introduce the notation that will be employed throughout the rest of the paper.

Notation

- Denote the sink as S and an interest as ζ. Specifically, the i-th distinct interest is ζ^i, where $i = 0, 1, \ldots$. A copy of ζ^i broadcast (not necessarily originated) by a node x is $\zeta^i_{\vec{x}}$.

- We express a gradient as a tuple of the form (ζ^i, x, γ^i) where ζ^i is the interest, x is the identifier of the downstream neighbour the gradient is directed at, and γ^i is the gradient value. Note that Intanagonwiwat et al [21] define gradient as a pair of direction and a 'generic value', but implement the 'generic value' as a tuple of direction, data rate and duration. Our 'gradient value' maps to their 'data rate', but we omit duration for the simplicity of our discussion. Intuitively, the gradient value γ^i is derived from the interest ζ^i, and we write this relationship as $\gamma^i = g(\zeta^i)$.

- For clarity we denote nodes that match the interest as u_i and nodes that do not match the interest as x_i (where $i = 0, 1, \ldots$).

- The i-th data message from a source u is denoted as ξ^i_u.

Gradient Setup The example begins by S broadcasting the 0-th interest $\zeta^0_{\vec{S}}$. When node x_0 (Figure 1) receives $\zeta^0_{\vec{S}}$, x_0 caches the tuple (ζ^0, S, γ^0) in its interest cache. Note that for the initial interest, the gradient value $\gamma^0 = g(\zeta^0)$ should be a low value. Moreover, before x_0 re-broadcasts the interest, if x_0 receives $\zeta^0_{\vec{x_1}}$ from x_1, x_0 would also cache the tuple (ζ^0, x_1, γ^0). Now x_0 re-broadcasts the interest as $\zeta^0_{\vec{x_0}}$ to all its neighbours. S and x_1 simply ignore $\zeta^0_{\vec{x_0}}$. If $\zeta^0_{\vec{x_1}}$ arrives after $\zeta^0_{\vec{x_0}}$ has been broadcast, x_0 would just ignore $\zeta^0_{\vec{x_1}}$. All other nodes behave likewise. One important point to summarize from this is that *all nodes only remember downstream interests.*

Gradient Reinforcement Suppose u_0 matches the interest. As a source, u_0 replies with data message $\xi^0_{u_0}$, which travels down the gradients to S, and suppose x_0 is unavailable and S receives $\xi^0_{u_0}$ only from x_1 and x_2. Depending on some system-dependent parameters (e.g. data rate, latency, energy etc.), S would choose to *reinforce* either x_1 or x_2. *Reinforcement is actually the selective/directional sending of a refined version of the initial interest.* If S chooses to reinforce x_1, then S would unicast ζ^1 to x_1, and x_1 would update the corresponding gradient entry in its interest cache from (ζ^0, S, γ^0) to (ζ^1, S, γ^1), where $\gamma^1 > \gamma^0$. At the same time, without reinforcement, x_2's gradient entry remains as (ζ^0, S, γ^0); x_5's gradient entries remain as (ζ^0, x_1, γ^0) and (ζ^0, x_2, γ^0). Future data messages from u_0 and u_1

would only travel down reinforced gradients, i.e. through nodes that store a gradient value larger than a certain system-defined threshold τ, where $\gamma^1 > \tau > \gamma^0$.

Multipath Delivery After x_4 has reinforced u_1, u_1 has to decide whether to reinforce u_0 since apart from x_3, $\xi^0_{u_0}$ also passes through u_1. Since u_1 does not know whether there are any other downstream gradients from u_0 to S, u_1 has to reinforce u_0. Consequently, u_0 have two reinforced gradients to S: one via x_3 and one via u_1. In other words, multipath delivery is inherent (but not inevitable) in the directed diffusion model. In fact, Ganesan et al. [15] have a multipath extension of directed diffusion. LKHW can be implemented on top of either the original or the multipath version of directed diffusion, but in our discussion, we only focus on the original version for simplicity's sake.

Data Aggregation When u_0's data messages pass through u_1, u_1 can potentially, instead of sending u_0's messages and its own messages separately, aggregate the data and perform some in-network processing before dispatching them downstream, thus saving bandwidth and energy. Data aggregation is an essential part of directed diffusion.

Gradient Maintenance The dynamics of WSN are characterized, among other things, by the addition and expiration of nodes, node mobility and link volatility. When for some reason (e.g. relative movement, environmental jamming, energy depletion) the link u_1-x_4 starts to deteriorate, while the link u_1-x_5 is still good, we would logically want x_5 to take over x_4's work of routing u_1's data messages toward S. Unless there is a feedback channel from x_4 to u_1, u_1 cannot tell when the link is failing. This is an important point because it means only downstream nodes can take appropriate measure. Assuming data rate is the criterion by which x_4 observes whether the link u_1-x_4 is failing or has failed, when x_4 detects much lower data rate than normal or even zero data rate, x_4 re-sends the interest ζ^i stored in its interest cache to its neighbours except x_1. On receiving $\zeta^i_{\vec{x_4}}$, x_5 starts routing u_1's data messages. It goes without saying that x_4 would then reinforce x_5.

In conclusion, 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) [39]. The re-keying scheme of LKHW is based on Wong et al's group-oriented re-keying scheme [40] 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

n	Number of users in a multicast group
a	Degree or a-rity of a LKH tree
h	Height of a LKH tree, i.e. $\lceil log_a n \rceil$
$\mathscr{L}$	The set of hierarchical levels of a LKH tree, i.e. $\mathscr{L} = \{0, \ldots, h-1\}$
$A\|B$	Concatenation of A and B, where A and B are strings
K'	Refreshed version of a key K
$E_K(M)$	Encryption function that takes key K and plaintext M
$MAC_K(M)$	Message Authentication Code (MAC) function that takes key K and plaintext M

Table 1: Notation.

In LKH, keys (symmetric keys unless stated otherwise, e.g. $K_0, \ldots, K_{15}$, $P_0, \ldots, P_{15}$ in Figure 2) 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 2). By 'user', we mean a user process on a sensor. 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*,

$$\mathscr{K}_i = \{K_j \mid j = j(i,l) = S_a(l) + \left\lfloor \frac{i}{a^{h-l}} \right\rfloor, \forall l \in \mathscr{L}\} \quad (1)$$

where i represents user u_i; keys of the form K_j are as illustrated in Figure 2; a, n, $\mathscr{L}$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, i.e.

$$S_a(l) = \begin{cases} 0 & \text{if } l = 0 \\ \sum_{i=0}^{l-1} a^i & \text{if } l > 0 \end{cases} \quad (2)$$

In Equation 1, although it may not be obvious now, we find it useful to express j as a function of i and l.

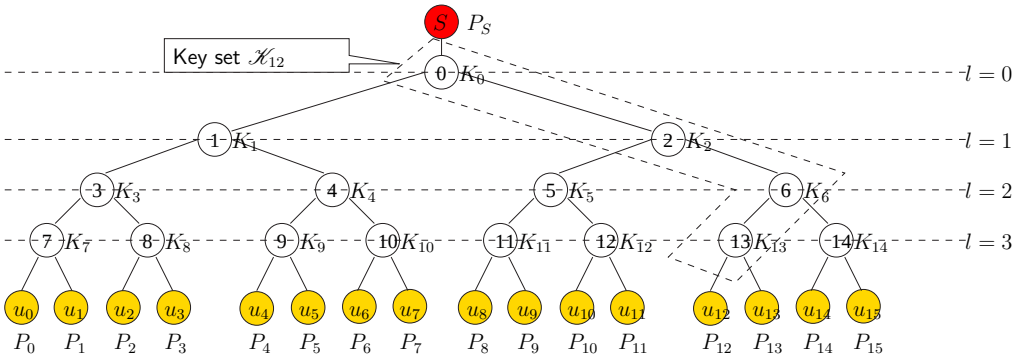


Figure 2: Logical key hierarchy ($a = 2, n = 16$).

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_0 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 [6] 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 are treated as multicast group members, whereas the sink is treated as the KDC.

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 algorithm 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. Formally, the protocol between a user u_i and a KDC S is:

Protocol 1

1. $S \rightarrow u_i$: T, GID, N_S
2. $u_i \rightarrow S$: $T, GID, ID, N_{u_i}, MAC_{P_i}(T|GID|ID|N_{u_i})$
3. $S \rightarrow u_i$: $GID, ID, N_S, E_{P_i}(i|C(\mathscr{K}_i))$, $MAC_{P_i}(GID|ID|N_S|N_{u_i}|E_{P_i}(i|C(\mathscr{K}_i)))$
4. $S \rightarrow u_i$: secure interest
5. $u_i \rightarrow S$: $E_{K_0}(D), MAC_{K_0}(N_S|E_{K_0}(D))$

where

- T specifies the task(s) that the sources should be capable of.
- GID is the group ID.
- ID is a generic identifier that identifies the key P_i. For example, it can be a node ID if the underlying key management architecture is such that a unique key is bound to a unique node ID [26, 30]; or it can be a key ID in a random key pre-distribution

model [10, 14]. As long as a shared key P_i exists between S and u_i, the protocol will succeed. We are not particular about the key pre-distribution model but we do assume that there is *no* single system-wide key. To emphasize, *LKHW is independent of the underlying key pre-distribution model.*

- N_{u_i} and N_S are nonces.
- i is the index S assigns to u_i.
- $C(\mathscr{K}_i)$ is a concatenation of all the keys in $\mathscr{K}_i$, i.e. the key set of u_i as expressed by Equation 1.
- D stands for an arbitrary piece of data that u_i sends to S in response to S's interest.

The protocol has been verified using the protocol verifier CoProVe (`http://wwwes.cs.utwente.nl/24cqet/`) but the security analysis of the protocol is deferred to Section 5. According to standard security practice, it is to be understood that the K's in $E_K(\cdot)$ and $MAC_K(\cdot)$ are in fact two different keys K^{enc} and K^{mac} derived from K [2]. The message at step 4 can piggyback on the message at step 3 for efficiency. The problem with protocol specification is that formal and actual operators cannot be shown, so Figure 3 is given for this purpose. Note how "interest to join" structurally matches "interest about interests to join" and how "data for joining" matches "interest to join".

Detailed description of the group initialization algorithm follows:

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 (cf Figure 1a).

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. 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 its 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 of u_i) and their respective 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 3d); and u_i ($i = 0, \ldots, n-1$) can reply with data, encrypted too with the same group key (Figure 3e), called *secure data*. Consequently, in-networking processing and data aggregation can start to take place, securely.

```
class EQ interest
task EQ ...
(more "task EQ ..." tuples)
groupID IS ...
```

(a) Interest about interests to join

```
class IS interest
task IS ...
groupID EQ ...
id IS ...
nonce IS ...
mac IS ...
```

(b) Interest to join

```
class IS data
groupID IS ...
id EQ ...
nonce IS ...
index IS ...
keyset IS <encrypted>
reinforce IS true
mac IS ...
```

(c) Data for joining

```
class IS interest
groupID IS ...
task EQ ...
task-specific1 EQ <encrypted>
task-specific2 EQ <encrypted>
(other encrypted task
 -specific tuples...)
mac IS ...
```

(d) Sample secure interest

```
class IS data
groupID EQ ...
task IS ...
task-specific1 IS <encrypted>
task-specific2 IS <encrypted>
(other encrypted task
 -specific tuples...)
mac IS ...
```

(e) Sample secure data

Figure 3: Message formats for group initialization (encrypted entries are marked as < `encrypted` >.

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 3 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.

As mentioned, since S is both a sink and a source, the keying materials S dispatches, double as data and reinforcements. To be precise, not only a "data for joining" (Figure 3c) structurally matches an "interest to join" (Figure 3b), but also 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 treats "data for joining" as reinforcements. This explains the `reinforce` tuple in Figure 3c.

3.3 Group Dynamics

In this section, we describe the algorithms for leave and join operations, starting with the leave operation. Multiple leaves and multiple joins are deferred to a later paper.

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. Below, we illustrate an example before giving the mathematical formalization.

Evicting $u_v = u_{12}$ in Figure 2 implies that all the keys in key set $\mathscr{K}_{12}$, 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 3:

$$K'_0 = H(K'_2) = H^{(2)}(K'_6) = H^{(3)}(K'_{13}) \tag{3}$$

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 $\mathscr{R}_v = \mathscr{R}_{12}$. The next step is to deliver these refreshed keys to appropriate members other than u_{12} securely and efficiently. In LKHW,

- $u_0, \ldots, u_7$ would receive $E_{K_1}(K'_0)$;
- $u_8, \ldots, u_{11}$ would receive $E_{K_5}(K'_2)$, and compute K'_0 according to Equation 3;
- u_{14}, u_{15} would receive $E_{K_{14}}(K'_6)$, and compute K'_2, K'_0 according to Equation 3;
- u_{13} would receive $E_{P_{13}}(K'_{13})$, and compute K'_6, K'_2, K'_0 according to Equation 3.

Note that according to Figure 2, 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 $\mathscr{T}_v^l$ as the *transmitted key set* at level l on the eviction of node u_v, then $\mathscr{T}_{12}^0 = \{E_{K_1}(K'_0)\}$, $\mathscr{T}_{12}^1 = \{E_{K_5}(K'_2)\}$, $\mathscr{T}_{12}^2 = \{E_{K_{14}}(K'_6)\}$, $\mathscr{T}_{12}^3 = \{E_{P_{13}}(K'_{13})\}$.

Similarly we define $\mathscr{U}_v^l$ as the *recipient set* at level l on the eviction of node u_v , i.e. the set of members who receive $\mathscr{T}_v^l$. So $\mathscr{U}_{12}^0 = \{u_0, \ldots, u_7\}$, $\mathscr{U}_{12}^1 = \{u_8, \ldots, u_{11}\}$, $\mathscr{U}_{12}^2 = \{u_{14}, u_{15}\}$, $\mathscr{U}_{12}^3 = \{u_{13}\}$.

Mathematical Formalization We now proceed to define the aforementioned notions mathematically. First the refreshed key set is

$$\mathscr{R}_v = \{K'_{j(v,l)} \mid K'_{j(v,l)} = \begin{cases} H(K'_{j(v,l+1)}) & \text{if } l = 0, \ldots, h-2 \\ \text{regenerated} & \text{if } l = h-1 \end{cases}, \forall l \in \mathscr{L}\} \tag{4}$$

where function $j(v,l)$ is as defined in Equation 1. The transmitted key set at level l, $\mathscr{T}_v^l$ is

$$\begin{aligned}\mathscr{T}_v^l = \{&E_\chi(K'_{j(v,l)}) \mid \\ &\chi = \begin{cases} K_k & \text{if } l = 0,\ldots,h-2 \\ P_{k-a^h+1} & \text{if } l = h-1 \end{cases}, \\ &k = aj(v,l)+m, \\ &\forall m \in \{1,\ldots,a\} \setminus \left\{ \left\lfloor \frac{v}{a^{h-l-1}} \right\rfloor - a \left\lfloor \frac{v}{a^{h-l}} \right\rfloor + 1 \right\}\}\end{aligned} \tag{5}$$

Lastly, the recipient set, $\mathscr{U}_v^l$ can be expressed by

$$\begin{aligned}\mathscr{U}_v^l = \{&u_r \mid r = a^{h-l-1}\left(a\left\lfloor \frac{v}{a^{h-l}} \right\rfloor + m - 1\right) \\ &\ldots a^{h-l-1}\left(a\left\lfloor \frac{v}{a^{h-l}} \right\rfloor + m\right) - 1, \\ &\forall m \in \{1,\ldots,a\} \setminus \left\{ \left\lfloor \frac{v}{a^{h-l-1}} \right\rfloor - a \left\lfloor \frac{v}{a^{h-l}} \right\rfloor + 1 \right\}\}\end{aligned} \tag{6}$$

Algorithm The only difference between voluntary leave and forced eviction is that for voluntary leave, the node, say u_v starts by flowing down an eviction message to the sink. Otherwise, from then on, both voluntary and forced eviction are the same. From a system point of view, the algorithm is:

1. $S \rightarrow *$: interest about interests to re-key
2. $\mathscr{U}_v \rightarrow S$: interests to re-key
3. $S \rightarrow \mathscr{U}_v^l$: $\mathscr{T}_v^l, \forall l \in \mathscr{L}$

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

1. **Interest about interests to re-key**: S broadcasts an "interest about interests to re-key", the format of which is depicted in Figure 4a, where `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, i.e. $u_i (i = 0,\ldots,n-1, i \neq v)$, upon receiving the "interest about interests", set off a timer with a random time-out value $\rho_i\Delta$, where ρ_i is the time-out ratio, $0 \leq \rho_i \leq 1$ and Δ 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 `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 specifies the transmitted key set, $\mathscr{T}_v^{l_i}$, it should receive. See Figure 4b for the message format and notice the use of the `level` tuple, which specifies l_i and essentially $\mathscr{T}_v^{l_i}$. Notice further that there can be more than one `level` tuple to cater for data aggregation. These interests travel down the gradients that have previously been established by S's "interest about interests".

To further illustrate the details, we borrow the help of a sample topology depicted in Figure 5, where node u_{12} is in the process of being evicted. First 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

$$\frac{N}{N+1}\left[1 - \sum_{l=0}^{L_{max}} \binom{(a-1)a^{h-l-1}}{N+1} \Big/ \binom{n-1}{N+1}\right]$$

where $L_{max} = \lfloor h - \lg_a [\frac{a(N+1)}{a-1}] \rfloor$.

The proof of Proposition 1 can be found in Appendix A. While the probability that u_i is able to perform at least one aggregation is given by Proposition 1, the other three conditions under which aggregation is not facilitated are summarized as follows:

(a) $l_j \neq l_i$ and $\rho_j \geq \rho_i$: u_i should ideally send both l_i and l_j downstream, but u_i will only receive l_j after sending l_i, due to the later time-out of u_j.

(b) $l_j = l_i$ and $\rho_j < \rho_i$: although u_j times out before u_i, the identical value of l_i and l_j does not require data aggregation anyway.

(c) $l_j = l_i$ and $\rho_j \geq \rho_i$: not only the identical value of l_i and l_j does not require data aggregation, but also u_j will time out later than u_i anyway.

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.

For example, pick a random node, say u_7 in Figure 5. According to Figure 2, $l_7 = l_0 = 0$, 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 rebroadcast 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 `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 $\mathscr{T}_{12}^0$ and $\mathscr{T}_{12}^1$ are needed and would deliver them upstream in the direction of u_0. u_0, remembering that $\mathscr{T}_{12}^0$ and $\mathscr{T}_{12}^1$ are needed, would likewise deliver the key sets upstream. Finally u_8 would send only $\mathscr{T}_{12}^0$ upstream since it is only u_7 who needs it (but of course u_8 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 $\mathscr{T}_{12}^0$ *before* u_7's "interest to re-key" arrives, u_0 can easily retrieve $\mathscr{T}_{12}^0$ from its data cache and hand it over to u_7 without going through S again.

```
class EQ interest
groupID IS ...
evictIndex IS ...
mac IS ...
```

(a) Interest about interests to re-key

```
class IS interest
groupID EQ ...
evictIndex EQ ...
level IS ...
(possibly more
 level tuples...)
mac IS ...
```

(b) Interest to re-key

Figure 4: Message formats for leaving.

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.

Algorithm As it happens in many secure multicast schemes, the join algorithm and the leave algorithm are asymmetrical with the join algorithm 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 higher

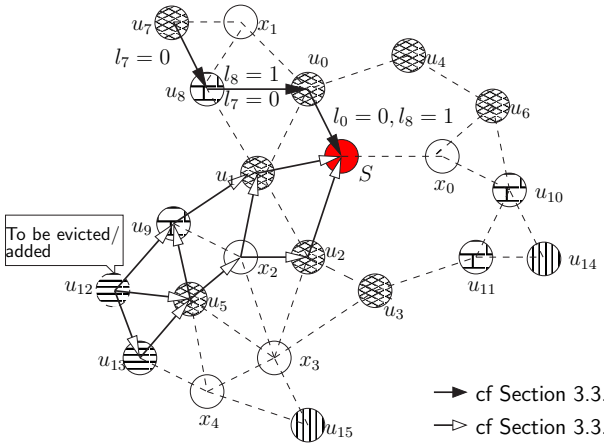


Figure 5: A sample random topology for LKHW. S denotes the sink/KDC; members are named u_i ($i = 0, \ldots, 15$); non-members are named x_j ($j = 0, \ldots, 4$); arrows represent sample "interests to re-key"; nodes with similar hatch pattern belong to the same recipient set $\mathscr{U}_{12}^l$, where $l = 0, 1, 2, 3$.

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 3b 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 matching will still work.

From a system point of view, the algorithm is:

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

where "seed" is the key regeneration seed. Detailed explanation of the algorithm 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 5, 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 $\mathscr{K}_v$ (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 $\mathscr{K}_i'$ from its existing key set $\mathscr{K}_i$ as follows:

$$\mathscr{K}_i' = \{K' | K' = MAC_K(\text{seed}), \forall K \in \mathscr{K}_i\} \tag{7}$$

The forward security of the seed can be ensured by Bellare and Yee's construction [5].

Formally, the protocol between u_v and S is the same as Protocol 1. In the next section, we discuss the performance of the join and leave algorithm.

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 [9], 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 [34], 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 5, S would potentially receive requests for the transmitted key set at level 0, $\mathscr{T}_v^0$, 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, $\mathscr{T}_v^0$ as data replies to u_0, u_1, u_2 and x_0 individually. So S would send $\mathscr{T}_v^0$ three times. On the other hand, requests for $\mathscr{T}_v^3$ would potentially only come from u_1 and u_2, and hence S would send $\mathscr{T}_v^3$ two times. Intuitively, we can certainly do better than what directed diffusion restricts us to. Observe that instead of unicasting to each requesting neigbhour, we might be able to save energy by broadcasting all the key sets $\mathscr{T}_v^l$, $l = 0, \ldots, h-1$ at once because each key set $\mathscr{T}_v^l$ 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

$$E_{unicasts} = (1+r)\sum_{i=1}^{N'} C_i E_{ks} + (1+r)N'E_o \tag{8}$$

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;
- 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

$$E_{broadcast} = (1+Nr)hE_{ks} + (1+Nr)E_o \tag{9}$$

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 $E_{broadcasts}$, if $E_{unicasts} > E_{broadcasts}$, 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 $(1+r)\sum_{i=1}^{N'} C_i$ to 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.

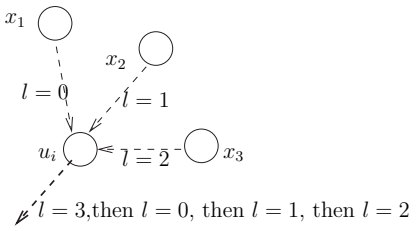


Figure 6: Worst case of receiving re-key levels from upstream neighbours ($\mathscr{L} = \{0, 1, 2, 3\}$).

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 $\mathscr{L}$ (Figure 6). The upper-bound energy cost is therefore:

$$\begin{aligned} E_{upper} &= r\sum_{i=1}^{N}(C_i E_l + E_o) + (E_l + E_o) + (h-1)(E_l + E_o) \\ &= (h + r\sum_{i=1}^{N} C_i)E_l + (h + Nr)E_0 \\ &\approx (h + Nr)E_0 \end{aligned} \tag{10}$$

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;
- E_o is the energy associated with the overhead of a single "interest to re-key" message.

Note that the number of bits to represent a re-key level is potentially very small compared with the the length of an "interest to re-key" message (Figure 4) 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 described 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:

$$E_{lower} = (E_l + E_o) \approx E_0 \tag{11}$$

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:

$$E = E_{ks} + E_{seed} + E_{o(unicast)} + E_{o(broadcast)} \tag{12}$$

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 Security Analysis

After evaluating the performance of LKHW, we now investigate the security aspects of LKHW. To model the attackers, we adopt, as usual, the Dolev-Yao model [12], in which an attacker can eavesdrop, intercept, modify and replay any message in the network. The backward and forward secrecy of our scheme relies on the security properties of MAC and pseudorandom generator which are standard cryptographic primitives. As for other properties, our analysis is performed against *security protocol attacks* and known *routing protocol attacks*.

5.1 Security Protocol Attacks

During group initialization, Protocol 1 is applied between pairs of S and u_i. As mentioned, the protocol has been verified using the protocol verifier CoProVe (`http://wwwes.cs.utwente.nl/24cqet/`), to be (1) secure in the confidentiality of K_0, (2) secure against replay attacks on S and (3) secure against replay attacks on u_i. The principles of verification have been detailed in our previous work [26]. Security against replay attacks on S means an adversary cannot replay any past message of S to impersonate S. The definition applies similarly to u_i. Note that the notion of node-to-node authentication is meaningless in the data-centric paradigm of directed diffusion, i.e. names are applied to data instead of to the nodes themselves.

5.2 Routing Protocol Attacks

Routing protocols that are not designed with security in mind are particularly susceptible to attacks (with energy deprivation or denial-of-service being the most significant result), and directed diffusion is no exception. According to Karlof and Wagner [24], directed diffusion

is especially susceptible to *selective forwarding*, *flow suppression*, *cloning*, *data tampering*, *flow redirection*, *bogus routing information*. Note that these attacks relate to each other in one way or another, hence this classification is by no means unique nor universal, we will however analyze LKHW according to this very classification.

Selective forwarding is the most generic and effortless form of attack in WSN, because any node in the network is capable of launching such attack without the knowledge of any cryptographic information. One specific instance of selective forwarding is **flow suppression**. Fortunately due to the exploratory nature of initial interests and initial data replies, there is usually too much redundancy in the interest diffusion and exploratory reply phase for a selectively forwarding adversary to wield its disruptive power. However once on one of the reinforced gradients, an adversary can arbitrarily influence the data flow. In response to the resultant flow irregularity, the sink has to initiate new rounds of interest diffusion. We expect that the multipath extension of directed diffusion [15] is able to resist this attack.

Cloning is an attack whereby an adversary clones the interest of a sink, thus inducing an alternative data flow to itself. In LKHW, outsider adversaries cannot forge interests nor data within a group because they do not know the group key. Insider adversaries are however able to forge interests and data with the group key. Apart from forging data, they are also capable of **data tampering**. Depending on the application scenario, forging interests might not mean more harm than being a passive eavesdropper, a role any insider adversary is readily capable of. To solve the problem of interest forging, we believe asymmetric cryptography is required, but it has more to do with key management than the current issue we are solving at hand. As for the problem of data forging or tampering, we believe in typical scenarios, there are many more legitimate than malicious sources, averaging out the undesirable effect of forged or tampered data.

Flow redirection can be achieved in the original direction diffusion model, by an attacker launching *sinkhole* and *wormhole* attacks. In general, an attacker can draw traffic towards itself (as if it were a sinkhole) by appearing attractive to neighbouring nodes in terms of routing cost. In directed diffusion, an attacker can easily achieve this by sending interests with high data rate and at a high frequency. Moreover, since the attacker can also spoof positive and negative reinforcements, it can suppress data flows to other parts of the network while concentrating the flows to itself.

Suppose there are attackers, with attacker A located near the sink, and attacker B located near the sources. If A and B collude to establish an exclusive, out-of-band channel known as wormhole [19] between each other, A can send the sink's interests directly to B through the wormhole. With A spoofing negative reinforcements and B spoofing positive reinforcements to their neighbouring nodes, they can push traffic away from the sink and towards B instead.

In LKHW, recall in Section 3.2 that reinforcements during group initialization are the directional sendings of "data for joining". "Data for joining" do not specify any data rate, thus the most an adversary can do is to disrupt the propagation of these reinforcements and not bump up or pin down the "degree" of these reinforcements. After a secure group is setup up, insider attackers, with the knowledge of the group key, are however able launch *sinkhole* or *wormhole* attacks to redirect data flows. On the bright side, we believe this problem can be mitigated by adopting the multipath extension of directed diffusion [15], and using link-layer encryption and authentication (for curbing Sybil attacks [13] that easily defeat multipath routing).

Bogus routing information includes arbitrary deviations from the normal flow of messages, e.g. the advertisement of bogus interests or the supply of bogus data. In LKHW, any node, insider or outsider, can advertise "interest about interests to join" in the form of plaintext tuple (T, GID) as in Protocol 1, thus any node can trigger unnecessary propagation of "interests to join". Our response to this is as follows:

First of all, we cannot encrypt (T, GID) because due to our "no system-wide key" assumption/requirement, we cannot make sure intermediate nodes (nodes between the sink and potential sources) can decrypt the interests and hence perform data-interest matching. In fact, insider attackers are still able to generate bogus (T, GID) even if we require the messages to be encrypted. A simple solution is

1. to limit the propagation of (T, GID), and
2. to make sure that algorithm-wise the generation of GID consumes much more energy and computational resources than the verification of GID.

To the effect of strategy (1), the original mechanism of directed diffusion already ensures that duplicates of (T, GID) are discarded, forcing the adversary to generate different T or GID each time. The number of choices for T, i.e. the number of task types, is conceivably limited, and strategy (2) makes sure that the adversary has to pay a heavier price compared to its victims to fulfill its own malicious intent. For the algorithm, we can use moderately hard, memory bound functions [1].

Lastly, outsider attackers are not able to forge "interests to join" because they are protected by the individual keys. Outsider attackers are also not able to forge "interests to re-key" because they are protected by the group key.

6 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 [17, 21] in Section 2 in details. Krishnamachari et al. [25] offer further insight into the energy efficiency of directed diffusion. In the following, we will compare LKHW with other group communication schemes from two perspectives, the distributed approaches and the hierarchical approaches.

6.1 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. [20] are the first to extend the Diffie-Hellman (DH) problem [11] to groups. Burmester and Desmedt [7] correct Ingermasson et al.'s security flaw by using cyclic instead of symmetric functions. Just and Vaudenay [23] patch the key authentication flaw of the Burmester-Desmedt scheme and generalize it. Steiner et al. [37, 38] 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 require less communication than the Burmester-Desmedt scheme does. Ateniese et al. [3] 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 [9, 18], and re-keying is inefficient. Blundo et al. [6] did the pioneering investigative work on the storage requirements of k-secure t-conference key distribution scheme. The proposed scheme is information-theoretically secure, 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. In particular, Carman et al. [9] have concrete figures of the storage requirements. Furthermore, Beimel and Chor [4] prove that *interaction* among the group members does not improve space efficiency. The conclusion is that both DH-based and information-theoretically secure schemes have their share of scalability problems.

6.2 Hierarchical Approaches

By imposing a hierarchial structure – binary tree being the mainstream – on dynamic peer groups, these approaches have been able to achieve better scalability than the distributed approaches. There is the pioneering work by Wallner et al. [39, 16] 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. [40] 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. [27] introduce the one-way function tree (OFT), where the KDC needs only dispatch $\lceil \lg_a n \rceil$ keys during re-keying, the same number as LKHW's, instead of the $2\lceil \lg_a n \rceil$ required by the basic LKH model. Canetti et al. [8] replace McGrew et al.'s non-standard cyptographic primitive with pseudorandom generator. The EHBT scheme proposed by Rafaeli et al. [33] 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 [29] however avoid that overhead by locally *and* periodically regenerating their keys. This is in the same spirit as Setia et al.'s Kronos [35], where re-keying is driven by time rather than membership changes. Di Pietro et al. [31] 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, ELK, EHBT 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. That said, time-driven re-keying strategy like the one proposed by Di Pietro et al. [32] is indeed an avenue of research for improving LKHW.

7 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 novel evaluation methodology that is entirely based on energy efficiency.

This paper involves a lot of details, so much so that the details for multiple join and multiple leave have been left out. We also have not addressed the efficient rebalancing of the LKH tree in the directed diffusion context. The other issues currently under assessment include *self-healing* key distribution schemes (e.g. [36]).

Acknowledgements

The authors would like to thank Pieter Hartel for his constructive comments on the drafts; Chan Haowen and Bhaskar Krishnamachari for their helpful reply to our query.

References

[1] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. In *Proc. 10th Annual Network and Distributed System Security Symposium (NDSS'03)*. Internet Society, February 2003.

[2] M. Abdalla and M. Bellare. Increasing the lifetime of a key: A comparitive analysis of the security of rekeying techniques. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 546–565. Springer-Verlag, 2000.

[3] G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4), 2000.

[4] Amos Beimel and Benny Chor. Communication in key distribution schemes. *IEEE Transactions on Information Theory*, 41(1):19–28, 1996.

[5] M. Bellare and B. Yee. Forward-security in private-key cryptography. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003*, volume 2612 of *LNCS*, pages 1–18. Springer-Verlag, 2003.

[6] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly secure key distribution for dynamic conferences. *Information and Computation*, 146(1):1–23, 1995.

[7] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In A.D. Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *LNCS*, pages 275–286. Springer-Verlag, 1995.

[8] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and efficient constructions. In *INFOCOM '99*, volume 2, pages 708–716, 1999.

[9] D.W. Carman, P.S. Kruus, and B.J. Matt. Constraints and approaches for distributed sensor network security. Technical Report #00-010, NAI Labs, 2000.

[10] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Research in Security and Privacy*, 2003. To appear.

[11] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[12] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, 29(2):198–208, Mar 1983.

[13] J.R. Douceur. The sybil attack. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Peer-to-Peer Systens. 1st Int. Workshop, IPTPS 2002*, volume 2429 of *LNCS*, pages 251–260. Sprinter-Verlag, 2002.

[14] L. Eschenauer and V.D. Gligor. A key-management scheme for distributed sensor networks. In *Proc. 9th ACM conference on Computer and communications security*, pages 41–47. ACM Press, 2002.

[15] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly resilient, energy efficient multipath routing in wireless sensor networks. *Mobile Computing and Communications Review (MC2R)*, 1(2), 2002.

[16] H. Harney and E. Harder. Logical key hierarchy protocol. Internet draft, IETF, April 1999.

[17] J.S. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Symposium on Operating Systems Principles*, pages 146–159, 2001.

[18] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System archtecture directions for networked sensors. *ACM SIGPLAN Notices*, 35(11):93–104, 2000.

[19] Y.-C. Hu, A. Perrig, and D.B. Johnson. Wormhole detection in wireless ad hoc networks. Technical Report TR01-384, Department of Computer Science, Rice University, Jun 2002.

[20] I. Ingemarsson, D.T. Tang, and C.K. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5):714–720, 1982.

[21] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *6th Annual Int. Conf. on Mobile Computing and Networking (MobiCOM '00)*, pages 56–67, Boston, Massachusetts, United States, 2000. ACM Press.

[22] D.B. Johnson and D.A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[23] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Advances in Cryptology – ASIACRYPT'96*, volume 1163 of *LNCS*, pages 36–49. Springer-Verlag, 1996.

[24] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *Ad Hoc Networks*, 2003.

[25] B. Krishnamachari, D. Estrin, and S. Wicker. Modelling data-centric routing in wireless sensor networks. Technical Report CENG 02-14, University of Southern California, 2002.

[26] Y.W. Law, R. Corin, S. Etalle, and P.H. Hartel. A formally verified decentralized key management architecture for wireless sensor networks. In *Personal Wireless Communications (PWC 2003)*, Venice, Italy, Sep 2003. IFIP WG 6.8 - Mobile and Wireless Communications, Springer-Verlag. To appear.

[27] D.A. McGrew and A.T. Sherman. Key establishment in large dynamic groups using one-way function trees. Technical Report 0755, TIS Labs, Network Associates, Inc., 1998.

[28] C.E. Perkins and E.M. Royer. Ad hoc on-demand distance vector routing. In *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100. IEEE Computer Society Press, 1999.

[29] A. Perrig, D. Song, and D. Tygar. ELK, a new protocol for efficient large-group key distribution. In *Proc. 2001 IEEE Symposium on Security and Privacy*, pages 247–262. IEEE Computer Society Press, 2001.

[30] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Proceedings of the 7th Ann. Int. Conf. on Mobile Computing and Networking*, pages 189–199. ACM Press, 2001.

[31] R. Di Pietro, L.V. Mancini, and S. Jajodia. Efficient and secure keys management for wireless mobile communications. In *Proc. 2nd ACM Int. Workshop on Principles of Mobile Computing*, pages 66–73. ACM Press, 2002.

[32] R. Di Pietro, L.V. Mancini, and A. Mei. A time driven methodolgy for keys dimensioning in secure multicast communications. In D. Gritzalis, S. De Capitani di Vimercati, P. Samarati, and S.K. Katsikas, editors, *Security and Privacy in the Age of Uncertainty. 18th IFIP Int. Information Security Conf.*, pages 121–132. Kluwer Academic Publishers, 2003.

[33] S. Rafaeli, L. Mathy, and D. Hutchison. EHBT: an efficient protocol for group key management. In Jon Crowcroft and Marcus Hofmann, editors, *Proceedings of the Third International COST264 Workshop (NGC 2001)*, volume 2233 of *LNCS*, pages 159–171. Springer-Verlag, 2001.

[34] RF Monolithics, Inc. TR1001: 868.35 MHz Transceiver. Datasheet. `http://www.rfm.com/products/data/tr1001.pdf`.

[35] S. Setia, S. Koussih, S. Jajodia, and E. Harder. Kronos: A scalable group re-keying approach for secure multicast. In *IEEE Symposium on Security and Privacy*, pages 215–228, 2000.

[36] J. Staddon, S. Miner, M. Franklin, D. Balfranz, M. Malkin, and D. Dean. Self-healing key distribution with revocation. In *Proc. 2002 IEEE Symposium on Security and Privacy (S&P'02)*, pages 224 –240. IEEE Computer Society Press, 2002.

[37] M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: A New Approach to Group Key Agreement. In *Proc. 18th Int. Conf. on Distributed Computing Systems*, pages 380–387, 1998.

[38] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, 2000.

[39] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. RFC 2627, IETF, June 1999.

[40] C.K. Wong, M. Gouda, and S.S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking (TON)*, 8(1):16–30, 2000.

Appendix A Proof of Proposition 1

Denote X as the event that u_i can perform at least one aggregation X, 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,

$$\begin{aligned} Pr[X] &= 1 - Pr[A] - Pr[B] + Pr[AB] \\ &= 1 - Pr[A] - Pr[B] + Pr[A]Pr[B] \end{aligned} \tag{13}$$

since events A and B are mutually independent. Recall that ρ_i is as defined in Section 3.3.1, the time-out ratio of u_i. We first derive $Pr[B]$, assuming ρ_i is uniformly distributed between 0 and 1:

$$\begin{aligned} Pr[B] &= \int_{\rho_i=0}^{1} (1-\rho_i)^N \\ &= \frac{1}{N+1} \end{aligned} \tag{14}$$

reducing (13) to the simpler form of (15).

$$Pr[X] = \frac{N}{N+1}(1 - Pr[A]) \tag{15}$$

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 2, 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

$$L_{max} = h - \lg_a \left\lceil \frac{a(N+1)}{a-1} \right\rceil$$

Constraining L_{max} to integer values, we have

$$L_{max} = \left\lfloor h - \lg_a \left\lceil \frac{a(N+1)}{a-1} \right\rceil \right\rfloor \tag{16}$$

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,

$$Pr[A] = \frac{\sum_{l=0}^{L_{max}} \binom{n'}{N+1}}{\binom{n-1}{N+1}} \tag{17}$$

where n is the total number of nodes in the group *before* eviction. Substituting $Pr[A]$ with (17), in (15), we get Proposition 1.

□