# A Deterministic Key Management Scheme for Securing Cluster-Based Sensors Networks

Mandicou Ba, Ibrahima Niang, Bamba Gueye
Département de Mathématiques et Informatique
Université Cheikh Anta Diop
Dakar, Senegal
mandicou.ba@ucad.edu.sn, iniang@ucad.sn, bamba.gueye@ucad.edu.sn

Thomas Noel
LSIIT Laboratory
Université de Strasbourg
Strasbourg, France
noel@unistra.fr

***Abstract*—The main goal of Cluster-based sensor networks is to decrease system delay and reduce energy consumption. LEACH is a cluster-based protocol for microsensor networks which achieves energy-efficient, scalable routing and fair media access for sensor nodes. However, the election of a malicious or compromised sensor node as the cluster head is one the most significant breaches in cluster-based wireless sensor networks. We propose a deterministic key management scheme, called *DKS-LEACH*, to secure LEACH protocol against malicious attacks. Our contributions are twofold. Firstly, we design and performed a theoretical evaluation of our security model which secures the setup and study phases of LEACH protocol. Secondly, using the *TOSSIM* simulator, we performed an evaluation of the power consumption of DKS-LEACH. The results indicate clear advantages of our approach in preventing the election of untrustworthy cluster head as well different kind of attacks from malicious sensor nodes.**

***Keywords*-Wireless sensor networks; Key management; Security; Energy consumption**

## I. Introduction

A Wireless Sensor Networks (WSN) is a network consisting of a large number of micro, low-cost, low power consumption and spatially distributed autonomous electronic devices using sensors to cooperatively monitor physical or environmental conditions [1].

The main drawbacks of sensor nodes is their energy limitation. To reduce the emission range of sensor nodes, and thus improve the node's lifetime, cluster-based WSN is proposed by the authors of [2], [3], [4]. In such case, each cluster is managed by a cluster head (CH). The role of the CH is to form the clusters and to gather the data sent by other sensor nodes. Furthermore, it sends the aggregated data to the sink which manages the WSN. In so doing, cluster-based WSN achieves scalability and energy efficiency.

Adding security in cluster-based WSN is challenging. In fact, sensor nodes organize periodically themselves into new cluster. In such case, we have a periodic rearranging of the network's cluster key distribution. Furthermore, in cluster-based WSN, it is mandatory to secure communication between sensor nodes and CH, and between CH and the sink. The role of the sink is to gather all information collected in the network through CHs. Quite often the sink is also called base station. In cluster-based WSN, the CHs are vulnerable to attacks [5]. In fact, they have the responsibility to route all messages generated in the network.

In order to secure data in WSN, encryption keys must be established among sensor nodes [6]. Key distribution refers to the distribution of multiple keys among the sensor nodes, which is typically a non-trivial security scheme. Key management plays a central role in data encryption and authentication. It is worth noticing that due to resource constrains of sensor nodes, security based on public key like "Diffie-Hellmanand", are not suitable for sensor networks [7]. Therefore, the most practical approach for bootstrapping secret keys in sensor networks is to use pre-deployed keying. In such case, keys are loaded into sensor nodes before their deployment. Few works propose to manage the keys, that will be pre-deployed in sensor nodes, either in a deterministic scheme [8], [9], or in a probabilistic scheme [10].

Note that techniques based on probabilistic scheme [11], [12] need to store an important subset of keys on each sensor node. Furthermore, sensor nodes should exchange a lot of messages to seek if they share the same key in order to have a secure communication. The pre-distribution of secret keys for all pairs of nodes is not suitable due to the large amount of memory used when the network size is large. In contrast, deterministic key management does not require to pre-distribute a large amount of subset keys. Nevertheless, it needs a time computation which is not negligible compared to probabilistic scheme. It should be noted that in WSN, the computation does not consume a lot of energy in contrast to exchanged messages. So far, all solutions proposed to secure LEACH [11], [12] are probabilistic-based.

In this paper, we propose a deterministic key management scheme, called *DKS-LEACH*, in order to secure the LEACH protocol. In fact, LEACH protocol is vulnerable to attacks such as jamming, spoofing, replay, etc [5]. Since LEACH is a cluster-based protocol, which relies fundamentally on the CH for data aggregation and routing, attacks involving CH are the most damaging. If a malicious node seeks to become a CH, it can use a kind of attacks such as sinkhole and selective forwarding.

In such case, it can disrupt the network. Note that, the malicious node can choose to not attack the CH, and thus try to inject erroneous information into the network. In so doing,

these wrong information will be relayed through the network. To overcome theses limitations, we propose in DKS-LEACH to set up dynamically cryptographic keys in an autonomous manner.

The rest of this paper is organized as follows. Section II reviews the related work on security with respect to cluster-based WSN. Section III describes DKS-LEACH and Section IV illustrates the performance of our deterministic key management. Finally, we conclude and present some research perspectives in Section V.

## II. Related Work

Some key management schemes [10], [9] have been specifically designed for WSNs. Few are dedicated for hierarchical sensors networks, whereas the remaining are appropriate for flat sensors networks. These schemes typically assume that a node interacts with a quite static set of neighbors and that most of its neighborhood is discovered right after the deployment. However, clusters in LEACH [2] are formed dynamically (at random) and periodically, which changes interactions among the nodes and requires that any node needs to be ready to join any CH at any time. For more details about LEACH protocol, the reader can refer to [2].

In the light of security challenge, the probabilistic scheme was first proposed by Eschenauer and Gligor [10]. The probabilistic key pre-distribution scheme, that they have proposed, enables scalability and network resiliency against compromising. In fact, in [10] any pair of nodes needs to find a single common key from their key rings to establish a secure link during the *setup phase*. The main drawback of this proposition is the use of one single key between two sensor nodes which is very compromising. Indeed, based on one single key, an attacker could easily discover the shared key.

The probabilistic key management schemes like [10], [12], suffer a lot from security threats and communication overhead. In fact, the nodes must set up secure link with some neighbor nodes. This is realized by flooding the *id* of their keys. With this approach, we notice a huge amount of exchanged messages. Also, the lack of identity authentication may cause several attacks including id replication. Finally, any pair of nodes should establish a bidirectional link before the keys exchange for future communication. Therefore, a key management based on a probabilistic approach like [10] is not appropriate.

Nevertheless, *SecLEACH* [12] proposes to secure LEACH by using a probabilistic scheme. In SecLEACH, each node has K pre-distributed keys obtained randomly from a set of keys $\mathcal{P}$. The main advantage provided by SecLEACH is the possibility to authenticate and to secure the communication between CH and cluster's members without the participation of the BS. Note that, the authors of SecLEACH have already proposed in [11] a protocol, called *S-LEACH*, in order to secure LEACH. SecLEACH is an improvement of SLEACH. Since there are only two keys per node, S-LEACH does not provide a complete and efficient solution to node-to-cluster-head authentication as said in [12].

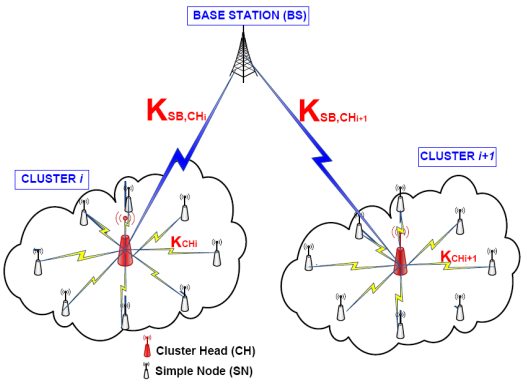



Fig. 1. LEACH protocol with secure links

Xu et *al.* by using as metric the "*Successful Attack Probability*" have shown that the probabilistic key management approaches have only limited performance advantages over deterministic approaches [13]. LEAP [9] also proposes a deterministic key distribution in hierarchical WSNs. The authors of LEAP establish four types of keys that must be stored in each sensor. Therefore, if LEAP is used to secure communication in LEACH, a new key distribution could be required in each round.

The approaches based on probabilistic key distribution generate a lot of messages, require much more memory space, and especially present compromising risks. In contrast, the deterministic key distribution requires more computation time for nodes. Note that computation consumes less energy compared to the exchange of messages between sensor nodes.

## III. Key Management Scheme for Securing LEACH Protocol

### A. DKS-LEACH architecture

DKS-LEACH is based on two types of keys: *(i)* a pairwise key ($K_{BS,CH_i}$) shared between the *Base Station* (BS) and the *Cluster Heads* (CH); *(ii)* a *Cluster's Key* ($K_{CH_i}$) shared between sensor nodes and the CH that form the same cluster. For instance, the distribution of keys between nodes that form our network is illustrated in Figure 1. In fact, these keys are set dynamically during the initialization phase and the cluster formation. For security reasons, keys are renewed in each round [2]. A round in LEACH is the process where cluster members are renewed and measurement tasks are done with respect to a predetermined duration. During the bootstrap phase, all sensor nodes share a same secret key. Note that, each sensor node knows this secret before its deployment. In contrast to [9], this secret key is changed after cluster's formation during the first round.

The obtained results show that, with respect to energy consumption, the gap between DKS-LEACH and LEACH is very low and DKS-LEACH does not require an important storage space. In fact, before the deployment, sensor nodes are preloaded with one global key. After the deployment, if one node becomes CH, it needs to store one pairwise key

shared between the base station and itself; and the cluster's key between SN and itself. In other words, the CH needs only three keys. For sensor nodes that are considered as SN, it needs just to store the cluster's key and the global key. Furthermore, the messages used by DKS-LEACH are piggy backed to LEACH messages, and thus, the number of exchanged messages are reduced.

### B. Assumption

In our study, we assume that the base station has unlimited resources and it is considered trustworthy. When the network is initialized, the BS generates a matrix $M_{kp}$ at random. The value of each element in the matrix $M_{kp}$ is 0 or 1.

$$M_{kp} = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1p} \\ m_{21} & \ddots & & \vdots \\ \vdots & & \ddots & m_{k-1p} \\ m_{k1} & \cdots & m_{kp-1} & m_{kp} \end{pmatrix}$$

As each column of $M_{kp}$ is dedicated to a sensor node, its number of columns should be upper than the total number of sensor nodes ($N$).

### C. Keys establishment phases in DKS-LEACH

**Algorithm 1:** Pairwise key Establishment

**begin**
  **if** *(isClusterHead)* **then**
    $CH_i \longrightarrow BS : E(K_{G_0}, id_{CH_i} \parallel id_{BS} \parallel nonce)$, $id_{CH_i} \parallel id_{BS}, MAC(K_{G_0}, id_{CH_i} \parallel id_{BS} \parallel nonce)$
    **(..1)**
  **else if** *(!isClusterHead)* **then**
    wait for cluster-head announcement
  **end**
  /* The base station computes the pairwise key as follows */
  $BS : columnResult = column[id_{CH_i}] \bigoplus column[id_{BS}]$
  $BS : K_{SB,CH_i} = F_{K_{G_0}(columnResult)}$
  /* The base station sends the pairwise to CH */
  $BS \longrightarrow CH_i :$
  $E(K_{G_0}, id_{CH_i} \parallel id_{BS} \parallel nonce \parallel K_{BS,CH_i})$, $id_{CH_i} \parallel id_{BS}, MAC(K_{BS,CH_i}, id_{CH_i} \parallel id_{BS} \parallel nonce)$
  **(..2)**
**end**

The DKS-LEACH approach has four phases: *(i)* initialization (key pre-distribution); *(ii)* announcement of a new round; *(iii)* pairwise key establishment; *(iiii)* cluster's key establishment. All these phases occur during the *Set-up* phase. The pairwise keys and the cluster's keys are used to encrypt the communications during the *Steady-state* phase. Both keys are encrypted with the encryption function $E(K, mess)$. Also, in order to ensure authentication and integrity of both keys, we use a *Message Authentication Code* $MAC(K, mess)$.

During the initialization phase, the BS computes a set of keys $\phi = K_{G_0}, K_{G_1}, \ldots, K_{G_n}$ by using a keyed one-way hash function $F_K(val)$. The $K_{G_i}$ key will be used during the Set-up phase of each round. The $K_{G_i}$ key can be seen as the *global key* of the current round *i*. The BS selects a key out of the set $\phi$, and then pre-loads it in all sensor nodes. This key will be used to encrypt the broadcast message sent for a new round. We argue that only legitimate nodes may decrypt this message. This key is deleted from the first round. It should be noted that each exchanged message has a timestamp called "*nonce*" that guarantee the freshness of information.

For the announcement phase of a new round, the BS encrypts a threshold value, called $proba$, with the key $K_{G_0}$, generates a MAC and sends these information in the network. It is worth noticing that in LEACH, this threshold value represents a given probability used by sensor nodes to become CH. In fact, if a sensor node generates a value lower than $proba$ it acts as CH [2].

**Algorithm 2:** Cluster key Establishment (*Step 1*)

**begin**
  **if** *(isClusterHead)* **then**
    $CH_i \longrightarrow * : E(K_{G_0}, id_{CH_i} \parallel nonce \parallel ADV), ADV, MAC(KG_0, id_{CH_i} \parallel ADV)$;
    **(..3)**
  **else if** *(!isClusterHead)* **then**
    $SN \longrightarrow CH_i :$
    $E(K_{G_0}, id_{SN} \parallel id_{CH_i} \parallel JoinREQ \parallel nonce), id_{SN} \parallel JoinREQ, MAC(K_{G_0}, id_{CH_i} \parallel JoinREQ)$;
    **(..4)**
  **end**
  /* The CH stores the id of each simple node */
  $CH_i : NodeMembers[i] = id_{SN}$ ;
  /* The CH sends the list of cluster's members to the BS */
  $CH_i \longrightarrow BS : E(K_{CH_i,BS}, id_{CH_i} \parallel NodesMembers \parallel nonce), id_{CH_i}, id_{BS}, MAC(K_{CH_i,BS}, id_{CH_i} \parallel id_{BS} \parallel nonce)$;
  **(..5)**
**end**

If a simple node (SN) becomes cluster head, then a pairwise key is established between itself and the BS. This is the unique key which is shared between each CH and the BS. The Algorithme 1 describes the processus of the pairwise key creation.

Finally, after the pairwise key computation, the CH initiates the cluster's key computation in order to secure the communication between CH and simple nodes. This phase is divided in two steps:

- During the first step, as illustrated by Algorithme 2, the cluster head retrieves the *id* of each simple node that belong the cluster. Afterwards, the CH sends the cluster's members to the BS.

- In the second step, the BS has already received the list of simple nodes that belongs to a given cluster. The BS uses the matrix $M_{kp}$ described in Section III-B in order to compute the argument that will be used by the keyed one-way hash function (Algorithme 3). Following that, the one-way hash function establishes the cluster's key, and thus, the BS can sends this key to the appropriate CH. Note that, the key needed to set-up the newt round is sent at the same time.

**Algorithm 3:** Cluster key Establishment (*Step 2*)

**begin**

/* The BS computes the cluster's key */

**for** ($i \in$ *NodesMembers*) **do**

$columnResult$ = $columnResult \bigoplus column[i]$;

**end**

$K_{CH_i}$ = $F_{K_{SB,CH_i}}(columnResult)$ ;

/* The BS chooses the global key of the next round */

$K_{G_{i+1}}$ = $\Phi$ [i+1] ;

/* The BS sends the cluster's key and the global key of the next round to the CH */

$BS \longrightarrow CH_i : E(K_{BS,CH_i}, K_{CH_i} \parallel K_{G_{i+1}} \parallel id_{BS} \parallel idCH_i \parallel nonce), id_{BS} \parallel id_{CH_i}, MAC(K_{SB,CH_i}, id_{BS} \parallel idCH_i \parallel nonce)$ ;

**(..6)**

/* The CH transmits the cluster's key and the key global of the next round to SN */

$CH_i \longrightarrow * : E(K_{G_0}, id_{CH_i} \parallel K_{CH_i} \parallel K_{G_1} \parallel nonce), id_{CH_i}, MAC(K_{G_0}, id_{CH_i} \parallel nonce)$;

**(..7)**

**end**

## IV. Security and performance analysis

### A. Theoretical analysis

In our DKS-LEACH, the keys are encrypted before transmission. Therefore we are protected against eavesdropping. During the announcement of a new round the BS encrypts the probability $proba$ with the general key $K_{G_i}$ of the current round. Only legitimated nodes that own the global key can decrypt this message. Since the encrypted message and the message authentication code (MAC) include the $nonce$, we argue that all messages are not out to date. We guarantee a freshness of messages exchanged in the network. Also, the MAC allows all nodes to authenticate the BS as well the integrity of the threshold $proba$ received from the BS. We recall that the goal of the threshold $proba$ is to allow nodes to become or not a CH (see Section III-C).

If a node becomes CH, it encrypts its $id$, generates a MAC and sends these information to the BS as labeled **(..1)** in Algorithm 1. This prevents a malicious node to attempt to establish a pairwise key. The main reason is due to the fact that malicious nodes will not be authenticate by the BS and thus will be rejected by the BS. Afterwards, the BS encrypts the pairwise key, generates a MAC and sends directly all information to the CH **(..2)** (Algorithm 1). The confidentiality of the pairwise key allows us to avoid eavesdropping attack. The MAC provides authentication of the BS and the integrity of the received key. Therefore, we establish a pairwise key between the CH and the BS in a secure manner.

To establish the cluster's key, the CH encrypts the message "$adv$", generates a $MAC$, and uses a broadcast message to send information to all SN. Only legitimated nodes can decrypt the message and return a $join\_req$ response which contains their $id$. In fact, we ensure the confidentiality, the authentication and the integrity of $adv$ and $join\_req$ messages (**..4**). This secure mechanism enables only the participation of safe nodes during the cluster's formation. These steps are illustrated in Algorithm 2. Note that the same security mechanism are ensured in Algorithm 3.

*Complexity Analysis* : DKS-LEACH has only two types of keys (cluster's key and pairwise key), and the number of keys does not depend to the number of sensor nodes. Therefore, DKS-LEACH is suitable for large WSN. In contrast to SecLeach and S-LEACH where the number of keys follows the number of nodes. Table I illustrates a comparison between DKS-LEACH, SecLEACH, and S-LEACH with respect to the complexity of key management. The parameter $m$ in Table I represents the number of keys located in each subset.

Furthermore, DKS-LEACH is suitable for large WSN. Due to most of operations to build, session key is computed by the base station, that it will reduce the overhead of simples nodes.

### B. Performance analysis

*1) Simulation setup:* We have implemented LEACH and DKS-LEACH using the *TinyOS* environment [14]. All simulations were carried out using the TOSSIM simulator [15]. TOSSIM is a simulator of WSN which compiles a TinyOS application and simulates a sensor network executing the target application. In order to analyze the output trace given by TOSSIM, we have used the *PowerTossim* extension. This tool gives accurate energy reports based on *mica2* motes consumptions: cpu, radio, sensors etc.

We have also used *TinySec*, which is implemented in TinyOS, as a cryptographic library. TinySec contains two cryptographic ciphers: *Skipjack* and *RC5*. It should be noted that in our simulations, we used the Skipjack algorithm for computing encryptions and for the MAC. The main reason is due to the fact that Skipjack is faster than RC5 with respect to computation time.

We have done our simulations following different grid topologies. Simulation time for all scenarios was fixed to $500$ seconds. We consider a network size of 50, 100, 150, and 200 sensor nodes. It should be noted that sensor nodes are deployed at random in the network. The number of chosen CH is fixed to 10% with respect to the number of sensor nodes inside the network. We consider two metrics : the energy consumption and the end-to-delay. The end-to-end delay represents the elapsed time between a measurement task, done by a sensor node, and its reception by the base station.

| | Space storage | Type of keys | Connectivity |
|---|---|---|---|
| S-LEACH | $m$ keys $\times$ key size | $pairwise$, $cluster$, $broadcast$ | probabilistic |
| SecLEACH | $m$ keys $\times$ key size | $pairwise$, $cluster$, $broadcast$ | probabilistic |
| DKS-LEACH | (3 keys for $CH_i$ and 2 keys for SN) $\times$ key size | $pairwise$, $cluster$ | 100% |

TABLE I
COMPARISON OF PROPERTIES BETWEEN THE KEY MANAGEMENT PROTOCOLS.

For each given network size $k$, we compute for each metric the average as the average of all values corresponding to 10 simulations results. In our simulations, we adopt *Lossy propagation model*.

*2) Energy consumption:* Based on TOSSIM, we measured the energy consumption of LEACH and DKS-LEACH. We compute the average, minimum, maximum energy consumed following different number of sensor nodes. Note that, we performed 10 simulations for each network size.

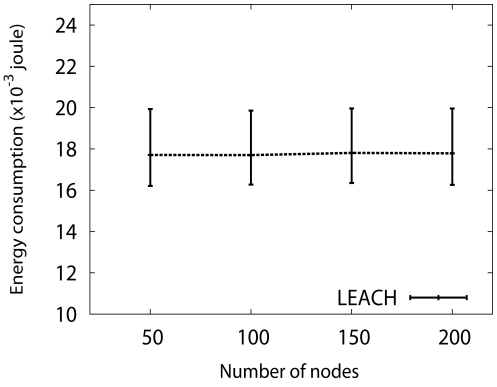
Figure 2 illustrates the impact of the number of sensor nodes in the performance of DKS-LEACH when we take into account the energy consumption as metric. Since we performed 10 simulations, error bars in Figure 2 indicates the minimum and the maximum energy consumed by a given sensor node according to our different simulations.

Figure 2(a) and Figure 2(b) show the average energy consumption of sensor nodes following different network size. Based on both figures, we remark that the gap between DKS-LEACH and LEACH is very low. DKS-LEACH by increasing a little bit the average power consumption allows to secure LEACH. By lack of space, the figures that illustrated these results are not shown.

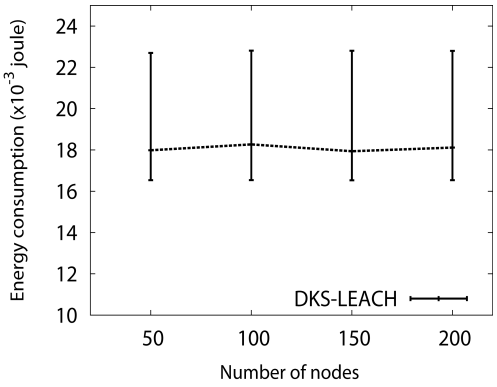
Figure 2(b) shows that DKS-LEACH respects the main characteristic of LEACH which expresses that the energy consumption is not related to number of sensor nodes. Note that DKS-LEACH consumes on average more than 1.775% compared to LEACH (Figures 2. This gap is due to exchanged messages used in order to secure LEACH. It should be noted that the energy consumption of CH is upper than the consumption of SN. The maximum values illustrated by error bars (Figure 2) represent the energy consumption of CHs. Based on Figure 2(b), we remark that the CH consumption according to DKS-LEACH is more important compared to LEACH's CH. The main reason is due to the fact that in DKS-LEACH CH do more computation.

*3) End to End Delay:* To evaluate the end-to-end delay (EED), we used the simulator PowerTossim in order to extract the delays, between sensor nodes, obtained from TOSSIM. Indeed, this file allows us to retrieve the time when packets are sent and received between sensor nodes.

Figure 3 illustrates the average EED computed overall sensor nodes in the network. As shown in the Figure 3, we note that the EED is not related to the number of nodes deployed in the network. In fact, a simple node can send its information in one hop. SN just need to send the information to its CH, and afterwards the CH aggregate data and send it to the BS. The gap noticed between DKS-LEACH and LEACH is also very low. This means that the encryption of messages does not take a lot of time. Also, the number of sensor nodes inside the network does not influence the end-to-end delay.




(a) LEACH




(b) DKS-LEACH

Fig. 2. Energy consumption of sensor nodes.

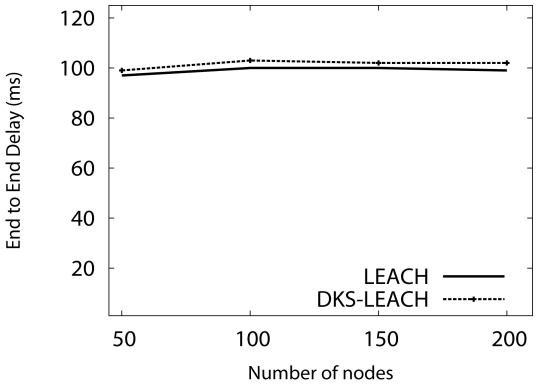



Fig. 3. Comparison of end-to-end delay between DKS-LEACH and LEACH

## V. Conclusion

We presented DKS-LEACH, a deterministic key management, for securing node-to-node communication in LEACH protocol. We performed a theoretical analysis of our deterministic key management. By securing the communications between simple nodes an cluster head as well the communication between cluster heads and the base station we enable LEACH to avoid a different type of attacks from malicious nodes.

Our results show that the overhead incurred by DKS-LEACH compared to LEACH is small. DKS-LEACH allows to level off the energy consumption and the end-to-delay even if the number of sensor nodes increase in the network. Furthermore, by using a limited number of keys DKS-LEACH minimize the memory usage in contrast to the previous works.

As future works, we plan to take into account the case where a sensor node and/or the base station are compromise by a malicious node. We plan to add self-resilience scheme to DKS-LEACH.

## References


[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Comput. Netw.*, vol. 38, no. 4, pp. 393–422, 2002.

[2] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, pp. 660–670, October 2002.

[3] M. Ye, C. Li, G. Chen, J. Wu, and M. Y. E. Al, "Eecs: An energy efficient clustering scheme in wireless sensor networks," in *In: Proc. of the IEEE Int'l Performance Computing and Communications Conf*. IEEE Press, 2005, pp. 535–540.

[4] "Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *IEEE Transactions on Mobile Computing*, vol. 3, no. 4, pp. 366–379, 2004.

[5] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasures," in *IEEE SNPA*, May 2003, pp. 113–127.

[6] J. Lee, V. Leung, K. Wong, J. Cao, and H. Chan, "Key management issues in wireless sensor networks: current proposals and future developments," *Wireless Communications, IEEE*, vol. 14, no. 5, pp. 76 –84, october 2007.

[7] Y. Wang, G. Attebury, and B. Ramamurthy, "A survey of security issues in wireless sensor networks," *IEEE Communications Surveys*, vol. 8, pp. 2–23, 2006.

[8] D. Liu, P. Ning, and R. Li, "Establishing pairwise keys in distributed sensor networks," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 1, pp. 41–77, 2005.

[9] S. Zhu, S. Setia, and S. Jajodia, "Leap: efficient security mechanisms for large-scale distributed sensor networks," in *in Proc. ACM CCS*, October 2003, pp. 62–72.

[10] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *in Proc. of ACM CCS*, 2002, pp. 41–47.

[11] O. H. W. Ferreira, Vilaça and Loureiro, "On the security of clusterbased communication protocols for wireless sensor networks," *in Proc of ICN*, pp. 449–458, April 2005.

[12] M. B. Leonardo B. Oliveira Hao C. Wong, "Secleach: A random key distribution solution for securing clustered sensor networks," in *IEEE NCA*, April 2006.

[13] D. Xu, J. Huang, J. Dwoskin, M. Chiang, and R. B. Lee, "Re-examining probabilistic versus deterministic key management," in *IEEE ISIT*, June 2007, pp. 2586–2590.

[14] P. Levis and D. Gay, *TinyOS Programming*. New York, NY, USA: Cambridge University Press, 2009.

[15] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *in Proc. of SenSys*, 2003, pp. 126–137.