An Efficient Key Establishment Scheme for Secure Aggregating Sensor Networks

Erik-Oliver Blaß
University of Karlsruhe
76128 Karlsruhe, Germany
+49 721 608-6416
blass@tm.uka.de

Martina Zitterbart
University of Karlsruhe
76128 Karlsruhe, Germany
+49 721 608-6400
zit@tm.uka.de

ABSTRACT

Key establishment is a fundamental prerequisite for secure communication in wireless sensor networks. A new node joining the network needs to efficiently and autonomously set up secret keys with his communication partners without the use of a central infrastructure. Most cited current research papers focus on a probabilistic distribution of sets of keys from larger *key pools* to new nodes. This results in unnecessary expensive communication and memory consumption, growing linearly with the size of the network, and guarantees secure connections only with a certain probability. This work presents a novel approach for efficient and secure key establishment of nodes joining the network by utilizing the fact that communication in sensor networks follows a paradigm called *aggregation*. Keys are split into shares and forwarded using disjoint paths in the network. The approach is self-organizing and minimizes memory consumption as well as radio transmissions efficiently – down to logarithmic behavior.

Categories and Subject Descriptors

C.2.0 [**Computer Systems Organization**]: Computer-Communication Networks—*Security and Protection*

General Terms

Security, Algorithms

Keywords

Sensor Networks, Key Establishment, Aggregation, Efficiency

1. INTRODUCTION

With the emergence of sensor networks in more and more situations of daily life, data security becomes an important aspect of this new technology. In many scenarios, confidentiality of transported data can be considered as critical, like data from sensors measuring patients health information, heartbeat or blood pressure details. Since all data is transported wirelessly between nodes, it is typically prone to interception and eavesdropping. As a result of cost or simple feasibility reasons, tiny sensor hardware is not tamper-proof and therefore an attacker willing to eavesdrop or even modify essential data might compromise a loyal node for his purposes or add a malicious node to the network.

Consequently, data transportation starting from a sensor node towards a data sink has to be protected against such threats. To protect data, a security relation between communicating nodes has to be initiated – usually by employing cryptographic keys. However, one of the crucial problems for security in wireless sensor networks lies in the distribution and establishment of these keys between nodes or in particular a new node joining a network and the nodes it is going to communicate with. Tiny microprocessor-based hardware of current sensor nodes with scarce memory and processing power prohibits the use of complex security protocols known from the PC/LAN world. Although public-key cryptography is becoming feasible on these platforms, its frequent use is still too expensive in terms of computing time and thus energy consumption. Therefore key establishment should generally rely on symmetric primitives. As wireless sensor networks lack permanent available infrastructure components, new key establishment protocols or schemes have to be self-organizing and must not rely on key-distribution centers or any central authorities.

This paper presents a new, efficient key establishment protocol for sensor networks. It utilizes the fact that nodes in a sensor network do not need to share keys with arbitrary other nodes in the network but only with those nodes lying on their *aggregation tree*. The benefits are:

- Efficient scaling with $O(\log n)$, n the number of nodes, behavior in terms of memory consumption and radio transmissions. Related publications consume memory and radio energy with linear $O(n)$ behavior. Our scheme uses only cheap symmetric encryption.
- Key establishment between a joining node and his partners in the network is done autonomously without usage of online infrastructures like key-distribution centers.
- The protocol is able to support dynamic changes of aggregation and nodes leaving the network.
- In contrast to related work, if no unexpected node failure occurs, this scheme guarantees secure key establishment not only with a probability $p < 1$.
- Finally, it operates securely even in the presence of malicious nodes.

The paper is structured as follows: After an overview about related work in Section 2, Section 3 describes the implications for key establishment that come with data aggregation. In Section 4, all prerequisites and assumptions for the key establishment are considered, especially a realistic attacker model, and the use of a so called *Master Device* is motivated. Section 5 introduces the new protocol, discusses its security, points out benefits, and finally compares it to the well-known work by Eschenauer and Gligor [7]. Conclusions are given in Section 6.

This publication is roughly based on preparatory work, published in an extended abstract [3].

2. RELATED WORK

Most current publications regarding key distribution or key establishment in wireless sensor networks assume that there is a requirement for every sensor node to be able to securely communicate with arbitrary other nodes within the network. It will be shown in Section 3 that this is a very strong and unnecessary assumption, which is not realistic in real-world sensor scenarios. However, as a result of the assumption to demand or support pairwise communication, pairwise symmetric keys between all n nodes within the network are necessary. Hence, each and every sensor node has to store about $O(n)$ symmetric communication keys in main memory.

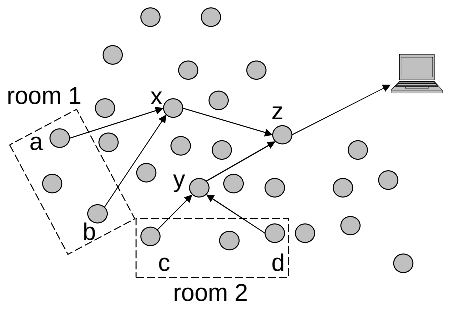
A typical representative for this class of distribution protocols is presented in [5, 7]. Every sensor node receives a subset of a very large set of *pool-keys* from the user. With a certain probability, it is then possible for two nodes willing to communicate to have at least one common key in their subset. To achieve a high probability for the availability of a pairwise key, the pool-size must be relatively large and grows with $O(n)$. In such a way, however, a lot of memory is wasted, which is especially critical for low memory sensor devices with, e.g., only 4KByte RAM. Also secure keying cannot be guaranteed and is only possible with a determined likelihood. If two nodes do not share a common key within their subset, $O(n)$ communication steps are required to find a key path to each other. Along this path, there might still be a compromised node, making key establishment insecure. In a similar way, ref. [4] uses 2-dimensional key-spaces, reducing memory overhead to about $O(\sqrt{n})$, but the user has to know the total number of nodes in advance. This scheme is also not able to cope with nodes leaving the network.

Other publications focus on the use of a permanently available *base station* to distribute keys, e.g., ref. [12]. Online base stations are often named differently, like *mother* [15] or *rich-uncle* [9]. Nevertheless, they have in common that for key establishment between new nodes a trusted third party is always necessary – an extraordinary node with more computing power or more memory than the average simple node. However, depending on a base station for assisting in every key exchange is quite unrealistic, as in a real world sensor network this base station might not be available at all times.

Finally, there is the idea of an attacker being limited in his capabilities: He should not be able to monitor key establishment, e.g., during initial sensor deployment [1]. This might be a realistic approach in certain scenarios in which a fixed number of nodes is statically deployed and the attacker does not know anything about the to-be-installed sensor network. However, in a dynamic environment with new nodes joining or leaving the network from time to time, this approach is inappropriate.

3. AGGREGATION

Typically, sensors report measured data, e.g., room temperature or the actual blood pressure, towards a *data sink*. On the way to the sink, data can be aggregated by so-called *aggregation nodes*. These nodes are able to collect data from other sensor nodes and process it. Aggregation nodes forward the resulting *aggregate* further to the sink.



(a) Temperature measurement in two rooms

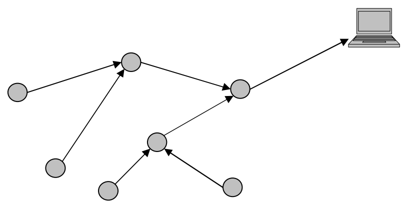

(b) Aggregation as a tree-like hierarchy

Figure 1: Example data aggregation in a sensor network

Figure 1(a) shows an example network. Sensor nodes a and b measure the temperature in room 1 at different positions, e.g., at the ceiling and the floor, and nodes c and d measure the temperature in room 2, respectively. Represented as a laptop, the sink is however only interested in the mean temperature of the complete building. Therefore, the sensor application or a sophisticated middleware establishes a tree-like scheme for data transmission. Aggregation node x collects temperature measurements from nodes a and b and computes their mean value. This is forwarded to aggregation node z. Aggregation node y does the same for nodes c and d. Finally, node z computes the mean temperature for the whole building and reports it to the sink. As shown in Figure 1(b), this communication scheme forms a hierarchy, sensor nodes (vertexes) and communications paths (edges) form a graph, more precisely a tree. The question whether communication within this tree is overlay communication or not, i.e., if there are multiple *hops* necessary to reach x from a, is negligible at this point. From an application point of view, it is just necessary to have *end-to-end* secured data transport between a and x.

Contrary to assumptions made in related work, the major observation here is that in this typical scenario, sensor nodes do not have to arbitrarily communicate with every other node. Nodes talk to each other only by a communication scheme, i.e., within the aggregation tree. Sensor node a for example has to exchange data only with node x, therefore a needs a shared key with x, aggregation node y needs a key with node z. Communication is, however, unlikely to happen between nodes from other categories, like communication between a temperature sensor and a light sensor. Also communication between nodes within the same category will never happen. Nodes a and b as well as c or d will most likely never exchange data among each other. As mentioned above, they might *route* or *forward* data in multi-hop situations, but there is no need for an applied end-to-end communication.

However, to ensure authenticity and data-origin-proof, it will also be necessary for, e.g., the sink to verify that certain received aggregated data has been aggregated in a "correct" way, set aside the meaning of correct. As an example, to verify aggregated data from z with reasonable certainty, the sink will have to talk securely to x and y. To check whether x aggregated correctly, the sink has to talk to a and b. It is not in the scope of this work to introduce an efficient algorithm for an authentic data aggregation, but important to point out that therefore secure communication has to be established not only between the vertexes in the aggregation respectively communication graph: Keys are also necessary between a vertex and its grandfather, great-grandfather etc. tree upwards. Every nodes needs keys with all other nodes lying on its aggregation path. So, sensor a needs to have a shared key with x but also different shared keys with z and the sink to enable possible additional authenticity verifications or data origin proofs.

4. PREREQUISITE

This section describes an *attacker* or *threat model*. We also introduce a *Master Device*, a mandatory device for user interaction during a new nodes network join, and motivate its use.

It is further assumed that the general existence of a sink in a sensor networks is quite differently than the existence of a base station. Although, in Figure 1 shown as a laptop, the sink may be the user's device for interaction with the network, it can still have similar hardware constraints or availability problems as any other node within the network. It should and does not take a special role during key establishment, like a base station.

4.1 Attacker Model

Data transmission in wireless sensor networks is done using a vulnerable channel: wireless radio. The danger of eavesdropping wireless communication and modification of confidential data demands encryption (with cryptographic keys). However, keys need to be established or distributed among communication partners prior to any communication. It is quite likely that the attacker is present not only during an applications communication phase, but also during key establishment. Because of cost and simple feasibility reasons, the simple sensor nodes hardware is not tamper-proof. Furthermore, as sensor devices are often placed in non-controlled or public environments, it will be therefore possible for the attacker to access them and to take over at least one or more regular nodes [14]. By compromising a node, the attacker will learn all its stored secrets, their keys, and will control their actions. Security protocols often assume only one malicious node in the network. However, it might be possible for the attacker to capture and re-program $k > 1$ nodes. In every scenario, the attacker has a certain amount of money, time, and effort he is able or willing to spend on compromising at most k nodes. As all nodes are assumed to be similar difficult to access and to compromise, his amount does not change with the total number of nodes available. Consequently, by capturing one or more nodes from the network, the attacker might pretend to be a legal player to a new node entering the network. By imitating good behavior, all compromised nodes might work together and take part in an establishment protocol to get to know the new key to be exchanged. Therewith, the attacker is capable of decrypting all further communications using this key. The major challenge for a secure key establishment protocol is to cope with the presence of compromised nodes, not exactly knowing which nodes are malicious.

In the following, firstly the presence of exactly one malicious compromised node is assumed for simplicity and better understanding. However, the key establishment scheme can be extended easily to k attackers. First, 5.2 describes the scheme with the presence of only one malicious node, Section 5.4 extends the scheme to cope with k malicious nodes.

4.2 Use of a Master Device

Assume the situation of a new sensor node, say i, going to join the network. Node i does not know anything about the network, because i has just been bought. There must be something like an initial deployment of trust or a *trust anchor* to i that will help i to further establish keys and make it securely join the network. Such a *pairing* for the network can never be done autonomously without any user interaction, because otherwise an attacker would also be able to legally add an arbitrary number of malicious nodes to the network. Consequently, the user or a special device, the so called Master Device (MD) representing the user, must establish the initial pairing. The act of pairing itself, as a preparation for a new node, can be done using a *location limited channel*, a secure channel, e.g., physical contact, between MD and a new node [2]. This allows secure information exchange between MD and the new node. Examples for appropriate Master Devices are rings or key fobs like *iButtons* [6]. See [8] for a discussion on Master Devices.

User or MD interaction with the sensor network is only necessary during an a priori pairing and the MD does not know anything about the current network configuration at all time: MD is stateless with respect to the network topology or configuration. During normal sensor network operation, MD is offline and can not assist nodes, so it is not an online server.

It is quite obvious that one has to take special care of an MD's security. If an MD is stolen or compromised, an attacker may misuse it to legally add malicious nodes to the network, give away secrets, and thereby compromise the whole network. However, it is fair to assume a tamper-proof, small, and handy MD which is carried around by the user and taken special care of, e.g., a key fob.

4.3 Notation

This paper uses quite a simple notation: A symmetric key K shared between node x and node y is called $K_{x,y}$. K is symmetric, that is, $K_{x,y} = K_{y,x}$. Encryption of data using K is denoted $C = E_K(\text{data})$. Decrypting C is not explicitly used here.

5. KEY ESTABLISHMENT SCHEME

The key establishment scheme is defined inductively. Assume that at a certain point in time, a subtree of an even larger sensor network's aggregation tree looks like the example in Figure 2. Nodes e, f and g send their measured values to b, which aggregates them and sends the resulting aggregate to a and so on. The vertexes in this graph simply indicate a direct aggregation-relation between nodes. This implies that secret keys have been established between adjacent vertices like g and b, but as discussed in Section 3, it is important to notice that there are also pairwise keys between, e.g., g and a – keys between a node and every predecessor on its aggregation tree. The actual network and key configuration at this point may be seen as an *induction hypothesis*. At this time, a new node, say i, wants to enter the network. It will have to talk to aggregation nodes $f, b, a, \ldots$ as seen in Figure 2 and therefore keys must be established between those nodes and i. Details about how i determines its aggregation path $f, b, a, \ldots$ will be given in Section 5.6. For now, it has to be assumed that i simply knows the nodes it needs to share keys with.

5.1 Pairing of New Nodes

As explained in Section 4.2, i has to be prepared or paired by MD to properly enter the network. In this protocol, pairing works as follows: Only MD knows a hard-coded key K_{MD}, which might have been generated during its fabrication and never leaves the device. Using K_{MD}, MD is able to compute a pairwise key with every sensor node at any time. Every node is assumed to have a

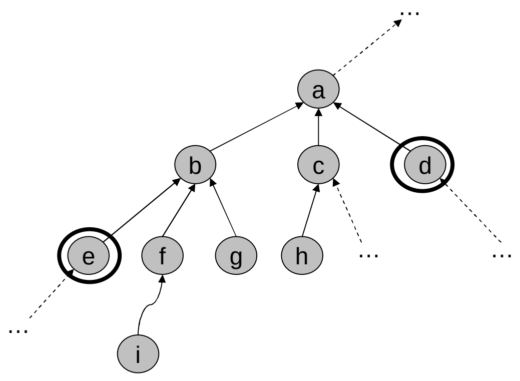


Figure 2: Node i joins the network and gets keys for communication with random nodes e and d.

unique identity ID, like f or g or any other unique identification *ID* respectively. MD computes the pairwise key with node ID by calculating $K_{\mathrm{MD,ID}} = E_{K_{\mathrm{MD}}}(\mathrm{ID})$. During the initial pairing with a new joining node ID, MD hands out $K_{\mathrm{MD,ID}}$ to ID using a secure location limited channel, e.g., physical contact. MD, however, does not need to store $K_{\mathrm{MD,ID}}$. MD can be completely stateless. $K_{\mathrm{MD,ID}}$ can easily be re-computed. As a result, every node that has ever joined the network successfully, possesses a pairwise secret key with MD. From now on, MD is in the position to send messages to node ID that are encrypted with $K_{\mathrm{MD,ID}}$. Receiving those messages and decrypting them with $K_{\mathrm{MD,ID}}$, the node knows that they are originated from MD. This technique allows MD to hand out *tickets* to new nodes. Tickets are encrypted specifically for older nodes already in the network and will help a new node to introduce itself to the older nodes, as described below.

So, with i going to join the network, MD generates $K_{\mathrm{MD},i}$ and gives it to i. After that, it randomly chooses two nodes from the existing network, e.g., e and d. Section 5.5 discusses how two nodes are randomly chosen by MD without knowing the exact network configuration. Again, assume for now MD to be in the position of picking two nodes like e and d randomly. MD computes and hands out two tickets T_e and T_d and two keys $K_{e,i}$ and $K_{d,i}$. This is similar to Kerberos with MD as KDS [11].

In conclusion, i gets from MD using a location limited channel:

- $K_{\mathrm{MD},i}$, a secret key between i and MD. Node i can thereby securely communicate with MD in the future.
- Random node IDs e and d as well as secret keys $K_{e,i}$ and $K_{d,i}$, so i knows whom to talk to during the rest of the protocol.
- Ticket $T_e = E_{K_{\mathrm{MD},e}}$(i, "is legal player", $K_{e,i}$) and Ticket $T_d = E_{K_{\mathrm{MD},d}}$(i, "is legal player", $K_{d,i}$)

5.2 Splitting and Distributing Keys

In the following, the term *key splitting* is used. The main idea is to split a key K perfectly into two key *shares* K_1 and K_2 by choosing a random number r of the same size as K and computing ($\oplus$ means XOR)

$$K_1 = r,$$

$$K_2 = K \oplus r.$$

Now K_1 and K_2 may be distributed to different nodes. This technique allows $K = K_1 \oplus K_2$ to be restored only if K_1 as well as K_2 are known to the same node. Knowledge of only one K_1 or K_2 will not reveal anything about K [13].

Node i generates a symmetric key $K_{i,f}$ for secret communication between i and f and splits $K_{i,f}$ into two shares K_1 and K_2. Because i possesses two tickets and keys for the two random nodes e and d, it uses these nodes to forward both shares towards f. Node i does the following:

- Node i sends T_e and T_d to e and d using its normal wireless communication facilities. As both tickets are encrypted with the corresponding key between MD and e or d, both nodes realize that the tickets are originally coming from MD. The term "is a legal player" convinces them about i going to be a legal node in the network. Using keys $K_{e,i}$ and $K_{d,i}$, they can now also securely communicate with i. Tickets T_e and T_d can be discarded.
- Node i sends
  $$C_1 = (i, E_{K_{e,i}}(K_1, f))$$
  to e. This is a request to forward K_1 to f. The preceding plain node ID i in C_1 just helps the receiving node e to understand that the following ciphertext is coming from i and can be decrypted with $K_{e,i}$.
- Node i requests d to forward K_2 to f in the same way. It sends
  $$C_2 = (i, E_{K_{d,i}}(K_2, f)).$$

Nodes e and d now accept i as a legal new node, both have shared keys with i, and possess i's key shares for forwarding to f.

Now, secure *key forwarding* begins. If e already had a pairwise key with f, say $K_{e,f}$, it would simply send

$$\gamma_1 = (e, E_{K_{e,f}}(i, K_1))$$

to f. However, as shown in Figure 2, there is no key between e and f. What e now does is to gradually find one of its predecessors with a secret key to f, simply by asking them. Node e starts asking its father b whether it has a key with f. If b did not have a key with f then e would go on and ask a etc. As f is taking part in the aggregation tree, one of e's predecessors will have a key with f in any case. Similar to the technique explained above, these forward request are secured with the pairwise keys all nodes along a tree-predecessor path possess: By the induction hypothesis, e.g., e and b share the key $K_{e,b}$ or e and a share the key $K_{e,a}$ and so on. In our case however, e's predecessor b already has the common key $K_{b,f}$ with f and reports this fact back to e. As e now has knowledge that b owns a key with f, it is quite clear to e that b's predecessor a must also have a common key with f. On the other hand, e knows a, because a is of course also along the path upwards the aggregation tree. And because a is a predecessor in e's aggregation tree, e has a common key $K_{e,a}$ with a as well.

Now e does the following:

- e splits K_1 into two shares, namely K_1^1 and K_1^2.
- It computes
  $$C_1^1 = (e, E_{K_{e,b}}(i, f, K_1^1))$$
  and
  $$C_1^2 = (e, E_{K_{e,a}}(i, f, K_1^2)).$$
- Finally, e sends C_1^1 to b and C_1^2 to a. This is shown in Figure 3(a).

After decryption, b and a send to f:

$$b:\ \gamma_1^1 = (b, E_{K_{b,f}}(i, K_1^1))$$

$$a:\ \gamma_1^2 = (a, E_{K_{a,f}}(i, K_1^2)).$$

This is shown in Figure 3(b).

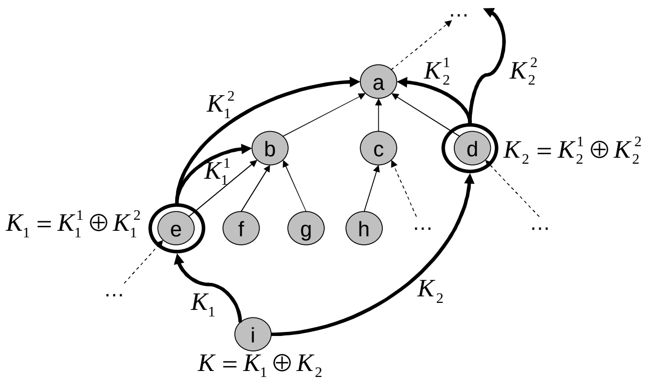
Node d carries out the same operations after receiving C_2 and generating K_2^1 and K_2^2. It will ask a and a's father (not shown in Figure 3) to forward K_2^1 and K_2^2 in a similar way to f. Note that it is no problem, if a has no father but is the root node, that is, the sink (cf. Section 5.3). Eventually, f receives $\gamma_1^1, \gamma_1^2, \gamma_2^1, \gamma_2^2$ and decrypts to $K_1^1, K_1^2, K_2^1, K_2^2$, see Figure 3(b). Therewith, it can compute $K_{i,f}$ by

$$K_{i,f} = K_1^1 \oplus K_1^2 \oplus K_2^1 \oplus K_2^2.$$

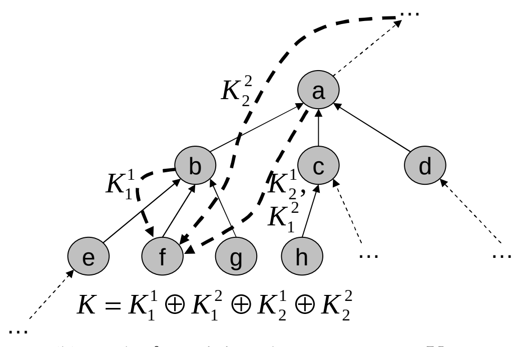
If one of e or d already had a key with f, say e, then f would have received only $\gamma_1, \gamma_2^1, \gamma_2^2$ giving K_1, K_2^1, K_2^2. Yet, f is able to compute $K_{i,f}$ by simple XOR:

$$K_{i,f} = K_1 \oplus K_2^1 \oplus K_2^2.$$

As a result, a secret key $K_{i,f}$ has been established between nodes i and f.



(a) Distribution of key shares tree upwards



(b) Node f receiving shares to restore $K_{i,f}$

Figure 3: Key split, distribution and receipt. Shares are sent encrypted, see text.

Using the above scheme, i establishes keys with b, a and all other predecessors along the aggregation path. It will ask e and d to forward a newly generated and split key $K_{i,b}$ to b and so on and so forth.

5.3 Security Discussion

This section explains the protocols resistance against *one* compromised malicious node for simplicity. As a result, all an attacker can do is some kind of *Denial-of-Service*-attack, i.e., denying a successful key establishment. He will, however, never be in the position to learn a new key, because all data exchange is individually encrypted.

What makes this protocol secure is the idea to split and distribute a new key in a way that no single node in the network can fully restore it. From a security point of view, it becomes obvious that therefore the following would be the worst scenario: Both initial random nodes are residing in the same subtree so that both of them find the same predecessor having a common key with f. For example, let h and c be the random nodes. Both nodes will recognize a as the predecessor with a key to f. However, even if a is malicious, it will not get all key shares, because h and c are also securely sending shares to a's father. The same holds for the situation where a's father is malicious. In another scenario with c being malicious, c might lie on h's request and might pretend having a key with f. Yet, h is going to send one key share to a, leaving attacker c with an incomplete set of key shares. Note: It may be possible that one of the predecessors during a key establishment is the sink itself. This is not a problem, because you generally assume to trust the sink. So if the sink turns out to be one of the predecessor taking part in an ongoing protocol cycle, there is no need to look for more predecessors – you already found a node which you can trust. Something similar holds, if the initially chosen nodes reside in different subtrees and find different predecessor with keys to f: It is impossible for one node to gather all the different key shares together.

It is, however, possible for a malicious node to intercept and drop encrypted key shares, i.e., to refuse forwarding, or to alter a key share before forwarding it. This will result in an incomplete or erratic set of key shares arriving at the destination node f. As a consequence, the destination node will assemble the shares incorrectly and compute a new key $K'_{i,f}$ which is different from $K_{i,f}$. Nodes i and f will thus not be able to communicate securely at all – ending in a Denial-of-Service attack on the establishment protocol. Detecting such a Denial-of-Service attack is quite simple: i and f will not understand each others encrypted data, yet they would not know which of the nodes taking part in the protocol misbehaved. In this case only unprotected communication between both nodes would be possible, as would be without any key establishment protocol. Wireless sensor networks are prone to Denial-of-Service in general. If an attacker successfully compromises a legal node in the network he can launch any type of Denial-of-Service attack.

Looking at Figure 2, you must not consider the vertexes between nodes as communications paths but only as application-wise aggregation flows, as mentioned above. Communication between, e.g., node b and a may be multi-hop communication involving a lot of other nodes in the whole network, maybe even e or f or whatever. This is not a security problem, as those multi-hop nodes are only forwarding end-to-end encrypted data between say b and a. Intermediate nodes will never learn anything from the data they forward.

5.4 Extension to Defend against Multiple Malicious Nodes

So far, the key establishment protocol is secure only in the presence of one compromised node. However, the user may assume the presence of up to k malicious nodes in the network. The proposed protocol can easily be extended to cope with k malicious nodes. The security goal here would be that even if k nodes may work together and assemble their key shares, they will never be able to completely restore the original secret key.

In the presence of k evil nodes, MD has to initially choose $k+1$ random nodes, see section 5.5 on how to do this. A new node i will split its new key into $k+1$ shares $K_1, \ldots, K_{k+1}$ distributing them to the initial nodes. Initial nodes will again split their received share further into $k+1$ new shares $K_1^1, K_1^2, \ldots, K_{k+1}^{k+1}$ using the algorithm above. Each of them has to find $k+1$ predecessors with a shared key to the destination node. As described above, this can be done gradually: If an initial node x finds a predecessor y claiming to possess a key, x will not only send one share to y but also different shares to all k predecessors of y.

Consequently, k nodes working together can only assemble k key shares. Because the secret is split into $k+1$ shares and distributed, there will however always be one non-compromised node ensuring security for the whole key.

5.5 Finding Multiple Initial Random Nodes

As described above, a new node i going to join the network initially needs tickets and keys for at least two (or $k+1$) random nodes already residing in the network. Finding these nodes is not easy for the user or his MD as the current network configuration is never known, i.e., the user is not aware which nodes are present in the network. However, there is a quite intuitive solution to this problem.

During pairing, nodes could get IDs in a sequence, i.e., a new node entering the network has a higher ID than the node that joined before. MD or the user would only need to store the total number of nodes or the ID of the last joining node. The current network configuration would still remain unknown to MD. If k malicious nodes are to be expected in the network, $k+1$ initial random nodes are required, e.g., two. To pair a new node i with ID ID_i entering the network, MD could determine $k+1$ initial nodes by selecting $k+1$ random numbers $r_1, \ldots, r_k \in \{1, \ldots, \mathrm{ID}_i - 1\}$. The initial nodes for i would therefore be $\mathrm{ID}_{r_1}, \ldots, \mathrm{ID}_{r_k}$.

For simplicity reasons, key establishment protocols such as [7] often implicitly assume immortal network hardware, where nodes will never leave the network they once joined. They are always reachable and can thereby assist in security protocol steps. In such an idealistic world, the proposed key establishment scheme would function perfectly. However, assuming static network configurations is quite unrealistic: In a real world scenario, nodes will exhaust their batteries, fail, are vandalized, and so on. Consequently, randomly choosing IDs could result in IDs of nodes that already left the network. In such a more dynamic sensor network, all required $k+1$ nodes are present only with a certain probability. If at least one of the nodes is missing, a secure key establishment is not possible.

It is therefore a good idea for MD to generally select more than $k+1$ nodes. The probability of randomly selecting at least $k+1$ nodes, which are still present in the network, rises by choosing $j > k+1$ nodes. Assume a network with a total number of n nodes that ever joined, i.e., n different nodes have ever been connected to the network. At the time node i is going to join, m out of these n nodes have already left, leaving $(n-m)$ nodes still present. The goal is to find, with a high probability, at least $k+1$ alive nodes by a random selection of j nodes. This is a combinatorial problem equivalent to the following question: Choosing randomly $j \geq k+1$ balls from an urn with n balls, m red and $(n-m)$ green, what is the likelihood of selecting at least $k+1$ green balls and $(j-k-1)$ red balls? This probability P can be computed as:

$$P = \sum_{k'=k+1}^{j} \frac{\binom{n-m}{k'}\binom{m}{j-k'}}{\binom{n}{j}}$$

P is a function which rises quickly with j as demonstrated in Figure 4. Three different scenarios are shown with the total number n of nodes ever joined being 100, 1000, and 10000. Every scenario

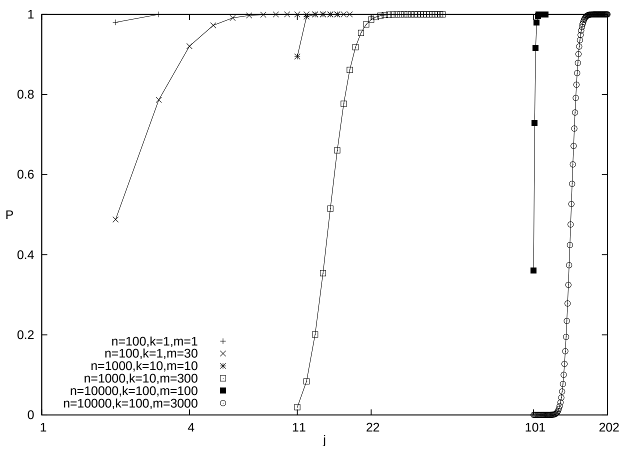


Figure 4: Probabilities of finding at least $k+1$ nodes alive in different network configurations

has a more powerful attacker: The number of malicious nodes k is $1, 10$ or 100. For every scenario m depicts the number of nodes which are non-reachable for a new node i, e.g., because of a wasted battery or failures. m is shown being 1% or 30% of n to represent a low and a high number of node failures. Note that the k malicious nodes are always part of the $(n-m)$ living nodes. In Figure 4, the y axis represents the probability of finding at least $k+1$ living nodes if m out of n are already dead and MD selects j out of n nodes randomly. The x axis is scaled logarithmically.

As a result, you can see P reaching high values of more than 99% as soon as j gets close to $2(k+1)$. For the three different scenarios with $k = 1, 10$ or 100 malicious nodes, $2(k+1)$ would be 4, 22, and 202 accordingly. It is important to notice that $2(k+1)$ is an upper bound regardless of n: If the user or MD assumes only $k = 1$ malicious node in $n = 1000$ sensors, the likelihood of finding $2(k+1) = 4$ living nodes within 1000 is far greater than finding 4 nodes in $n = 100$. If k malicious nodes are assumed to be in the network at a given time and node failure is possible, MD will select not only $k+1$ but $2(k+1)$ nodes from the network randomly. $2(k+1)$ is constant and does not depend on the total number n of sensor nodes. It therefore scales perfectly.

5.6 Finding Key Paths to the Sink

Before a new node i can establish secure keys with all nodes on its aggregation path $\mathbb{P} = \{f, b, a, ..\}$, i has to know for sure that these are actually future communications partners and i needs keys for them. i has to know $\mathbb{P}$. From a security perspective, this is again delicate: If i simply broadcasts and asks for its path, an attacker might intercept and provide i with a fake path consisting only of malicious nodes. This is *not* a particular problem for a key establishment protocol and one could argue about this being application dependent in general. A clever and secure middleware mechanism might provide i with the aggregation path setup, this information might be distributed to i by *out-of-band* mechanisms, e.g., i could ask a secure distributed service directory for this [8]. Also the user knows that i is a new heartbeat sensor and simply knows the first aggregation point for heartbeat sensors in the network.

On the other hand, if all nodes on the aggregation path $\mathbb{P}$ for heartbeat sensors are aware of their actual function, they will know that a new heartbeat sensor i has to share secret keys with them. Therefore you could again use the technique of initial random nodes and secret distribution to learn about f, b, and a. As i has introduced itself to at least e and d, it could ask both of them to look for the heartbeat-aggregation path $\mathbb{P}$. e would gradually and securely ask its predecessors in the tree, i.e., b first, about the aggregation path for a new heartbeat sensor. Node b would answer with the path f, b, a and so on. As b already knows the path, a would go on and further ask k predecessors of b. Only if all received answers do not differ, e will respond to i with the path $\mathbb{P}$. If d would do the same and answer to i with the same aggregation path $\mathbb{P}'$, i can be sure even in the presence of k malicious nodes.

5.7 Aggregation Changes

A sensor node like b may fail or die unexpectedly. In this case, all of b's keys are lost. This is not a significant problem, because communication between all predecessors and successors of b will continue securely, as they have secure keys with each other. All nodes along b's aggregation path will even quite likely notice b's failure, if b is not responding to encrypted messages. So, node failure will not impose more problems than node failure without using the key establishment protocol would do.

On the other hand, nodes may leave the network voluntarily. The user, application or software component responsible for all aggregation setups may decide a change of a small part or the whole aggregation tree. Assume node b has to be replaced by node c, as of Figure 2, and node b and c are aware of the configuration change. As both nodes do not have a shared secret key with each other, the idea is again to gradually find $k+1$ common predecessors. In the presence of k malicious nodes, b and c can then securely exchange a key. As described above, b would generate a random key K_b and split it into $k+1$ parts $K_b^1, \ldots, K_b^{k+1}$ distributing them to the predecessors. The predecessors will forward all key shares securely to c. c itself would do the same with its own generated key K_c and $K_c^1, \ldots, K_c^{k+1}$. Both nodes will receive all key shares from their counterpart and compute $K_{bc} = K_b \oplus K_c = K_b^1 \oplus \ldots \oplus K_b^{k+1} \oplus K_c^1 \oplus \ldots \oplus K_c^{k+1}$.

5.8 Comparison and Benefits of the Protocol

The performance of this protocol depends on the configuration of the aggregation tree. Typically, an aggregation tree will not be a degenerated tree, i.e., nodes will have more than one child. Therefore, it is fair to assume d, the mean degree of nodes in an aggregation tree, to be $d > 1$. The height h of the tree will be $h = \lceil \log_d n \rceil$. Therefore about $\log_d n$ keys are necessary for a new node entering the network.

Every aggregation node needs pairwise keys with every node in its d subtrees. To prevent nodes higher up in the aggregation tree from storing a lot of keys for all nodes in their subtree, for example, the sink would have to store n keys, the following technique offers further memory savings: similar to MD, each node knows a unique secret hard-coded key K_{ID}, e.g., node b knows K_b. These hard-coded keys may be generated during nodes' fabrication time. Assume node i enters the network and needs to establish a key with node b, residing higher up in the aggregation tree. As mentioned before, they establish key $K_{i,b}$. However, now both nodes do *not* store $K_{i,b}$ as their pairwise secret key, but use it as temporary key to establish a permanent Key $K'_{i,b}$ as follows: the node higher up in the aggregation tree, in this case b, computes $K'_{i,b} = E_{K_b}(i)$. Using the already established temporary $K_{i,b}$, node b securely sends $E_{K_{i,b}}(K'_{i,b})$ to i and discards $K'_{i,b}$. Node i is able to decrypt $K'_{i,b}$ and stores it. From now on, both nodes will use $K'_{i,b}$ as their pairwise secret key. If i wants to communicate with b later on, b can easily restore $K'_{i,b}$ by re-computing $K'_{i,b} = E_{K_b}(i)$, just like MD in Section 5.1. Consequently, this helps to drastically reduce the amount of required memory. Nodes on the lth-layer of aggregation need only to store l keys, i.e., $l-1$ keys of their predecessors in the aggregation tree plus their own hard-coded key. For example, the sink only needs to store 1 key, leaves, i.e., nodes on the lowest layer of the aggregation tree, need to store at most $h = \lceil \log_d n \rceil$ keys.

In conclusion, the total number of keys each node has to store in its valuable main memory scales with $d \log_d n = O(\log_d n)$. This is far superior compared with the $O(n)$ pairwise keys necessary in related work.

If k malicious nodes are assumed a priori, the establishment of a key needs $2(k+1)$ initial random nodes. Each random node will again split its share into $k+1$ further shares and find predecessors. So about $2(k+1)^2 \log_d n = O(\log_d n)$ communications steps are necessary to securely establish a new key. Again this scales far better than the $O(n)$ communication steps of related work, especially [7], which can not even guarantee secure multi-path communication. The *agreement* on a shared key is also quite simple in this paper, as it simply needs the generation of a random number on $k+1$ XOR-operations. In [7] a lot of symmetric encryptions and decryptions up to the *poolsize* (can grow to $O(n)$) are mandatory per key establishment.

Furthermore, if nodes fail, the probability of finding a path for a shared key can degrade rapidly – this is not covered at all in [7]. If $2(k+1)$ initial random nodes are chosen, this protocol guarantees a probability of $> 99\%$ for a secure key establishment, even in the case of 30% failed nodes.

Finally, in [7], the user must discover all compromised nodes for the protocol to function properly and securely, which is very unrealistic and doubtful to assume. Our protocol can deal with compromised nodes knowing all secret keys they are supposed to know anyway due to normal operation and it even copes with malicious nodes not revealing themselves to the user.

The proposed scheme utilizes only symmetric cryptography, like encryption or decryption. This is very time and memory efficient [10], so the use on a variety of different sensor platforms is possible.

6. CONCLUSION

This work presents a new efficient key establishment scheme for wireless sensor networks. Communication flow in sensor networks typically forms a hierarchical, tree-like aggregation. Within such a tree, an efficient, memory and radio transmission saving key establishment is proposed, which scales only with logarithmic behavior compared to the more expensive linear behavior of related work.

Every node establishes and stores only those keys it would need because of its task anyhow. Even in the presence of k malicious nodes, the scheme allows secure and autonomous key exchange without relying on any online central server or base station. The user does not need to know the exact network configuration at any time. Finally, the scheme is able to support dynamic network behavior, not only joining nodes, but also node failure, voluntary node leave, and changes of aggregation.

An efficient establishment of keys however only provides the basis for a secure and authentic data transport, which is taking aggregation into account. It is an open question, how data origin verification and authentication can be protected efficiently in an aggregating sensor network.

7. ACKNOWLEDGMENTS

The authors wish to thank Roland Bless, Artur Hecker, Michael Conrad, Curt Cramer, and André Huttel for their support.

8. REFERENCES

[1] R. Anderson, H. Chan, and A. Perrig. Key infection – smart trust for smart dust. In *ICNP*, 2004.

[2] D. Balfanz, D. Smetters, P. Stewart, and H. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Symposium on Network and Distributed Systems Security*, 2002.

[3] E.-O. Blaß, M. Conrad, and M. Zitterbart. A tree-based approach for secure key distribution in wireless sensor networks. In *Real World Sensor Networks*, June 2005.

[4] H. Chan and A. Perrig. PIKE: Peer intermediaries for key establishment in sensor networks. In *INFOCOM*, 2005.

[5] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Security and Privacy Symposium*, 2003.

[6] Dallas Semiconductor Corp. The iButton, 2005. `http://www.ibutton.com`.

[7] L. Eschenauer and V. Gligor. A key management scheme for distributed sensor networks. In *ACM CCS*, 2002.

[8] H.-J. Hof, E.-O. Blaß, T. Fuhrmann, and M. Zitterbart. Design of a secure distributed service directory for wireless sensornetworks. In *European Workshop on Wireless Sensor Networks*, 2004.

[9] Y. Law, R. Corin, S. Etalle, and P. Hartel. A formally verified decentralized key management architecture for wireless sensor networks. In *Personal Wireless Communication*, 2003.

[10] Y. Law, J. Doumen, and P. Hartel. Benchmarking block ciphers for wireless sensor network. In *IEEE Mobile Ad-hoc and Sensor Systems*, 2004.

[11] S. Miller, B. C. Neuman, J. I. Schiller, and J. Saltzer. Kerberos authentication and authorization system. Project Athena Technical Plan, MIT Project Athena, 1998.

[12] A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar. SPINS: security protocols for sensor netowrks. In *Mobile Computing and Networking*, 2001.

[13] G. J. Simmons. *An introduction to shared secret and/or shared control schemes and their application*. Contemporary Cryptology. IEEE Press, 1992.

[14] F. Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, 2002.

[15] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *7th International Workshop on Security Protocols*, 1999.