Ann. Telecommun.
DOI 10.1007/s12243-009-0123-0

MUQAMI+: a scalable and locally distributed key management scheme for clustered sensor networks

Muhammad Khaliq-ur-Rahman Raazi Syed · Heejo Lee · Sungyoung Lee · Young-Koo Lee

Received: 10 November 2008 / Accepted: 19 June 2009

Abstract Wireless sensor networks (WSN) are susceptible to node capture and many network levels attacks. In order to provide protection against such threats, WSNs require lightweight and scalable key management schemes because the nodes are resource-constrained and high in number. Also, the effect of node compromise should be minimized and node capture should not hamper the normal working of a network. In this paper, we present an exclusion basis system-based key management scheme called MUQAMI+ for large-scale clustered sensor networks. We have distributed the responsibility of key management to multiple nodes within clusters, avoiding single points of failure and getting rid of costly inter-cluster communication. Our scheme is scalable and highly efficient in terms of re-keying and compromised node revocation.

Keywords Locally distributed · Key management · Sensor networks · Key revocation · Scalable key management · Flexible scheme

M. K. R. R. Syed · S. Y. Lee (✉) · Y. K. Lee
Department of Computer Engineering,
Kyung Hee University, Seoul, Korea
e-mail: sylee@oslab.khu.ac.kr

M. K. R. R. Syed
e-mail: raazi@khu.ac.kr

Y. K. Lee
e-mail: yklee@khu.ac.kr

H. Lee
Divison of Computer & Communication Engineering,
Korea University, Seoul, South Korea
e-mail: heejo@korea.ac.kr

1 Introduction

Wireless sensor networks (WSN) are employed in various application areas, which include habitat monitoring, military surveillance, border monitoring, and health care. WSNs differ from other distributed network systems in such a way that they have to work in real-time with given constraints, which include energy, storage, computation, and communication. WSNs are mostly data centric and are used to monitor their surroundings, gather information, and filter it [3]. A sensor network typically consists of a large number of sensor nodes working together to collect data and gather it in a central node, using wireless communications [27].

Security is an important aspect in WSN. Adversaries may try to listen to communications, disrupt them, or even block them. Also, an adversary may attack externally, e.g., capture the node or jam the traffic signals. In addition to being secure, WSN should also be cost effective because their batteries may not be recharged. This also limits their memory and computational power. So the WSN require security mechanisms that are also resource-efficient. Highly effective security mechanisms, such as TLS [7] and Kerberos [18], exist, but they cannot be applied to the WSN paradigm because they are not resource-efficient.

In WSN, group communications are performed to increase efficiency. Groups of nodes share common secret keys. If a node is compromised, it must be evicted from the group and keys must be refreshed in such a way that the compromised node does not get to know the new key values. A single key cannot be used for the whole network because, in that case, if even a single node is compromised, it compromises the whole network with it. On the other extreme, all pairs of sensor nodes can

Published online: 16 July 2009

have separate keys. This provides high security, but it hampers network processing [15, 17] because some schemes use passive participation of nodes, i.e., they decide their actions after overhearing the messages [16, 22]. So, we need a lightweight scheme, which also enables sharing of a key with large numbers of nodes.

In this paper, we proposed MUQAMI+,[1] a lightweight, scalable, and locally distributed key management scheme for clustered sensor networks. In MUQAMI+, a large number of nodes in a cluster share common keys. MUQAMI+ is efficient not only for periodic key refreshment but also for revocation of a compromised node. Also, our scheme allows the role of being a cluster head to be shifted from node to node with the passage of time. MUQAMI+ is based on exclusion basis system (EBS) matrix [10] and key chains [8]. The key chain is an authentication mechanism based on Lamport's one-time passwords [19].

The rest of the paper is organized as follows: Section 2 outlines background and related work. Section 3 describes models and assumptions. Section 4 presents our scheme. Section 5 and Section 6 contain theoretical and simulation-based analysis and evaluation, respectively. Section 7 concludes the paper and suggest some future work.

2 Background and related work

Static key management is the primitive form of key management in WSN. It is sometimes also referred to as key pre-distribution, in which keys are calculated and pre-loaded in the nodes before the deployment of the WSN. Intensive research has been done in devising efficient methods for distributing keys before the network deployment [6, 9, 12, 21]. Camtepe et. al. [5] have proposed a key distribution approach based on combinatorial design using balanced incomplete block design (BIBD) and generalized quadrangles (GQ). These are static key management schemes and they work on the assumption that WSN are very short-lived networks. However, a real-life example of WSN Mica2 has a life-time of 2 weeks at full power [17]. If keys are not refreshed periodically, there are always chances of cryptanalytic attacks on the WSN.

Many dynamic key management schemes have been proposed, which emphasize the refreshment and revocation of keys periodically. Riaz et al. [26] proposed a scheme that actively involves the base station for communication among sensor nodes using public keys. Drawback of this scheme is the frequent communication between sensor nodes and the base station as it incurs a lot of communication overhead. G. Dini [8] proposed a tree-based key revocation protocol for WSNs based on key chains. Apart from increased storage overhead, another drawback of their scheme is that there is a lot of communication and computation overhead in case of node compromise.

LEAP+ [31] and SHELL [13] are two state-of-the-art schemes for key management in WSN. Also, K.J. Paek et al. [24] proposed key management based on regional and virtual groups. Drawbacks of Paek et al. [24] and LEAP+ [31] are that they assume that the network is safe during some initial time period. Also, all the nodes have to generate keys, which consume a lot of energy. SHELL [13] does not require all the nodes to generate keys, but it has a lot of inter-cluster communication, which is also not desirable. Later, Eltoweissy et al. briefly proposed LOCK [11], which eliminates inter-cluster communication by distributing key generation responsibilities among few nodes within the cluster. However, LOCK requires some nodes to have more capabilities than normal sensor nodes so that they can generate keys. Otherwise, it causes the key generating nodes to die down more quickly. Our scheme assumes no initial safe time period. Also, there is no inter-cluster communication and very few nodes are involved in key management. Moreover, our scheme is flexible and allows the key management responsibilities to rotate among different nodes within the cluster. In addition to that, we use key chains [8] to keep the key generation cost low.

3 Models and assumptions

3.1 System model and assumptions

WSN consist of a command node connected to a number of sensor nodes, which can be grouped into clusters. We assume that the constraints of WSN do not apply to the command node. We are assuming clustered sensor networks, in which cluster head node aggregates information from other sensor nodes and sends it back to the command node. Clustering can be based on some criteria like in [1, 14], where nodes do not know their locations. Sensor nodes relay their messages directly or indirectly, depending upon their communication ranges [2, 20]. There are many applications of WSN-like soil moisture monitoring and battlefield monitoring, in which nodes do not change their location after initial deployment. It is also important to minimize key

[1] In the conference version, we named this protocol MUQAMI. Since it is much improved in this journal version, we have named it MUQAMI+.

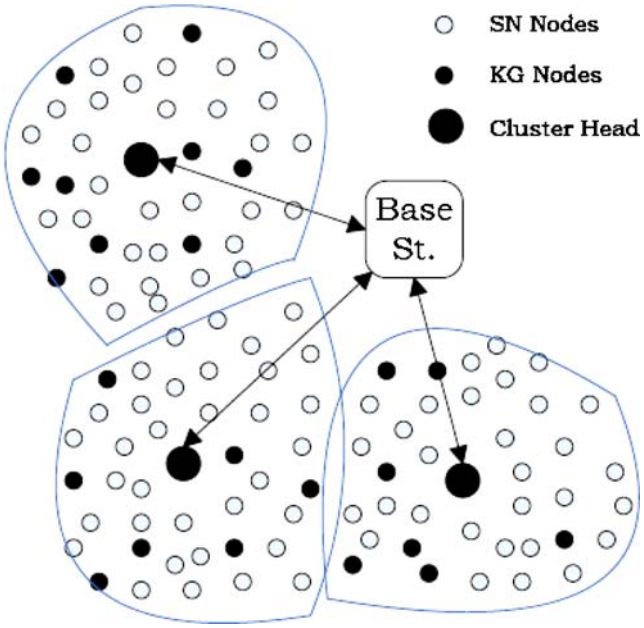


Fig. 1 Clustered sensor networks architecture

management overhead in such applications. We are assuming that all nodes, including the cluster heads, are stationary. Authors of other state-of-the-art schemes, like LEAP+ and SHELL, have also assumed the sensor nodes to be stationary. Communication range and physical locations of all nodes are known to the nodes at the higher levels. The role of being cluster head requires more capabilities in a node as compared to the simple sensor nodes. The role of being a cluster head node can be shifted between nodes, which can bear the responsibility of a cluster head node. Figure 1 depicts the network architecture assumed in our scheme. Each node can be a member of only one cluster. If it falls within the boundaries of more than one cluster head node, it must choose one cluster for its membership.

3.2 Adversity model and assumptions

According to our model, the adversary will try to get hold of keys so that it can attack actively or passively. In order to get hold of the keys, an adversary can even capture a node and use it in any way possible. It may use its memory, spread false messages, and sabotage communications. The sensor nodes are assumed not to be tamper-resistant. Another assumption is that the higher we go in the hierarchy of nodes, the more difficult it gets for an adversary to capture a node and that the command node is completely secure. Moreover, we assume that the compromised nodes do not collude, i.e., they do not collaborate with each other to carry out an attack. Also, a compromised node is revoked as soon as it is detected, and we have a reasonable compromised node detection technique employed in the network. Our last assumption is that compromised nodes cannot communicate with each other through any external communication channel. Detection of attacks is out of the scope of this paper. Readers interested in attack detection can refer to [4, 30] for further knowledge.

4 MUQAMI+

In our scheme, the command node (*CN*) stores all node IDs. Since the CN does not have energy constraints, we have tried to move as much load to the CN as possible. The CN is responsible for managing basic keys (K_{bsc}) and discovery keys (K_{disc}) for all the nodes. It is also responsible for managing keys between cluster heads and the command node. In order to facilitate in network processing and reduce the overall security overhead, our scheme secures all communications using K_{comm}, which is a group key used for providing group confidentiality. However, using only K_{comm} is very risky, and it also cannot secure a network against insider attacks. Therefore, we use administrative keys K_{admin} to secure K_{comm} and to protect the network against insider attacks. Apart from that, every node in the network shares a pair-wise key with its CH. If some key, other than K_{comm}, is required to secure communication between a pair of sensor nodes, the CH node sends a pair-wise key to that pair of sensor nodes directly. In this paper, we use different notations, which are mentioned in Table 1.

Our scheme uses the EBS system of matrices [10] to manage keys. In EBS, a small number of keys are required to manage a large number of nodes. Every node knows a distinct set of k keys out of a set of $k+m$ keys. We have proposed a little change in the representation of the EBS matrix. Usually, we use "0" if a node does not know a key and "1" if a node knows a key. In our scheme, we also use "2," which means that a node generates a key. Table 2 shows an example of an EBS matrix. Over 3,000 key combinations are available if 14 keys are used.

After the CH nodes are deployed in the initial phase, the CN sends K_{disc} of all the nodes to their respective CH nodes, so that the CH nodes can recognize the newly deployed nodes in their respective clusters. After the nodes are deployed, the CN computes the details of EBS matrices according to the locations of all the nodes in the network and shares them with the respective CH nodes. CN also sends initial values of relevant administrative keys to each node. While forwarding the encrypted administrative keys to the respective nodes

Table 1 List of used notations

CN	Command node or the base station
CH^i	Cluster head node i
KG^i	Key generating node i. KG nodes compute keys using lightweight one-way hash functions, rather than generating them
SN^i	Sensor node i
$\{CH\}$	Set of all the CH nodes
$\{SN_{CH^i}\}$	Set of the SN nodes belonging to CH i
$\{KG_{CH^i}\}$	Set of the KG nodes belonging to CH i
K^i_{bsc}	Basic key of node i. Used for communication with the command node. It is preloaded in every node of the network and refreshed after being used once
K^i_{disc}	Discovery key of node i. Used for initial discovery of the node. It is preloaded in every node of the network and refreshed after being used once
$K^{i,j}_{ch,kg}$	Key used for communication between CH i and KG j
$K^{i,j}_{ch,sn}$	Key used for communication between CH i and SN j
K_{comm}	Communication key
K^i_{admin}	Administrative key i
$K^i_{cn,ch}$	Key used for communication between CN and CH node i
mi	Message number i in a particular communication sequence
$E_{K\{A\|B\}}$	Values A and B is put together in a block/chunk and then the chunk is encrypted using key K

in its cluster, CH nodes also establish pairwise keys with all nodes in its cluster. For key refreshment, CH asks the KG nodes to send new keys to sensor nodes in its cluster. KG nodes compute key values with the help of lightweight one-way hashing functions [23] and broadcast them in the cluster. Keys, other than the administrative and communication keys, are very rarely used. Figure 2 elaborates the working of our scheme MUQAMI+ with the help of a flow diagram.

4.1 Initial deployment

CH nodes are deployed in the first phase. Following is the first message that a newly deployed CH i sends to the CN:

$$m1 : \forall CH^i \in \{CH\} : CH^i \rightarrow CN : E_{K^i_{disc}}\{ID|Auth\_Code\}$$

Table 2 Example of an EBS matrix for MUQAMI+

	N_0	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9
K_1	2	1	1	1	1	1	0	0	0	0
K_2	1	2	1	0	0	0	1	1	1	0
K_3	1	0	0	2	1	0	1	1	0	1
K_4	0	1	0	1	0	2	1	0	1	1
K_5	0	0	1	0	2	1	0	1	1	1

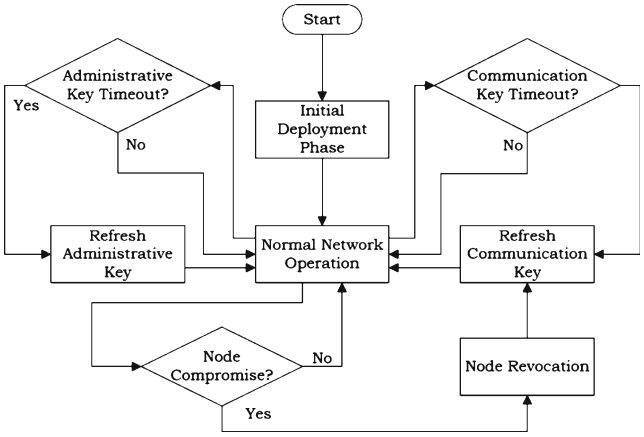


Fig. 2 Outline of the proposed scheme MUQAMI+

Then for every CH i, the CN authenticates it and sends to it the $K^i_{cn,ch}$ and the EBS matrix of its cluster along with IDs and K_{disc} of all nodes j, which are to be deployed in the cluster of CH i:

$$m2 : \forall CH^i \in \{CH\} : CN \rightarrow CH^i : E_{K^i_{disc}}\Big\{K^i_{cn,ch}|EBS\_Matrix \; |\forall SN^j \in \{\{SN_{CH^i}\} \cup \{KG_{CH^i}\}\} : \Big\{ID(SN^j)|K^j_{disc}\Big\}\Big\}$$

In the above message, IDs and K_{disc} of all the relevant nodes are put together in a block along with the $K^i_{cn,ch}$ and the relevant EBS matrix, then encrypted using K_{disc} of the CH node and then sent to the CH i from the CN. SN and KG nodes are deployed in the second phase. The following messages are exchanged for every KG node j that is deployed in the cluster i:

$$\begin{aligned}
&\forall CH^i \in \{CH\} \wedge \forall KG^j \in \{KG_{CH^i}\} : \\
&m1 \; : \; KG^j \rightarrow CH^i : E_{K^j_{disc}}\Big\{ID(KG^j)|Auth_Code\Big\} \\
&m2 \; : \; CH^i \rightarrow CN : E_{K^i_{cn,ch}}\Big\{ID(KG^j)|Auth_Code\Big\} \\
&m3 \; : \; CN \rightarrow CH^i : E_{K^i_{cn,ch}}\Big\{E_{K^j_{bsc}}\Big\{K^j_{bsc_new} \\
&\qquad |K^j_{disc_new}|K^1_{admin}|K^2_{admin}|...|K^{k-1}_{admin}\Big\}\Big\} \\
&m4 \; : \; CH^i \rightarrow KG^j : E_{K^j_{disc}}\Big\{K^{i,j}_{ch,kg}|E_{K^j_{bsc}} \\
&\qquad \Big\{K^j_{bsc_new}|K^j_{disc_new}|K^1_{admin}|K^2_{admin}|...|K^{k-1}_{admin}\Big\}\Big\}
\end{aligned}$$

In the above messages, after authenticating a new KG node j, i.e., after the first two messages, CN puts together all the K_{admin} relevant to the KG node j in a block along with the new values of K_{bsc} and K_{disc}, encrypts this block first using the current value of K_{bsc} and then using $K^i_{cn,ch}$, and then sends it to the CH i in message $m3$. After receiving $m3$, CH i generates the seed value for $K^{i,j}_{ch,kg}$ and computes the whole key chain associated with $K^{i,j}_{ch,kg}$. CH i then adds the seed value of $K^{i,j}_{ch,kg}$ into the block and sends it to KG j in $m4$. Note that the CN sends new values for K_{bsc} and K_{disc} every

time they are used. Also, $k-1$ administrative keys are communicated to a KG node as it is responsible for generating one key by itself. After receiving $m4$, KG j computes the associated key chain for $K^{i,j}_{ch,kg}$. Similar message exchanges take place for every SN node j that is deployed in cluster i:

$$\forall CH^i \in \{CH\} \wedge \forall SN^j \in \{SN_{CH^i}\}:$$
$$m1 : SN^j \rightarrow CH^i : E_{K^j_{disc}}\left\{ID(SN^j)|Auth\_Code\right\}$$
$$m2 : CH^i \rightarrow CN : E_{K^i_{cn,ch}}\left\{ID(SN^j)|Auth\_Code\right\}$$
$$m3 : CN \rightarrow CH^i : E_{K^i_{cn,ch}}\left\{E_{K^j_{bsc}}\left\{K^j_{bsc\_new} |K^j_{disc\_new}|K^1_{admin}|K^2_{admin}|...|K^k_{admin}\right\}\right\}$$
$$m4 : CH^i \rightarrow SN^j : E_{K^j_{disc}}\left\{K^{i,j}_{ch,sn}|E_{K^j_{bsc}} \left\{K^j_{bsc\_new}|K^j_{disc\_new}|K^1_{admin}|K^2_{admin}|...|K^k_{admin}\right\}\right\}$$

No key chain is associated with $K^{i,j}_{ch,sn}$ as it is rarely used. Sometimes, a node is not deployed in its expected cluster. In that case, messages $m2$ and $m3$ in the above message exchanges will be changed as follows:

$$m2 : CH^i \rightarrow CN : E_{K^i_{cn,ch}}\left\{E_{K^j_{disc}}\left\{ID(SN^j)|Auth\_Code\right\}\right\}$$
$$m3 : CN \rightarrow CH^i : E_{K^i_{cn,ch}}\left\{K^j_{disc}|E_{K^j_{bsc}}\left\{K^j_{bsc\_new} |K^j_{disc\_new}|K^1_{admin}|K^2_{admin}|...|K^k_{admin}\right\}\right\}$$

In the end, CN shares the final version of the EBS matrix with the CH node as follows:

$$m1 : CN \rightarrow CH^i : E_{K^i_{cn,ch}}\{EBS\_Matrix\}$$

Note that the cluster heads do not know the administrative keys being used in their clusters. This is important to avoid a single point of failure, i.e., revelation of all K_{admin} in case of compromise of CH. Next, the initial values of communication keys are distributed. Every CH node i sends communication keys to all KG nodes in its cluster. In turn, every KG node broadcasts the communication key in the cluster using the administrative keys that it manages. The message exchanges for broadcasting the initial values of communication keys are as follows:

$$\forall CH^i \in \{CH\} \wedge \forall KG^j \in \{KG_{CH^i}\}:$$
$$m1 : CH^i \rightarrow KG^j : E_{K^{i,j}_{ch,kg}}\left\{K^i_{comm}\right\}$$
$$m2 : KG^j \rightarrow *: E_{K^j_{admin}}\left\{K^i_{comm}\right\}$$

4.2 Re-keying and node addition

In order to avoid the cryptanalytic attacks on the network, keys need to be refreshed regularly. Communication keys are refreshed in the same manner as they were distributed initially, i.e., using the administrative keys. Administrative keys are refreshed using their previous values. In order to refresh K^l_{admin} of its own cluster, CH node i sends a refresh message to the KG node j, which manages K^l_{admin}. KG node j then broadcasts the new administrative key encrypted in the old one. The following message exchanges take place:

$$m1 : CH^i \rightarrow KG^j : E_{K^{i,j}_{ch,kg}}\{Refresh\_Message\}$$
$$m2 : KG^j \rightarrow *: E_{K^l_{admin}}\left\{K^l_{admin\_new}\right\}$$

When a sensor node receives a new value of an administrative key, it verifies the new value through the one-way hashing function as follows:

$$K^l_{admin} = F\left(K^l_{admin_new}\right)$$

where F is the one-way hashing function that is used to compute the administrative keys. Since we use key chains to manage $K_{ch,kg}$ and K_{admin}, it becomes necessary for the KG node to get the new seed value from CH node or the CN node, respectively. A KG node j gets the new value of K^l_{admin}, which it manages, from the CN through its CH i, as follows:

$$m1 : KG^j \rightarrow CH^i : E_{K^{i,j}_{ch,kg}}\left\{E_{K^j_{bsc}}\left\{Auth\_Code|Refresh\_Msg\right\}\right\}$$
$$m2 : CH^i \rightarrow CN : E_{K^i_{cn,ch}}\left\{E_{K^j_{bsc}}\left\{Auth\_Code|Refresh\_Msg\right\}\right\}$$
$$m3 : CN \rightarrow CH^i : E_{K^i_{cn,ch}}\left\{E_{K^j_{bsc}}\left\{K^j_{bsc\_new}|K^l_{admin\_seed}\right\}\right\}$$
$$m4 : CH^i \rightarrow KG^j : E_{K^{i,j}_{ch,kg}}\left\{E_{K^j_{bsc}}\left\{K^j_{bsc\_new}|K^l_{admin\_seed}\right\}\right\}$$

KG node j can get the new seed value for $K^{i,j}_{ch,kg}$ from the CH node i using the last value of $K^{i,j}_{ch,kg}$ in the key chain.

For the addition of SN node j in cluster i, CN sends the following message to the CH i: -

$$m1 : CN \rightarrow CH^i : E_{K^i_{cn,ch}}\left\{ID(SN^j)|K^j_{disc}\right\}$$

After getting this message, CH i waits for the discovery message from the SN node j. When deployed, SN node

j will contact the CH i with its discovery key K^j_{disc}. The following message exchanges will take place to add the new SN node j in cluster i:

$$m2 : SN^j \rightarrow CH^i : E_{K^j_{disc}}\{ID(SN^j)|Auth\_Code\}$$

$$m3 : CH^i \rightarrow CN : E_{K^i_{cn,ch}}\{ID(SN^j)|Auth\_Code |Cur\_Admin\_Chain\_Indexes\}$$

$$m4 : CN \rightarrow CH^i : E_{K^i_{cn,ch}}\left\{E_{K^j_{bsc}}\left\{K^j_{bsc\_new}|K^j_{disc\_new} |K^1_{admin}|K^2_{admin}|...|K^k_{admin}\right\}\right\}$$

$$m5 : CH^i \rightarrow SN^j : E_{K^j_{disc}}\left\{K^{i,j}_{ch,sn}|E_{K^j_{bsc}}\left\{K^j_{bsc\_new}|K^j_{disc\_new} |K^1_{admin}|K^2_{admin}|...|K^k_{admin}\right\}\right\}$$

where $Cur\_Admin\_Chain\_Indexes$ represents the number of times K_{admin}, related to SN node j, has been refreshed. Based on $Cur\_Admin\_Chain\_Indexes$, the CN calculates and sends to the SN j the current values of the K_{admin} related to the SN node j. In case a new KG node j is to be deployed in cluster i, CN sends initial value of the new key "l," which KG j manages, to all the relevant SN nodes in the cluster through the CH i. Then, the new KG node j is deployed in the same manner in which a new SN node is deployed. The only difference is that $k-1$ admin keys are sent to the new KG node as it generates one by itself. The fact that the CN is often solicited through CH nodes has an impact on the energy consumption of CH nodes. A single node may not be able to act as a CH node throughout the network lifetime. Therefore, our scheme has the flexibility to shift the responsibility of being the CH node from the current CH node to another node, which has the capability of becoming CH. Refer to Section 4.3.1 for details regarding the addition a new CH node.

4.3 Node compromise

If a node is compromised, we need to refresh the keys in such a way that the new keys are not known to the compromised node and it can only act as an outsider when trying to interfere in the network operation. We assume that an efficient mechanism to detect an attack is already in place and the relevant CH node starts the recovery procedure. In case of CH node compromise, CN starts the procedure. We have three types of node in our network. We will discuss the implications of the compromise of each type of node one by one.

4.3.1 Cluster head compromise

If a CH node is compromised, CN can either deploy a new CH node or designate an existing node from the network to act as a CH node. Apart from sharing the discovery keys of all nodes in the cluster and the EBS matrix of the cluster i with the new CH i, CN sends a validation message to each node in the cluster through the new CH node i. CH i cannot decrypt the validation messages as they are encrypted using K_{bsc} of the related nodes. The following message is exchanged between CN and the new CH node i:

$$m1 : CN \rightarrow CH^i : E_{K^i_{bsc}}\left\{EBS|\forall l \in \{\{SN_{CH^i}\} \cup \{KG_{CH^i}\}\} : \left\{K^l_{disc}|E_{K^l_{bsc}}\left\{K^l_{bsc\_new}|K^l_{disc\_new}|CH\_Valid\right\}\right\}\right\}$$

Then, the CH node sends the validation messages to all the SN nodes k along with the new value of $K^{i,k}_{ch,sn}$. For all the KG nodes j, it will send the validation message along with a new seed value of $K^{i,j}_{ch,kg}$. The new CH node i will send the following messages to each SN node k and KG node j, respectively:

$$m2 : \forall SN^k \in \{SN_{CH^i}\} : CH^i \rightarrow SN^k : E_{K^k_{disc}} \left\{K^{i,k}_{ch,sn}|E_{K^k_{bsc}}\left\{K^k_{bsc\_new}|K^k_{disc\_new}|CH\_Valid\right\}\right\}$$

$$m2 : \forall KG^j \in \{KG_{CH^i}\} : CH^i \rightarrow KG^j : E_{K^j_{disc}} \left\{K^{i,j}_{ch,kg}|E_{K^j_{bsc}}\left\{K^j_{bsc\_new}|K^j_{disc\_new}|CH\_Valid\right\}\right\}$$

4.3.2 Sensor node compromise

If an SN node is compromised, we need to distribute the set of K keys known to the compromised SN node using the M keys not known to the compromised SN node. So, in the first phase, the CH node will ask all the KG nodes, which generate those K keys, to generate new values, encrypt them using the previous ones, and send them back to the CH node. If the SN node is compromised in cluster i, the following communications will take place between CH i and the KG nodes, which manage those K keys in the cluster:

$$m1 : \forall p \in K : CH^i \rightarrow KG^p : E_{K^{i,p}_{ch,kg}}\{Revoc\_Msg\}$$

$$m2 : \forall p \in K : KG^p \rightarrow CH^i : E_{K^{i,p}_{ch,kg}}\left\{E_{K^p_{admin}}\left\{K^p_{admin\_new}\right\}\right\}$$

Now, the CH node will aggregate these K encrypted values and send the aggregated message to the M KG nodes, i.e., those KG nodes, which manage keys that are not known to the compromised node. Each one of those M KG nodes will then broadcast the aggregated messages using the key it manages. SN nodes, which

use any of those K compromised keys, will get the new value using some key that it knows other than those K keys. Note that no two nodes know the same set of K keys in the EBS matrix (refer to Table 2). The following message exchanges will take place to distribute the refreshed keys:

$$m3 : \forall q \in M : CH^i \rightarrow KG^q :$$
$$E_{K^{i,q}_{ch,kg}} \left\{ \forall p \in K : E_{K^p_{admin}} \left\{ K^p_{admin\_new} \right\} \right\}$$
$$m4 : \forall q \in M : KG^q \rightarrow * :$$
$$E_{K^q_{admin}} \left\{ \forall p \in K : E_{K^p_{admin}} \left\{ K^p_{admin\_new} \right\} \right\}$$

Note that, in all these communications, the compromised SN node can not use the keys known to it in order to interfere in the network operations.

4.3.3 Key-generator compromise

If a KG node is compromised, either a new node will be deployed or an existing node will be given the responsibility of managing the key, which was previously managed by the compromised node. If a new node is deployed, key chain will be pre-loaded into it before deployment. It will only need to know the current key value so that it can use it to send the new value. On the other hand, if an existing node is given the responsibility, it will also need a seed value to compute the new key chain. In order to award the responsibility to an existing SN j in cluster i, the following messages are exchanged:

$$m1 : CH^i \rightarrow CN : E_{K^i_{cn,ch}} \{ Revoc\_Msg | Key\_ID | Cur\_Refr\_Iter \}$$
$$m2 : CN \rightarrow CH^i : E_{K^i_{cn,ch}} \left\{ ID(SN^j) | K^j_{disc} | E_{K^j_{bsc}} \left\{ K^j_{bsc\_new} | K^j_{disc\_new} | Cur\_Key\_Val | Seed\_Val \right\} \right\}$$
$$m3 : CH^i \rightarrow SN^j : E_{K^j_{disc}} \left\{ E_{K^j_{bsc}} \left\{ K^j_{bsc\_new} | K^j_{disc\_new} | Cur\_Key\_Val | Seed\_Val \right\} \right\}$$

where $Seed\_Val$ is the seed value of the K_{admin} that SN^j has to manage. After these messages are exchanged, SN^j becomes one of the K KG nodes, which know one compromised key each in the cluster i. A similar procedure, as in the previous section (Section 4.3.2), is followed to distribute the set of K compromised keys using the remaining set of M keys.

Since a compromised node is not bound in its behavior, it may happen that it refreshes a key without consent of the CH node. If the compromised KG node has already refreshed the compromised key without the instructions of the CH i, then a new initial value of the compromised key, encrypted with the respective K_{bsc} of all the relevant SN nodes, is sent to all the relevant SN nodes through the CH node i. In such a scenario, the following message exchanges will take place instead of the above message exchanges:

$$\mathbf{m1} : CH^i \rightarrow CN : E_{K^i_{cn,ch}} \{ Revoc\_Msg | Key\_ID \}$$
$$\mathbf{m2} : CN \rightarrow CH^i : E_{K^i_{cn,ch}} \left\{ ID(SN^j) | K^j_{disc} | E_{K^j_{bsc}} \{ K^j_{bsc\_new} | K^j_{disc\_new} | Seed\_Val \} | \forall SN^l \in Key\_ID : ID(SN^l) | K^l_{disc} | E_{K^l_{bsc}} \left\{ K^l_{bsc\_new} | K^l_{disc\_new} | New\_Init\_Key\_Val \right\} \right\}$$
$$m3 : CH^i \rightarrow SN^j : E_{K^j_{disc}} \left\{ E_{K^j_{bsc}} \left\{ K^j_{bsc\_new} | K^j_{disc\_new} | Seed\_Val \right\} \right\}$$
$$m4 : \forall SN^l \in Key\_ID : CH^i \rightarrow SN^l : E_{K^l_{disc}} \left\{ E_{K^l_{bsc}} \left\{ K^l_{bsc\_new} | K^l_{disc\_new} | New\_Init\_Key\_Val \right\} \right\}$$

5 Performance analysis and comparison

In this section, we will provide a brief comparison of MUQAMI+ with other schemes and try to establish our claims. Two factors contribute towards the usage of power in a node: communication overhead and computation overhead. However, we need to consider the storage overhead first as the sensor nodes are also limited in their storage capacity.

5.1 Storage overhead

EBS-based key management schemes are inherently able to support a large number of nodes with a small number of keys using combinatorics (see Fig. 3). Note that the graph is drawn on logarithmic scale because the number of nodes that can be supported increases exponentially with respect to the number of keys used. This is particularly helpful when compromised nodes need to be revoked. The number of nodes n that can be supported using EBS matrix is given by the formula:

$$n = \frac{(k+m)!}{k!m!}, \tag{1}$$

where k and m are EBS parameters. As opposed to SHELL [13], cluster heads in our scheme need not store any key to communicate with other cluster

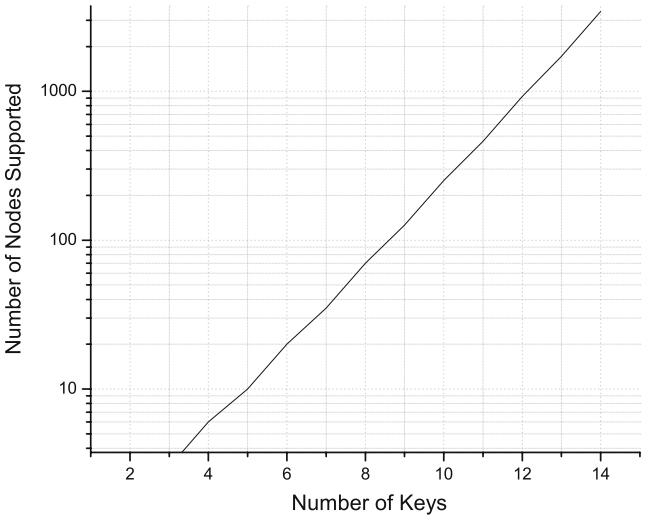


Fig. 3 Number of nodes that can be supported using the EBS-based scheme

heads. Moreover, gateways in our scheme also need not generate and store EBS keys for other clusters. In MUQAMI+, each CH node has to store one K_{comm} and $K_{cn,ch}$ apart from $K_{ch,sn}$ of all SN nodes and the key chains $K_{ch,kg}$ of all KG nodes in the cluster. So, the average storage requirement of a CH node (in number of keys) in MUQAMI+ can be expressed with the following formula:

$$SR_{CH}^{MUQAMI+} = (l \times (k+m)) + r - (k+m) + 2, \tag{2}$$

where l is length of the key chain and r is the number of nodes in a cluster. SN nodes have to store k admin keys apart from four other keys: $K_{ch,sn}$, K_{comm}, K_{bsc}, and K_{disc}. So, the average storage requirement of an SN node in MUQAMI+ can be expressed with the following formula:

$$SR_{SN}^{MUQAMI+} = k + 4 \tag{3}$$

KG nodes have to store two key chains: one for the admin keys, which it generates, and one for $K_{ch,kg}$. Also, it has to store $k-1$ EBS keys along with three other keys: K_{comm}, K_{bsc}, and K_{disc}. So the storage requirement of a KG node can be expressed as:

$$\begin{aligned} SR_{KG}^{MUQAMI+} &= 2l + (k-1) + 3 \\ &= 2(l+1) + k \end{aligned} \tag{4}$$

Since we have $k+m$ KG nodes out of r nodes inside the cluster, average storage requirement of each node within a cluster comes out to be:

$$\begin{aligned} SR_{SN \cup KG}^{MUQAMI+} &= \frac{(r-(k+m))(k+4)+(k+m)(2(l+1)+k)}{r} \\ &= \frac{r(k+4)+(k+m)(2(l+1)-4)}{r} \\ &= (k+4) + \frac{2(l-1)(k+m)}{r} \end{aligned} \tag{5}$$

Note that the ratio $((k+m):r)$ is very small as $(k+m) << r$ (see Eq. 1 and Fig. 3). Therefore, average storage requirements of a node inside a cluster are not too much as compared to SHELL.

In our scheme, we use one-way hashing functions to compute key chains. Also, we know that key-chain length l is a variable, which can change according to the storage capabilities of a node. However, l must be a value such that the cost of computing the keys does not exceed the cost of generating them on the nodes. In other words, the following inequality must hold:

$$\begin{aligned} & Cost_{seed} + l(Cost_{comp} - 1) < l(Cost_{gen}) \\ & \Rightarrow Cost_{seed} < l(Cost_{gen} - Cost_{comp} + 1) \\ & \Rightarrow l < \frac{Cost_{seed}}{Cost_{gen} - Cost_{comp} + 1} \end{aligned} \tag{6}$$

where $Cost_{seed}$, $Cost_{comp}$, and $Cost_{gen}$ are the costs of getting new seed value, computing a key value through one-way hashing function, and generating a key on a node, respectively.

Table 3 compares the storage requirements of MUQAMI+ with other schemes. In Table 3, b is the average number of neighboring nodes with whom SN node has to share pair-wise keys and h is the number of neighboring CH nodes, with whom a CH node communicates. The value of h can vary depending upon the value of EBS parameters k and m and the extent to which the distribution of keys is desired. Our

Table 3 Storage requirements (in number of keys) of each type of node in all the three schemes

	CH	SN
MUQAMI+	$(l \times (k+m)) + r - (k+m) + 2$	$(k+4) + [(2(l-1)(k+m))/r]$
LEAP+	$r+2$	$b+2$
SHELL	$l+r+h+k+m+1$	$l+k+3$

Table 4 Average number of bytes transmitted by each type of node on each link during initial deployment phase (an asterisk means a broadcast within a cluster)

	MUQAMI+	LEAP+	SHELL
$CH \rightarrow CH$	–	–	$x(h+r+k+m)$
$CH \rightarrow CN$	$2z(r+1)$	–	$2z$
$CN \rightarrow CH$	$g(x+y+r(z+x)+x(k+2))$	$g(2x)$	$g(x+(h+r)(z+x)+z)$
$CH \rightarrow *$	–	$2x$	$x(k+m)$
$CH \rightarrow SN$	$r(k+3)x$	rx	rkx
$SN \rightarrow CH$	$2z$	$(d+2)x$	$2z$
$SN \rightarrow SN$	–	$(2b+3d)x$	–
$SN \rightarrow *$	–	$2x$	–

scheme is completely distributed in nature, i.e., one node manages not more than one key. The value of b also varies according to the size and density of the network. However, the values of parameters h and b are always such that they are comparable to the value of $k+m$. In Table 3, we use SN nodes to represent both SN and KG nodes of MUQAMI+ because other schemes do not have any KG nodes. Details of the storage analysis of LEAP+ and SHELL are presented in Appendix A.

We can decrease the key-chain length in MUQAMI+ at the cost of more computations and communications, but we cannot store more keys in LEAP+ to reduce the computation and communication costs. This proves our scheme to be more adaptable to the capabilities of nodes as compared to LEAP+. Also, storage requirements of SN nodes are very low in our scheme MUQAMI+ as compared to SHELL.

5.2 Communication and computation overhead

Since communication is the most energy-consuming activity, we will analyze and discuss it in more detail. While designing our scheme, we have tried to minimize and localize the communication as much as possible. This also helps in reducing the computation overhead because fewer message exchanges will result in fewer encryptions/decryptions. Now, we will try to compare our scheme with other schemes with respect to the number of messages exchanged between various types of nodes during different phases.

If we assume key length to be x bytes, EBS matrix size to be y bytes, and average length of other types of messages to be z bytes, then Table 4 shows the average number of messages transmitted by each type of node on each link during the initial deployment phase of each scheme. Here, g is the total number of CH nodes in the network and d is the number of nodes, which communicate with the CH node through a particular node. The exact value of d depends upon the network topology and varies from node to node even within a network. One important thing to note in Table 4 is that there is much less load on CH nodes in LEAP+. In LEAP+, SN nodes have to bear extra loads instead of the CH node in the initial deployment phase. LEAP+ is designed in such a way that if two nodes have to share a key, one of them broadcasts and the other one unicasts. We could have assumed that the CH node uses the broadcasts of SN nodes. However, that exercise would have only increased the load on the CH node without any significant effect on the load of SN nodes. The reason is that every SN node would have to broadcast with even more power so that its broadcast message could reach the CH node. New nodes are added in the same way as they are deployed initially in all the three schemes. So the comparison of new node addition would be similar to the one shown in Table 4.

Table 5 shows the average number of bytes transmitted by each type of node during the key refreshment phase. Communication keys are refreshed in the same way as the administrative keys except in LEAP+. In LEAP+, communication key refreshment is comparatively efficient for CH nodes. However, this efficiency comes at the cost of increased load on all SN nodes. The average number of messages broadcasted by an SN node for rekeying in our scheme can be expressed with the following formula:

$$Avg\_Msg\_Count\_Rekey_{SN\rightarrow *}^{MUQAMI+} = \frac{k+m}{r} \tag{7}$$

Table 5 Average number of bytes transmitted by each type of node on each link during key refreshment phase (an asterisk means a broadcast within a cluster)

	MUQAMI+	LEAP+	SHELL
$CH \rightarrow CH$	–	–	$x(h+k+m)$
$CN \rightarrow CH$	$(zg)+((2xg)/l)$	xg	$xg(h+1)$
$CH \rightarrow CN$	$(2z)/l$	–	–
$CH \rightarrow *$	–	–	$x(k+m)$
$CH \rightarrow SN$	$x(k+m)+((2z)/l)$	xr	–
$SN \rightarrow CH$	$(2z)/l$	$x(d+1)$	–
$SN \rightarrow SN$	–	$x(b+2d)$	–
$SN \rightarrow *$	$x((k+m)/r)$	–	–

Table 6 Average number of bytes transmitted by each type of node on each link in case the CH node of the cluster is compromised

	MUQAMI+	LEAP+	SHELL
$CH \rightarrow CH$	–	–	$x(h+(k+m)(2r+1))$
$CN \rightarrow CH$	$y+r(3x+z)$	$2x$	$(2xh)+r(2x+z)$
$CH \rightarrow CN$	$2z$	$2z$	$2z$
$CH \rightarrow *$	–	$2x$	$(k+m)x$
$CH \rightarrow SN$	$(r(3x+z))+x(k+m)+((2z)/l)$	rx	$r((k+3)x+z)$
$SN \rightarrow CH$	–	$(d+2)x$	–
$SN \rightarrow SN$	–	$(2b+3d)x$	–
$SN \rightarrow *$	–	–	–

All communications are within the cluster of the compromised CH except $CH \rightarrow CH$ communication (an asterisk means a broadcast within a cluster)

Apart from that, we have added an expression $1/l$ in the number of communications from $SN \rightarrow CH$, $CH \rightarrow SN$, and $CH \rightarrow CN$. This expression caters to the communications that are required to get a new seed value from the CH node.

If a CH node is compromised, the recovery procedure of our scheme is fairly straight forward as compared to other schemes. In Table 6, we have compared the communication overhead of our scheme with other schemes in case of CH node compromise.

If an SN node is compromised, we do not assume that a new SN node is deployed. However, in case of a KG node in MUQAMI+, we have to give the responsibility of key generation to some other node. In MUQAMI+, if a KG node is compromised, the CH sends one message to the CN and the CN replies with the compromised administrative key's new seed value. Apart from that, there is only one extra communication from the CH to the SN node, after which the SN node becomes a KG node. We assume that all nodes in a cluster have equal probabilities of being compromised. So, the average number of communications between CH and CN, in case of the compromise of the SN or the KG node, will be similar to Eq. 5, i.e., $(k+m)/r$. Similarly, the average number of communications from CH to SN node can be calculated with the following formula:

$$\begin{aligned} &Avg\_Msg\_Count\_SNCompr^{MUQAMI+}_{CH\rightarrow\{SN\cup KG\}} \\ &= \frac{((r-(k+m))(k+m))+(k+m)(k+m+1)}{r} \\ &= \frac{(k+m)(r-k-m+k+m+1)}{r} \\ &= \frac{(k+m)(r+1)}{r} \end{aligned} \tag{8}$$

Whereas the average number of bytes transferred from the CH to the SN node for the above task can be expressed with the following formula:

$$\begin{aligned} &Avg\_Byte\_Count\_SNCompr^{MUQAMI+}_{CH\rightarrow\{SN\cup KG\}} \\ &= \frac{((r-(k+m))(kz+mx))+(k+m)(kz+mx+2z+2x)}{r} \\ &= \frac{krz+mrz+2zk+2xk+2zm+2xm}{r} \\ &= \frac{2x(k+m)+z(r(k+m)+2(r+m))}{r} \end{aligned} \tag{9}$$

Table 7 Average number of bytes transmitted by each type of node on each link in a cluster in case a SN node is compromised in that cluster

	MUQAMI+	LEAP+	SHELL
$CH \rightarrow CH$	–	–	$x(h+k+m)+zk$
$CN \rightarrow CH$	$3(x+z)((k+m)/r)$	–	–
$CH \rightarrow CN$	$3z((k+m)/r)$	–	–
$CH \rightarrow *$	–	x	xm
$CH \rightarrow SN$	$(2x(k+m)+z(r(k+m)+2(r+m)))/r$	xr	–
$SN \rightarrow CH$	$x(k/r)$	$x(d+1)(b/r)$	–
$SN \rightarrow SN$	–	$x(b+2d)(b/r)$	–
$SN \rightarrow *$	$x(m/r)$	x	–

All communications are within the cluster of the compromised CH except $CH \rightarrow CH$ communication (an asterisk means a broadcast within a cluster)

In Table 7, we have compared the communication overhead of our scheme with other schemes in case of SN node compromise. Details of the communication and computation analysis of LEAP+ and SHELL are presented in Appendix B.

Note that, in all comparisons, there are no inter-cluster communications in our scheme as opposed to SHELL, and there are no unicast communications among SN nodes as opposed to LEAP+. There is communication between CH nodes and the CN node, but it is minimal and not very frequent. Due to these factors, our scheme has less communication and computation overhead than other schemes, which we will further establish in Section 6.

6 Simulation results

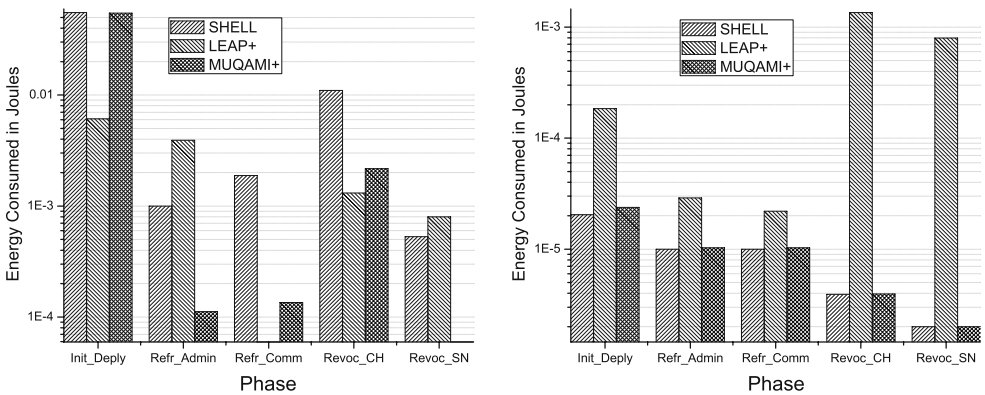
For simulation, our proposed network architecture is similar to the one shown in Fig. 1. We have compared our scheme with two other schemes, SHELL[13] and LEAP+[31], which are the most appropriate ones for WSNs proposed so far according to the best of our knowledge. In SHELL, CH nodes need to contact the neighboring CH nodes, so we have assumed a total of five clusters ($g = 5$) with 412 nodes in each cluster, i.e., $r = 412$. We have assumed $k = m = 6$, so that there are ample key combinations left for the addition of new nodes in the network. Also, we have assumed $b = 10$ and $d = 0.5$ on average. In case of SHELL, each CH node divides the EBS matrix into four equal parts and shares one part with each of the other CH nodes, i.e., $h = 4$. The neighboring CH nodes in turn manage keys for the part of EBS matrix shared with them. Simulation was programmed in "Tools Command Language (tcl8.0)," which is used to program ns-2 simulations.

G. Xing et al. [29] state that the range of data transmission from sensor nodes is between −20 and 10 dBm. We have assumed that the maximum distance between a CN and a CH or between two CH nodes is about ten times that of a cluster size. Moreover, the maximum cluster size is around ten times the maximum distance between two neighboring nodes. In order to record the power consumed during message exchanges, we have assumed three power levels for message transmission: one for communication outside the cluster, one for communication with a node inside the same cluster, and one for communication with a neighboring node. Also, we assume that the power levels are directly proportional to the distance of communication.

CH nodes transmit at 10 dBm (10 mW) to communicate outside the cluster and 0 dBm (1 mW) to communicate within its cluster. SN and KG nodes transmit at 0 dBm to communicate with the CH or to broadcast within the cluster. Communication with the neighboring node within the cluster is done at −10 dBm (0.1 mW). Karlof et al. [17] suggests that the application level bandwidth in WSN is around 19.2 kbps, and [29] suggests that it is around 6 kbps. We have assumed the application level bandwidth to be 19.2 kbps in our simulations. We also simulated keeping it at 6 kbps and found similar results.

The power level of a node during message reception and computation phase is assumed to be 0.1 mW. It is important to assume a power level for computation because we have also taken into account computation costs in our simulation. We have assumed that MICA2 motes are used, and they have ATMEGA128L CPU, as mentioned in [17]. Further, we have assumed that MD5 hashing scheme and IDEA cipher algorithm are used. For 16 bytes, MD5 takes 1.45 ms, encryption using IDEA takes 0.68 ms, and decryption using the same algorithm takes 2.42 ms on ATMEGA128L CPU according to [28].

We have assumed key size and key-chain seed size to be 16 bytes in our simulation. Seetharam and Rhee



(a) Cluster Head Node (b) Sensor Node

Fig. 4 Comparison of average energy consumed by different types of nodes in different phases of each scheme (**a**, **b**)

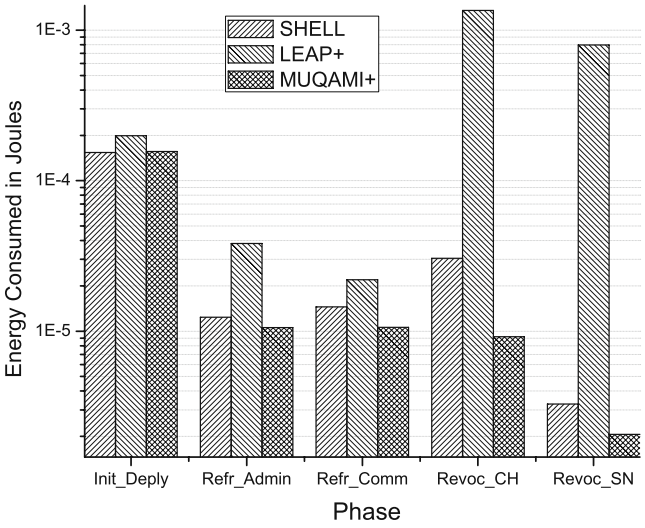


Fig. 5 Comparison of average energy consumed by a node in different phases of each scheme

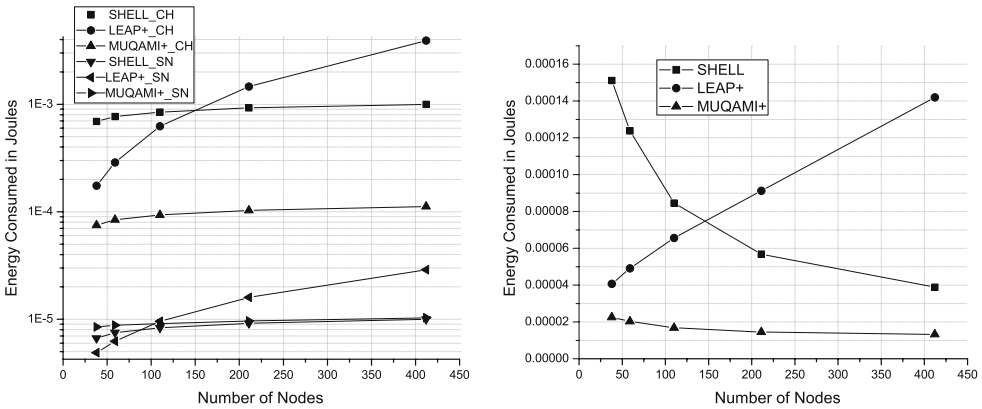
[25] states that an 8-MHz processor can generate 50,000 random bytes in 1 min. ATMEGA128L CPU also has a speed of 8 MHz. So, we have calculated the time to generate a key or seed value as 19.2 ms according to the calculations of [25]. Lastly, we have assumed the key-chain length l to be 32 in our simulations.

With the above set of simulation parameters, we recorded the average energy consumed by CH and SN nodes during these five phases: initial deployment, refreshment of administrative keys, refreshment of communication keys, revocation of a compromised CH node, and revocation of a compromised SN node. Over 70 iterations were carried out for every phase. In case of MUQAMI+, we took the weighted average of SN and KG nodes and recorded it as average energy consumed by SN nodes. Weights were set according to the number of number of KG nodes in a cluster, i.e., 12/412 in our case. We have plotted our graphs on logarithmic scale because of large differences in the readings recorded.

Figure 4a compares the average energy consumed by a CH node in each of the three schemes in all five phases. Apart from the initial deployment phase, in which SHELL and MUQAMI+ are comparable, MUQAMI+ outperforms SHELL in all other episodes by a comprehensive margin. MUQAMI+ outperforms LEAP+ in the refreshment phase of the admin key and revocation of a compromised SN node, but the CH node in LEAP+ consumes less energy than the CH node in our scheme if we consider the phases of initial deployment, refreshment of communication key, and revocation of a compromised CH node. However, it comes at the cost of additional burden on the other SN nodes in those three phases, as is evident in Fig. 4b. Also, note that there is no additional burden on SN nodes of our scheme as compared to SHELL, whose CH nodes consume more energy as compared to our CH nodes.

We believe that the role of being a CH node can be transferred from one node to another from time to time. Also, the CH node of LEAP+ consumes less energy than that of our scheme, and the SN node of our scheme consumes less energy than that of LEAP+. So we find it necessary to compare the average energy consumed by a node considering all types of nodes in the network. Figure 5 compares the average energy consumed by a node considering all types of nodes, i.e., CH nodes and SN nodes together. Except for the initial deployment phase, our scheme outperforms both of the other schemes. MUQAMI+ outperforms LEAP+ due to the use of combinatorics, and it outperforms SHELL due to the local distribution of key management responsibilities. Also, there is no single point of failure in our scheme.

Fig. 6 Comparison of average energy consumed in administrative key refreshment phase of each scheme with respect to the number of nodes in the network (**a**, **b**)



(a) Plotting consumption of CH and SN nodes separately

(b) Plotting consumption of CH and SN nodes collectively (Average)

Finally, we need to show the most significant improvement of our scheme and its scalability. For that purpose, we changed the number of nodes in our simulation code and then recorded the readings. Since the management of the administrative key is most critical in key management, we have plotted, in Fig. 6a, trends of average energy consumed in the phase of administrative key refreshment by both type of nodes considering each of the three schemes while varying the number of nodes in the network. Figure 6b shows the average of both types of node. It is clear that our scheme is scalable, as well as more efficient, than the other two schemes. It is important to note the convergence of SHELL with MUQAMI+ in Fig. 6b. The improvement of our scheme over SHELL is in the energy consumed by the CH node. With increased network size, energy consumed by CH nodes averages with larger numbers of nodes. For LEAP+, we assume that the network density increases with the number of nodes. A similar trend is followed by all the schemes in communication key refreshment and node revocation phases.

7 Conclusion and future work

We have presented the MUQAMI+ key management scheme, which uses EBS matrix to manage large numbers of nodes with small numbers of administrative keys. In MUQAMI+, the load of key management is distributed within clusters. We have compared MUQAMI+ with two other state-of-the-art schemes and found that our scheme is more efficient in key refreshment and node revocation phases. Also, our scheme is scalable and flexible, as we can transfer the responsibility of being cluster head or key generator from node to node with the passage of time.

Apart from defence from attacks, it is also important to focus on the detection of attacks in the network. A future direction of this research is to devise a scheme for attack detection in WSN and then couple it with the attack prevention and mitigation schemes, proposed in this paper, to form a complete lightweight security solution for WSN.

Acknowledgements This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement) (IITA-2009-(C1090-0902-0002)). This work also, was supported by the Korea Science & Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. 2008-1342), and was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2009-0076798). This work is supported by the IT R&D program of MKE/KEIT, [10032105, Development of Realistic Multiverse Game Engine Technology].

Appendices

Appendix A. Storage overhead

The average storage requirements of a CH node in LEAP+ are fairly straightforward. Apart from the pair-wise key shared with each node in the cluster, it has to store two more keys, i.e., its cluster key and the communication key. So, if there are r nodes in a cluster, storage requirements of a CH node in LEAP+ turn out to be:

$$SR_{CH}^{LEAP+} = r + 2 \tag{10}$$

In LEAP+, the SN nodes only establish pair-wise keys with only their b neighbors. However, they have to store two keys, i.e., the cluster key and the communication key. So, the storage requirement of SN node in LEAP+ becomes:

$$SR_{SN}^{LEAP+} = b + 2 \tag{11}$$

Now, we will discuss the storage requirement of a CH node in SHELL. In SHELL, each node has to store a key chain of length l for the key it shares with the CN. Also, it has to store pair-wise keys with h neighboring CH nodes, with whom it shares the EBS matrix. Also, it has to store pair-wise keys with the r nodes, i.e., the average number of nodes in a cluster. Finally, it has to store the $k+m$ administrative keys and the communication key. So, the storage requirement of a CH node in SHELL can be written as

$$SR_{CH}^{SHELL} = l + r + h + k + m + 1 \tag{12}$$

Apart from the key chain of length l for the key shared with the CN and the k administrative keys, SN nodes have to store three other keys, i.e., one pair-wise key shared with a neighboring CH node, one shared with its own CH node, and one communication key. So, the average storage requirements of a SN node in SHELL can be written as:

$$SR_{SN}^{SHELL} = l + k + 3 \tag{13}$$

Appendix B. Communication and computation overhead

In this section, we will discuss the communication and computation overhead of LEAP+ and SHELL and explain the formulas listed in Tables 4 to 7. In the initial deployment phase, every CH node receives one

message each from h neighboring CH nodes to establish the communication path. Also, for each node in its cluster, the CH node receives one message from one of the neighboring CH nodes in order to establish pair-wise keys. Finally, for the distribution of the communication key, every CH node receives the communication keys $k+m$ times, i.e., encrypted separately in each administrative key. So, the average message exchanges between CH nodes during the initial deployment phase of SHELL comes out to be:

$$Avg\_Msg\_Count\_Init^{SHELL}_{CH\rightarrow CH} = h + r + k + m, \quad (14)$$

where $Avg\_Msg\_Count\_Init^{SHELL}_{CH\rightarrow CH}$ is the average message exchanges between CH nodes during initial deployment phase of SHELL.

In the initial deployment phase of LEAP+, every SN node sends two unicast messages to the CH node. One message establishes a pair-wise key with the CH node and the other one transfers the cluster key to the CH node. Apart from that, it also has to forward d messages to the CH node, where d is the average number of nodes that establish their pair-wise keys with the CH node through each node. So, the average number of messages transmitted from a SN to its CH in the initial deployment phase turns out to be:

$$Avg\_Msg\_Count\_Init^{LEAP+}_{SN\rightarrow CH} = d + 2 \quad (15)$$

where $Avg\_Msg\_Count\_Init^{LEAP+}_{SN\rightarrow CH}$ is the average number of messages transmitted from a SN to its CH in the initial deployment phase of LEAP+. Also, it is important here to mention the average number of message exchanges between SN nodes in LEAP+. There are b neighbors of each node and each node has to send two messages to each of its neighbors. One message is for the pair-wise key establishment and the other message is for the communication of its cluster key. Apart from that, three messages are exchanged for every node that establishes a pair-wise key with the CH node through this node, i.e., three messages are exchanged for d nodes from each node on average. In one message, the d nodes send a message to the CH node through this node to establish a pair-wise key with the CH node. In the other two messages, cluster keys are exchanged between the CH and the d nodes, i.e., CH and the d nodes send their cluster keys to each other through this node. So, the average number of message exchanges between SN nodes in LEAP+ for the initial deployment phase comes out to be:

$$Avg\_Msg\_Count\_Init^{LEAP+}_{SN\rightarrow SN} = 2b + 3d, \quad (16)$$

where $Avg\_Msg\_Count\_Init^{LEAP+}_{SN\rightarrow SN}$ is the average number of messages exchanged between SN nodes in the initial deployment phase of LEAP+. It is not difficult to understand the rest of the expressions in Table 4.

For key refreshment in SHELL, every CH sends a message to h neighboring CH nodes. In turn, it receives $k+m$ messages from them to broadcast in the cluster. So, the average number of messages exchanged between the CH nodes in the key refreshment phase can be written as:

$$Avg\_Msg\_Count\_Rekey^{SHELL}_{CH\rightarrow CH} = h + k + m, \quad (17)$$

where $Avg\_Msg\_Count\_Rekey^{SHELL}_{CH\rightarrow CH}$ is the average message exchanges between CH nodes during the key refreshment phase of SHELL. Also, the keys between the neighboring CH nodes and the keys between CH nodes and the CN will also be refreshed. So, in order to refresh those keys, the CN will send $h+1$ messages to all the CH nodes, i.e., to g CH nodes. h messages contain new keys for communication with the neighboring CH nodes and one message contains a key for communication with the command node. So, the total number of messages transferred from CN to the CH nodes in key refreshment phase of SHELL comes out to be:

$$Msg\_Count\_Rekey^{SHELL}_{CN\rightarrow CH} = g \times (h + 1) \quad (18)$$

where $Msg\_Count\_Rekey^{SHELL}_{CN\rightarrow CH}$ is the total number of messages transferred from CN to the CH nodes in key refreshment phase of SHELL.

In order to refresh its cluster key, each node will send one message to the CH node. Also, it will forward one message each from d nodes, so the average number of messages transmitted from a SN to its CH in the key refreshment phase turns out to be:

$$Avg\_Msg\_Count\_Rekey^{LEAP+}_{SN\rightarrow CH} = d + 1 \quad (19)$$

where $Avg\_Msg\_Count\_Rekey^{LEAP+}_{SN\rightarrow CH}$ is the average number of messages transmitted from a SN to its CH in the key refreshment phase of LEAP+. In the same way, when a SN node refreshes its cluster key in LEAP+, it send one message each to its b neighbors. Also, the CH node exchange cluster keys with d nodes through every node on average. On a particular SN node, one message is received from each of the d nodes and forwarded to the CH node and one message from the CH node is forwarded to them. So, the average number of message exchanges between SN nodes in LEAP+ for the key refreshment phase comes out to be:

$$Avg\_Msg\_Count\_Rekey^{LEAP+}_{SN\rightarrow SN} = b + 2d \quad (20)$$

where $Avg\_Msg\_Count\_Rekey^{LEAP+}_{SN\rightarrow SN}$ is the average number of messages exchanged between SN nodes in the key refreshment phase of LEAP+. The rest of the

expressions in Table 5 are trivial and easily understandable. Expressions in Table 6 are similar to the ones in Table 4 except that $h+1$ messages are sent from the CN to the CH nodes in case a CH node is compromised. One message is sent to the new CH node and h messages are sent to its neighboring CH nodes, so that keys can be established between the new CH node and its neighboring CH nodes. Explanations for the rest of the expressions in Table 6 are similar to the ones in Table 4.

In case a SN node is compromised, k compromised keys are refreshed using m remaining keys that the compromised SN node does not know. So, the number of broadcast messages in the cluster by the CH node is m. However, the CH has to send k messages to the neighboring CH nodes, which manage the k compromised keys. k messages will be returned to the CH node with the new key values encrypted in the old ones. Then, those k keys will be aggregated and sent to the h neighboring CH nodes, so that they can encrypt the aggregated message using them using all keys that they manage other than the k compromised keys. In turn, the CH will receive m messages, which it will broadcast in its cluster. So, the average number of messages exchanged between the CH nodes in case of SN node revocation in SHELL can be written as:

$$Avg\_Msg\_Count\_Revoc\_SN_{CH\rightarrow CH}^{SHELL} = h + 2k + m \tag{21}$$

where $Avg\_Msg\_Count\_Revoc\_SN_{CH\rightarrow CH}^{SHELL}$ is the average message exchanges between CH nodes when a SN node is compromised in SHELL. In LEAP+, if a SN node is compromised, only its neighbors perform the $SN \rightarrow CH$ and $SN \rightarrow SN$ communication. So, in order to calculate the average message count, we divide the expressions for the count of both such messages by b/r. The rest of the expressions in Table 7 do not require much elaboration.

References

1. Younis O, Fahmy S (2004) Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. IEEE Trans Mob Comput 3(4):366–379. doi:10.1109/TMC.2004.41
2. Youssef A, Agrawala A, Younis M (2005) Accurate anchor-free localization in wireless sensor networks. In: First IEEE workshop information assurance in wireless sensor networks (WSNIA '05)
3. Akyildiz I, Su W, Sankarasubramaniam Y, Cayirci E (2002) Wireless sensor networks: a survey. Comput Networks 38(4):393–422
4. Amin S, Siddiqui M, Hong C (2008) Detecting jamming attacks in ubiquitous sensor networks. In: IEEE sensors applications symposium 2008, pp 40–45
5. Çamtepe SA, Yener B (2007) Combinatorial design of key distribution mechanisms for wireless sensor networks. IEEE/ACM Trans Netw 15(2):346–358. doi:10.1109/TNET.2007.892879
6. Chan H, Perrig A, Song D (2003) Random key predistribution schemes for sensor networks. In: SP '03: proceedings of the 2003 IEEE symposium on security and privacy. IEEE Computer Society, Washington, DC, p 197
7. Dierks T, Allen C (1999) The TLS protocol version 1.0
8. Dini G, Savino IM (2006) An efficient key revocation protocol for wireless sensor networks. In: WOWMOM '06: proceedings of the 2006 international symposium on on world of wireless, mobile and multimedia networks. IEEE Computer Society, Washington, DC, pp 450–452. doi:10.1109/WOWMOM.2006.23
9. Du W, Deng J, Han YS, Varshney PK (2003) A pairwise key pre-distribution scheme for wireless sensor networks. In: CCS '03: proceedings of the 10th ACM conference on computer and communications security. ACM, New York, pp 42–51. doi:10.1145/948109.948118
10. Eltoweissy M, Heydari MH, Morales L, Sudborough IH (2004) Combinatorial optimization of group key management. J Netw Syst Manage 12(1):33–50. doi:10.1023/B:JONS.0000015697.38671.ec
11. Eltoweissy M, Moharrum M, Mukkamala R (2006) Dynamic key management in sensor networks. IEEE Commun Mag 44(4):122–130. doi:10.1109/MCOM.2006.1632659
12. Eschenauer L, Gligor VD (2002) A key-management scheme for distributed sensor networks. In: CCS '02: proceedings of the 9th ACM conference on computer and communications security. ACM, New York, pp 41–47. doi:10.1145/586110.586117
13. Ghumman K (2006) Location-aware combinatorial key management scheme for clustered sensor networks. IEEE Trans Parallel Distrib Syst 17(8):865–882. doi:10.1109/TPDS.2006.106 (Senior Member-Mohamed F. Younis and Senior Member-Mohamed Eltoweissy)
14. Gupta G, Younis M (2003) Load-balanced clustering of wireless sensor networks. In: International conference on communications (ICC '03), pp 1848–1852
15. Intanagonwiwat C, Govindan R, Estrin D (2000) Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: ACM mobile computing and networking (Mobicom'00), pp 56–67
16. Karlof C, Li Y, Polastre J (2003) Arrive: an architecture for robust routing in volatile environments. Tech. Rep. CSD-03-1233, University of California at Berkeley
17. Karlof C, Sastry N, Wagner D (2004) TinySec: a link layer security architecture for wireless sensor networks. In: SenSys '04: proceedings of the 2nd international conference on embedded networked sensor systems. ACM, New York, pp 162–175. doi:10.1145/1031495.1031515
18. Kohl J, Neuman C (1993) The Kerberos network authentication service (v5)
19. Lamport L (1981) Password authentication with insecure communication. Commun ACM 24(11):770–772. doi:10.1145/358790.358797
20. Langendoen K, Reijers N (2003) Distributed localization in wireless sensor networks: a quantitative comparison. Comput Networks 43(4):499–518
21. Li G, He J, Fu Y (2006) A hexagon-based key predistribution scheme in sensor networks. In: ICPPW '06: proceed-

ings of the 2006 international conference workshops on parallel processing. IEEE Computer Society, Washington, DC, pp 175–180. doi:10.1109/ICPPW.2006.9

22. Madden S, Szewczyk R, Franklin MJ, Culler D (2002) Supporting aggregate queries over ad-hoc wireless sensor networks. In: WMCSA '02: proceedings of the fourth IEEE workshop on mobile computing systems and applications. IEEE Computer Society, Washington, DC, p 49
23. Menezes AJ, Vanstone SA, Oorschot PCV (1996) Handbook of applied cryptography. CRC, Boca Raton
24. Paek KJ, Kim J, Hwang CS, Song US (2007) An energy-efficient key management protocol for large-scale wireless sensor networks. In: MUE '07: proceedings of the 2007 international conference on multimedia and ubiquitous engineering. IEEE Computer Society, Washington, DC, pp 201–206. doi:10.1109/MUE.2007.74
25. Seetharam D, Rhee S (2004) An efficient pseudo random number generator for low-power sensor networks. In: LCN '04: proceedings of the 29th annual IEEE international conference on local computer networks. IEEE Computer Society, Washington, DC, pp 560–562. doi:10.1109/LCN.2004.18
26. Shaikh R, Lee S, Khan M, Song Y (2006) LSec: lightweight security protocol for distributed wireless sensor networks. In: 11th IFIP international conference on personal wireless communications PWC'06. LNCS, vol 4217, Spain, pp 367–377
27. Tilak S, Abu-Ghazaleh N, Heinzelman W (2002) A taxonomy of wireless microsensor network models. ACM Mobile Comput Commun 6(2):1–8
28. Venugopalan R, Ganesan P, Peddabachagari P, Dean A, Mueller F, Sichitiu M (2003) Encryption overhead in embedded systems and sensor network nodes: modeling and analysis. In: CASES '03: proceedings of the 2003 international conference on compilers, architecture and synthesis for embedded systems. ACM, New York, pp 188–197. doi:10.1145/951710.951737
29. Xing G, Lu C, Zhang Y, Huang Q, Pless R (2005) Minimum power configuration in wireless sensor networks. In: MobiHoc '05: proceedings of the 6th ACM international symposium on mobile ad hoc networking and computing. ACM, New York, pp 390–401. doi:10.1145/1062689.1062738
30. Xu W, Ma K, Trappe W, Zhang Y (2006) Jamming sensor networks: attack and defense strategies. IEEE Netw 20(3): 41–47. doi:10.1109/MNET.2006.1637931
31. Zhu S, Setia S, Jajodia S (2006) LEAP+: efficient security mechanisms for large-scale distributed sensor networks. ACM Trans Sen Netw 2(4):500–528. doi:10.1145/1218556.1218559