MUQAMI: A Locally Distributed Key Management Scheme for Clustered Sensor Networks

[1]Syed Muhammad Khaliq-ur-Rahman Raazi, [1]Adil Mehmood Khan, [2] Faraz Idris Khan, [1]Sung Young Lee,[3]Young Jae Song, [1]Young Koo Lee†
[1]Ubiquitous Computing Lab, Department of Computer Engineering, Kyung Hee University, 449-701 Suwon, South Korea {raazi, adil, sylee, yklee}@oslab.khu.ac.kr
[2]Internet Computing and Security Lab, Department of Computer Engineering, Kyung Hee University, 449-701 Suwon, South Korea {faraz}@khu.ac.kr
[3]Software Engineering Lab, Department of Computer Engineering, Kyung Hee University, 449-701 Suwon, South Korea, {yjsong}@khu.ac.kr

Abstract. In many of the sensor network applications like natural habitat monitoring and international border monitoring, sensor networks are deployed in areas, where there is a high possibility of node capture and network level attacks. Specifically in such applications, the sensor nodes are severely limited in resources. We propose MUQAMI, a locally distributed key management scheme for resilience against the node capture in wireless sensor networks. Our scheme is efficient both in case of keying, re-keying and node compromise. Beauty of our scheme is that it requires minimal message transmission outside the cluster. We base our Scheme on Exclusion Basis System (EBS).

1 Introduction

Wireless Sensor Networks (WSN) differs from other distributed systems in a way that they have to work in real-time with an added constraint of energy. They are mostly data centric and are used to monitors their surroundings, gather information and filter it [1]. A sensor network will typically consist of a large number of sensor nodes

† Corresponding author.

working together for collection of data in a central node, using wireless communications [2].

WSN should also be cost effective, as their energy may not be replenished. This limits their memory and computational power also. In effect, resource conscious techniques should be employed in the WSNs. Sensor networks can be deployed in different areas for surveillance activities. WSNs are required to work unattended. Adversary may attack externally i.e. capture the node or jam the traffic signals. Internal attacks such as collusion can be made through loopholes in protocols. In addition to the energy constraint, WSNs have dynamic topologies. Due to these constraints, traditional security techniques can not be applied to WSNs.

Sensor nodes work collectively to perform a common task. Group communications are performed for efficient operations. In this case, groups of nodes share common keys. If a node is compromised and acts abnormally, it should be evicted from the group. In case of node compromise, re-keying must be done. During the re-keying process all the communication and administrative keys, known to the compromised node, should be revoked. After re-keying, the group should be in such a state that the compromised node is not part of the group i.e. it can’t infer anything from the communication going on within the group.

In this paper we present MUQAMI, a lightweight and locally distributed key management scheme for clustered sensor networks. MUQAMI is based on Exclusion Basis System (EBS) matrix [3] and key-chains [4], which is an authentication mechanism based on Lamport’s one-time passwords [5]. Our scheme is an improvement on SHELL [6], which is a hierarchical, distributed and collusion resistant key management scheme for WSN. In addition to the advantages offered by SHELL, our scheme offers lesser communication and computation overhead. Also, it is more resilient and scalable as compared to SHELL.

This paper is organized as follows. Section 2 describes models and assumptions of our system. Section 3 outlines related work and relevant schemes, which will help in understanding our solution. In section 4, we will present our scheme. In section 5, we will analyze our scheme and compare it with SHELL. We’ll conclude our discussion in Section 6.

2 Models and Assumptions

2.1 System Model

WSN consist of a command node connected to a number of sensor nodes, which can be grouped into clusters. In case of clusters, there is a cluster head, which aggregates information from other sensor nodes and sends it back to the command node. Cluster head is also called a gateway. We will use both the terms interchangeably. Clustering of nodes can be based upon their location and other criteria [7] [8]. We are assuming clustered sensor networks in our scheme. Sensor nodes sense their environment and relay information to their cluster heads. Sensor nodes relay their messages directly or indirectly, depending upon their reach [9] [10].

We are assuming that all nodes, including the cluster heads, are stationary. Communication range and physical locations of all nodes are known to the nodes at the higher layer. We assume that sensors can only communicate within their clusters. Their processing and memory capabilities are also very limited. The cluster heads are designed with more energy capabilities as compared to lowest level sensor nodes. Cluster heads have to communicate with the command node, which can be situated at a larger distance.

Data aggregation is carried out at the clustered heads and not low level sensor nodes, due to their limited processing capabilities. Data aggregation at cluster heads considerably reduces size of messages, which need to be transmitted to the command node. This hierarchy can be extended to any number of levels depending upon the requirements. Command node is even more resource rich as compared to the cluster heads.

Higher we go in a hierarchy, higher is the energy consumed in transmitting a message. Due to this reason, we were motivated to delegate the message exchange for security as low as possible in the hierarchy of sensor nodes. Delegating message to lower levels is not as trivial. Special care should be taken not to overload sensor nodes in this process.

2.2 Adversity Assumptions

We assume that an adversary can capture a node and use its memory in any way possible. The sensor nodes are not assumed to be tamper resistant. We also assume that initially, the adversary does not have any knowledge about the contents of nodes' memory or messages, being exchanged between nodes. Another assumption is that higher we go in hierarchy, difficult it gets for an adversary to capture a node. Moreover, command node can not be compromised and every node has a unique identifier.

Moreover, we assume that a compromised node does not have any information about any other compromised nodes except its neighbours. Our last assumption is that compromised nodes can not communicate through any external communication channel. No assumptions are made on trust and collusion. 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, adversary can even capture a node and read or edit its memory.

3 Relevant schemes

In this section, we will briefly describe Exclusion Basis System (EBS) and Key-chains, which are the basic building blocks of MUQAMI. We will also briefly explain SHELL as our scheme is an improvement over SHELL.

3.1 Exclusion Basis System (EBS)

EBS was developed by Eltoweissy et. al[3], in which they used combinatorial optimization for key management. EBS is found to be very scalable for large networks. Basically, EBS plays between two variables ‘k’ and ‘m’. To support a set of ‘N’ nodes, a set of “k+m” keys are required in EBS. Out of the total of “k+m” keys, each node knows ‘k’ keys. No two nodes should know the same set of ‘k’ keys. Any new node can be added if a distinct set of ‘k’ keys is still available. Values of ‘k’ and ‘m’ can be adjusted according to the network requirements. In order to evict a compromised node, new keys are distributed using ‘m’ keys that the node does not know. Clearly, communication overhead increases with the value of ‘m’.

EBS scheme is very susceptible to collusion attacks. Younis et. al[6] has devised a scheme, which tends to mollify collusion attacks on EBS-based key management schemes. Details of EBS scheme can be found in [3].

Table 1: Example of an EBS Matrix

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

3.2 Key-chains

G. Dini et al [4] uses key-chains, whose authentication mechanisms are based on one-way hash functions. These one-way hash functions are light weight and several orders of magnitude more efficient than RSA [11], an example of Public Key Cryptography.

One-way hash function [12] uses a key to compute its previous one. We can’t use the current key to compute the next one. If we give the end key and the start key, sensor node can iteratively apply the one-way hash function to find the intermediate keys starting from the end key. We can adjust the number of keys that the sensor node produces before it needs to be given the new set of start and end keys. The number of keys produced by a sensor node in one such episode can be optimized. Optimum value will depend upon the node’s power and memory capabilities.

3.3 SHELL

In SHELL, each sensor node has a discovery key K_{sg} and two preloaded keys KS_{CH} and KS_{Key} initially. K_{sg} is recomputed with one-way hashing function, such as SHA1 [13] or MD5 [14], stored in the node. The one-way hashing function is only known to the sensor node and the command node. K_{sg} helps later on, when the network needs to be recovered after gateway compromise. KS_{CH} and KS_{Key} are used for initial key distribution.

Apart from K_{sg} of every node in its cluster, the gateway also has a preloaded key K_{gc}. K_{gc} is used for communication between the gateway and the command node. Gateways can also communicate between themselves. In SHELL, gateway is responsible for the following: -

- Formation of EBS matrix and generation of communication keys of its own cluster. Also, it is responsible for refreshment of its cluster's data keys.
- On request generation of administrative keys of other clusters.
- It is also responsible for detection and eviction of compromised nodes within its cluster.

Whenever a node is deployed initially or afterwards, it is authenticated by the command node. Inter-gateway communication keys are also generated and renewed by the command node.

Key Distribution: In the key distribution phase, gateways form their EBS matrices first. Each EBS matrix, along with the list of sensors in that cluster, is shared between the gateways and the command node. Command node designates more than one cluster for each cluster to generate administrative keys. The gateway or the cluster head shares its EBS matrix with the gateways designated by the command node.

Command node shares the key KS_{CH} of each sensor node with its cluster head. It also shares their keys KS_{Key} with the neighbouring cluster heads i.e. the one's who are supposed to generate their administrative keys. For key distribution, each neighbouring gateway generates one message per individual administrative key in its cluster for each sensor node. The message is first encrypted with the KS_{Key} of the node and then the administrative key of the sensor node's gateway. Gateway decrypts the message, encrypts it with KS_{CH} of the node and sends it to the sensor node.

Cluster heads/gateways share their communication keys in a similar fashion. They generate communication keys and send them to their neighbouring cluster heads. Neighbouring clusters then send them to sensor nodes in the way described above.

For addition of new sensor nodes, command node informs the relevant gateways about their IDs and keys K_{sg}. Gateways register the new nodes and authenticate them from the command node. Command nodes then provide the gateways with keys KS_{CH} and KS_{Key} of the new nodes. Rest of the procedure is same as in the initial key distribution phase.

Failure Recovery: If a gateway is compromised, it is either replaced or its sensors are redistributed. If replaced, the command node establishes new keys for communication between existing gateways and the newly deployed one. Then the new gateway makes a new EBS matrix, repeats initialization process and generates administrative keys for other clusters. The other option is that sensor nodes join other clusters with the help of new K_{sg} that they generated after initial deployment.

If a sensor node is compromised, keys known to the compromised node must be changed so that the compromised node doesn't get to know the new key. For this purpose, we encrypt new keys with their older versions and then again encrypt them in a key that is not known to the compromised node. This way, the compromised node can not decrypt the message. Assumptions of SHELL scheme and its complete details can be found in [6].

4 MUQAMI

In our scheme also, the command node stores the database of all node IDs as in SHELL. All sensor nodes have a basic key K_{bsc} along with a one-way hashing function. The one-way hashing function is used to compute new values of K_{bsc}. We assume that the nodes have enough memory to store the keys required for their normal operation. K_{bsc} is used for communication between the sensor and the command node in case the gateway or a key-generating node is compromised.

Apart from K_{bsc}, each sensor also has another set of keys $K_{kgs}[n]$ for administrative purposes. $K_{kgs}[n]$ are used by key-generating nodes to communicate with the sensor nodes. Some of the sensor nodes are given an additional responsibility of generating a key. We refer to such nodes as key-generating nodes. Key-generating nodes use lightweight one-way hashing functions to generate keys. One-way hashing functions are as described in [4]. As required by the one-way hashing function, the key-generating nodes also need to store the hashing function and the keys it generate in a chain. Key-generating nodes store one other key for administrative purposes K_{chs}, which is used by the cluster head to communicate with the key-generating node.

The same scheme is not employed in the upper hierarchy i.e. at the level of gateways. Analogy of a gateway and a sensor can not be applied to the command node and a gateway. The reason is that we assume the command node to have unlimited energy unlike gateways. In effect we think that we should move as much responsibility to the command node as possible. If we have more than three-level hierarchy, we can use the same scheme at every level, except the top most.

Initially, the gateway has its K_{bsc} stored in it. For key management, communication between gateways is not necessary in our scheme. Command node authenticates all the nodes, which are initially deployed or added to the network later on.

4.1 Network Initialization

Gateways are deployed in the beginning. They communicate with the command node with the help of their basic keys K_{bsc}. The command node then sends the gateway a list of sensor IDs along with their basic keys K_{bsc}. These are the sensors that are to be deployed within the cluster of this gateway. We assume that the sensors are deployed carefully under their desired cluster heads.

In the second phase, sensors are deployed in the desired regions. Sensor nodes try to communicate with their cluster heads with the help of their basic Keys K_{bsc}. On getting the initial messages from sensor nodes, cluster heads authenticate them from the command node. Command node authenticates valid sensors to their cluster heads and also informs which one will also generate keys in their cluster. In addition to that, the command node communicates the K_{chs} of nodes to their cluster head. In case of simple sensors i.e. those that are not key-generating nodes, command node sends list of IDs of key-generators that are responsible for generating its keys. After authentication, the cluster heads form EBS matrices for key management. EBS matrices are shared between the cluster heads and the command node.

4.2 Initial Key Distribution

In the first phase, EBS matrices are formed in gateways and the command node. We know that in EBS matrix, values of ‘k’ and ‘m’ are to be decided, which depends upon factors like storage requirements, number of re-keying messages, size of the cluster and frequency with which the sensor nodes are to be evicted. It is evident that in our case, command node predominantly decides the whole EBS matrix and communicates with the gateways during the authentication phase.

We propose a little change in the representation of EBS matrix. Usually, we use ‘0’ if a node does not know a key and ‘1’ if a node knows a key. In this case, we make an addition. We use a ‘2’ if a node generates a key. For our case, example shown in table 1 can be modified as shown in table 2.

Table 2: Updated EBS matirx example for MUQAMI scheme

	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

Note that after the formation of EBS matrix in the gateway and the command node, initial keys of the EBS matrix are already distributed among the sensor nodes. Moreover, also the gateway does not have any idea about the administrative keys being used in the sensor nodes just as in SHELL. In order to send its communication key to the cluster, the cluster head encrypts it with keys K_{chs} of all key-generating nodes and send it to all of them. In turn, the key-generating nodes decrypt, then encrypt with their generated keys and broadcast the encrypted communication key in their cluster.

4.3 Node addition and Re-keying

Re-keying is not very complicated in our scheme. If the cluster head wants to refresh its communication key, it encrypts in keys K_{chs} and sends the new communication key to the key-generating nodes. The key-generating nodes then broadcast the new communication key in the cluster using their generated keys. In order to refresh the administrative keys, the cluster head just needs to send a very short message to the key-generator, whose key needs to be refreshed. In turn, the key-generator sends new key encrypted in the old one. The short message need not be encrypted as it does not contain any key. Even if an adversary comes to know that a key is being refreshed, it will not be able to do anything as it wouldn't know any of the new and old keys. We know that the key-generator uses one-way hashing function. If the key-generating node is running out of keys it has generated and stored, it communicates with the command through cluster head using its K_{bsc}. Command node sends it a new seed for calculating the key.

Sometimes, it may be necessary to add new nodes into the cluster. For this purpose, we should have an EBS matrix such that there is a key combination available for the new node. The command node possesses a copy of EBS matrix. The new node can be a simple node or it can be a node with key-generating capability. A group of new nodes including both can also be added.

First of all, adjustments in the EBS matrices are made in the command node. Then relevant gateways are sent IDs and K_{bsc} of the nodes, which it should expect in its cluster. The gateway halts re-keying and for every key, tells the command node how many times it has been changed. Command node calculates the current keys relevant to the new sensors and encrypts them with the second key in their respective key rings. Then the new keys are sent to the cluster head, which stores until the new node is authenticated and registered completely.

In case of simple sensor nodes i.e. without key-generating capability, the sensor node communicates with its cluster head. Cluster head authenticates the new node from the command node. Command node authenticates and specifies which nodes should generate key for it, so that the gateway can also update its EBS matrix. Then the cluster head forwards current administrative keys to the new sensor node. In the end, the network returns to its normal state. The command node also sends its key K_{chs} to the cluster head. As already stated, cluster head uses K_{chs} keys to spread its communication key in the cluster.

4.4 Node compromise

We assume that the system has enough capability to find out the compromised node. In this subsection, we describe how to evict compromised node from the network. We need to keep all further communications secret from the compromised nodes such that they can only act as an outsider when trying to interfere in the network's normal operation. There are three types of nodes, so we will discuss each of them one by one.

Cluster head compromise: Command node is responsible for detecting the compromise of gateways. There are three methods to cater for the compromise of gateways. First option is that we deploy a replacement. Second one is that we redistribute nodes among neighbouring clusters. These two methods are discussed in detail in [6].

If our scheme is applied in the network, command node can designate another nearby gateway as a caretaker in case of gateway compromise. We assume that this is possible because we have taken off the burden of administrative key generation from cluster heads. In this case, command node evicts the compromised gateway and communicates ID of the caretaker to the nodes in the compromised gateway's cluster. Then the command node assigns a new K_{chs} to the key-generating nodes in the cluster of compromised gateway. This is done with the help of K_{bsc} of the nodes. Then the command node provides the cluster's EBS matrix and all K_{chs} keys, to the caretaker node. Network continues its normal operation until new gateways are deployed. In case there is no nearby cluster head, or it does not have enough resources to manage two clusters, we can use other methods.

Sensor Compromise: In case of sensor node compromise, all the keys that are known to the compromised sensor node must be changed. Cluster head informs all the key-generators, whose keys need to be changed, to change their keys, encrypt them using previous ones and then further encrypt in their respective K_{chs} and send them back to the cluster head. Cluster head is not able to find out the new keys as it does not know the old ones. Also the compromised node is not able to find out the new key as it does not know the K_{chs} keys of the key-generating gateways. All this is possible because in our scheme, the same key is used both for encryption and decryption.

After receiving all the keys, which were demanded from key-generating nodes, the gateway aggregates them into one message. From the EBS matrix, it finds out the key-generating nodes, whose keys are required for spreading the new keys. It then sends this aggregated message to each of those key-generating nodes using their respective K_{chs}. Upon receiving the aggregated message, the key-generator node decrypts the message and then encrypts it again with the key it generates. Eventually, the message is forwarded to all the cluster in such a way that only the relevant node knows about the new key.

As an example, consider the scenario in table 2. Suppose the node N_2 is compromised. Cluster head will ask the nodes N_0, N_1 and N_4 to send the new keys encrypted in the old ones. Cluster head will then aggregate all three messages into one and send to nodes N_3 and N_5. N_3 and N_5 use their keys to spread the new keys to whole cluster. Cluster head does not come to know the new keys as it does not know the old keys. All the nodes, which know the old keys, can use them to find out the new ones except N_2. As N_2 does not know keys K_3 and K_4, it does not come to know the new values of keys K_1, K_2 and K_5 and thus it is effectively evicted from the network.

Key-generator Compromise: In case a key-generating node is compromised, it can either be replaced by a new key-generating node, or some other node can take over the responsibility of key-generation. Cluster head can also hold this responsibility temporarily but cluster heads can only take this responsibility up to a certain number of keys. After that, they might become a single point of failure. We assume that the cluster head and nodes, for which it generates key, immediately come to know that the node has been compromised. In effect, they cease any re-keying and wait for corrective actions. If a key-generator is compromised, the cluster head tells the command node how many times the key has been changed. Command node then calculates the current value of the key.

In case the compromised node is to be replaced, a new key-generator node is added to the network. Method for addition of a new node is described in section 4.3. After addition of the new node, command node sends the current value of key to the newly deployed key-generator node. The newly deployed key-generator node changes the key, encrypts in the old one and send it to the cluster head using its K_{chs} key.

In case some other node is to be given the responsibility of generating the key, then the command node informs the cluster head and provides it with the current value of key encrypted in its current K_{bsc}. Cluster head forwards it to the responsible node. The responsible node then calculates the new value of key, encrypt it in the old one and send it to the cluster head using its K_{chs} key.

In case the cluster head is given the responsibility of this key, the command node simply provides the current value of key to the cluster head. Cluster head then calculates new value and encrypts it in the old one. It is recommended to use this method temporarily and that too when above two methods can't be applied. As soon as a capable node is deployed, responsibility should be shifted on it and administrative keys should be deleted from the cluster head. This is because in our scheme, the management of administrative keys is not the responsibility of gateways.

The compromised node also knows some other keys, which it does not generate. Cluster head asks generators of those keys to change them encrypt in the previous ones and send using their K_{chs} to the cluster head. After getting all the keys, the gateway aggregates them into one message. From the EBS matrix, it finds out the key-generating nodes, whose keys are required for spreading the new keys. It then sends this aggregated message to each of those key-generating nodes using their respective K_{chs}. Upon receiving the aggregated message, the key-generator node decrypts the message and then encrypts it again with the key it generates. Eventually, the message is forwarded to all the cluster in such a way that only the relevant node knows about the new key.

5 Comparison

As compared to shell, we have moved the generation of administrative keys to key-generating nodes within our cluster. We have been able to take this responsibility down to sensor nodes due to the use of key-chains [4]. Also, we have maintained the condition that there is no single point of failure in our scheme. In this section, we will compare the number and length of messages exchanged in each stage in both SHELL and MUQAMI.

5.1 Storage Requirements

Sensor Nodes: Equal numbers of EBS keys are stored in each node in both schemes. However, if we compare storage requirement of the two schemes on lowest level sensor nodes, we see that we have to store three administrative keys (K_{sg}, KS_{CH} and KS_{Key}) in case of SHELL and only two (K_{bsc} and K_{chs}) in case of our scheme

MUQAMI. On the contrary, storage requirements of the key-generating nodes are more in case of MUQAMI. In addition to the keys that are stored in the simple sensor nodes, key-generating nodes also have to store K_{chs} along with the series of key values that it uses for managing the key it generates.

In key-generating nodes, we use one-way hashing function to generate and store 'n' key values at a time. When all 'n' values are exhausted, command node provides the key-generator with another seed value to generate 'n' keys. Value of 'n' depends upon storage capabilities of the key-generating node. We have to store only K_{chs} and K_{bsc} in MUQAMI as compared to SHELL, in which we had to store Ksg, KSCH and KS_{Key}. This provides us with one more slot for storing a key. We assume that in the key-generating node, each key uses 'm' bits of memory.

$$SR_{kg} = m \times (n - 1)\ bits \qquad (1)$$

where SR_{kg} is the additional storage requirement for the key-generating node in MUQAMI as compared to SHELL. If we assume that every key takes 128 bits (16 bytes) and re-keying is done after every 500 seconds as in SHELL, then a key-generating node requires mere 512 bytes for managing its generated keys for more than four and a half hours before contacting the command node for a new seed. We should also keep in mind that the number of such key-generating nodes wouldn't be too high as we show in the storage requirements analysis of the cluster heads.

Cluster Heads: As opposed to SHELL, cluster heads in our scheme need not store any key to communicate with other cluster heads. Moreover, gateways in our scheme also do not need to generate and store EBS keys for other clusters. If we assume that the key management capability was equally distributed among all gateways in SHELL and size of clusters both in our scheme and SHELL are same then

$$SR_{ch}^{SHELL} - SR_{ch}^{MUQAMI} = (\sum_{i}^{k+m+r} n[i])keys \qquad (2)$$

where SR_{ch}^{SHELL} and SR_{ch}^{MUQAMI} are the storage requirements of cluster head nodes in SHELL and MUQAMI schemes respectively, 'k' and 'm' are the EBS parameters, 'r' is the number of gateways, with which a gateway communicates in SHELL, and 'n[i]' is the number of key values that SHELL stores for key. We assume that SHELL also uses one-way hashing function for key management. In SHELL, cluster size is assumed to be between 500 and 900. Assuming the value of 'k' and 'm' to be 7 each we get more than 3000 key combinations, which looks to be quite safe if we need to deploy more sensor nodes afterwards. This also shows that we do not require a large ratio of key-generating nodes either. If we assume value of 'r' to be 2 and each n[i] to be 32 on average (as assumed above),

$$SR_{ch}^{SHELL} - SR_{ch}^{MUQAMI} = 16 \times 32 = 512\ keys \qquad (3)$$

Further, if we assume each key to be 128 bits (16 bytes) as in SHELL, storage difference becomes

$$SR_{ch}^{SHELL} - SR_{ch}^{MUQAMI} = 8KB \qquad (4)$$

Storage cost at cluster head is increased a little bit in our scheme. In addition to '0' or '1', now we also have to incorporate '2' to indicate the key-generating nodes. Storage cost also depends upon the storage scheme applied for storing the EBS matrix.

If we apply SHELL, then our cluster heads need to have double the size of memory that we have in typical MICA mote just for key management purposes. On the contrary, if we apply MUQAMI, we do not require very high memory both in cluster head and the sensor nodes.

5.2 Computation Requirements

Sensor Nodes: In SHELL, sensor nodes used to do two decryptions per administrative message received, while in our scheme, only key-generating nodes are required to do one encryption and one decryption. Other nodes only need to do one decryption. However, key-generating nodes would have to bear some additional computation cost due to the calculation of keys through one-way hashing function.

Cluster Heads: Computation requirements are also higher in case of SHELL. Computation cost of calculating the EBS matrix is lower in MUQAMI than SHELL. This is because in our scheme, the command node explicitly informs the cluster head about the keys stored and generated by every node. For the distribution of keys in SHELL, it requires four encryptions and four decryptions. Two of the decryptions are done at sensor level and rest is done at gateway level. On the contrary, our scheme requires only two encryptions and two decryptions for the distribution of keys. Out of these four computations, only one is done at the gateway level.

In SHELL, all sensor nodes required two decryptions, while in our scheme only key-generating gateways require one decryption and one encryption. Other sensor nodes only require one decryption. So, if we have to distribute 'k+m' administrative keys inside a cluster,

$$COMP_{ch}^{SHELL} = ((r \times 2) + ((k+m) \times 4))computations \qquad (5)$$

where $COMP_{ch}^{SHELL}$ denotes the number of computations required by cluster head in SHELL for computing all the administrative keys once. Rest of the variables is the same as above. By computation, we mean one encryption or decryption. In MUQAMI, the number of computations comes out to be

$$COMP_{ch}^{MUQAMI} = (k+m)computations \qquad (6)$$

as the cluster head just needs to encrypt and send one message for key distribution. In effect, we see the following difference in the number of computations for each cluster head on average

$$COMP_{ch}^{SHELL} - COMP_{ch}^{MUQAMI} = ((r \times 2) + ((k+m) \times 3))computations \qquad (7)$$

Considering similar values of 'r', 'k' and 'm' as above the average computation difference at the cluster head, for computing all administrative key once, becomes 46. We need 60 computations in case of SHELL and just 14 in case of MUQAMI.

In case of node compromise also, MUQAMI overshadows SHELL. For redistributing the keys in SHELL, we need to initiate the jilting process that requires $r \times 2$ computations. Then for every key in ‘k’, neighbouring cluster head encrypts twice and sends to the cluster head. Then the cluster head decrypts and aggregates. This requires $k \times 3$ computations. Then for every ‘m’, cluster head encrypts and sends to neighbouring cluster head. Neighbouring cluster head decrypts, encrypts in another key and then further encrypts in the cluster head’s key. Cluster head decrypts and then broadcasts the message. This requires $m \times 5$ computations. Sensor nodes only need to decrypt twice for finding out a new key. $\text{COMP}_c^{\text{SHELL}}$ denotes the computations in case of node compromise. $\text{COMP}_c^{\text{SHELL}}$ can be calculated as

$$COMP_c^{SHELL} = ((r \times 2) + (k \times 3) + (m \times 5))\,computatio\,ns \qquad (8\,)$$

For sensor node compromise in MUQAMI, only ‘k’ key generating nodes are required to do one decryption and two encryptions (one extra computation). No extra computation is required in case of key-generating node compromise. Cluster head is required to do at maximum ‘k’ encryptions for asking key-generating nodes for new keys. This is followed by ‘k’ decryptions of reply messages and then further m encryptions after aggregation. So $\text{COMP}_c^{\text{MUQAMI}}$ comes out to be

$$COMP_c^{MUQAMI} = (2k + m)\,computations \qquad (9\,)$$

we see that the difference in the computations at each node comes out to be

$$COMP_c^{SHELL} - COMP_c^{MUQAMI} = ((r \times 2) + k + (m \times 4))\,computatio\,ns \qquad (10\,)$$

Considering similar values of ‘r’, ‘k’ and ‘m’ as above, the difference comes out to be 39. In case of SHELL, we need 60 computations while in case of MUQAMI, we just need 21. The difference in the cost of encryption and decryption is evident from these figures. Despite the fact that cluster heads have more power, we think it is a good idea to take off a large burden from cluster heads and put a small burden on a few key-generating nodes inside the cluster.

5.3 Communication Requirements

In this section, we will compare the number of communication messages that the sensor nodes have to transmit in each phase of our scheme. We assume that most of the energy is consumed in transmitting a message and thus it should be minimized.

Sensor Nodes: Network initialization phase for the sensor nodes is similar in both schemes, so there is not much to compare. As evident from the scheme, administrative keys are already stored on each node. Each key-generating node is required to broadcast one message encrypted in its generated key for re-keying of administrative keys, initial distribution of communication keys and re-keying of communication keys. Note that the transmission of one such message from each key-generating node causes significant decrease in the communication overhead of the cluster heads. Similar analysis is applicable to the re-keying methodology in both schemes. For node addition, similar analysis is required as in network initialization and initial key distribution. In case of the compromise of sensor nodes, no transmission is required by sensor nodes in SHELL scheme. In our scheme, all key-generating nodes are required to broadcast one message each.

Cluster Heads: In the network initialization phase of MUQAMI, command node computes most of the EBS matrix itself and communicates it to the cluster heads. In case of SHELL, command node sends inter-gateway keys to the cluster heads. Cluster heads also use these keys to find out the working and broken links. In the initialization phase, at least ‘r’ communication messages are transmitted from each node in SHELL where ‘r’ is the number of neighbouring cluster heads, with which each cluster head communicates. On the other hand, MUQAMI requires no communication messages to be transmitted from cluster heads. MUQAMI requires the cluster heads to receive significant EBS related information from the command node. On the other hand, before the initial key distribution phase in SHELL, cluster head transmits the complete EBS matrix to the command node. This requires almost the exact opposite of what happens in MUQAMI. MUQAMI is better because it requires command node to transmit and cluster head to only receive.

Apart from communicating the EBS matrix to the command node, SHELL requires each cluster head to share its EBS matrix with ‘r’ other cluster heads. On average, each cluster head transmits and also receives one complete EBS matrix just before the initial key distribution phase. In SHELL, each cluster head has to receive one additional key per sensor node in some other cluster. Since we are not considering the reception of messages, we establish that one extra inter-cluster transmission of whole EBS matrix by each cluster head is required in SHELL just before initial key distribution phase. Analysis of the overhead due to EBS matrix exchange depends upon the storage scheme used. Storage scheme are out of the scope of this paper.

Initial administrative keys are already stored in MUQAMI. Analyzing the administrative key distribution of SHELL, we see that neighbouring cluster heads generate one message per individual administrative key. This message is transmitted to the cluster head. Keeping in mind that the number of administrative keys is ‘k+m’, we observe

$$TRANSMISSIONS_{CH,CH}^{ADMIN,INI} = (k+m)transmissions \qquad (11)$$

where $TRANSMISSIONS_{CH,CH}^{ADMIN,INI}$ is the number of inter-gateway transmissions required from each cluster head in the initial distribution of administrative keys. Each neighbouring cluster can aggregate all such messages into one large message. After the cluster head receives one such message, it sends one message to each of the sensor nodes in its cluster head. So,

$$TRANSMISSIONS_{CH,S}^{ADMIN,INI} = ((k+m)\times S_n)transmissions \qquad (12)$$

where $TRANSMISSION_{CH,S}^{ADMIN,INI}$ is the number of transmissions required by cluster head to communicate with the sensor nodes in initial distribution of administrative keys and Sn is the average number of sensor nodes inside a cluster. For each communication key, cluster head first transmits it to ‘r’ neighbouring cluster heads. Each neighbouring cluster head encrypts each communications key in the administrative keys that it has, and sends it back to the cluster head. If we assume that there are ‘l’ communication keys,

$$TRANSMISSIONS_{CH,CH}^{COMM,INI} = (l \times (r + k + m))transmissions \tag{13 }$$

where $\text{TRNASMISSIONS}_{\text{CH,CH}}^{\text{COMM,INI}}$ is the number of transmissions required by cluster head to communicate with other cluster heads in initial distribution of communication keys. Value of $\text{TRANSMISSIONS}_{\text{CH,S}}^{\text{COMM,INI}}$ is also same as $\text{TRNASMISSIONS}_{\text{CH,CH}}^{\text{COMM,INI}}$ because every message in broadcasted inside the cluster.

In case of MUQAMI, there is no inter-cluster communication required. For each communication key, cluster head is required to send just one transmission to each key-generating node. So,

$$TRANSMISSION_{CH,S}^{COMM,INI} = (l \times (k + m))transmissions \tag{14 }$$

In case of communication between cluster head and sensor nodes, difference between the two schemes for initial distribution of communication keys become

$$DIFF\_TRANS_{CH,S}^{COMM,INI} = (l \times r)transmissions \tag{15 }$$

In both schemes, node addition phase is the same as network initialization phase. So, the same analysis is applicable in both phases.

When a sensor node is compromised, SHELL sends one message per neighbouring cluster head, informing them about the keys to be changed. Neighbouring cluster heads send new keys encrypted in old ones to the cluster heads. Cluster head aggregates them and sends back to the neighbouring cluster heads, so that the aggregated message can be encrypted in keys that are not known to the compromised node. Neighbouring nodes encrypt the aggregated message and send back to the original cluster head. This shows that we require four inter-cluster transmissions in case of SHELL. Eventually, the cluster head broadcasts the aggregated message one by one in every key that is not known to the compromised node. One broadcast communication in every key is required in MUQAMI also. Effectively, in case of node compromise the inter-cluster communications are avoided in our scheme.

6 Conclusion and Future work

From our discussion in section 5, we see that if we give a little responsibility to a small ratio of nodes in a cluster, we can take off a lot of burden from cluster heads. We have freed the cluster heads of inter-cluster communication and key management. In addition to that, we also take away from the cluster heads, the burden of communicating EBS matrix to the command node and neighbouring cluster heads. We achieve two goals by doing this. Firstly, our cluster heads will have a longer life as their power will mainly be used for long-range communications with the command node. Secondly, we have reduced the vulnerability of cluster heads as it is more costly to deploy cluster heads rather than simple sensor nodes.

Acknowledgement

This research was supported by the MIC (Ministry of Information and Communication), Korea, Under the ITFSIP (IT Foreign Specialist Inviting Program) supervised by the IITA(Institute of Information Technology Advancement).

7 References

1. I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, Wireless Sensor Networks: A Survey, Computer Networks 38(4), 393-422 (2002).
2. S. Tilak, N.B. Abu-Ghazaleh, and W. Heinzelman, A Taxonomy of Wireless Microsensor Network Models, ACM Mobile Computing and Comm. Rev. 6(2), 1-8 (2002).
3. M. Eltoweissy, H. Heydari, L. Morales, and H. Sadborough, Combinatorial Optimization of Group Key Management, J. Network and Systems Management 12(1), 33-50 (2004).
4. G. Dini and I.M. Savino, An Efficient Key Revocation Protocol for Wireless Sensor Networks, International Workshop on Wireless Mobile Multimedia, Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks, 450-452 (2006).
5. L. Lamport, Password authentication with insecure communication, Communications of the ACM 24(11), 770–772 (1981).
6. M. Younis, K. Ghumman, and M. Eltoweissy, Location aware Combinatorial Key Management Scheme for Clustered Sensor Networks, IEEE Trans. Parallel and Distrib. Sys. 17(8), 865-882 (2006).
7. G. Gupta and M. Younis, Load-Balanced Clustering of Wireless Sensor Networks, Proc. Int'l Conf. Comm. (ICC '03), 1848-1852 (2003).
8. O. Younis and S. Fahmy, HEED: A Hybrid, Energy-Efficient, Distributed lustering Approach for Ad Hoc Sensor Networks, IEEE Trans. Mobile Computing 3(4), 366-379 (2004).
9. K. Langendoen and N. Reijers, Distributed Localization in Wireless Sensor Networks: A Quantitative Comparison, Computer Networks 43(4), 499-518 (2003).
10. A. Youssef, A. Agrawala, and M. Younis, Accurate Anchor-Free Localization in Wireless Sensor Networks, Proc. First IEEE Workshop Information Assurance in Wireless Sensor Networks (WSNIA '05), (2005).
11. R.L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Comm. of the ACM 21(2), 120-126 (1978).
12. A.J. Menezes, P.C.V. Oorschot, and S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, (1996).
13. D. Eastlake and P. Jones, US Secure Hash Algorithm 1 (SHA-1), RFC 3174, IETF, (2001).
14. R. Rivest, The MD5 Message-Digest Algorithm, RFC 1320, MIT and RSA Data Security Inc., (1992).