

Fast and Scalable Key Establishment in Sensor Networks

Tassos Dimitriou, Ioannis Krontiris and Fotios Nikakis

Athens Information Technology

[tdim, ikro, fnik]@ait.gr

Abstract

We present a protocol for key establishment in sensor networks. Unlike random key pre-distribution schemes our protocol does not rely on probabilistic key sharing among sensors. During a quick bootstrapping phase, nodes use their pre-deployed key material to form groups or clusters of small size that share a common key. Inter-cluster communication is then achieved by nodes sharing cluster keys. Our scheme is *scalable* and provides *resiliency* against node capture and replication. This is due to the fact that keys are localized; keys that appear in some part of the network are not used again. So, even if a node is compromised and its keys exposed, an adversary can have access only to a very small portion of the network centered around the compromised node. What is more important however, is that our protocol is optimized for *message broadcast*; when a node has to broadcast a message it doesn't have to encrypt it each time with a key targeted for a specific neighbor. This saves energy and makes re-transmissions unnecessary. Furthermore, our scheme is suited for *data fusion* and aggregation processing; if necessary, nodes can “peak”

at encrypted data using their cluster key and decide upon forwarding or discarding redundant information.

Index Terms

Sensor Networks, Wireless Communication, Network-level security and protection, Authentication

I. INTRODUCTION

Sensor networks have attracted much scientific interest during the past few years. These networks use hundreds to thousands of inexpensive wireless sensor nodes over an area for the purpose of monitoring certain phenomena and capture geographically distinct measurements over a long period of time (see [1] and [2] for a survey). Nodes employed in sensor networks are characterized by limited resources such as storage, computational and communication capabilities. The power of sensor networks, however, lies exactly in the fact that their nodes are so small and cheap to build that a large number of them can be used to cover an extended geographical area. In most sensor networks there are one or more base stations to which data collected by sensor nodes is relayed for uploading to external systems.

Many critical applications require that the data must be exchanged in a secure way. Establishing secure communication between sensor nodes becomes a challenging task, given the limited processing power, storage, bandwidth and energy resources. Public-key algorithms, such as RSA are undesirable, as they are computationally expensive. Instead,

symmetric encryption/decryption algorithms and hashing functions are between two to four orders of magnitude faster [3], and constitute the basic tools for securing sensor networks communication.

For these algorithms a shared key between the two communicating parties is required. Since we do not have prior knowledge of which nodes will be neighbors of each other after deployment, a solution would be for every pair of sensor nodes in the network to share a unique key. However this is not feasible due to memory constraints. A more scalable solution is the use of a key common to all sensor nodes in the network [4]. The problem with this approach is that if a single node is compromised then the security of the whole network is disrupted. Furthermore, refreshing the key becomes too expensive due to communication overhead.

There exist several schemes [5], [6], [7], [8] proposed in the literature that suggest random key pre-distribution: Before deployment each sensor node is loaded with a set of symmetric keys that have been randomly chosen from a key pool. Then nodes can communicate with each other by using one or more of the keys they share according to the model of random key pre-distribution used. These schemes offer network resilience against node capture but they are not “infinitely” scalable. As the size of the sensor network increases, the number of symmetric keys needed to be stored in sensor nodes must also be increased in order to provide sufficient security of links. Unfortunately, the more keys are stored in a network, the more links become compromised (even not neighboring ones) in case of node capture. Hence these schemes offer only “probabilistic” security as other links

may be exposed with certain probability. A desirable feature is therefore that compromise of nodes should result in a breach of security that is *constrained* within a small, localized part of the network.

Another problem that is not handled well by current key pre-distribution schemes is that of simple message broadcast. Many sensor network protocols require a node to broadcast a message to a subset or all of its neighbors. If a random key pre-distribution scheme is used, it is highly unlikely that a node can share the *same* key (or set of keys) with *all* its neighbors. Hence nodes have to make multiple transmissions of messages, encrypted each time with a different key. Thus such schemes suffer from this message broadcast problem and focus only on establishing node to node security. In general, we believe that transmissions must be kept as low as possible because of their high energy consumption rate.

Finally, a closely related problem to that of broadcasting encrypted messages is the ability to perform aggregation and data fusion processing [9]. This however can be done only if intermediate nodes have access to encrypted data to (possibly) discard extraneous messages reported back to the base station. The use of pair-wise shared keys effectively hinders data fusion processing.

Our contribution

In this work we present a security protocol that has the following properties:

- *Resilience to Node Capture.* Our scheme offers deterministic security as a single

compromised node disrupts only a *local* portion of the network while the rest remains fully secured. We designed our protocol without the assumption of tamper resistance. Once an adversary captures a node, key materials can be revealed. Building tamper-resistance into sensor nodes can substantially increase their cost, and besides, trusting such nodes can be problematic [10].

- *Resistance to Node Replication.* Even if a node is compromised and be used to populate the network with its clones, an adversary cannot gain control of the network as key material from one part of the network cannot be used to disrupt communications to some other part of it.
- *Energy efficiency.* We enable secure communication between a node and its neighbors by requiring only *one* transmission per message. Thus messages do not have to be encrypted multiple times with different keys to reach all neighbors. This saves energy as transmissions are among the most expensive operations a sensor can perform [11].
- *Intermediate Node Accessibility of Data.* An effective technique to extend sensor network lifetime is to limit the amount of data sent back to reporting nodes since this reduces communications energy consumption. This can be achieved by some processing of the raw data to discard extraneous reports. However, this cannot be done unless intermediate sensor nodes have access to the protected data to perform *data fusion* processing. Although existing random key pre-distribution schemes provide a secure path between a source and a destination, nearby nodes cannot have access to this information as it is highly unlikely they possess the right key to decrypt data.

- *Scalability.* The number of keys stored in sensor nodes is *independent* from the network size and the security level remains unaffected.
- *Easy Deployment.* Our protocol enables a newly deployed network to establish a secure infrastructure quickly using only local information and total absence of coordination.

The organization of the rest of the paper is as follows: In Section II, we describe our security protocol. We break this discussion into multiple subsections to ease the readability and understanding of the protocol. Furthermore, we prove each of the claims we made in this introduction about the features of our protocol. In Section III, we provide experimental evidence about the scalability of our protocol (in terms of the keys stored in each node) as well as the local resiliency (in terms of nodes in each cluster). In Section IV we show that our protocol is secure against certain types of attacks and in Section V we discuss related work on similar architectures. Finally we summarize our results in Section VI.

II. SECURITY PROTOCOL

In this section we first describe a localized algorithm for key management in wireless sensor networks (Sections II-A and II-B) and then provide a scheme that utilizes the established keys in order to provide secure communication between a source node and the base station (Section II-C). In general, our scheme can provide secure communication between *any* pair of nodes by building transitive keying relationships, but we demonstrate our protocol with respect to the “node to base station” data delivery model, since this is the most commonly used in sensor networks. The protocol is divided into the following

phases:

- 1) Initialization phase that is performed before sensor nodes are deployed.
- 2) Cluster key setup phase that splits the network into disjoint sets (clusters) and distributes a unique key to each cluster. That key is shared between all the cluster members, as well as the nodes that are one-hop away from the cluster.
- 3) Secure communication phase that provides confidentiality, data authentication, and freshness for messages relayed between nodes towards the base station.

In what follows, we describe in details the three phases of our protocol. In our discussion the following notation is used:

$M_1 M_2$	Concatenation of messages M_1 and M_2 .
$E_{K,IV}(M)$	Encryption of message M , with key K and the initialization vector IV , which is used in counter mode (CTR).
$MAC_K(M)$	MAC of message M using key K .

A. Initialization

Sensor nodes are assigned a unique ID that identifies them in the network, as well as three symmetric keys. Since wireless transmission of this information is not secure, it is assigned to the nodes during the manufacturing phase, before deployment. In particular the following keys are loaded into sensor nodes:

- K_i : Shared between each node i and the base station. This key will be used to secure information sent from node i to the base station. It is not used in establishing the

security infrastructure of the network but only to encrypt the sensed data D that must reach the base station in a secure manner. If we are interested in data fusion processing this key should not be used to encrypt D as otherwise intermediate nodes will not be able to evaluate and possibly discard the data.

- K_c^i : Shared between each node i and the base station. This key will be used only by those nodes that will become clusterheads and it will be the cluster key. These are the keys used to forward information to the base station in a hop-by-hop manner.
- K_m : A master key shared among all nodes, including the base station. This key will be used to secure information exchanged during the cluster key setup phase. Then it is erased from the memory of the sensor nodes.

The base station is then given all the ID numbers and keys used in the network before the deployment phase. Since the base station stores information used to secure the entire network, it is necessary to include it in our trusted computing base.

B. Cluster key setup

In this section we describe how sensor nodes use the pre-deployed key material in order to form a network where nodes can communicate with each other using a set of trusted keys.

The cluster key setup procedure is divided into two phases: organization into clusters and secure link establishment. During the first phase the sensor nodes are organized into clusters and agree on a common cluster key, while in the second phase, secure links are

established between clusters in order to form a connected graph.

An implicit assumption here is that the time required for the underlying communication graph to become connected (through the establishment of secure links) is smaller than the time needed by an adversary to compromise a sensor node during deployment. As security protocols for sensor networks should *not* be designed with the assumption of tamper resistance [10], we must assume that an adversary needs more time to compromise a node and discover the master key K_m . In the experimental section we give evidence that this is indeed the case.

1) *Organization into clusters:* In this phase, the creation of clusters happens in a probabilistic way that requires the nodes to make at most one broadcast. Each node i waits a random time (according to an exponential distribution) before broadcasting a HELLO message to its neighbors declaring its decision to become a cluster head. This message is encrypted using K_m and contains the ID_i of the node, its key K_c^i and an authentication tag:

$$E_{K_m}(ID_i|K_c^i|MAC_{K_m}((ID_i|K_c^i)))$$

Upon receiving a HELLO message, a node decrypts and authenticates the message. Then reacts in the following way:

- 1) If the node has not made any decision about its role yet, it joins the cluster of the node that sent the message and cancels its timer. No transmission is required for that node. The key that it is going to be using to secure traffic is $K_c = K_c^i$.

- 2) If the node has already decided its role, it rejects the message. This will happen if the node has already received a HELLO message from another node and became a cluster member of the corresponding cluster, or the node has sent a HELLO message being a cluster head itself.

Upon termination of the first phase, the network will have been divided into clusters. All nodes will be either cluster heads or cluster members, depending on whether they sent a HELLO message or received one. We are assuming here that collisions are resolved at a lower level otherwise acknowledgements must be incorporated in this simple protocol. There is, however, the possibility that two neighboring nodes send HELLO before they receive the same message from their neighbor, thus becoming clusterheads of themselves. Furthermore, there is a case for a node to send a HELLO message after all its neighboring nodes have decided their role, and thus become a head of a cluster with no members. Although these possibilities can be minimized by the right exponential distribution of the time delays that nodes send the HELLO messages, they do not affect the proper running of the protocol. In Fig. 1 we show the distribution of nodes to clusters for densities (average number of neighbors per sensor) equal to 8 and 20. As it can be seen in the figure, for smaller densities a larger percentage of nodes forms clusters of size one. However, the probability of this event decreases as the density becomes larger.

At the end of this phase each cluster will be given an identifier CID , which can be the cluster head's ID. All nodes in a cluster will be sharing the same key, K_c , which is the key K_c^i of the cluster head. From this point on, cluster heads turn to normal members, as

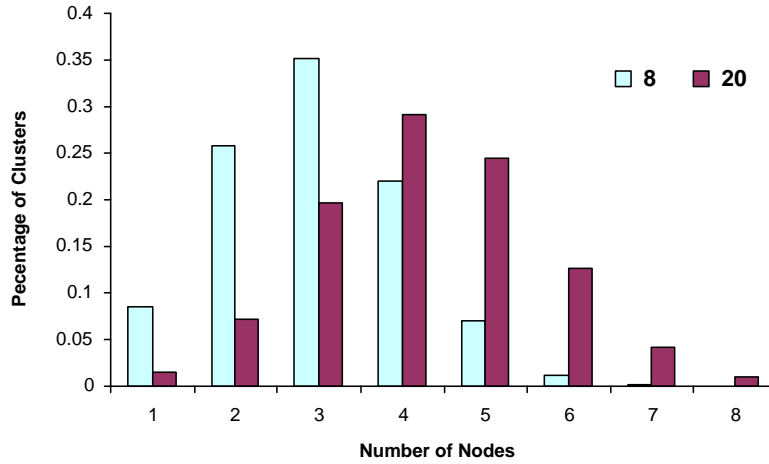


Fig. 1. Distribution of nodes to clusters.

there is no more need for a hierarchical structure. This is important since cluster based approaches usually create single points of failure as communications must usually pass through a clusterhead. Fig. 2 shows an example topology where three clusters have been created with *CIDs* 13, 9 and 19.

As it can be seen from Fig. 2, the maximum distance between two nodes in a cluster is two hops. Since all nodes in a cluster share a common key K_c , we need to keep the size of the clusters as small as possible in order to minimize the damage done by the compromise of a single node. In the experimental section, we give evidence that indeed clusters contain in average a small number of nodes that is independent of the size of the network.

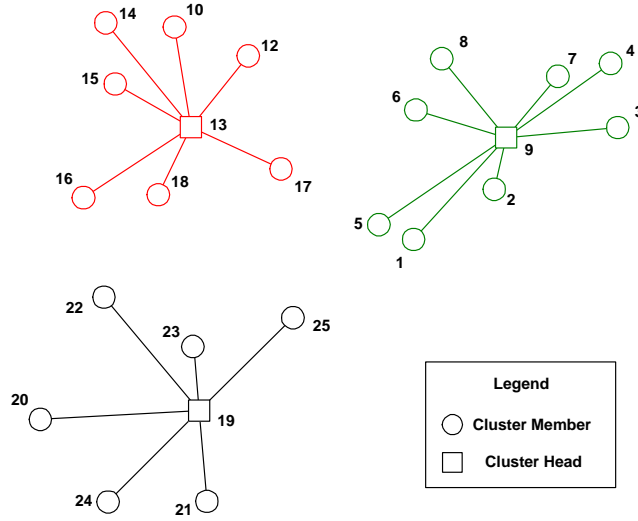


Fig. 2. An example topology after organization into clusters

2) *Secure link establishment*: In the second phase, all nodes get informed about the keys of their neighboring clusters. We need this phase in order to make the whole network connected since up to this point it is only divided into clusters whose nodes share a common key. We say that a node is *neighbor* of a cluster CID when that node has within its communication range at least one member of that cluster. This phase is executed with a simple local broadcast of the cluster key by all nodes. The message sent contains the tag and the CID , encrypted using K_m :

$$E_{K_m}(CID_i|K_c|MAC_{K_m}(\langle CID_i|K_c \rangle))$$

Nodes of the same cluster simply ignore the message, while any nodes from neighboring clusters will store the tuple $\langle CID, K_c \rangle$ and use it to decrypt traffic coming from that cluster, as explained in the next section. If the message has been sent from a member of the same

cluster, then that message should be ignored.

We must emphasize again that the total time of both steps is too short for an adversary to capture a node and retrieve the key K_m (see also Fig. 9 in Section III for a justification of this claim.) Nevertheless an adversary could have monitored the key setup phase and by capturing a node at later time it could retrieve all cluster keys. Therefore after the completion of the key setup phase, all nodes erase key K_m from their memory.

At this point, each node i of the sensor network will have its key K_i and a set \mathcal{S} of cluster keys that includes its own cluster key and the keys of its neighboring clusters. The total number of the keys that a node will have to store depends on the number of its neighboring clusters, thus not all cluster members store the same number of keys. (In the experimental section we give evidence that each node needs to store on average a handful of cluster keys). Most importantly however, the number of keys that each node gets is *independent* of the network size and therefore there is no upper limit on the number of sensor nodes that can be deployed in the network.

We illustrate the operations of the cluster key setup phase with the following example. Consider the sensor network depicted in Fig. 3. Three clusters with *CIDs* 13, 9 and 19 have been formed from the first step. The figure also shows the transmission radius of nodes 25, 17 and 5. As it can be seen, node 25 has two neighboring clusters, since node 17 from cluster 13 and nodes 5 and 1 from cluster 9 are within its communication range. Therefore node 17 will store 3 cluster keys. Likewise, nodes 17, 5 and 1 also have two neighboring clusters and will store 3 cluster keys each. On the other hand, node 6 is within

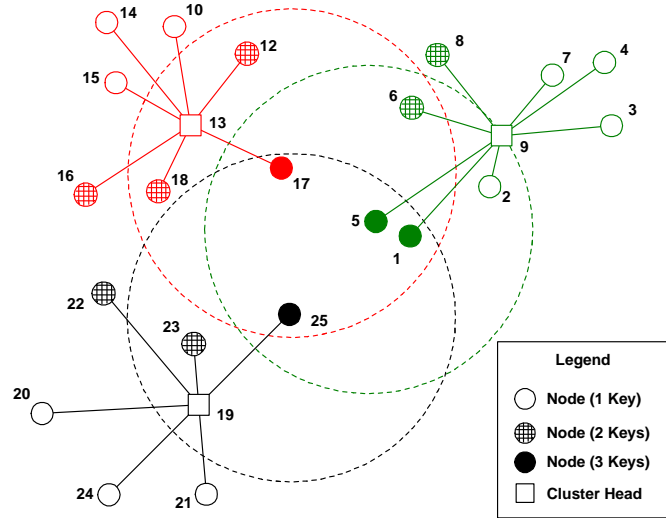


Fig. 3. An example topology during the key setup phase.

the range of node's 17 but outside the radius of any node from cluster 19, therefore it will only store 2 cluster keys.

C. Secure message forwarding

In this section we describe how information propagating towards a base station can be secured to guarantee confidentiality, data authentication, and freshness. *Confidentiality* guarantees that sensor readings and other sensitive information (e.g. routing information) are protected from disclosure to unauthorized parties. This can be achieved by encrypting data with a secret key. *Data authentication* allows the base station to verify that data was really send by the claimed sender and not from an adversary that injected it in the network. Even if messages cannot be inferred or forged, an adversary can capture them and replay them at a later time (message replay attack). *Data freshness* ensures that messages are

fresh, meaning that they obey a message ordering and have not been reused.

Here we make the assumption that sensor readings must first be encrypted (Step 1 in the description below) and then authenticated in a hop-by-hop manner (Step 2) as data is forwarded to the base station through intermediate nodes. If we are interested in data fusion processing then Step 1 should be omitted. It is only used when we want to make sure that sensor readings can only be seen by the base station.

It is worth mentioning here that while we describe our protocol for the case that a sensed event must be securely forwarded to a base station, *the same ideas can be used with any routing protocol that requires secure communication between one-hop neighbors*. The key translation character of our protocol is very important in establishing a transitive chain of trust.

1) *Step 1 (Optional)*: To achieve the security requirements for the data D that will be exchanged between the source node and the base station, we use a SNEP [11] like exchange, as shown in Fig. 4. A good security practice is to use different keys for different cryptographic operations; this prevents potential interactions between the operations that might introduce weaknesses in a security protocol. Therefore we use independent keys for the encryption and authentication operations, K_{encr} and K_{MAC} respectively, which are derived from the unique key K_i that node shares with the base station. For example we may take $K_{encr} = F_{K_i}(0)$ and $K_{MAC} = F_{K_i}(1)$, where F is some secure pseudo-random function.

As it can be seen from Fig. 4, we use the encrypt-then-authenticate method to construct

$$\begin{array}{l}
y_1 \leftarrow E_{K_{encr}, C}(D) \\
t_1 \leftarrow MAC_{K_{MAC}}(y_1) \\
c_1 \leftarrow y_1 | t_1
\end{array}$$

Fig. 4. Step 1 for secure communication between source node and base station. This step is applied by the source node alone.

a “secure channel” between source node and base station. It is shown in [12] that this is the most secure method for symmetric encryption and authentication. Encryption is performed through the use of a counter C that is shared between the source node and the base station. We do this in order to achieve semantic security; an adversary will not be able to obtain partial information about a plaintext, even if it is the same plaintext that is encrypted multiple times. This can also be achieved through randomization but then the random value used in the encryption of the message must also be transmitted. The counter approach results in less transmission overhead as the counter is maintained in both ends. If counter synchronization is a problem (usually the receiver can try a small window of counter values to recover the message) then the counter or the random value used can be sent alongside the message. We leave the choice to the particular deployment scenario as one alternative may be better than the other.

2) *Step 2 (Required)*: Since the encrypted data must be forwarded by intermediate nodes in order to reach the base station, we need to further secure the message so that an adversary cannot disrupt the routing procedure. Thus, no matter what routing protocol

is followed, *intermediate* nodes need to verify that the message is not tampered with, replayed or revealed to unauthorized parties, before forwarding it.

To secure the communication between one-hop neighbors, we use the protocol described in Fig. 5. Each node (including the source node) uses its cluster key to produce the encryption key K'_{encr} and the MAC key K'_{MAC} . These keys are used to secure the message produced by Step 1, before it is further forwarded. (As we emphasized previously, if we are only interested in hop-by-hop encryption and authentication, Step 1 should be omitted in which case c_1 , in message 2 below, is simply the data D .) Since the nodes that will receive that message don't know the sender and therefore the key that the message was encrypted with, the cluster ID is included in c_2 . This way intermediate sensors will use the right key in their set \mathcal{S} to authenticate the message.

$$\begin{array}{l}
 \tau \leftarrow \text{time()} \\
 y_2 \leftarrow E_{K'_{encr}}(c_1, \tau, CID) \\
 t_2 \leftarrow MAC_{K'_{MAC}}(y_2) \\
 c_2 \leftarrow CID|y_2|t_2
 \end{array}$$

Fig. 5. Step 2 for secure communication between source node and base station. This step is applied by all intermediate nodes, besides the source node.

If authentication is not successful, the message should be dropped since it is not a legitimate one. Otherwise, each node will apply Step 2 with its own cluster key to further forward the message. The fact that this key is shared with all of its neighbors, allows

the node to make only *one* transmission per message. Notice that this is the point where our protocol differs from random key pre-distribution schemes. To broadcast a message in such a scheme the transmitter must encrypt the message multiple times, each time with a key shared with a specific neighbor. And this, of course, is extremely energy consuming.

To continue the example shown on Fig. 3, assume that node 14 must send a message m towards the base station that lies in the direction of node 4. It first encrypts and tags the message to produce a ciphertext c_1 according to the protocol shown on Fig. 4 and then wraps this to produce an encrypted block c_2 according to the specifications shown on Fig. 5. When ready, it broadcasts c_2 to its neighbors. Eventually an encapsulation of c_1 will reach node 12, maybe through node 10. This node will decrypt and authenticate the message since it shares the same cluster key as node 14 and once all the checks are passed, it will re-encrypt c_1 and forward it to its neighbors. One of them is node 8 which is a member of cluster with $CID = 9$. This node will look at its set of cluster keys \mathcal{S} and use the one which it shares with node 12 (the one corresponding to $CID=13$). Upon success it will re-encrypt the message with its cluster key and forward it towards its neighbors. Thus nodes that lie at the edge of clusters will be able to “translate” messages that come from neighboring clusters and be able to *authenticate* them in a hop-by-hop manner.

In summary, every node decrypts and authenticates the packets it receives by using the keys it derived from each cluster key. If the node belongs to a different cluster than the transmitter of the message, it will find the right cluster key from those stored in its memory and use it to re-encrypt the message and pass it along to its neighbors. Thus messages get

authenticated as they traverse the network. However, while this approach defends against adversaries who do not possess the required cluster keys, it falls prey to insider attacks since an adversary can easily inject spurious messages after it has compromised a node and discovered its cluster key. Unfortunately this kind of attack is not only hard to prevent but is also hard to detect.

To increase security and avoid sending too much traffic under the same keys, cluster keys may be refreshed periodically. To support such functionality, sensor nodes can repeat the key setup phase with a predefined period in order to form new clusters and new cluster keys. Since K_m is no longer available to the nodes, the current cluster key may be used by the nodes instead. The fact that each node can communicate with all of its neighbors using the current cluster key makes it possible to broadcast a HELLO message in a secure way. The message will contain the new cluster key, created by a secure key generation algorithm embedded in each node. Since the key setup phase requires very low communication overhead (as it will be showed in the next section) and takes only a short time to complete, the refreshing period can be as short as needed to keep the network safe. Alternatively, if we don't like the fact that certain nodes are assigned the task of creating new keys (as they may be the compromised ones), we can renew the cluster keys by periodically hashing these keys at fixed time intervals.

D. Addition of new nodes

The new sensors are arbitrary deployed. As they cannot be preassigned to a specific cluster, they must i) associate themselves to an existing cluster, ii) become informed about neighboring clusters and iii) retrieve and store the corresponding cluster keys. Each new node comes equipped with a master key K_{MC} that can be used to generate the relevant cluster keys as it will be explained below.

Every new node transmits a hello message to its neighbors indicating its will to become a member of some existing cluster. The message contains the ID of the new node. Nodes receiving this message will respond with the cluster id they belong to, authenticated using their cluster key K_c . This will prevent an adversary for realizing the following attack. The adversary will send fake messages containing various cluster ids. When the new node makes the association between the cluster id and the cluster key and store it in its memory, the adversary can later compromise the node thus having acquired the cluster key of any cluster in the network. To prevent this type of impersonation attack the response sent by existing nodes is simply

$$CID, MAC_{K_C}(CID).$$

A new node receiving such a collection of cluster id's will consider itself a member of the first such cluster while the rest will be the neighboring ones. We need now a way to associate each cluster id CID with the corresponding cluster key. We assume here that the cluster keys K_c^i of the original nodes were formed using a master key K_{MC} through the application of some pseudorandom function F . The use of a secure one way function

F will prevent an adversary who compromised a node and found its cluster key to recover the master key K_{MC} and hence the cluster keys of other nodes. Using F , the cluster key of the i -th node is simply given by

$$K_c^i = F(K_{MC}, i).$$

Each new node can use K_{MC} to generate the various cluster keys and store them in its memory. Then it can participate in encrypting and forwarding messages just like the original nodes. When this phase is over, the master key K_{MC} is deleted from the memory of the nodes.

III. EXPERIMENTAL ANALYSIS

We simulated a sensor network to determine some parameter values of our scheme. We deployed several thousands of nodes (2500 to 3600) in a random topology and run the key setup phase. Of particular interest is the scalability, the communication overhead and memory requirements of our approach.

The storage requirements of our approach are determined by the number of cluster keys stored in each node. We performed experiments with various network sizes and we found that the curves matched exactly (modulo some small statistical deviation). So, we experimented with respect to network *density*. Fig. 6 shows the average number of cluster keys that each node stores as a function of the average number of neighbors per node (density of network). The number of cluster keys also indicates the number of neighboring clusters that each node has. As is obvious from the figure, the number of stored keys is

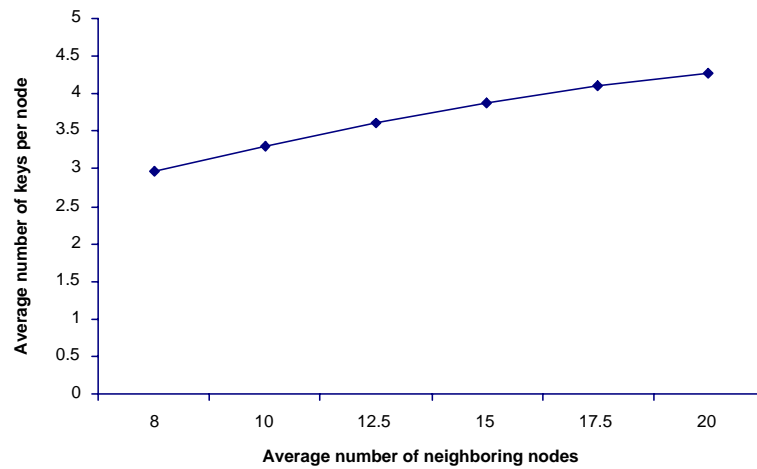


Fig. 6. Average number of cluster keys held by sensor nodes as a function of network density

very small and increases with low rate as the number of neighbors increases, requiring negligible memory resources from the sensor node. The point to be made is that the number of required keys remains *independent* of the actual network size. Thus our protocol behaves the same way in a network with 2000 or 20000 nodes.

In Fig. 7 we further show the average number of nodes per cluster for various network densities. Nodes of the same cluster share a common cluster key, and thus an adversary, upon compromising such a node, can also control the communication links of the rest of cluster nodes. Thus, having small clusters, as is indicated in the figure, minimizes the damage inflicted by the compromised node and prevents its spreading to the rest of the network.

The communication traffic required by the key setup phase is partly due to the number

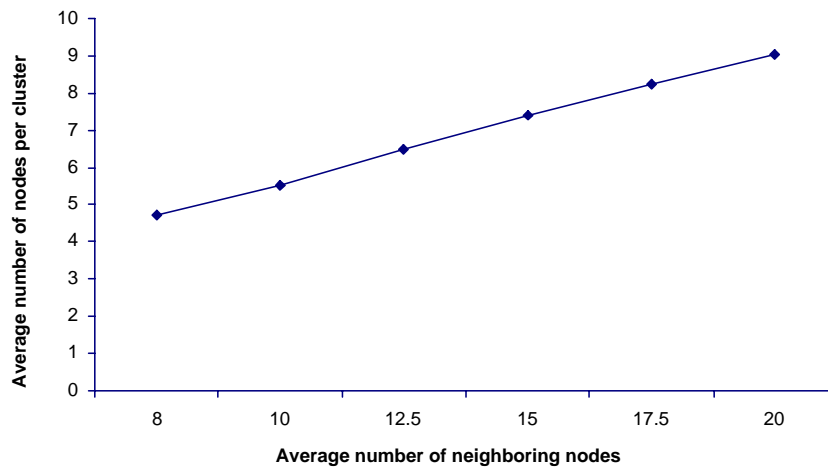


Fig. 7. Average number of nodes in clusters as a function of network density

of messages sent by the cluster heads to their cluster members during phase one, and partly due to the messages sent by all nodes of the network during the link establishment phase. The former quantity depends on the number of clusterheads and is shown in Fig. 8. The second quantity, is always constant and equal to n , the number of nodes in the network. Bearing in mind that the key setup phase is executed only once, the total communication overhead due to that phase is kept very low. Further evidence to this fact is given in Fig. 9, where the average number of messages required *per node* to set up the keys is shown. Thus the overall time needed to establish the keys is a little more than transmission of one message plus the time to decrypt the material sent during this phase.

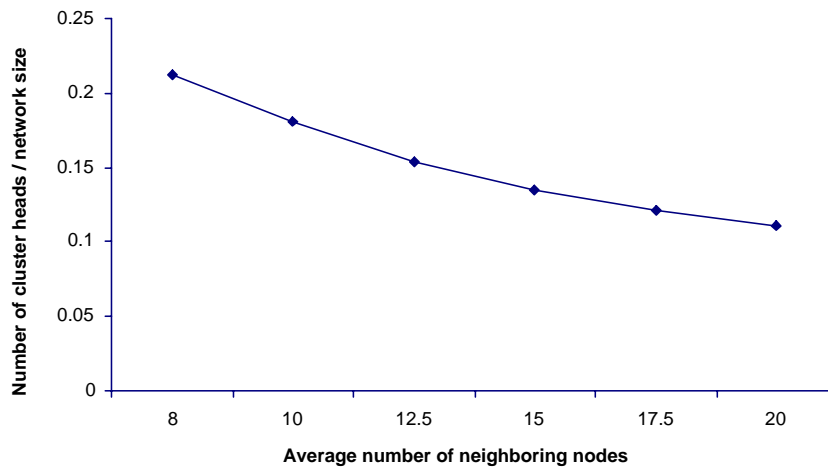


Fig. 8. Percentage of cluster heads with respect to total sensor nodes in the network

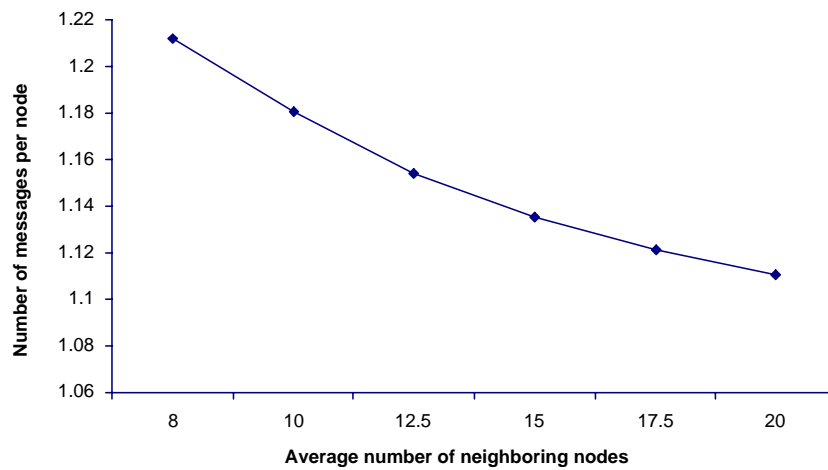


Fig. 9. Number of messages exchanged *per node* for organization into clusters and link establishment in a network of 2000 nodes and various densities.

IV. SECURITY ANALYSIS

We now discuss one by one some of the general attacks [15] that can be applied to routing protocols in order to take control of a small portion of the network or the entire part of it.

- *Spoofed, altered, or replayed routing information.* As sensor nodes do not exchange routing information, this kind of attack is not an issue.
- *Selective forwarding.* In this kind of attack an adversary selectively forwards certain packets through some compromised node while drops the rest. Although such an attack is always possible when a node is compromised, its consequences are insignificant since nearby nodes can have access to the same information through their cluster keys.
- *Sinkhole and wormhole attacks.* Since all nodes are considered equal and there is not a distinction between more powerful and weak nodes, an adversary cannot launch attacks of this kind. Furthermore, in our protocol such an attack can only take place during the key establishment phase. But the authentication that takes place in this phase and its small duration, as we described in the previous section, makes this kind of attack impossible.
- *Sybil attacks.* Since every node shares a unique symmetric key with the trusted base station, a single node cannot present multiple identities. An adversary may create clones of a compromised node and populate them into the same cluster or the node's neighboring clusters but this doesn't offer any advantages to the adversary with respect to the availability of the information to the base station.

- *Hello flood attacks.* In our protocol, nodes broadcast a HELLO message during the cluster key setup phase in order to announce their decision to become clusterheads and distribute the cluster key. Since, however, messages are authenticated this attack is not possible. (A necessary assumption for all key establishment protocols is of course that the duration of this phase is small so that an adversary cannot compromise a node and obtain the key K_m . In the previous section we presented evidence that this is indeed the case).

However, this kind of attack is possible during key-refresh. If we assume that a laptop-class attacker has compromised a node and retrieved its cluster keys then she could broadcast such a HELLO message during a key refresh phase and could attract nodes belonging to neighboring clusters as well and form a new larger cluster with himself as a clusterhead. One way to defend against this is to constraint the key-refresh phase within clusters, i.e. not allow new clusters to be created. Therefore, cluster keys will be refreshed within the same clusters, and an adversary cannot take control of more nodes than she already has, that is the nodes within the same cluster. A better way, however, which makes this kind of attack useless, is to refresh the keys by hashing instead of letting nodes generate new ones.

- *Acknowledgment spoofing.* Since we don't rely on link layer acknowledgements this kind of attack is not possible in our protocol.

The power of our proposal lies exactly in its simplicity. No location information, path reinforcements or routing updates are used. By strengthening security with encryption and authentication, it becomes really hard for an adversary to disrupt the routing procedure. Not many sensor network routing protocols currently proposed in the bibliography resist so well (or at all) to such security attacks as [15] demonstrated and their design remained up to now an open problem.

V. RELATED WORK

We have already mentioned Basagni *et al.*'s pebblenets architecture [4] that uses a global key shared by all nodes. Having network wide keys for encrypting information is very good in terms of storage requirements and energy efficiency as no communication is required among nodes to establish additional keys. It suffers, however, from the obvious security disadvantage that compromise of even a single node will reveal the universal key. Since one cannot have keys that are shared pair-wise between all nodes in the network, a key pre-distribution scheme must be used.

Random key pre-distribution schemes [5], [6], [7], [8] offer a trade-off between the level of security achieved using shared keys among nodes and the memory storage required to keep at each node a set of symmetric keys that have been randomly chosen from a key pool. Then, according to the model used, nodes use one or more of these keys to communicate securely with each other. A necessary requirement of a good key-distribution scheme is that compromise of nodes should *not* result in a breach of security that is

spread in the whole network. Our approach guarantees this since it's not a probabilistic scheme; compromised nodes cannot expose keys in another part of the network. In our work, however, we emphasize on another desirable characteristic: when a node wants to broadcast a message to a subset or all of its neighbors it should *not* have to make multiple transmissions of the same message, encrypted each time with a different key. We now review some other proposals that use security architectures similar to ours.

In LEAP [13] *every* node creates a cluster key that distributes to its immediate neighbors using pair-wise keys that shares with each one of them. In this case, however, clusters highly overlap so every node has to apply a different cryptographic key before forwarding the message. While this scheme offers deterministic security and broadcast of encrypted messages, it has a more expensive bootstrapping phase and increased storage requirements as each node must set up and store a number of pair-wise *and* cluster keys that is proportional to its actual neighbors.

Slijepcevic *et al.* [14] propose dividing the network into hexagonal cells, each having a unique key shared between its members. Nodes belonging to the bordering region between neighboring cells store the keys of those cells, so that traffic can pass through. The model works under the assumption that sensor nodes are able to discover their exact location, so that they can organize into cells and produce a location-based key. Moreover, the authors assume that sensor nodes are tamper resistant, otherwise the set of master keys and the pseudo-random generator, pre-loaded to all sensor nodes, can be revealed by compromising a single node and the whole network security collapses. Those assumptions are usually

too demanding for sensor networks.

VI. CONCLUSIONS

We have presented a key establishment protocol that is suitable for sensor network deployment. The protocol provides security against a large number of attacks and guarantees that data securely reaches the base station in an energy efficient manner. Our protocol is based on hop-by-hop encryption, allowing nodes to share keys only with neighboring nodes. The protocol has a number of important characteristics among which are:

- Resiliency against node capture and replication. This is due to the fact that keys are localized. After a deployment phase, nodes share a handful of keys to securely communicate with their neighbors. Thus compromised keys in one part of the network do not allow an adversary to obtain access in some other part of it.
- Efficient broadcasting of *encrypted* messages. When a node wants to broadcast a message to its neighbors it does not have to make multiple transmissions encrypted each time with a different key. We achieve this by encrypting messages with a cluster key which is shared between neighboring nodes. This makes our scheme very energy efficient.
- Intermediate node accessibility of data. When multiple nodes receive the same message, some of them may decide not to forward it. However, this is not possible unless nodes can have access to encrypted data. Using our approach, nodes can “peak” at encrypted information using their cluster key and decide upon forwarding or discarding

redundant messages thus enabling data aggregation processing.

- Scalability. Our protocol scales very well as the key establishment phase requires only local information and no global coordination. Furthermore the keys that need to be stored at each node do not depend on the size of the sensor network but only on its *density* (the average number of neighbors per node). Thus our protocol behaves similarly in networks of 2000 or 20000 nodes as long as the density is the same.

ACKNOWLEDGEMENTS

The authors wish to thank the anonymous reviewers for the useful comments that helped improve the readability of this paper.

REFERENCES

- [1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, March 2002.
- [2] C.-Y. Chong and S. Kumar, "Sensor networks: evolution, opportunities and challenges," *Proceedings of the IEEE*, vol. 91, pp. 1247–1256, August 2003.
- [3] D. Carman, P. Kruus, and B.J.Matt, "Constraints and approaches for distributed sensor network security," Tech. Rep. 00-010, NAI Labs, June 2000.
- [4] S. Basagni, K. Herrin, D. Bruschi, and E. Rosti, "Secure pebblenet," in *Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking & Computing, MobiHoc 2001*, pp. 156–163, October 2001.
- [5] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 41–47, 2002.
- [6] H.Chan, A.Perrig, and D.Song, "Random key predistribution schemes for sensor networks," in *IEEE Symposium on Security and Privacy*, pp. 197–213, May 2003.

- [7] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in *Proceedings of the 10th ACM conference on Computer and communication security*, pp. 52–61, October 2003.
- [8] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks," in *Proceedings of the 10th ACM conference on Computer and communication security*, pp. 42–51, October 2003.
- [9] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *ACM/IEEE Transactions on Networking*, vol. 11, February 2002.
- [10] R. Anderson and M. Kuhn, "Tamper resistance – a cautionary note," in *Proceedings of the Second Usenix Workshop on Electronic Commerce*, pp. 1–11, November 1996.
- [11] A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar, "SPINS: Security protocols for sensor networks," in *Seventh Annual International Conference on Mobile Computing and Networks (MobiCOM)*, pp. 189–199, 2001.
- [12] H. Krawczyk, "The order of encryption and authentication for protecting communications (or: How secure is SSL?)," *Lecture Notes in Computer Science*, vol. 2139, pp. 310–331, 2001.
- [13] S. Zhu, S. Setia, and S. Jajodia, "LEAP: efficient security mechanisms for large-scale distributed sensor networks," in *Proceedings of the 10th ACM conference on Computer and communication security*, pp. 62–72, October 2003.
- [14] S. Slijepcevic, M. Potkonjak, V. Tsiatsis, S. Zimbeck, and M. Srivastava, "On communication security in wireless ad-hoc sensor networks," in *11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 139–144, June 2002.
- [15] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasures," *Elsevier's Ad Hoc Network Journal, Special Issue on Sensor Network Applications and Protocols*, vol. 1, pp. 293–315, September 2003.