Book Title: XXXXXXXXXXXXXXXXXXXXXXXXXXXX

Editors

May 19, 2008

Contents

Chapter 1

Key Establishment Protocols for Wireless Sensor Networks

Ioannis Chatzigiannakis, Elisavet Konstantinou

In this book chapter we will consider *key establishment protocols* for *wireless sensor networks*. Several protocols have been proposed in the literature for the establishment of a shared group key for wired networks. The choice of a protocol depends whether the key is established by one of the participants (and then transported to the other(s)) or agreed among the participants, and on the underlying cryptographic mechanisms (symmetric or asymmetric). Clearly, the design of key establishment protocols for sensor networks must deal with different problems and challenges that do not exist in wired networks. To name a few, wireless links are particularly vulnerable to eavesdropping, and that sensor devices can be captured (and the secrets they contain can be compromised); in many upcoming wireless sensor networks, nodes cannot rely on the presence of an online trusted server (whereas most standardized authentication and key establishment protocols do rely on such a server).

In particular, we will consider five distributed group key establishment protocols. Each of these protocols applies a different algorithmic technique that makes it more suitable for (i) static sensor networks, (ii) sensor networks where nodes enter sleep mode (i.e. dynamic,

with low rate of updates on the connectivity graph) and (iii) fully dynamic networks where nodes may even be mobile. On the other hand, the common factor for all five protocols is that they can be applied in dynamic groups (where members can be excluded or added) and provide forward and backward secrecy. All these protocols are based on the Diffie-Hellman key exchange algorithm and constitute natural extensions of it in the multiparty case.

1.1 Introduction

Group key management mainly includes activities for the establishment and the maintenance of a group key. Secure group communication requires scalable and efficient group membership with appropriate access control measures to protect data and to cope with potential compromises. A secret key for data encryption must be distributed with a secure and efficient way to all members of the group. Another important requirement of group key management protocols is key freshness. A key is fresh if it can be guaranteed to be new. Moreover, the shared group key must be known only to the members of the group. Four important cryptographic properties must be encountered in group key agreement [31, 35]. Assume that a group key is changed m times and the sequence of successive keys is $\mathcal{K}=\{K_0, \ldots, K_m\}$.

Computational group key secrecy: It guarantees that it is computational infeasible for any passive adversary to discover any group key $K_i \in \mathcal{K}$ for all i.

Decisional group key secrecy: It ensures that there is no information leakage other that public blinded key information.

Key independence: It guarantees that a passive adversary who knows a proper subset of group keys can not discover any other of the remaining keys. Key independence can be decomposed into **forward secrecy** and **backward secrecy**. Forward secrecy guarantees that a passive adversary who knows a contiguous subset of old group keys cannot discover any subsequent group key. Backward secrecy guarantees that a passive adversary who knows a contiguous subset of group keys cannot discover preceding group key.

Group key establishment can be either centralized or distributed. In the first case, a member of the group is responsible for the generation and the distribution of the key. In distributed group key establishment all group members contribute to the generation of the key. Clearly, the second approach is suited for sensor networks because problems with centralized trust and the existence of single point of failure can be avoided. In this chapter, we consider distributed group key establishment protocols [7, 14, 29, 44] which can be applied in dynamic groups (where members can be excluded or added) and provide forward and backward secrecy. Moreover, all these protocols are based on the Diffie-Hellman key exchange algorithm [21] and constitute natural extensions of it in the multiparty case.

Many cryptographic protocols have been developed to provide security for group communication [5, 12, 13, 27]. Unfortunately, most of these protocols either require a particular structure for the network (that is neither desired nor available in ad hoc networks) or are resource intensive.

Most group key establishment protocols are based on generalizations of Diffie-Hellman key exchange protocol [21]. The first attempt for the construction of such protocols was made by Ingemarsson, Tang and Wong [29] that arrange the participants in a standard form like a logical ring via a synchronous start up phase. The protocol completes in $n-1$ rounds, where n is the number of the participants.

Burmester and Desmedt presented in [14] a more efficient scheme which requires only two rounds. However, the protocol's disadvantage is that (i) every participant must perform $n+1$ exponentiations and (ii) communication is based on concurrent broadcasts that lead to high number of collisions, a situation very common in wireless sensor networks that affects performance [17]. Moreover, the authors do not provide a proof of security (in the stronger sense of semantic security). Recently, Katz and Yung [30] proposed a more general framework that provides a formal proof of security for this protocol. In Hypercube protocol [7] the participants in the network are arranged in a logical hypercube. This topology decreases the number of transmitted data and exponentiation operations, but still the protocol is very demanding for use in sensor networks.

One of the most efficient protocols in the literature for group key management is the third protocol GDH.3 of Steiner, Tsudik and Waidner presented in [44]. This protocol requires serial execution of computations that makes it inefficient for highly dynamic networks with large number of nodes. More precisely, this protocol may not be a good choice for a dynamically evolving ad hoc environment since the last node in the protocol's computation would have to know the whole structure of the network.

A performance analysis of all the above mentioned protocols is presented in [3, 4] which clearly shows the superiority of GDH.3 protocol in the number of transmitted data and exponentiation operations required. In particular, the number of messages and exponentiations is linear to the number of the participants in the protocols, while for all other protocols are of order $n \log n$ or n^2.

A very efficient protocol is also presented in [31]. In this recent work, a logical key tree structure is used to improve the scalability of the key agreement protocol. Any device can calculate the group key if it knows all the keys in its co path. This requirement makes the protocol quite expensive in storage memory that is critical for sensor networks. For these reasons, we believe that the simplicity and the limited memory requirements of GDH.3 protocol make it more suitable and applicable in sensor networks. Moreover, the recent papers of Bresson et al. [10, 11] were the first to present a formal model of security for group authenticated key exchange and the first to give rigorous proofs of security for particular protocols.

Based on the above, we distinguish a category of protocols (e.g., [14, 29, 44]) which rely on communication primitives that provide global ordering of the devices, e.g. such as a (virtual) ring-based topology and enable many-to-many message exchanges. In fixed infrastructure based networks, such communication primitives can be provided by the fixed part (i.e. base stations). However, in wireless sensor networks the fixed infrastructure is sparse (or even non existing), making it difficult (or even impossible) to implement such primitives via external coordination. Certainly one can assume that the participating devices are capable of transmitting at long ranges, allowing them to communicate directly with each other. Still, in the light of the dense deployment of sensor devices, a traditional single hop

communication scheme consumes a lot of power compared to distributed short-range hop-by-hop propagation ([28]). In addition, multi-hop communication can effectively overcome some of the signal propagation effects in long-distance wireless transmissions and may help to smoothly adjust propagation around obstacles. Finally, the low energy transmission in hop-by-hop propagation may enhance security, protecting from undesired discovery of the data propagation operation.

In this direction, a number of protocols have been presented (e.g., [18, 22, 31]) that construct a connectivity related distributed tree structure representing the topology of the network by a series of message passes. The network is viewed as a dynamically changing, directed graph, with devices as vertices and *edges* (*virtual links*) between vertices corresponding to devices that can currently communicate. Each device is required to store some small amount of information regarding the data structure (i.e. not the complete graph) and uses this information for group key establishment. Since these protocols do not require that all group members communicate directly with each other via long-range transmissions, the silent adversary will only be able to listen to a limited number of messages given its actual physical location.

Communication in wireless sensor networks usually occurs in ad hoc manner [9] and is subject to frequent, unpredictable changes: network connectivity changes by time as devices adjust their radio range [33] and duty cycle [32], sensor devices die and new sensor devices may be added to the network. In order to guarantee the correctness of the process of group key establishment, the protocols are required to exchange messages to update the data structure in order to reflect the changes to the topology of the network. If the *rate of connectivity changes* is low ("quasi-static" networks) or medium, then adaptive algorithmic techniques can apply, e.g. like the work of [37]. However, if the *rate* is high, the devices end up exchanging large amounts of information (that waste the wireless medium and the energy resources) and might even fail to react fast enough forcing the group key establishment to fail.

1.1.1 Modelling Assumptions

In the sequel we present five key establishment protocols [14, 18, 19, 31, 44] using elliptic curves terminology. We abstract the technological specifications of existing wireless sensor systems [1] as a system consisting of n devices connected through bidirectional channels allowing direct communication between pairs of neighbor processes linked by a channel. We assume that devices have distinct identities, and they know their own identities together with those of their neighbors. To simplify the presentation of the protocols, we here assume that channels are safe, that is, messages are delivered without loss or alteration after a finite delay, but they do not need to follow a first-in-first-out rule. Moreover, the protocols can be distributed (e.g. a structure based algorithm, see [18, 44]) or centralized (i.e. by a controlling center). We here consider general wireless sensor networks that are composed by a base station and simple devices where no kind of hierarchy exists. The base station represents the authorities of this remote surveillance system (i.e. where the wireless sensors report), has very large storage and data process capabilities and is usually a gateway to another network (i.e. the internet). Typically, the sensors are deployed around the area of the base station and form groups given the needs of the base station. Data flow in our network is group-wise within a group of sensor devices and the base station.

In Sec. 1.8 we implement (in nesC code running in TinyOS) safe communication demonstrating that the underlaying technology can fulfill these assumptions.

1.2 Preliminaries of Elliptic Curve Theory

In this section we review some basic concepts regarding elliptic curves and their definition over finite fields. The interested reader may find additional information in e.g. [8, 41]. We also assume familiarity with elementary number theory (see e.g. [16]).

The elliptic curves are usually defined over *binary fields* F_{2^m} ($m \geq 1$), or over *prime fields* F_p, $p > 3$. In the experimental results we used elliptic curves defined over prime fields. An *elliptic curve* $E(F_p)$ over a finite field F_p, where $p > 3$ and prime, is the set of points

$(x, y) \in F_p$ (represented by affine coordinates) which satisfy the equation

$$y^2 = x^3 + ax + b \tag{1.1}$$

and $a, b \in F_p$ are such that $4a^3 + 27b^2 \neq 0$. The set of solutions (x, y) of Eq. (1.1) together with a point $\mathcal{O}$, called the *point at infinity*, and a special addition operation define an Abelian group, called the *Elliptic Curve group*. The point $\mathcal{O}$ acts as the identity element (for details on how the addition is defined see [8, 41]).

The *order* m of an elliptic curve is the number of the points in $E(F_p)$. The *order of a point* P is the smallest positive integer n for which $nP = \mathcal{O}$. Application of Langrange's theorem (see e.g. [16]) on $E(F_p)$, gives that the order of a point $P \in E(F_p)$ always divides the order of the elliptic curve group, so $mP = \mathcal{O}$ for any point $P \in E(F_p)$, which implies that the order of a point cannot exceed the order of the elliptic curve.

The security of elliptic curve cryptosystems is based on the difficulty of solving the discrete logarithm problem (DLP) on the EC group. The Elliptic Curve Discrete Logarithm Problem (ECDLP) is about determining the least positive integer k which satisfies the equation $Q = kP$ for two given points Q and P on the elliptic curve group. A user A in an elliptic curve cryptosystem can choose a random integer $0 < k < p - 1$ and send Q to a user B with whom he wants to communicate secretly. A's public key is Q and his private key is k. Then an encryption algorithm can be applied (e.g. ElGamal encryption [24]) so that B can encrypt the message he wishes to send to A with the public key Q and A will decrypt it using his private key k.

The Elliptic Curve Diffie-Hellman algorithm [21] is based on the difficulty of solving the discrete logarithm problem in an elliptic curve group. The algorithm is as follows. Let A and B be two entities that wish to share a secret key. Both A and B agree a priori on an elliptic curve group, a generator P of this group and generate a pair of private/public key $(k_A, Q_A = k_A P)$ and $(k_B, Q_B = k_B P)$ respectively. Then A sends Q_A to B and B sends Q_B to A. A computes the value $S = k_A Q_B$ and B the value $S = k_B Q_A$, where S is now their shared secret key. For an appropriately chosen elliptic curve group, an adversary who observes Q_A and Q_B can not find the shared point S. The only weakness of this algorithm is

that there must be an authentication process between A and B so that there is a guarantee that every entity is who he claims to be.

There is a family of protocols that are referred as "natural" extensions of the original, 2-party DH key exchange to n parties. Like in the 2-party case, all participants $M_1, \ldots, M_n$ agree on an elliptic curve group and a generator P. Each member M_i chooses randomly a value k_i and the final shared key among all the members will be equal to $Q_n = k_1 \ldots k_n P$. It can be proven that any adversary who observes any subproduct $k_1 \ldots k_{i-1} k_{i+1} \ldots k_n P$, can not compute the shared key Q_n [44]. In particular, in [44] is proven that *if a 2-party key is indistinguishable from a random value, the same is true for n-party keys.*

1.3 Burmester-Desmedt Protocol

In 1994, Burmester and Desmedt proposed an efficient protocol for the establishment of a common secret key among the members of a group which requires a constant number of rounds [14]. Suppose that n users $M_1, \ldots, M_n$ wish to establish a common group key. The steps of the protocol are the following where the indices are taken modulo n so that member M_0 is M_n and member M_{n+1} is M_1.

BD protocol

1. In the first stage every group member M_i generates a random secret value k_i and broadcasts the point $Q_i = k_i P$ where P is the base point.
2. Each group member M_i computes and broadcasts the point $X_i = k_i(Q_{i+1} - Q_{i-1})$.
3. In the last stage, every group member M_i computes the group key as $K = nk_i Q_{i-1} + (n-1)X_i + (n-2)X_{i+1} + \ldots X_{i+n-2}$.

If all group members follow the above steps, they will compute the same group key $K = (k_1 k_2 + k_2 k_3 + \ldots k_n k_1)P$. For example, suppose that four group members M_1, M_2, M_3 and M_4 wish to establish a common secret key. They compute and broadcast the points

$Q_1 = k_1P$, $Q_2 = k_2P$, $Q_3 = k_3P$ and $Q_4 = k_4P$. Then, M_1 broadcasts the point $X_1 = k_1(Q_2 - Q_4)$, M_2 broadcasts $X_2 = k_2(Q_3 - Q_1)$, M_3 broadcasts the point $X_3 = k_3(Q_4 - Q_2)$ and finally M_4 the point $X_4 = k_4(Q_1 - Q_3)$. Every group member now knows all points Q_i, X_i and using its secret value k_i can compute the group key. For instance, M_1 computes $K = 4k_1Q_4 + 3X_1 + 2X_2 + X_3 = k_1k_2P + k_2k_3P + k_3k_4P + k_4k_1P$.

The security of the protocol is based on the Decisional Diffie-Hellman problem. The protocol is unauthenticated and consequently is secure only against passive adversaries. The authors of [14] provided the full security proof of their protocol later in [15]. Recently, Katz and Yung [30] proposed a more general framework that provides a formal proof of security for this protocol. They also proposed a scalable compiler which transforms an unauthenticated group key agreement protocol into an authenticated group key agreement protocol preserving in the same time the forward secrecy of the original protocol. The modification that Katz and Yung proposed to this protocol adds one more round, two signature generations and $2n - 2$ signature verifications.

The main advantage of this protocol is that it completes in only two rounds. However, the protocol's disadvantage is that (i) every participant must perform $n + 1$ exponentiations and (ii) communication is based on concurrent broadcasts that lead to high number of collisions, a situation very common in wireless sensor networks that affects performance [17]. Another disadvantage of the protocol is that for any new dynamic event that may happen in the group (either join or leave) the protocol has to be executed again from the start.

Many variants of the Burmester-Desmedt protocol have been proposed. Two of the most recent variants are [20, 23]. In [20] the authors proposed a bilinear variant of the Burmester-Desmedt protocol whose security relies on the Computational Diffie-Hellman problem in the random oracle model. Another variant was presented in [23] which improves on the efficiency and flexibility of the original Burmester-Desmedt protocol.

1.4 Group Diffie-Hellman Protocols

A class of three, generic n-party protocols was presented in [44], namely GDH.1, GDH.2 and GDH.3 protocols. Here we will present the elliptic curve analog of the protocols. Suppose that every member in the group has agreed on the use of the same elliptic curve parameters. The number of participants is n and we will denote by M_i the i-th participant.

GDH.1 protocol

1. In the first stage every group member M_i generates a random secret value k_i. The M_1 participant selects a base point P and sends to M_2 the point $Q_1 = k_1P$. Then M_2 sends to M_3 the points $(Q_1 = k_1P, Q_2 = k_1k_2P)$, member M_3 sends to M_4 the points $(Q_1 = k_1P, Q_2 = k_1k_2P, Q_3 = k_1k_2k_3P)$ and so on until the protocol reaches member M_n. In this stage, every member of the group performs one scalar multiplication and the message send by M_i to M_{i+1} contains i intermediate values.

2. Group member M_n computes the point $Q_n = k_1k_2 \dots k_nP$, which is the intended group key. Then, he sends to M_{n-1} the points $(k_nP, k_1k_nP, k_1k_2k_nP, \dots, k_1k_2...k_{n-2}k_nP)$. In other words, he multiplies all the points he had received from M_{n-1} in the previous stage with k_n, removes the formed group key Q_n and adds the point k_nP.

3. In the following stage every group member M_i, $i \in [1, n-1]$ multiplies all points which were received from M_{i+1} with k_i, computes the group key Q_n, removes it from the set of points and sends the rest of them to group member M_{i-1}. In this stage every member M_i performs i scalar multiplications and the message he sends to M_{i-1} contains $i-1$ points.

For example, assume that the size of the group is $n = 5$. Then , in the first stage, M_1 sends to M_2 the point k_1P, M_2 to M_3 the set (k_1P, k_1k_2P), M_3 to M_4 the set $(k_1P, k_1k_2P, k_1k_2k_3P)$, and M_4 to M_5 the set $(k_1P, k_1k_2P, k_1k_2k_3P, k_1k_2k_3k_4P)$. Then, member M_5 calculates the final group key $Q_5 = k_1k_2k_3k_4k_5P$ and sends to M_4 the message $k_5P, k_1k_5P, k_1k_2k_5P, k_1k_2k_3k_5P$. M_4 can compute now the group key by multiplying the last value of the message with its secret value k_4. Then, he sends to M_3 the set $k_4k_5P, k_1k_4k_5P, k_1k_2k_4k_5P$ who calculates the

group key and sends to M_2 the points $k_3k_4k_5P, k_1k_3k_4k_5P$. Finally, M_2 acquires the key and forwards to M_1 the point $k_2k_3k_4k_5P$.

In summary, GDH.1 protocol requires $2(n-1)$ rounds and $2(n-1)$ messages are sent. Computationally, the total number of scalar multiplications required in the protocol are $(n+3)n/2-1$, while every member M_i has to compute $i+1$ scalar multiplications (except from M_n who computes n).

One of the main drawbacks of GDH.1 protocol is the large number of rounds. In order to reduce the number of rounds, the authors of [44] modified GDH.1 and proposed the GDH.2 protocol:

GDH.2 protocol

1. Similarly to GDH.1, in the first stage every group member M_i generates a random secret value k_i. In round i, $1 \leq i \leq n$, member M_i sends to M_{i+1} the points $\{(k_1k_2 \ldots k_i/k_j)P \,|j \in [1,i]\}$ and $k_1k_2 \ldots k_iP$. For example, M_3 sends to M_4 the points $(k_1k_2k_3P, k_1k_2P, k_1k_3P, k_2k_3P)$. In this stage, every member M_i of the group performs i scalar multiplication and the message send by M_i to M_{i+1} contains $i+1$ intermediate values.
2. In the second stage, member M_n computes the point $Q_n = k_1k_2 \ldots k_nP$, which is the intended group key and broadcasts the values $\{(k_1k_2 \ldots k_n/k_i)P \,|i \in [1,n]\}$ to the rest of the users. In this way, every group member M_i can compute the group key by multiplying the point $(k_1k_2 \ldots k_n/k_i)P$ with its secret value k_i.

GDH.2 protocol is executed in n rounds, its total computational cost is $(n+3)n/2-1$ multiplications and every member M_i computes $i+1$ scalar multiplications (like in GDH.1 protocol).

GDH.3 protocol

1. In the first stage every group member M_i generates a random secret value k_i. The M_1 participant selects a point P and sends to M_2 the point $Q_1 = k_1P$. Then M_2 sends to M_3 the point $Q_2 = k_1k_2P$ and so on until the protocol reaches member M_{n-1}. Notice

here that the protocol must pass only one time from every participant.

2. Group member M_{n-1} computes the point $Q_{n-1} = k_1 k_2 \dots k_{n-1} P$ and sends it to all M_i, with $i \in [1, n]$.

3. In the third stage every group member M_i, $i \in [1, n-1]$ computes a point $G_i = k_i^{-1} Q_{n-1}$ and sends it to the last group member M_n.

4. M_n calculates the values $k_n G_i$ and send them to the corresponding members M_i.

After these stages, every group member M_i can calculate the group key $Q_n = k_1 k_2 \dots k_n P$ by multiplying the value $k_n G_i$ with its secret number k_i. Despite its efficiency, the disadvantage of GDH.3 protocol is that it does not offer symmetric operation, because all the participants in the protocol do not perform the same number of operations. If the number of the participants is large, then the computational effort in member M_n can be devastating for its energy.

1.5 Tree-based Group Diffie-Hellman (TGDH) Protocol

In [31], an efficient group key agreement protocol was presented that arranges the group members in a binary tree structure. For example, four members M_1, M_2, M_3 and M_4 will be arranged in a binary tree as shown in Figure 1.1. The tree should be kept balanced. Every node in the tree is labelled with a pair $< l, u >$, where l is the level in which the node belongs to and u is a number indicating the place of the node in the particular level. Obviously, $0 \leq l \leq h$ where h is the height of the tree and $0 \leq u \leq 2^l - 1$. Each node $< l, u >$ is associated with a secret key $k_{<l,u>}$ and a public key $K_{<l,u>} = k_{<l,u>} P$. Initially, all group members generate their secret keys and compute their public keys $K_{<h,u>}$. The public keys $K_{<l,u>}$ of the upper levels are computed using Diffie-Hellman key exchange between the left and right child of the particulat node $< l, u >$, that is nodes $< l+1, 2u >$ and $< l+1, 2u+1 >$. The secret key $k_{<l,u>}$ is computed by $k_{<l,u>} = map(k_{<l+1,2u>} K_{<l+1,2u+1>})$ or $k_{<l,u>} = map(k_{<l+1,2u+1>} K_{<l+1,2u>})$ where $map()$ is a function which maps an elliptic curve point to its x-coordinate. The final secret group key is $k_{<0,0>}$. Clearly, this key can

be computed by every group member if he knows all public keys in the tree.

The original description of TGDH [31] does not specify clearly the setup phase of the protocol, but it focuses on the group membership events. However, we can summarize the previously mentioned steps in the following setup procedure [38]:

TGDH protocol

1. In the first stage every group member M_i generates a random secret value $k_{<l_i,u_i>}$ and broadcasts the point $K_{<l_i,u_i>} = k_{<l_i,u_i>}P$ where P is the base point.
2. Each group member M_i computes the point $K_{<l,u>} = k_{<l,u>}P$, where $l = l_i - 1$ and $u = \left\lfloor \frac{u_i}{2} \right\rfloor$ and the rightmost member of the (sub)tree rooted at node $< l, u >$ broadcasts the point $K_{<l,u>}$.
3. All members of the group repeat step 2 with $l = l - 1$ and $u = \left\lfloor \frac{u}{2} \right\rfloor$ until they compute the group key $k_{<0,0>}$.

Consider, for example, a group with four members arranged in the tree structure of Figure 1.1. In the first step, the members of the group compute and broadcast the points $K_{<2,0>}$, $K_{<2,1>}$, $K_{<2,2>}$ and $K_{<2,3>}$. Then, members M_1 and M_2 compute the value $k_{<1,0>} = map(k_{<2,0>}K_{<2,1>}) = map(k_{<2,1>}K_{<2,0>})$ using the Diffie-Hellman key exchange protocol and member M_2 (the rightmost member of the subtree) broadcasts the point $K_{<1,0>} = k_{<1,0>}P$. Similarly, members M_3 and M_4 compute the value $k_{<1,1>}$ and M_4 broadcasts the point $K_{<1,1>}$ to the network. Finally, all members can compute the group key $k_{<0,0>} = map(k_{<1,0>}K_{<1,1>}) = map(k_{<1,1>}K_{<1,0>})$. Concluding, the main disadvantage (like in the case of Burmester-Desmedt protocol) of this protocol is that it requires many broadcasts that lead to high number of collisions, a situation very common in wireless sensor networks that affects performance [17].

The tree-based structure of the protocol eases the handling of additive (such as join or merge) and subtractive (such as leave or partition) events. In additive events, the new member or the rightmost member of the new tree broadcast their own public keys to the

network. Then, all nodes update the tree structure and the rightmost member M_s of the updated tree changes its secret value $k_{<l_s,u_s>}$, computes the new secret and public keys in its path up to the root and broadcasts them to rest of the group members. A similar procedure is followed in the case of subtractive events.

1.6 Distributed Sequential Traversal Protocol

The protocols presented above rely on communication primitives that provide global ordering of the devices (stage 1), e.g. such as a (virtual) ring-based topology and enable many-to-many message exchanges (stages 2,3). In fixed infrastructure based networks, such communication primitives can be provided by the fixed part (i.e. base stations). However, in wireless sensor networks the fixed infrastructure is sparse (or even non existing), making it difficult (or even impossible) to implement such primitives via external coordination. Certainly one can assume that the participating devices are capable of transmitting at long ranges, allowing them to communicate directly with each other. Still, in the light of the dense deployment of sensor devices close to each other, a traditional single hop communication scheme consumes a lot of power when compared to distributed short-range hop-by-hop propagation ([28]). In addition, multi-hop communication can effectively overcome some of the signal propagation effects in long-distance wireless transmissions and may help to smoothly adjust propagation around obstacles. Finally, the low energy transmission in hop-by-hop propagation may enhance security, protecting from undesired discovery of the data propagation operation.

In this section we present a distributed protocol that does not require many-to-many message exchanges as in the case of the second and third stages of the GDH.3 protocol and does not rely on any global ordering of the devices. This protocol was presented in [18] and is based on the observation that in the first stage of the GDH protocols, group member M_n can compute the shared group key Q_n by acquiring the point Q_{n-1} from M_{n-1} and multiply it with its secret value k_n. Moreover, the points $Q_i = k_1 k_2 \dots k_i P$ which are generated by each group member M_i can be used as their public keys while their private keys are the values k_i. Using this observation, the protocol totally avoids the third and fourth stages of

GDH.3 protocol.

In particular, a distributed sequential traversal algorithm is used that visits each participant of the group in order to build the shared secret key (starting from M_1 and reaching M_n). This is similar to the first stage of GDH protocols, without necessarily requiring direct communication among all the group members and still guaranteeing that each participant are visited only one time. When the traversal is finished and all participants are visited, the protocol reaches member M_n that calculates the point Q_n, the shared secret key. Finally, in order for M_n to communicate the shared secret key to all the participants of the group, it repeats the distributed traversal but in reverse order. M_n now encrypts Q_n with M_{n-1}'s public key Q_{n-1} and sends it to M_{n-1}. M_{n-1} can decrypt the message with his private key k_{n-1}, acquire the secret value Q_n, encrypt it with the public key of M_{n-2} and send the result to M_{n-2}. The same process will be followed by M_{n-2} and so on, until the protocol reaches member M_1.

Assigning Groups. This group key distribution protocol is applicable to hierarchical wireless sensor networks, in the sense that among the nodes there is a hierarchy based on their capabilities. The hierarchical network is composed by a base station, group leaders and simple nodes (group members). The base station represents the authorities of this remote surveillance system (i.e. where the wireless sensors report), has very large storage and data process capabilities and is usually a gateway to another network (i.e. the internet). Typically, the sensors are deployed around the area of the base station and form groups given the needs of the base station. Group leaders are ordinary sensor nodes which can collect local traffic and send it to the base station. Also, they are trusted components and sensors in their groups get routing information from them. Data flow in the network is group-wise within a group of sensor nodes.

Initial Group formation. Based on the initial assignment of the sensors in a given group M (defined by the base station), every group member M_i generates a random secret value k_i while the group leader M_1 selects a point P and calculates the point $Q_1 = k_1 P$ which is its public key. Then, based on a distributed sequential traversal algorithm that visits all partipants at least once, and particularly a distributed *depth-first* traversal algorithm,

it sends Q_1 to M_2 (i.e. a neighboring group member) via special SEARCH<M_2, Q_1> message. When M_2 receives the message, it becomes *active for the first time*, it becomes *visited* and defines participant M_1 as its father (we say M_2 joins the traversal). Moreover, when M_2 is active, it calculates the point $Q_2 = k_1 k_2 P$ (which is the public key of M_2) and shifts the control to a non visited neighbor M_3, through a special SEARCH<M_3, Q_2> message.

When the protocol reaches member M_u with all its neighbors being visited, M_u encrypts Q_u with M_{u-1}'s public key Q_{u-1} and sends it to M_{u-1} (its father) using a special PARENT<M_{u-1}, encrypt(Q_u, Q_{u-1})> message. The function encrypt is defined as encrypt(data, key). M_{u-1} can decrypt the message with his private key k_{u-1}, acquire the secret value Q_u and either continue the distributed sequential traversal by shifting the control to the next non visited neighbor (if any), again through a special SEARCH<M_v, Q_u> or if all its neighbors have been visited, send a PARENT<M_{u-2}, encrypt(Q_u, Q_{u-2})> message to its parent. This process continues until the protocol reaches the last member of the group M_n that calculates the point Q_n, the shared secret key. In a similar way with M_u, M_n encrypts Q_n with M_{n-1}'s public key Q_{n-1} and sends it to M_{n-1} (its father) using a special PARENT<M_{n-1}, encrypt(Q_n, Q_{n-1})> message.

In the case where M_u has more than one children (in the *depth-first* virtual tree), upon receiving the PARENT message from the last child (let this be M_v) that contains the shared key, it first sends the PARENT<M_{u-1}, encrypt(Q_n, Q_{u-1})> message to its father and then informs its children by sending the special message UPDATE<M_i, encrypt(Q_n, Q_u)>. The child M_i that receives an UPDATE<M_i, encrypt(Q_n, Q_u)> message, decrypts it to acquire the secret key and forwards it to its child via a UPDATE<M_{i+1}, encrypt(Q_n, Q_i)> message. This ensures that the shared key will traverse the *depth-first* virtual tree all the way up to the group leader M_1 but also reach all the nodes that belong to a branch of the tree.

Implementation of this traversal technique requires for the active process to know exactly which of its neighbors are visited. To do so, the distributed algorithm apart from the SEARCH and PARENT message, uses a special VISITED message that allows each visited process to inform its neighbors (by broadcasting the message) that it has joined the traversal.

This protocol essentially builds a depth-first search spanning tree of a network, given

a distinguished node as its root (i.e. M_1) and its correctness essentially follows from the correctness of the sequential DFS algorithm, because there is no concurrency in the execution of this algorithm [6].

Handling JoinGroup Events. When a new member M_{n+1} wants to join a group, it must first be authenticated by the base station and get an ID and then contact the group leader (via the nearest group member M_u and through the virtual tree structure) a `JOIN<`M_u, M_{n+1}`>` message. The group leader replies by sending the old group key Q_n (again via the tree structure). Then M_{n+1} generates a random value k_{n+1}, computes the new group key $Q_{n+1} = k_{n+1}Q_n$ and sends it back to the group leader. Finally, the group leader sends an `UPDATE` message to all group members, again by using the virtual tree. The need to contact the group leader is necessary in cases of more than one nodes joining the group simultaneously, in which case, the group leader delays the `UPDATE` message until all new nodes have joined.

Handling LeaveGroup Events. In the case that a member M_u leaves the group, the group leader generates a random value $\overline{k_n}$ and computes a new group key $\overline{Q_n} = \overline{k_n}Q_n$. Then, as in the case of the *JoinGroup* event, it informs all group members about the new shared by sending an `UPDATE` message using the virtual tree. However, since the removal of the old member will disrupt the tree structure, the children of M_u must now contact the parent of M_u and update the tree structure. If this is not possible, i.e. because the parent of M_u cannot directly communicate with the children of M_u, the handling of the event fails, and M_1 is signaled to restart the *depth-first* tree construction and generate a totally new shared key.

Handling MergeGroup Events. When a group M' of nodes want to join group M, the procedure followed essentially expands the *depth-first* search tree to include the members of M'. The group leader of M' contacts the leader of M (via the nearest group member of M, M_u) by sending a `MERGE<`M_u, M'_1`>`. The group leader M_1 replies by sending the old group key Q_n (again via the tree structure) to M'_1 that is now denoted as M_{n+1}. When the message reaches M_{n+1}, the distributed sequential traversal algorithm continues as if the members of M' where unvisited members of M. In this sense, M_{n+1} computes a new random value k_{n+1}, calculates the point $Q_{n+1} = k_1 k_2 \ldots k_{n+1} P$ (which is now the new public key of M_{n+1}) and

shifts the control to the next non visited neighbor M_{n+2} (an old member of M'), through a SEARCH<M_{n+2}, Q_{n+2}> message. When all the old members of M' have been visited, the new shared key is propagated to the merged group through the use of the *PARENT* and *UPDATE*.

Handling PartitionGroup Events. Instead of trying to compute a new group key for the two resulting groups, when a *PartitionGroup* event is signaled the protocol simply reconstructs the *depth-first* search spanning tree and generates a new shared key for each group.

Periodic Group Maintenance. We here note that in order to handle the above events, the virtual tree can degenerate into a spanning tree that no longer fulfills the *depth-first* search criteria. Therefore, in order to balance the tree and also in order to guarantee key freshness the group leader periodically restarts the *depth-first* search and generates a new shared key.

Discussion. This group key protocol satisfies the first two cryptographic properties mentioned in Sec. 1.1 and in particular the *JoinGroup* event accomplishes forward secrecy while backward secrecy is guaranteed by the *LeaveGroup* event. Regarding the computational group key secrecy this is satisfied since, if an adversary silently overhears radio communication and captures data, he can not discover the group key as it is computationally infeasible to find any secret value k_i from the transmitted data (he has to solve an elliptic curve discrete logarithm problem). Additionally, since the protocol does not require that all group members communicate directly with each other via long-range transmissions. the silent adversary will only be able to listen to a limited number of messages given its actual physical location.

All group events are handled in $O(n)$ time (as in the case of GDH.3, assuming that a bounded number of retransmissions are required due to collisions) and require $O(n)$ message exchanges (again similar to GDH.3, although it is expected that $2n$ less messages need to be transmitted in the network). However, in contrast to GDH.3, this protocol evenly distributes energy consumption among the participants as each device has similar roles in terms of required computations and communication exchanges (energy-wise, the two most

demanding events). Balancing the energy dissipation among the sensors in the network avoids the early energy depletion of certain sensors (i.e. in GDH.3 participant M_{n-1}) and thus increases the lifetime of the system by preventing from early network disconnection [42]. In contrast to the GDH.3 protocol, this protocol does not assign different roles to the participating devices nor requires some of them to transmit more messages than others. The distributed sequential traversal ensures that the devices consume more or less equal amounts of energy as they perform the same number of events and communication exchanges leading to better energy balance.

1.7 Random Traversal Protocol

We now present a distributed protocol that totally avoids constructing and maintaining a distributed structure that reflects the topology of the network, does not require any strong communication primitive and relies only on simple short-range hop-by-hop message exchanges. This protocol [19] is particularly suitable for dense ad-hoc networks where the topology is subject to frequent and unpredictable changes. The devices are coordinated in a distributed manner using minimum communication overhead through the use of a mobile agent (software, mobile code) that traverses the network. The protocol imposes minimum communication overhead as the mobile agent is of small size (it fits in a single packet). The mobile agent moves *randomly* through the devices of the network and *no specific traversal strategy* (e.g. strictly sequential, in both directions) or global information (e.g. such as the structure of the network) is required. This software agent (mobile code) is responsible for constructing a common key among the participants of the group (by taking into account the contribution of all members of the group) and for the delivery of the established key to all participants.

Initially, all the participants of the group execute a protocol that generates a unique group ID, an initial secret shared key and calculates the group size. This constitutes the *setup phase* of the protocol. This information is delivered to all participants. The shared key is used to encrypt the information exchanged between any two parties in order to authenticate each other. We assume that there is no structure that reflects the topology of the network and

thus, there cannot be any guarantee that a member truly belongs to the network. No further global information is available to the participants. Note that the initial keys are used only for a short period of time (that we estimate) after which they are replaced by the keys established during the first round of execution of our protocol.

The protocol is executed in two stages. At the *first stage* all the sensor nodes contribute their random information to construct a shared secret key, which is shared to all nodes at the second stage. We suppose that every member in the group has agreed on the use of the same elliptic curve parameters, the number of participants is n and we will denote by M_i the i-th participant visited by the mobile agent. The final shared group key will be equal to $Q_n = k_n k_{n-1} \dots k_1 P$ where k_i are random values selected from every group member M_i and P is a fixed point in the elliptic curve. More precisely, the two stages of our protocol are as follows.

First Stage: This stage starts by activating participant M_1. M_1 selects a point P, generates a random value k_1 and calculates the point $Q_1 = k_1 P$. Then, he constructs a mobile agent in which he puts the point Q_1, encrypts the agent with the shared key and transmits it to a random neighbor. Suppose that this neighbor is participant M_2. M_2 decrypts agent's data, acquires point Q_1, generates a random value k_2 and computes the point $Q_2 = k_2 Q_1$. Then, he updates agent's information with the point Q_2, encrypts the agent and sends it to a random neighbor. The same process is followed by every member M_i that is reached by the agent.

Continuing this way, the mobile agent will eventually reach the last unvisited device. This device will be the first to calculate the shared key $Q_n = k_n k_{n-1} \dots k_1 P$. We here note that the encryption and decryption of agent's critical information is accomplished using as key the secret shared point Q_{old} which had been generated and shared at the previous round of our protocol.

Second Stage: At this stage the produced secret key is injected back into the network with a new random walk, in order to inform all participants about the new key. Suppose that the last unvisited participant of the first stage was M_n. M_n constructs a mobile agent in which he puts the point $Q_{n-1} = k_n^{-1} Q_n$ and transmits the agent to a random

neighbor M_i. Member M_i will multiply the point Q_{n-1} with his secret value k_i^{-1} and updating the agent's information will send it back to M_n. M_n will multiply the agent's value with k_n, send it again to M_i who will now be able to acquire the shared value Q_n by multiplying agent's context with k_i. Now M_i is ready to send the agent to another participant M_j by following the same process. If an eavesdropper wants to reveal the shared secret key, he will have to know one of the produced random values k_is. Note that the encryption and decryption of agent's critical information in the first stage can be avoided, provided that every participant of the group can authenticate its neighbors.

This scheme is strong enough for an environment where group membership is highly dynamic, key lifetime is short, and little data is available for cryptanalysis. The protocol produces a shared key using a contributory component from each group member. This component is incorporated along with the other members' components forming the shared group key. Since the group key must be changed periodically due to security requirements, each member must have the ability to quickly generate a random value and perform a decryption operation in order to decode the agent's message. We now show how to guarantee the fulfillment of the cryptographic properties presented in Sec. 1.1 using the main membership events. Moreover, these procedures can be applied in all cases of our protocol, i.e. either there is one agent or more. The main membership events include single member addition, single member deletion, group merge and group partition. In all cases, the members that are added or deleted must first be authenticated by the base station using their ID.

A Join Event occurs when a single member wants to join the existing group. In this case, the new member generates a random value k_{new}, contacts the nearest group member and computes the new group key $Q_{new} = k_{new}Q_{old}$. Then the new group key is shared among all the members by the agent.

A Leave Event occurs when a member wishes to leave the group, or is forced to leave it. In order to handle this event, a random member of the group is chosen to generate a random value and calculate a new shared key by multiplying this value with the old key. This participant will then generate a new mobile agent in order to distribute the new key to all existing members.

A Group Merge Event occurs when multiple potential members want to join an existing group. In order to handle this event, we have to develop a `MERGE` procedure in which a new agent is generated and initialized with the original group's shared key. Then the new agent must visit all new members and include them to the old group. After the establishment of the new shared key, the agent continues the walk to the participants of the expanded group.

A Group Partition Event occurs when multiple members leave the group with or without forming their own subgroup. To handle this event, a random member is chosen in each partitioned subgroup (e.g. using a leader election algorithm, see [37]), it generates a new random value and computes the new group key for the partitioned subgroup by multiplying this value with the old group key. These new sub-group keys are delivered to all members of each subgroup by the corresponding agents.

Let's now cosider some *correctness* issues of the protocol, i.e., some fundamental properties for any protocol that tries to establish a common shared key among the participants of a group. We assume here that the duty cycle of the sensor devices of the network are determined by application protocols (i.e. when to enter sleep mode and when to wake-up) and that the decision is *independent* of the motion of the mobile agent (i.e. we exclude the case where the devices are deliberately trying to avoid the mobile agent, or enter sleep mode when the mobile agent is located in the device). Moreover, we assume that the sensor devices of the network have sufficient power to support communication. Recall that we assume that channels are safe (messages are delivered without loss or alteration after a finite delay).

In such a case, the mobile agent will eventually meet all the participants of the group with probability 1. In fact, and by using the Borel-Cantelli Lemmas for infinite sequences of trials, given an unbounded period of (global) time (not necessarily known to the sensor devices) the mobile agent will meet the devices *infinitely often* with probability 1 (since the events of meeting the devices are mutually independent and the sum of their probabilities diverges). This guarantees that the mobile agent will meet all the participants and collect their contribution to the shared key and, then, correctly distribute the established key within the group.

In order to estimate the *time-efficiency* of the protocol we model wireless sensor networks

by abstracting away the physical layer details as a graph $G(V, E)$ in Euclidean space. The set V is the set of all devices. The set E contains an edge from device u to v if u can directly transmit to v (this can be determined by considering path loss and the signal-to-noise ratio). We refer to G as the *transmission graph*. Under this model, we assume that the mobile agent employed by our protocol does a *continuous time random walk* on $G(V, E)$, without loss of generality (if it is a discrete time random walk, all results will transfer easily). We define the random walk of the mobile agent on G that induces a continuous time Markov chain M_G as follows: The states of M_G are the vertices of G and they are finite. Let s_t denote the state of M_G at time t. Given that $s_t = u$, $u \in V$, the probability that $s_{t+dt} = v$, $v \in V$, is $p(u, v) \cdot dt$ where

$$p(u,v) = \begin{cases} \frac{1}{d(u)} & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

and $d(u)$ is the degree of vertex u.

We know (Theorem 6.8, [39]) that the cover time of the random walk on G, i.e. the time required for the walk that started at vertex i to visit all vertices of G, is bounded from above by $2m(n-1)$, where $m = |E|$. This can also be expressed as $\delta(G)n(n-1)$, where $\delta(G)$ is the average degree of a vertex.

Now, by assuming that the sensor devices are random uniformly distributed on the area, the density of the network can be calculated according to [12] as $\mu(R) = \frac{n\pi R^2}{A}$, where R is the transmission range of the devices and A is the size of the area covered by the sensor network. Basically, $\mu(R)$ gives the number of sensor devices within the transmission radius of each device in region A. In our transmission graph model, this implies that the average degree $\delta(G) = \mu(R)$. Therefore the time required for the mobile agent to visit all devices (to collect their contributions) and then revisit all devices (to deliver the calculated key) is equal to two times the cover time of the random walk on G. Thus the time is bounded from above by $2 \cdot \frac{n^2\pi R^2}{A} \cdot (n-1)$, implying an $O(n^3)$ time efficiency. Note that Theorem 6.8 gives the same $O(n^3)$ upper bound for the complete graph K_n, whereas it is known that in such graphs the cover time is $\Theta(n \log n)$. In this sense, this time-efficiency bound can be further improved.

If we assume a more controlled sensor deployment strategy [2], such as a lattice shaped

network (e.g. where $\delta(G) = 4$), then the resulting transmission graph will be a *regular graph* and the corresponding cover times drop to $O(n^2)$.

1.8 Algorithmic Engineering

There is generally a considerable gap between the theoretical results and the implemented protocols. Advancements have been made in the physical hardware level, embedded software in the sensor devices, systems for secure sensing applications and fundamental research in new communication and security paradigms. Although these research attempts have been conducted in parallel, in most cases they were also done in isolation, making it difficult to converge towards a unified global framework. Most currently deployed solutions lack the necessary sophistication, innovation, and efficiency, while state-of-the-art foundational approaches are often too abstract, missing a satisfactory accuracy of important technological details and specifications. To be effective and to produce applicable results, it is important to encourage interaction and bridge the gap between fundamental approaches and technological/practical solutions.

The development of a wireless sensor network application requires a significant amount of resources not only in hardware but also in software development. In most of the traditional systems and network architectures the software development process is more or less standardized or at least enhanced in several ways. There are specifications and standardized protocols, well defined application programming interfaces and software libraries that programmers can built upon. Such facilities are absent in new and emerging technologies, thus the means to develop applications are limited. Developers frequently need to reinvent the wheel porting applicable components from other paradigms or implementing new solutions from scratch. On the other hand, in emerging paradigms and new networking architectures, there are many open problems and design aspects that are still under active research, new algorithmic solutions as well as software and hardware components are developed. In such systems ideally, scientific research progresses through an iterative process; new concepts are modeled and analyzed with theoretic tools. The results of the theoretic research are transub-

stantiated into algorithms and protocols that are implemented usually in some mainstream programming language through an algorithmic engineering process. Algorithms and protocols are evaluated through simulation and/or deployed in experimental testbeds; through the experimental evaluation, flaws in the initial design or unanticipated problems are traced, thus providing incentive to the theorists to improve and develop new analytical methods and models.

For this reason, it is expected that a systematic theory of *distributed algorithm engineering* will be developed. The term distributed algorithm engineering was first introduced in [43] and essentially it involves the considerable effort required to convert theoretically efficient and correct distributed algorithms to effective, robust and easily used software implementations on a simulated or real distributed environment, usually accompanied by thorough experimentation, fine-tuning and testing. Such a conversion process may lead to improved distributed algorithms through the experimental discovery of behaviors and properties that were not exploited in the initial theoretical version of the algorithm.

In order to evaluate the suitability of the protocols in wireless sensor networks we here consider the implementation of the protocols on the MICA2 mote architecture [1]. Currently, these devices represent the state of the art in wireless sensor networks technology based on commercial off-the-shelf hardware components and offer an 8-bit, 7.3 MHz ATmega 128L processor, 4 KB of primary memory (RAM) and 128 KB of program space (ROM) and 512 KB secondary memory (EEPROM) and a ChipCon CC1000 radio capable of transmitting at 38.4 KBps powered by 2 AA batteries.

1.8.1 Implementing Elliptic Curve Cryptography

In order to demonstrate the applicability of the protocols presented in this chapter, we used the elliptic curve version of Diffie-Hellman problem [21]. The reason is that elliptic curve cryptosystems use much smaller keys than conventional, discrete logarithm based cryptosystems (an 160-bit key in an elliptic curve cryptosystem provides equivalent security with a 1024-bit key in a conventional cryptosystem). This fact makes elliptic curves the

	Addition	Multiplication	Random	Encryption	Decryption
Running Time	$2.250sec$	$36.114sec$	$0.22sec$	$74.481sec$	$38.365sec$

Table 1.1: Running times of `EccM-2.0` for operations using 163-bit multiprecision integers

only reasonable choice for sensor networks, where the resources are very limited. Moreover, recent research has shown that public key cryptography based on elliptic curves is feasible to be used in sensor networks [25, 26, 36].

In software, based on the nesC programming language, the Elliptic Curve Cryptography module EccM [36], implemented specifically for TinyOS, allows to represent and carry out basic operations with multiprecision integers of 160-bit size. Given this particular selection of hardware/software we can evaluate the running times for generating random secret values k_i, multiplying the secret values with a point P and encrypting/decrypting them based on a given set of public/private keys. The running times shown in Table 1.1 were measured by the MICA2 device using the `SysTime`, TinyOS component that provides a 32-bit system time based on the available hardware clock. The results indicate that elliptic curves implementation is feasible in sensor devices, as time to perform an encryption and decryption averages out to $74.481sec$ and $38.365sec$.

1.8.2 Protocol Evaluation

Given the above running times for performing the necessary cryptography operations, we can investigate the performance of the key establishment protocols via simulation. The experimental evaluation is conducted with Power-TOSSIM [40] that simulates the wireless network at the bit level, using TinyOS component implementations almost identical to the MICA2 CC1000-based radio stack. Note that the amount of time spent executing instructions is not captured by TOSSIM. Regarding the generation of various different physical topologies of wireless sensor networks TOSSIM provides the `LossyBuilder` tool [34].

The evaluation presented here considers two different types of network topologies: (i) *lattice shaped* networks of $n = \sqrt{n} \times \sqrt{n}$ devices, where the spacing of the devices is set to $45feet$ and (ii) *random uniform* networks of n devices deployed in a square area of 50×50

feet. For both types of topologies the network size is set to $n = [16, 25, 36, 49, 64]$. The transmission range of the devices is set to 50 feet. Thus in the lattice shaped networks, the maximum degree is 4 and for the random uniform networks, the resulting transmission graph is fully connected and the degree is $n - 1$

All protocols rely on certain assumptions on the underlaying network. In order to provide the necessary high-level primitives for the protocol to be operational two additional modules are required: (i) the `Reckon` module that provides information on the identities of the neighboring nodes (that belong in the same group with the node) based on short `HELLO` messages and (ii) the `SafeSend` module that guarantees that messages are finally delivered to their destinations by periodically retransmitting messages until the destination confirms their safe reception. The GDH protocols require all devices to communicate directly with M_{n-1}. To be able to provide this primitive, a basic algorithm (the `Flood` module) is used that simply floods the messages in the network until they are received by their final destination. We here note that the `Bcast` module provided by TinyOS implements many-to-1 multi-hop routing and is thus unsuitable. Finally, since in GDH the devices need to know identities of all the group members (so that stage 1 can be carried out) we also implemented the `ReckonGlobal` module that provides information on the identities of all the nodes (that belong in the same group with the node) based on a simple flooding protocol of `HELLO` messages.

Given the above configuration, the performance of GDH.3, distributed sequential traversal (or DFS-based) and random traversal (or agent-based) protocols is evaluated. In our evaluation we included only these protocols as we believe that they are the most suitable for wireless sensor networks. Note that all implementations follow closely the protocol descriptions, are simple and have limited memory requirements.

As a starting point, let's consider the effect of network size on the communication efficiency of the protocols. The total number of transmissions performed by the devices when executing each protocol is shown in Fig. 1.2. In this graph, the results for lattice-shaped networks are included. Similar results hold for the random uniform networks considered, see Fig. 1.4 and Fig. 1.5. The results indicate that the random traversal, agent-based, protocol performs a large number of transmissions compared to the other protocols. This is explained by the

fact that while the agent is moving randomly within the network, each device may be visited multiple times until all the network is covered. On the other hand, the other two protocols build a tree-based structure and thus improve communication as each device is visited only a couple of times. Therefore, as the network increases in size, although the performance of all protocols increases, the random traversal protocol performs significantly more message transmissions than the other.

In order to have a better view on the energy efficiency of the protocols, consider Fig. 1.3 that depicts the power consumed by each device separately. Based on these results, the random traversal protocol consumes more energy than the other two, it manages to evenly distributes energy consumption among the participants. This is a result of the random motion of the agent that does not discriminate devices but more or less visits each device the same number of times (on average). This behavior is similar to the one achieved by the distributed sequential traversal protocol that similarly, assigns the same roles in terms of required computations and communication exchanges (energy-wise, the two most demanding events). On the other hand, in the GDH protocol, the energy consumed by devices 15 and 16 is extremely higher than the other devices; these devices may "die" much earlier than the others, especially if the protocol is executed multiple times.

We conclude by evaluating the time efficiency of the protocols. We observe that the time required for a point multiplication (used by all three protocols) dominates the overall execution time. In this sense, since all three protocols require $O(n)$ point multiplication operations, their execution times are more or less the same. However, more importantly we observe that GDH.3 and the distributed sequential traversal protocol operate based on a data structure that reflects the topology of the network, in order to be correct and establish a common key, the network must remain fixed during the whole protocol execution. So, even for small networks, we must guarantee that during the execution of these protocols, no device enters sleep mode, or is destroyed, and no additional devices are deployed in the network, otherwise the procedure must be repeated leading to further delays and certainly further power consumption.

1.9 Conclusions and Open Issues

This chapter presented five group key agreement protocols which can be used in wireless sensor networks. All protocols studied require that the participants are active throughout the establishment of the common key. It is realistic to consider that some of the devices might stop (due to a failure, or due to capture by an adversary). A future research direction is to come up with distributed protocols that tolerate stopping failures by establishing a common key even if a fraction of the devices fails.

Another characteristic property of all protocols is the sequential traversal of the group members. A future research direction is to parallelize this process and improve the time efficiency of the system. What's more, the approach of using an agent performing a random walk over the participants of the group seems to overcome fault tolerance issues in dynamic environments. On the other hand, the agent process of visiting every member of the group takes a considerable amount of time compared to other protocols. An interesting research direction will be the improvement of the protocol's time efficiency using a different methodology or more agents.

References

[1] Crossbow Technology Inc. MICA2 notes. http://www.xbow.com/Products/productsdetails.aspx?si
2005.

[2] I. Akyildiz, Y. Sankarasubramaniam W. Su, and E. Cayirci, "Wireless sensor networks: a survey", Journal of Computer Networks, 38:393–422, 2002.

[3] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik, "On the performance of group key agreement protocols", ACM Transactions on Information and System Security, volume 7(3), pages 457–488, 2004.

[4] E. Anton and O. Duarte, "Performance analysis of group key establishment protocols in ad hoc networks", Technical Report GTA-03-06, Universidade Federal do Rio de Janeiro, Brazil, 2006.

[5] G. Ateniese, M. Steiner, and G. Tsudik, "Determining the optimal configuration for the

zone routing protocol", IEEE Journal on Selected Areas in Communications, volume 18(4), pages 1–13, 2000.

[6] H. Attiya and J. Welch, "Distributed Computing: Fundamentals, Simulations and Advanced Topics", McGraw-Hill Publishing Company, 1998.

[7] K. Becker and U. Wille, "Communication complexity of group key distribution" 5th ACM Conference on Computer and Communications Security (CCS 1998), pages 1–6. ACM Press, 1998.

[8] I. Blake, G. Seroussi, and N. Smart, "Elliptic curves in cryptography", Technical report, London Mathematical Society Lecture Note Series 265, Cambridge University Press, 1999.

[9] A. Boukerche and S. Nikoletseas, "Wireless Communications Systems and Networks", Chapter: Protocols for Data Propagation in Wireless Sensor Networks: A Survey, Kluwer Academic Publishers, 2004.

[10] E. Bresson, O. Chevassut, and D. Pointcheval, "Dynamic group diffie-hellman key exchange under standard assumptions", Eurocrypt 2001, pages 321–336. Springer-Verlag,Lecture Notes in Computer Science 2332, 2002.

[11] E. Bresson, O. Chevassut, D. Pointcheval, and J. Quisquater, "Provably authenticated group diffie-hellman key exchange", ACM-CCS 2001, pages 255–264, New York, 2001. ACM Press.

[12] N. Bulusu, D. Estrin, L. Girod, and J. Heidemann, "Scalable coordination for wireless sensor networks: self-configuring localization systems", International Symposium on Communication Theory and Applications (ISCTA 2001), 2001.

[13] M. Burmester and Y. Desmedt, "Efficient and secure conference-key distribution", Security Protocols Workshop, pages 119–129. Springer-Verlag,Lecture Notes in Computer Science 1189, 1989.

[14] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system", Advances in Cryptology (EUROCRYPT 1994), pages 275–286, Springer-Verlag,Lecture Notes in Computer Science 950, 1994.

[15] M. Burmester and Y. Desmedt, "A Secure and Scalable Group Key Exchange System", in Information Processing Letters, Vol. 94, No. 3, pp.137-143, 2005.

[16] D. Burton, "Elementary Number Theory", McGraw-Hill Publishing Company, 4th edi-

tion, 1998.

[17] I. Chatzigiannakis, A. Kinalis, and S. Nikoletseas, "Wireless sensor networks protocols for efficient collision avoidance in multi-path data propagation", ACM Workshop on Performance Evaluation of Wireless Ad Hoc Sensor, and Ubiquitous Networks (PE-WASUN 2004), pages 8–16, 2004.

[18] I. Chatzigiannakis, E. Konstantinou, V. Liagkou, and P. Spirakis, "Design, analysis and performance evaluation of group key establishment in wireless sensor networks", ICALP Workshop on Cryptography for Ad hoc Networks - WCAN, Electronic Notes in Theoretical Computer Science, volume 2nd, 2006.

[19] I. Chatzigiannakis, E. Konstantinou, V. Liagkou, and P. Spirakis, "Agent-based distributed group key establishment in wireless sensor networks", TSPUC Workshop, IEEE Press, 2007, to appear.

[20] K. Y. Choi, J. Y. Hwang, and D. H. Lee, "Efficient ID-based Group Key Agreement with Bilinear Maps", in Public Key Cryptography - PKC' 04, LNCS 2947, pp. 130 - 144, 2004.

[21] W. Diffie and M. Hellman, "New directions in cryptography", IEEE Transactions on Information Theory, 22:644–654, 1976.

[22] T. Dimitriou, "Securing communication trees in sensor networks", 2nd International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2006), Springer-Verlag,Lecture Notes in Computer Science, 2006.

[23] R. Dutta and R. Barua, "Constant Round Dynamic Group Key Agreement", in *th International Conference in Information Security - ISC' 05, LNCS 3650, pp. 74-88, 2005.

[24] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", IEEE Transactions on Information Theory, 31:469–472, 1985.

[25] G. Gaubatz, J. Kaps, and B. Sunar, "Public key cryptography in sensor networks – revisited", 1st European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS 2004), pages 2–18. Springer-Verlag, Lecture Notes in Computer Science 3313, 2004.

[26] N. Gura, A. Pate, A. Wander, H. Eberle, and S. Shantz, "Comparing elliptic curve cryptography and RSA on 8-bit CPUs", Cryptographic Hardware and Embedded Systems (CHES 2004), pages 119–132. Springer-Verlag, Lecture Notes in Computer Science

3156, 2004.

[27] L. Harn and T. Kiesler, "Authenticated group key distribution scheme for a large distributed network", IEEE Symposium on Security and Privacy, pages 300–309, 1989.

[28] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks", 33rd IEEE Hawaii International Conference on System Sciences (HICSS 2000), page 8020, 2000.

[29] I. Ingemarsson, D. Tang, and C. Wong, "A conference key distribution system", IEEE Transactions on Information Theory, 28:714–720, 1982.

[30] J. Katz and M. Yung, "Scalable Protocols for Authenticated Group Key Exchange", in Advances in Cryptology – CRYPTO 2003, LNCS 2729, pp. 110-125, 2003.

[31] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement", ACM Transactions on Information and System Security, 7(1), pp. 60-96, 2004.

[32] P. Leone, L. Moraru, O. Powell, and J. Rolim, "A localization algorithm for wireless ad-hoc sensor networks with traffic overhead minimization by emission inhibition", 2nd International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2006), Springer-Verlag, Lecture Notes in Computer Science, 2006.

[33] P. Leone, S. Nikoletseas, and J. Rolim, "An adaptive blind algorithm for energy balanced data propagation in wireless sensor networks", 1st IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS 2005), Springer-Verlag, Lecture Notes in Computer Science 3650, pages 35 – 48, 2005.

[34] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications", 1st ACM International Conference On Embedded Networked Sensor Systems (SENSYS 2003), pages 126–137, 2003.

[35] L. Liao, "Group key agreement for ad hoc networks", Master's thesis, Ruhr-University Bochum, Germany, 2005.

[36] D. Malan, M. Welsh, and M. Smith, "A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography", 2nd IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004), pages 71–80, 2004.

[37] N. Malpani, J. Welch, and N. Vaidya, "Leader election algorithms for mobile ad hoc networks", 3rd ACM Annual Symposium on Discrete Algorithms and Models for Mobility (DIALM 2000), pages 96–103, 2000.

[38] M. Manulis, "Contributory Group Key Agreement Protocols, Revisited for Mobile Ad-Hoc Groups", Proceedings of the 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems - MASS'05, pp. 811-818, 2005.

[39] R. Motwani and P. Raghavan, "Randomized Algorithms", Cambridge University Press, 1995.

[40] V. Shnayder, M. Hempstead, B. Chen, G. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications", 2nd ACM International Conference on Embedded Networked Sensor Systems (SENSYS 2004), pages 188–200, 2004.

[41] J. Silverman, "The Arithmetic of Elliptic Curves", Springer Verlag, 1986.

[42] M. Singh and V. Prasanna, "Energy-optimal and energy-balanced sorting in a single-hop wireless sensor network", 1st IEEE International Conference on Pervasive Computing and Communications (PERCOM 2003), pages 50–59, 2003.

[43] P. Spirakis and C. Zaroliagis, "Distributed algorithm engineering", Experimental Algorithmics - The State of the Art, to appear.

[44] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to group communication", 3rd ACM Conference on Computer and Communications Security (CCS 1996), pages 31–37. ACM Press, 1996.

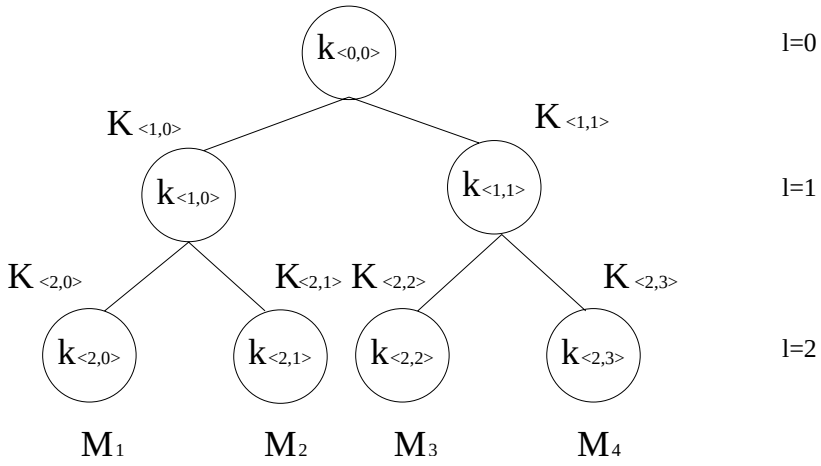

Figure 1.1: TGDH Binary Tree (n=4)

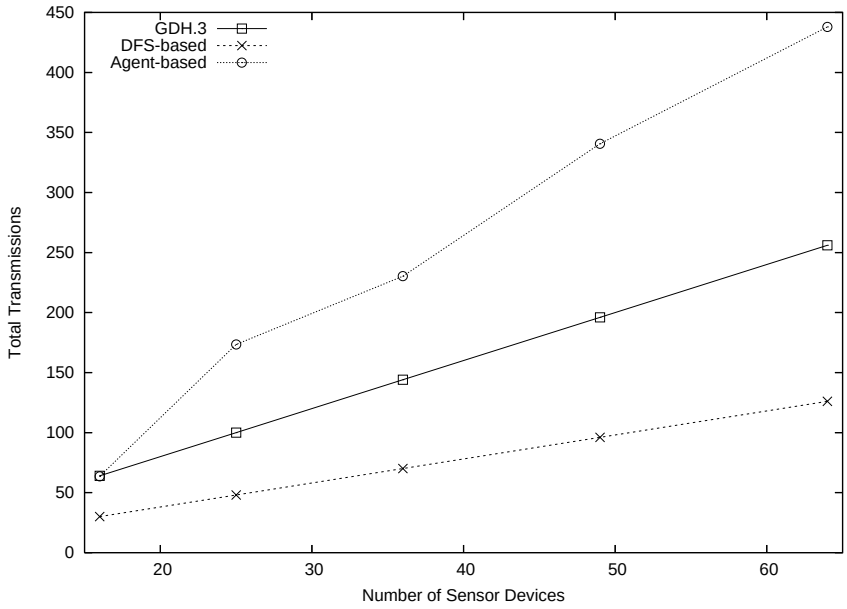

Figure 1.2: Total Transmissions when executing each protocol, for lattice-shaped networks

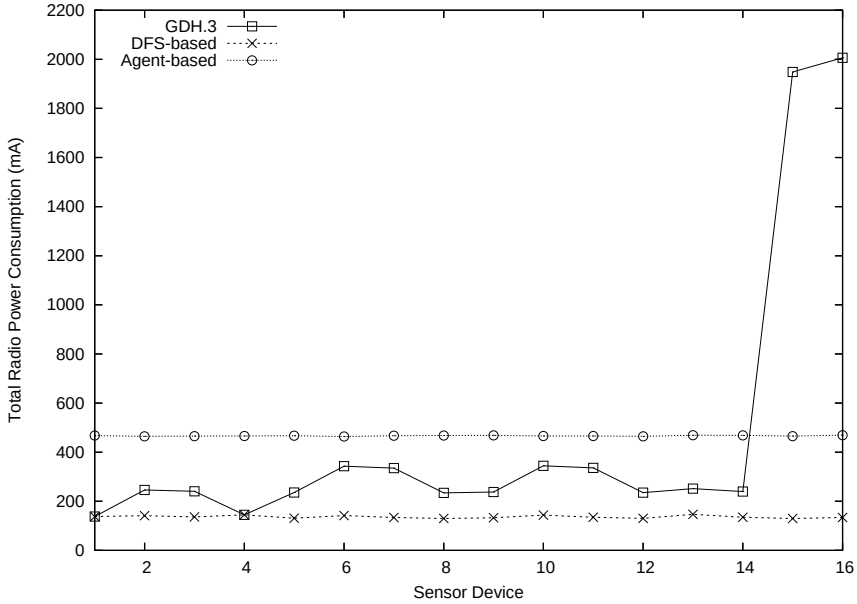

Figure 1.3: Power Consumption of Radio Equipment (mA) for each device separately, when executing each protocol, for lattice-shaped networks and $n = 16$

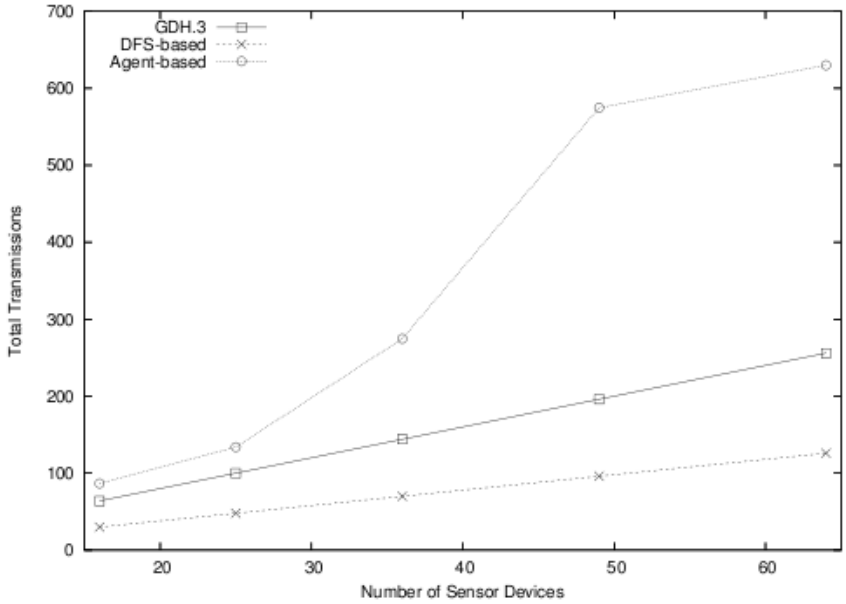

Figure 1.4: Total Transmissions when executing each protocol, for single-hop networks

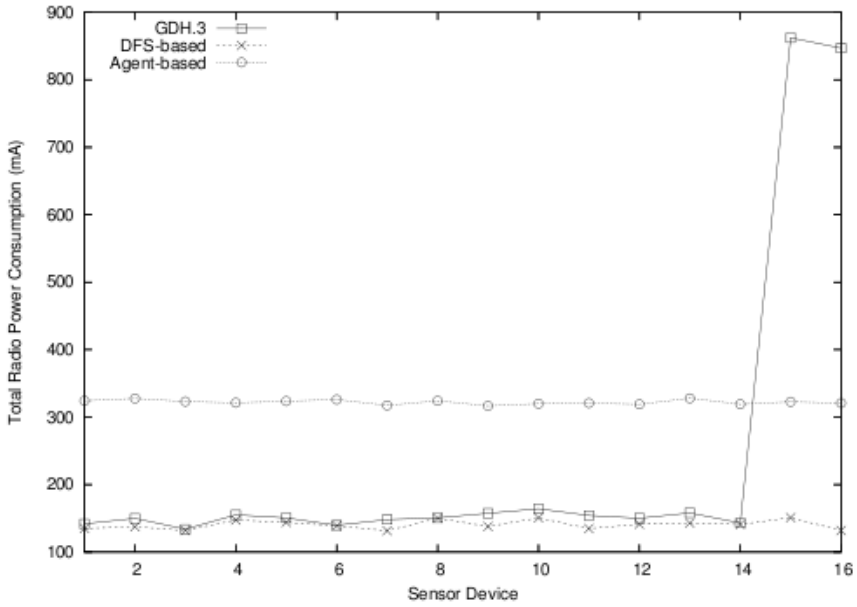

Figure 1.5: Power Consumption of Radio Equipment (mA) for each device separately, when executing each protocol, for single-hop networks and $n = 16$