Key Management in Wireless Ad Hoc Networks: Collusion Analysis and Prevention

Mohamed Younis and Kajaldeep Ghumman
Department of Computer Science and Elect. Eng.
University of Maryland Baltimore County
Baltimore, MD 21250
{younis, kajal1}@cs.umbc.edu

Mohamed Eltoweissy
Department of Computer Science
Virginia Tech
Falls Church, VA 22043
eltoweissy@vt.edu

Abstract

Due to the dynamic nature of WAHN communications and the multi-node involvement in most WAHN applications, group key management has been proposed for efficient support of secure communications in WAHNs. Exclusion Basis Systems (EBS) provide a framework for scalable and efficient group key management where the number of keys per node and the number of re-key messages can be relatively adjusted. EBS-based solutions, however, may suffer from collusion attacks, where a number of nodes may collaborate to reveal all system keys and consequently capture the network. In this paper we investigate the collusion problem in EBS and demonstrate that a careful assignment of keys to nodes reduces collusion. Since an optimal assignment is NP hard, we propose a location-based heuristic where keys are assigned to neighboring nodes depending on the hamming distance between the strings of bits representing the used subset of the keys employed in the system. Simulation results have demonstrated that our proposed solution significantly boosts the network resilience to potential collusion threats.

Keywords: Secure Group Communications, Wireless Security, Key Management, Collusion Attacks, Exclusion Basis Systems.

1. Introduction

Ad-hoc networking of wireless devices allows establishing communication links among participating nodes without the existence of a pre-arranged infrastructure [1][2]. The recent few years have witnessed a growing interest in Wireless Ad-Hoc Networks (WAHNs), motivated by a number of civil and military applications and by the continual advancement in wireless technologies. Digital battlefield, luggage handling in airports, context aware computing and mobile commerce are examples of a growing list of potential applications of WAHNs. In most WAHN setups, nodes are usually constrained in energy, communication and computation capacities. In addition, many of the WAHN applications may employ a large set of nodes with a dynamically changing network membership. These characteristics have made the design and management of WAHNs significantly challenging in comparison to contemporary networks.

One major challenge to realizing the full potential of WAHNs is their vulnerability to security attacks [3][4]. Indeed, for applications such as mobile commerce and tactical defense operations, security tops the list of design objectives. In such cases the network vulnerability will diminish the value of WAHNs and curtail their use. Thus, secure communications in WAHNs is vital to their acceptance and broader adoption in these information-sensitive applications. Confidentiality, authenticity, availability and integrity are typical security goals of WAHNs. WAHNs can be subject to different types of attacks from a variety of adversaries that target one or more of these security goals. Attacks may be external such as traffic jamming to node damaging, or internal such as collusion among some nodes to reveal the fundamentals of the employed security scheme. Applying message cryptography is the most common countermeasure for security attacks, with which nodes are assigned a data key(s), also called session or communication keys, to encrypt and decrypt messages and an administrative key(s), also known as key encryption keys to confidentially change the data key(s).

Since operations in WAHNs are inherently collaborative, where the autonomous nodes forming the WAHN collectively perform tasks, secure group communications are necessary for supporting efficient node cooperation. Key management is a core component in any secure group communications. The use of message encryption in group communications requires that each authorized member of a secure network has knowledge of one or more

communication key(s) shared by the group of nodes. The message source uses the communication keys for encrypting data packets before sending them to the group. Numerous conditions may introduce the need to evict a node, or a set of nodes. For example, nodes may experience failure due to energy depletion. Also nodes may be compromised and exhibit anomalous behavior [5][6]. When a member node is evicted from the group, the communication keys known to that member must be changed in order to maintain message secrecy. All remaining group members receive the new communication keys by secure transmission, which is typically accomplished by broadcasting a message containing the new key(s) to the group. Such a message must be indecipherable to the evicted member in order to prevent it from obtaining the new key. Therefore, re-keying messages are encrypted using administrative keys that are not known to the expelled nodes.

Exclusion Basis System (EBS) offers a combinatorial group key management protocol that exploits the trade-off between the number of administrative keys, "*k*", and the number of re-keying messages "*m*" [7]. A set of (*k*+*m*) administrative keys is used to support a set of *N* nodes, where each node is assigned a distinct combination of "*k*" keys. A node can be simply admitted to the group by assigning one of the unused set of "*k*" keys out of the total of $C(k+m, k)$ distinct combinations. Eviction of a compromised node can be performed by broadcasting replacement of the "*k*" keys that the evicted node knows using the "*m*" keys that the node does not know. The EBS approach proves to be very scalable for large networks and enables great flexibility in network management by controlling the values of "*k*" and "*m*" [7]. Large "k" increases the storage requirements at the node and slows down message decryption while large "*m*" increases communication overhead for key management.

However, EBS based key management can be prone to collusion attacks [8][9]. Two nodes collude when they share their keys with each other. In other words, colluding nodes would grow their knowledge about the network security measures. When using the EBS scheme, keys are reused in multiple nodes and only key combinations are unique. Therefore, it is conceivable that few compromised nodes can collude and reveal all the keys employed in the network to an adversary. Such scenario is considered as capturing the entire network since the adversary would be capable of revealing all encrypted communications in the network. Optimal assignment of key combinations to nodes so that the scope of collusion is prevented from widening to the level of capturing the network is a classical resource allocation problem and is thus NP-hard in nature. In this paper, we present an efficient heuristics for key assignment that reduces the probability of capturing the network. Our approach exploits the physical proximity of nodes so that a node would share most keys with reachable nodes. The main idea is to increase the number of nodes that need to be compromised for revealing all the employed keys.

The remainder of this paper is organized as follows. In the balance of this section we describe the considered system and threat models, overview the Exclusion Basis Systems framework and discuss related work. Section 2 analyzes the collusion problem in EBS and describes our solution for collusion prevention. Validation and performance evaluation of our approach can be found in Section 3. Finally, the paper concludes in Section 4 with a summary and an outline of future research plans.

1.1. System Model

A WAHN is a collection of autonomous nodes that together set up a topology without the support of a physical networking infrastructure. Each node functions as both a host and a router, and the control of the network is usually distributed among the nodes. The number of nodes in such a WAHN varies between a few tens in Mobile Ah hoc Networks (MANETs) and up to hundred thousands in Wireless Sensor Networks (WSNs). Communications among nodes are via multi-hop wireless broadcast with limited transmission range. In this paper, we consider applications in which nodes are stationary. We also assume that the physical location and communication range of all nodes in a WAHN are known. A good match for such a model is an unattended wireless sensor networks employed for a remote military reconnaissance or in a disaster relief environment.

Users may join or leave a WAHN at any time resulting in routing and security problems. This may be due to energy levels, as in sensor networks, or just the dynamic nature of the users, as in the ad hoc Internet network connection. Based on membership restrictions, WAHNs may be classified into open or closed classes. In an open network, any user can simply contact nearby network members and join as a node in the network. That node then may receive messages and access other nodes through its neighbors. An example would be an Armature Packet

Radio Network (APRN). Anyone with a ham radio license can be a member in the network. Obviously, the ability for anyone to join a network is itself a serious security concern. In a closed network, only valid nodes can be part of the network. A node is either pre-configured with keys or has some special information that only the ad hoc group knows. Node joining is then done in a very similar manner to open networks with the exception that the initialization may take extra steps. No assumption can be made about node collaboration. Nodes might refuse to get involved in any collaborative group management function either deliberately or unwillingly due to limited resources.

In this paper we focus on WAHN security using cryptographic keys. We further narrow our focus to address the collusion problem in key management schemes that use EBS as an efficient means for key management. We assume a closed WAHN with a key controller that is responsible for distributing the keys to WAHN nodes.

1.2. Threat Model

The wireless nature of WAHNs renders them susceptible to various attacks, especially when deployed in hostile and unattended environments. An adversary may try to eavesdrop on the messages exchanged in the system, intercept these messages as well as inject false messages. In this paper we mainly consider an adversary that tries to manipulate the system through compromising some network nodes. We do not assume the nodes to be tamper resistant and therefore they can be captured and tampered with. When a node gets compromised, we assume that all the information including the keys of the node is revealed. A simplifying assumption in our design is that the adversary does not launch a coordinated attack, i.e., if more than one node were to be captured, a compromised node would not be aware of the location of other compromised ones unless they are their immediate neighbors. In addition, we assume that the adversary does not have any prior knowledge of what is stored at each node and thus cannot selectively direct the attack to a particular node based on what that node has.

As mentioned earlier, the goal of the adversary is to uncover the keys used in the system so that he/she can discern the network operation. To achieve such a goal, the adversary attacks individual nodes and fosters collusion among compromised nodes. The main objective of node collusion is to incrementally aggregate the uncovered keys of individual nodes to a level that allows revealing all encrypting traffic in the network. The following scenarios can be identified for the compromised nodes to collude:

a) The compromised nodes have direct communication links: In such a case, collusion can be very subtle and hard to prevent.
b) The compromised nodes can only communicate through multi-hops: This would require the compromised nodes to be aware of the location/ID of other compromised nodes.
c) The adversary sets up separate channels for compromised nodes to communicate: This scenario implies a coordinated network attack and is not handled by our approach. We argue that the open communication among compromised nodes or with a coordination center would make it easy to detect and thwart the attack. In addition, the adversary cannot simply use the radio of the compromised node and would have to provide an independent radio, which may not be always possible.

Our focus in this paper is to counter the scenario in (a) above through careful assignment of keys to communicating neighboring nodes. A fitting example of the assumed model of attacks is in a battlefield or in a border monitoring applications. In such environments, attacks are typically covert and start localized in scope before spreading. The goal of our key assignment approach is to assign the keys in such a way, that even though compromised nodes may collude and share their keys, an adversary would not be able to access all the keys of the network unless he/she remains without detection for an extended period of time. In other words, we try to reduce the probability that the entire network will be captured. It should be noted that we assume that the network is empowered with some intrusion detection mechanisms so that a persistent adversary would not eventually be able to capture the network after attacking for an extended duration.

1.3. Exclusion Basis Systems (EBS) Framework

An EBS as defined in [8] is a collection Γ of subsets of the set of members. Each subset corresponds to a key and the elements of a subset $A \in \Gamma$ are the nodes that have that key. An EBS Γ of dimension (N, k, m) represents a situation in a secure group where there are N members numbered 1 through N, and where a key server holds a

distinct key for each subset in Γ. In this subsection, we will use the terms "key" and "subset" interchangeably. If the subset A_i is in Γ, then the key A_i is known by each of the members whose number appears in the subset A_i. Furthermore, for each $t \in [1, N]$ there are m elements in Γ whose union is $[1, N] - \{t\}$. From this it follows that the key server can evict any member t, re-key, and let all remaining members know the replacement keys for the k keys they are entitled to know, by multicasting m messages encrypted by the keys corresponding to the m elements in Γ whose union is $[1, N] - \{t\}$. Each new key is encrypted by its predecessor in order to limit decipherability only to the appropriate members.

To construct *EBS(N, k, m)* for feasible values of *N*, *k* and *m*, we employ a canonical enumeration of all possible ways of forming subsets of *k* objects from a set of *k + m* objects. We do this because our construction is based on such an enumeration. There are several algorithms for producing such a sequential enumeration. We choose an enumeration where each element of the sequence is a bit string of length *k + m*, where a 1 in the i^{th} position of a string means that object *i* is included in that subset, for all *i* ($1 \leq i \leq k + m$). Note that, every bit string in this enumeration will have exactly *k* ones.

For any *k* and *m*, let *Canonical(k, m)* be the canonical enumeration of all *C(k + m, k)* ways to form a subset of *k* elements from a set of *k + m* objects. For the sequence of bit strings in *Canonical(k, m)* we form a matrix *A*, where *k* and *m* are understood, and whose *C(k + m, k)* columns are the successive bit strings of *k + m* length, each with *k* ones. *"A"* is called the canonical matrix for *EBS(N, k, m)*. For example, the canonical matrix *A* for *EBS(10, 3, 2)* contains the enumeration of all *C(5,3)* ways to form a subset of 3 keys from 5 keys, as shown in the Table 1.

Table 1: The canonical matrix A for EBS(10, 3, 2)

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_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

The Exclusion Basis System scales well with the number of nodes, and depending on the system specifications and requirements, the parameters can be selected such that we can either have fewer keys at every node or fewer re-key messages. According to EBS, every node has a unique set of keys stored at it. The total number of keys in the system can be a maximum of *k + m* and the maximum number of key combinations can be *C (k + m, k)*. However, the number of nodes in the system is greater than *k + m*. Therefore, it is likely that when nodes in the system collude, their combined set of keys will reveal many keys to the two nodes. This makes EBS prone to collusion. Although collusion cannot be completely avoided, we present an approach, which aims at limiting the chances of the entire network being captured in case of collusion.

1.4. Related Work

Group key management protocols can be classified as centralized, decentralized, or distributed [14]. Examples of centralized protocols are Group Key Management Protocol (GKMP) [15], Logical Key Hierarchy (LKH) [16] [17], One-way Function Chain Tree (OFCT) [18], and Efficient Large Group ELK [19]. Examples of decentralized group key management protocol are MARKS [20] and Koronos [21]. Examples of distributed protocols include Conference Key Agreement CKA [22] and Distributed LKH [23]. The objective of most of these protocols is to balance communications with storage. ELK and Koronos also attempt to balance security and efficiency by using small-size hints and batch re-keying respectively. A major drawback that hampers the use of most of these group key management protocols in WAHNs is the lack of support for faulty and misbehaving nodes, and the overhead incurred to support key management activities including setup and re-keying. Protecting against collusion attacks also is a problem that has not been adequately addressed in WAHN environments.

Quite recently, the Hybrid Key Tree (HKT) was proposed in [24] to balance security and efficiency using a two level hybrid key tree. HKT has a sub-linear storage complexity at the controller due to the use of clusters. Cluster sizes are adjusted to resist collusion. Another effort is reported in [8][9] proposing an EBS-based distributed key

management solution for collusion resistance among evicted nodes. However, their work does not consider collusion among active nodes currently in session. Similar to [8] and [9], the work presented in this paper is based on key management using EBS. A distinguishing feature of our solution, however, is the use of node proximity in a WAHN to reduce the probability of collusion.

A number of pair-wise key pre-distribution schemes have been proposed for WSNs [26][27][28][29][31]. The original schemes in [26] and [27] do not scale to large networks since an increase in network diameter would boost the chances for a network compromise. In [28], Liu et al. present location-based pair-wise key pre-distribution techniques based the key pre-distribution protocols of Eschenauer [26] and Chan [27]. The schemes in [28] assign secrets to sensor nodes using some picks from a pool of polynomials. The Liu et al's scheme improves both the scalability and efficiency of key pre-distribution in WSNs. Another effort by Du et al. [29] presents a key pre-distribution approach in which if the number of compromised nodes is less than a threshold, the probability that nodes, other than the ones already compromised, will be affected is close to zero. A drawback of pair-wise key pre-distribution schemes is that they may impede in-network processing, which is highly recommended in WSNs to reduce communications among nodes with limited resources [30][31].

Another class of key management schemes is based on threshold secret sharing. For example, in the MOCA framework [32], some mobile nodes (known as MOCA) within the network are designated to act as a distributed certification authority. A locality driven key management scheme based on the principles of threshold cryptography has also been proposed in [33]. This scheme uses a number of mutually trusting certification authorities each of which provides authentication for its own community. Meanwhile, the key establishment scheme of [34] is based on the principles of probabilistic key sharing and threshold secret sharing. The proposed protocol is distributed in nature and is secure against collusion up to a certain number of nodes. Threshold-based cryptographic solutions generally incur high communication and computation overhead.

Motivated by the efficiency of EBS-based group key management schemes, we attempt to address issues that may impede winder-scale adoption of EBS in secure WAHN communications. In this paper, we focus on reducing the problem of collusion between compromised nodes in WAHNs that use EBS-based group key management protocols. We propose a key assignment approach which assigns key combinations to nodes according to their distance from each other.

2. Collusion Prevention

Collusion implies that two or more parties collaborate by secretly sharing their knowledge in order to gain access to certain information that they are not authorized to have. In our system a collusion attack can be possible when compromised nodes are in the transmission range of one another. Such nodes can collude by combining their set of known keys and thus expanding their capabilities in overcoming the network security measures. For example, consider nodes n_1 and n_2. Suppose n_1 knows keys K_1, K_2 and K_3, and n_2 knows keys K_2, K_4 and K_5. If n_1 and n_2 decide to collude, they will both know keys K_1, K_2, K_3, K_4 and K_5. Now, if $K_1 \ldots K_5$ are the only keys that are used in the network, n_1 and n_2 are said to have captured the network. In the next subsection we analyze the potential for a collusion attack to lead to capturing the network and develop a metric for quantifying such potential. Subsection 2.2 builds on this analysis and presents a key assignment algorithm that increases the network resilience to collusion.

2.1. Probability of Capturing the Network

We define a neighborhood of a node n_1 as all those nodes, which are in the transmission range of n_1. In order to collude, two nodes must be in the transmission range of one another, otherwise they have to collude through a third party. Collusion is most likely to occur when two neighboring nodes are compromised. The compromise of a node implies that the node has been captured by an adversary and can be manipulated. Thus, when two nodes in the same neighborhood are compromised, they may be manipulated to collude. As a result of collusion, the two colluding nodes would each know their own keys as well as the keys of the node it has colluded with. Collusion is transitive in nature. If n_1 is a neighbor of n_2, n_2 is a neighbor of n_3, and n_1 colludes with n_2, the resultant keys known to both of them would be Keys(n_1) $\cup$ Keys(n_2). Thereafter if n_2 and n_3 collude, the keys known to n_2 and n_3

would be Keys(n_1) ∪ Keys(n_2) ∪ Keys(n_3). Thus, it can be seen that if multiple nodes collude, they are likely to uncover all employed keys.

In an EBS system, each key combination can be represented in the form of bit strings of k 1's and m 0's, where k is the number of keys stored at each node and m is the number of re-key messages required. A value of "1" indicates the node knows the corresponding key. The hamming distance between any two combinations is defined as the number of bits that the two combinations differ in. Let d be the hamming distance between a pair of key combinations. The value of d is bounded by:

$$2 \le d \le 2k \qquad k < m \quad \text{------ 1(a)}$$
$$2 \le d \le 2m \qquad m < k \quad \text{------ 1(b)}$$
$$2 \le d \le k + m \qquad k = m \quad \text{------ 1(c)}$$

When two nodes collude, they both will know at least d keys, since d is the number of keys that they differ in. In addition they will also know all the keys that are common to both nodes. The common keys are equal to $k - d/2$. Thus, we get the number of keys known to the two colluding nodes as $k + d/2$. This leads us to the conclusion that the lower the hamming distance (the value of d) is, the fewer the total number of potentially revealed keys will be. This is illustrated with the help of the following example.

Consider three nodes with the following key combinations:

n_1: 1001100, n_2: 1001010, n_3: 0110001

Hamming distance between n_1 and n_2 = 2

Hamming distance between n_1 and n_3 = 4

If n_1 and n_2 colluded, they would both have a key combination 1001110.

If n_1 and n_3 colluded, they would both have a key combination 1111101.

Therefore, collusion of n_1 and n_3 cause more keys to be uncovered than the collusion of n_1 and n_2.

Since EBS is based on the distinction between the key combinations assigned to nodes rather than the keys themselves, a minimum of two nodes would be required to collude in order to reveal all keys used in the system, and a maximum of N nodes will be required when every node knows just one key. Even though the idea of every node storing just one key seems attractive in terms of increasing the resilience of the network, having the parameter $k = 1$ leads to numerous re-key messages which increases the overhead for the network and is therefore not a desirable option. Therefore, if 'p' is the probability that a node can be compromised, the probability P for capturing a network of N nodes would be (assuming statistical independence): $p^2 \ge P \ge p^N$

We define a collusion chain to be the transitive closure of a set of colluding nodes so that the union of assigned keys includes all the $k + m$ keys of the network. For example, in Fig 1(a), although node 1 and node 3 have been compromised, they cannot collude since node 2 is an uncompromised node and there is no direct communication link between node 1 and 3. However, in Fig 1(b) all three nodes have been compromised and we have a collusion chain. Therefore, it is essential that the key combinations be assigned in a careful manner, so that the cardinality (length) of the collusion chain increases. Long chains will decrease the probability of capturing the network (If x nodes are required in the collusion chain, the probability that the network is captured will be p^x. Since $p < 1$, as x increases p^x decreases).

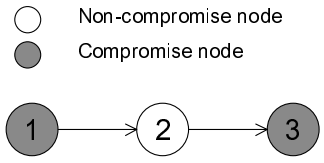

Fig 1(a): Node 1 and 3 do not collude since they are not in direct communication range

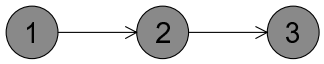

Fig 1(b): Node 1, 2 and 3 can collude since they are in direct communication range

Since two nodes cannot share their keys without being in direct communication range, we exploit this requirement as well as the location of nodes when assigning keys. We assign key combinations to neighboring nodes such that combinations having a lower hamming distance are assigned to nodes that are physically closer to each other. The advantage gained by this assignment scheme is that even if these nodes collude, they will not have knowledge of all the keys in network. Further collusions will be required to unveil all the keys, which would lower the probability of capturing the network, as we explained above.

2.2. Key Assignment Approach

In this section we present a novel approach for diminishing the probability of network capturing through collusion among compromised nodes. Before presenting our algorithm, we formulate the assignment problem and analyze its complexity.

2.2.1 Problem Formulation

Per the EBS methodology, we need to pick for each of the N nodes in the network a unique combination of "k" out of a set of $k + m$ keys, where k is the number of keys stored at each node and m is the number of re-key messages. Assuming that the physical location and transmission range of all nodes are known, the approach exploits the reasoning explained earlier while assigning keys to nodes. Nodes that are in such a close physical proximity that make them in the reachable communication range of each other are assigned key combinations with a lower hamming distance than those that are far apart. In other words, we try to increase the required number of colluding nodes for capturing the network.

We represent the assignment problem as a graph $G\ (V, E)$ where V is the set of N nodes and E represents the set of edges. An edge is said to exist between two nodes if the nodes are in communication range of one another, in which case we refer to them as neighbors. Each edge is assigned a weight, which is equal to the Euclidean distance between the nodes at each end of the edge. The goal is to assign a unique key combination to each node such that neighbor nodes are assigned combinations with the smallest mutual hamming distances. Edge weights are used to prioritize the neighbors. Close neighbors are favored for lower hamming distance combinations since they generally stay reachable for long time compared to further neighbors whose reachability may deteriorate over time due to increased noise, battery exhaustions or even the placement of obstacles.

Performing such key assignment can be modeled as a register-allocation problem, widely considered in compiling programs, and which tries to keep most live variables stored in registers for faster execution time [10]. Variables are modeled as vertices and edges exist between variables that are live at the same time. When possible, distinct registers are assigned to the connected nodes to avoid overwrite so that the use of non-register memory is minimized. In our case the key combinations with the smallest hamming distances are to be assigned to connected nodes in order to minimize the probability of network capturing. The register allocation problem is a known NP-hard problem [11][12] and therefore we pursue a heuristic approach, as we explain next.

2.2.2 Algorithm

We propose an algorithm that strives to optimize the key assignment locally and proceeds in a greedy fashion to cover the entire network. For every combination KC_a of k keys, we calculate the hamming distance d_{ab} to every other combination KC_b, i.e. $\forall b\ d_{ab} \mid a \neq b\ \&\ b \in C(m+k, k)$. We then sort these combinations (i.e., $KC_b\ \forall b$) in a non-decreasing order of the calculated hamming distances (d_{ab}). The problem can now be stated as "given the keys of a node "i", assign key combinations to the nodes $\{\forall j \mid (i,j) \in E\}$ picked in an ascending order of their hamming distance". That is, we start assigning the key combination with the least hamming distance and so on. Since we employ homogenous key combinations, i.e. every combination has k 1's and m 0's, the hamming distances will be bounded by equations 1(a), 1(b), 1(c) given earlier.

The algorithm starts with the node having the largest number of neighbors, designating it as a root, and proceeds in a breadth first manner. After assigning a key combination CK_{root} to the root, we identify the nodes that are reachable to it (its neighbors) and assign some unused key combinations that have the least hamming distance to CK_{root}. As we mentioned earlier, closer neighbors are favored for combinations that has the least hamming distances to CK_{root}. Subsequently, for every node that has not been visited, the node's neighbors are identified and key combinations are assigned to each of them. Analogous to graphs, we refer to the neighbors of a node under consideration as its children. Each time a key combination is assigned to a node, we try to assign keys in such a way that the hamming distance to the key combination of the node's parent is minimal, needless to say that every node is assigned a unique key combination.

Very often, two nodes may share the same parent and be neighbors to each other at the same time. In this case the key assignment has to be done carefully. Our approach calls for assigning key combinations to the nodes so that not only the hamming distances between the parent and the children are minimized, but also those between any two children that are connected with an edge. This is done by either selecting another unused combination that satisfies the above condition or by swapping the key combinations between the node under consideration and any previously assigned combinations. Since this can lead to potentially trying all combinations and making the assignment heuristics lean towards an exhaustive search, we limit the swapping to sibling nodes. Such restriction is logical since we pursue a breadth-first ordering and the impact of key assignment to one node will be limited to those nodes that are on the same level or one level above in the parsing hierarchy [13]. Again, a swap is accepted only if it does not increase the hamming distance between the key combination of the node chosen for swapping and the key sets of all its neighbors. Thus, our approach tries to minimize the maximum hamming distance between any two combinations.

The swapping process is applied first to a child node and if it turns to be unsuccessful, swapping the keys of parent node (that is being visited) is tried. Preference would be given to swapping with a neighbor sibling that matches the visiting/non-visiting status of the child node under consideration. Two neighbor nodes that were visited, most probably, would have been assigned key combinations with the least possible hamming distance. Thus, swapping the keys of these visited neighbors has a high potential of being acceptable. On the other hand, it is better for non-visited nodes to be swapped with non-visited siblings, especially neighboring ones, since the scope of the validation of such swapping will be limited to their parent in the parsing tree. It should be noted that a node, whose keys were swapped before with the same node, is marked as ineligible for consideration. When a suitable swap is not possible, an used key combination with the least possible hamming distance is picked. In the balance of the discussion we will illustrate the idea with an example. The pseudo-code for the algorithm along with the definition of the used notation can be found in Appendix A.

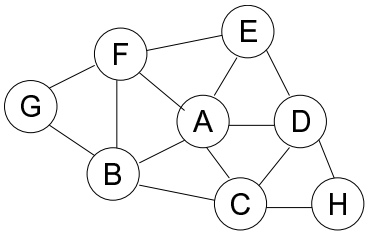

Fig. 2: An example of an 8-node network with an assumed equal link length.

Table 2: *Canonical(2, 3)*

KC					
1	1	1	0	0	0
2	1	0	1	0	0
3	1	0	0	1	0
4	1	0	0	0	1
5	0	1	1	0	0
6	0	1	0	1	0
7	0	1	0	0	1
8	0	0	1	1	0
9	0	0	1	0	1
10	0	0	0	1	1

Table 3: *Mutual hamming distance among KC's*

KC	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	2	2	2	2	2	2	4	4	4
2	2	0	2	2	2	4	4	2	2	4
3	2	2	0	2	4	2	4	2	4	2
4	2	2	2	0	4	4	2	4	2	2
5	2	2	4	4	0	2	2	2	2	4
6	2	4	2	4	2	0	2	2	4	2
7	2	4	4	2	2	2	0	4	2	2
8	4	2	2	4	2	2	4	0	2	2
9	4	2	4	2	2	4	2	2	0	2
10	4	4	2	2	4	2	2	2	2	0

Let us consider the scenario in Fig 2. For simplicity we assume all links span the same Euclidian distance. We employ the EBS scheme with k=2 and m= 3. Tables 2 and 3 enumerate the possible key combinations and their mutual hamming distances. Since the node *A* has the largest degree (5 links), we pick it as a root and assign KC_1 to it. Then nodes *B*, *C*, *D*, *E* and *F* are assigned KC_2 to KC_5 respectively and "*A*" is marked as visited. Next we consider node *B* and assign KC_8 to *G*. While the keys of *B* and *C* fit nicely, *B* and *F* do not since the hamming distance between KC_2 and KC_6 is high. Because KC_7 would not make things better we look for a swap. Since node *F* has not been visited yet, we can swap its keys with either node *C*, *D* or *E* all of whom have a hamming distance of 2 to the keys of *B*. Let's just pick *E*. Thus, *F* switches to KC_5 and *E* gets KC_6. It is now the turn for node C, for which the combinations assigned to its neighbors *B* and *D* are

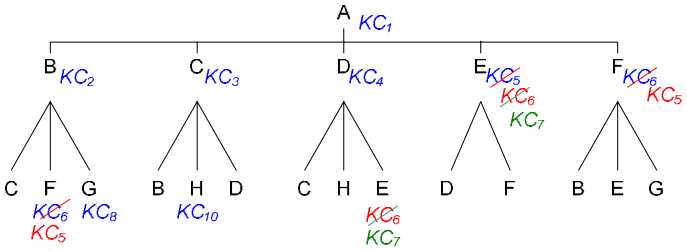

Fig 3: Illustration for how the algorithm is applied

perfect. Node *H* is assigned KC_{10}. Node *D* has neighbors *C* and *H* with key combinations of hamming distance of 2. However, the hamming distance to the keys of the other neighbor *E* is 4. KC_7 is a perfect match for *E* since it fits nicely with node *A* and *D*. When visiting *E* no action is to be taken. Finally, node F is visited with all its neighbors having good key combinations (each with a hamming distance of 2 to KC_5). A pictorial illustration of the described steps is shown in Fig. 3.

3. Validation and Performance Evaluation

We implemented the key assignment algorithm using C++. A network was simulated by assigning random positions to the nodes and creating an adjacency matrix assuming unified transmission range for all nodes. The deployment region was selected to be 500×500 meter and the node communication range was set to be 55 meter. In this section we discuss the performance metric and results.

3.1. Assessing the Potential for Collusion

As explained earlier, our approach strives to minimize the potential of network capturing by extending the length of collusion chain necessary to reveal all keys. In the experiments once the keys are assigned, we check for the shortest collusion chain in the network. Basically, we start checking what if a node colludes with each individual node in its neighborhood. We repeat this step for every node in the network generating all possible collusion sets of 2 nodes. We check whether any set would have gained knowledge to all keys. If not, we repeat for another iteration to form all possible collusion sets of 3 nodes. We repeat these steps until a collusion chain is formed. The length of the chain (number of iterations + 1) indicates the quality of the key assignment. While this procedure is analogous to finding a route from a node to every other node in the network, the complexity is still polynomial in N and |E|. In addition, we stop when all keys are known to a set of colluding nodes, which usually happen after a few iterations. It should be noted that we finish when all *used*, not all possible, keys in the network are revealed. This important for small networks, for which only a subset of the keys are assigned to nodes. We will revisit this point when we discuss the experimental results.

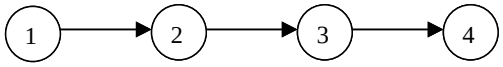

Fig. 4: A collusion chain of 4 nodes

The implementation of the described collusion assessment procedure is significantly simplified by the representation of key combinations as a bit vector. Two nodes colluding would mean that they collectively have knowledge of the keys known to each of them and thus can be performed by a logical OR operation of the 2 vectors. We illustrate the implementation with the help of an example. Fig. 4 shows the collusion chain. Let the total number of keys used in the system = 5.

$KC_1 = 10001$ $KC_3 = 10100$ $KC_2 = 10010$ $KC_4 = 11000$
$KC_1 \cup KC_2 = 10011$
$(KC_1 \cup KC_2) \cup KC_3 = 10111$
$((KC_1 \cup KC_2) \cup KC_3) \cup KC_4 = 11111$

The pseudo-code for the collusion assessment procedure can be found in Appendix B.

3.2. Analysis of Results

We ran experiments for varying number of nodes in the system, and observed how the system performs for different values of *k* and *m*. Each set of readings was averaged over 10 test runs, for each a distinct seed is applied to generate a random network topology. We observed that with 90% confidence level, the simulation results stay within 6%-10% of the sample mean.

We have chosen the *k* and *m* values such that the sum of $k + m$ is constant. The results plotted in Fig. 5 show the relation between different values of *k* and *m* and the corresponding number of nodes required in the collusion chain for capturing the network. As indicated $k + m = 10$ and the number of nodes in the network varies from 20 to 200. The right vertical axis reflects the length of the collusion chain while the left vertical axis indicates the total number of keys with "*k*" reported in the bottom part of the bar. The experiments were repeated using random assignment of keys to nodes. The results of such random allocation of keys are plotted in Fig. 6 and are used as a baseline for comparison.

It should be noted that for small networks and large number of key combinations, our approach tends to use the least number of keys since it minimizes the hamming distances among neighbors. In the experiment we have considered the network to be captured when the set of "employed" keys are revealed, which may not be all the allowed $k + m$ keys. For example in Fig. 5, for k=4 there are 210 different combinations of which 20 are assigned to the nodes. When applying our approach only 7 distinct (out of the 10) keys were employed. We also like to clarify that for large networks, it is infeasible to use small values of k since insufficient number of key combinations would be available. Based on figures 5 and 6, the following observations can be made:

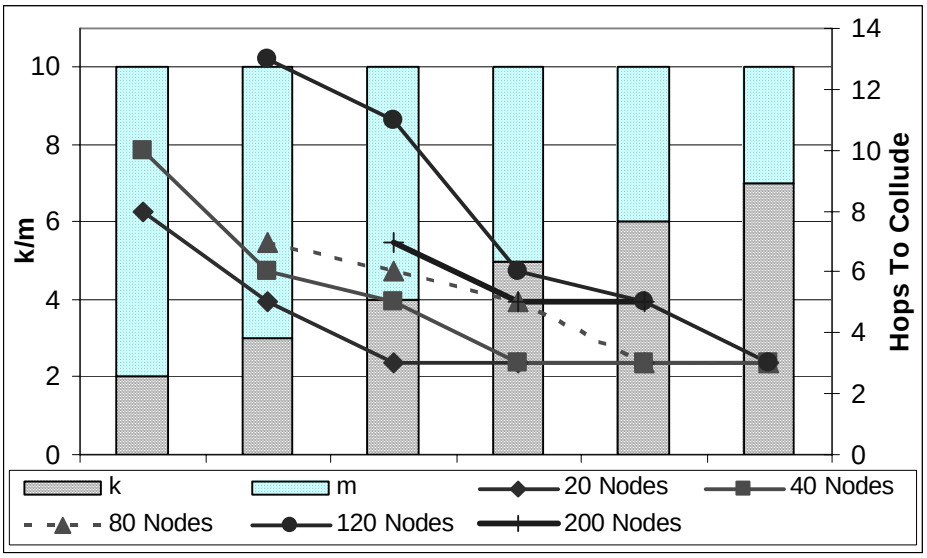

Fig. 5: The length of the collusion chain for different "k" with k+m=10 while using our proposed key assignment algorithm and for different networks sizes

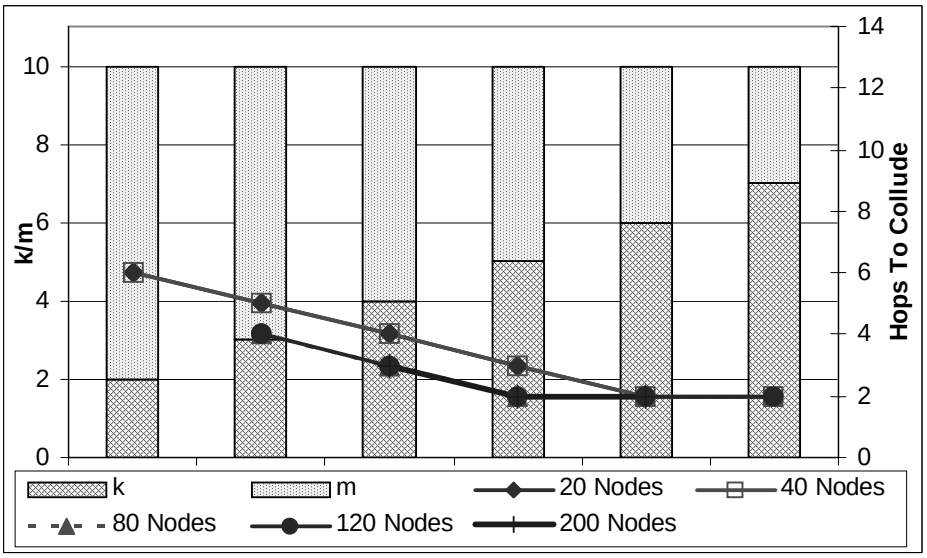

Fig. 6: The length of the collusion chain for different "k" with k+m=10 under random key assignment and for varying networks sizes

1. Our approach consistently outperforms the random assignment and is distinctly more superior for small k values. To illustrate, let us consider the case for k=4, N=120. Our approach achieves a collusion chain length of 11 nodes compared to 3 when a random assignment is pursued. For a probability of a node compromise "p" of .01, the probability of capturing the network using our approach is lower by a factor of 10^{16}.
2. For a given number of nodes, as the k value increases the number of hops required to collude is observed to decrease. This is in agreement with the analytical reasoning since uncovering all keys in the network would need the collusion of a fewer number of compromised nodes when the number of keys per node increases (while keeping the $k + m$ constant). In other words the union of a fewer number of key combinations will yield all the keys in the system. However, our approach makes this less threatening compared to the random key assignment.
3. As the network size grows, Fig. 5 indicates that the length of the collusion chain decreases in most cases with one exception corresponding to the 200-nodes network. For small networks, e.g. 20, 40 and 80 nodes, a boost in the number of nodes would increase the number of distinct keys employed by our approach and thus would have a positive impact (recall our earlier note regarding the tendency of our approach in using the least

number of distinct keys for small networks). However, a significant enlargement of the network can have a negative effect since most key combinations would have to be assigned to nodes and our key assignment heuristics become somewhat constrained. For example, when increasing the network size from 120 to 200 nodes, the length of the collusion chain went down from 11 to 5 when k = 4, and from 5 to 4 when k = 5.

4. The results of Fig. 6 show that a random key allocation, in contrary to our approach, does not take advantage of small networks in doing a better job in collusion prevention. It is conceivable that the network will be captured by as little as 2 compromised nodes.
5. The results also can be insightful in determining the *k* and *m* parameters for the EBS scheme. While contemporary system design issues such as storage, communication-related energy consumption and frequency of re-keying are usually considered, the implication of the choice of *k* and *m* on the network resilience to collusion attacks can lead to a trade-off.

To further validate our results and confirm these observations, we have performed another set of experiments on a network of 100 nodes, again deployed in 500 x 500 meters area and setting a transmission range of 55 meters. We have randomly picked a set of compromised nodes and checked to see if they could successfully collude to uncover all the keys used in the system. We ran the simulation for different numbers of compromised nodes. The experiments were run 100 times using both our key assignment and the random key assignment methods. We then counted the number of times the network was captured in both cases. The results are shown in the Fig. 7. From the figure we can see that the number of times the network was captured under the random key assignment is greater than the number of times it was captured using our approach. When the number of compromised nodes is few, our approach increases the resilience of the network by at least a factor of two compared to the case of random assignment of keys. As more nodes are compromised, the gain starts to decline since it becomes more difficult to defend the network under such high level of attacks.

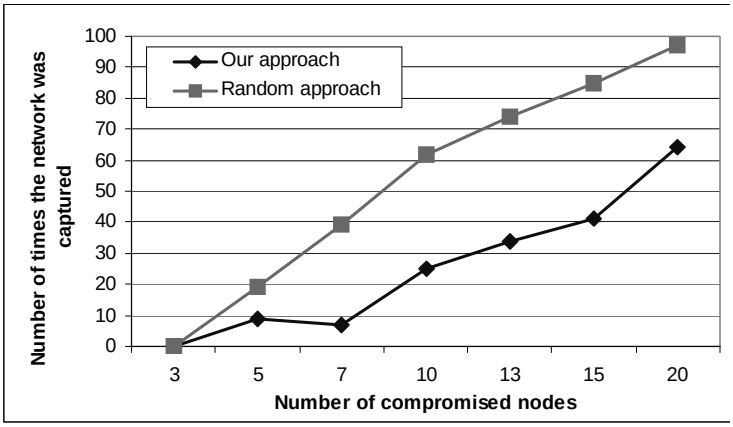

Fig 7: The number of times, during 100 runs, the network was captured due to collusion for varying levels of node compromises

4. Conclusion

EBS provides an efficient way to manage keys in WAHNs allowing the trade-off between the number of keys stored per node and the number of re-key messages. The basic EBS scheme, however, is prone to collusion attacks that can lead to capturing the entire network. In this paper, we have analyzed the problem of collusion in WAHNs, in particular when EBS is employed, and proposed a solution for diminishing the threat posed by collusion. We have shown that optimal assignment of key combinations to nodes in order to limit the potential of capturing the network through collusion among compromised nodes is a classical resource allocation problem and is NP-hard in nature. We have introduced a novel heuristic for key assignment that decreases the probability of capturing the network. Our approach exploits the physical proximity of nodes so that a node would share most keys with reachable nodes and thus very few additional keys would be revealed when colluding. Simulation results have demonstrated that our proposed solution significantly boosts the network resilience to collusion threats.

We are currently investigating the applicability of EBS and the presented collusion prevention approach to securing unattended wireless sensor network. Future research directions include extending our algorithm to handle mobile nodes, applying our collusion prevention methodology to other key management solutions and

investigating the support of other factors such as membership duration, member arrival time, and member departure time.

References

[1] C. E. Perkins, *Ad Hoc Networking*. New York: Addison-Wesley, 2001.

[2] S. Giordano. *Mobile ad-hoc networks*. In Handbook of Wireless Network and Mobile Computing, ed. I. Stojmenovic. John Wiley & Sons, 2001.

[3] H. Yang, et al., "Security in Mobile Ad-Hoc Wireless Networks: Challenges and Solutions," *IEEE Wireless Communications Magazine*, February 2004.

[4] L. Zhou and Z. J. Haas "Securing ad hoc networks," *IEEE Networks Special Issue on Network Security,* Vol. 13, No. 6, November/December, 1999

[5] D. Carman, P. Kruus, and B. Matt, "Constraints and Approaches for Distributed Sensor Networks Security," NAI Technical Report #00-010, Sep. 2000.

[6] G. Jolly, et. al, "A Low-Energy Key Management Protocol for Wireless Sensor Networks," in the *Proceedings of the 8th IEEE Symposium on Computers and Communications (ISCC'2003)*, Antalya, Turkey, June 2003.

[7] M. Eltoweissy, H. Heydari, L. Morales, H. Sadborough., "Combinatorial Optimization of Key Management in Group Communications," *Journal of Network and Systems Management: Special Issue on Network Security*, March 2004.

[8] M. Moharram, R. Mukkamala, and M. Eltoweissy, "TKGS: Threshold-based Key Generation Scheme for Wireless Ad Hoc Networks," in the *Proceedings of the IEEE International Conference on Computer Communications and Networking (ICCCN'2004)*, October 2004.

[9] R. Mukkamala, M. Moharram, and M. Eltoweissy, "A Novel Architecture for Secure Group Communication in Wireless Ad-Hoc Networks with Application-Level Multicast," in the *Proceedings of Workshop on Trusted Internet*, Bangalore, India, December 2004.

[10] M. Eltoweissy, M. Younis and K. Ghumman "Lightweight Multi-Granularity Key Management for Secure Wireless Sensor Networks," in the *Proceedings of the IEEE Workshop on Multi-hop Wireless Networks (MWN'04)*, Phoenix, Arizona, April 2004.

[11] P. Briggs, K. Cooper, K. Kennedy, and L. Torczon. Coloring heuristics for register allocation. In *ASCM Conference on Program Language Design and Implementation*, pages 275-284, 1989.

[12] F. C. Chow and J. L. Hennessy. The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems*, 12(4): 501-536, 1990.

[13] R. Diestel, Graph Theory, 2nd Edition, Springer-Verlag, New York, February 2000.

[14] S.Rafaeli and D. Hutchison. A survey of key management for secure group communication, *ACM Computing Surveys, Volume 35, Issue 3, Pages: 309 - 329*, 2003.

[15] H. Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification," RFC 2093, 1997.

[16] D. Wallner, E Harder, R. Agee, "Key Management for Multicast: Issues and Architectures," RFC 2627, 1999.

[17] K. Wong, M. Gouda, and S. Lam "Secure group communications using key graphs," *IEEE/ACM Transactions on Networking*, Vol 8, No. 1, pp. 16-30, February 2000.

[18] R. Canetti, T. Malkin, and K. Nissim, "Efficient communication-storage tradeoffs for multicast encryption," in Advances in Cryptology- EUROCRYPT '99, J. Stem, Ed. Lectures Notes in Computer Science, vol. 1599. Springer-Verlag, New York, pp. 459-474, 1999.

[19] A. Perrig, D. Song, and J. Tygar, "ELK, A new protocol for efficient large-group key distribution," in the *Proceedings of the* IEEE Symposium on Security and Privacy, Oakland, California., May 2001.

[20] B. Brisco, "MARKS: Multicast key management using arbitrarily revealed key sequences," in the *Proceedings of the 1st InternationalWorkshop on Networked Group Communication*, Pisa, Italy, November 1999.

[21] S. Setia, S. Koussih, and S. Jajodia, "Kronos: A scalable group re-keying approach for secure multicast," in the *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland California, May 2001.

[22] C. Boyd, "On key agreement and conference key agreement," *in Proceedings of the Information Security and Privacy: Australasian Conference*, Lecture Notes in Computer Science, vol. 1270. Springer-Verlag, New York, 294- 302, 1997.

[23] O. Rodeh, K. Birman, and D. Dolev, "Optimized group rekey for group communication systems", *Network and Distributed System Security*, San Diago, CA, 2000.

[24] C. Duma, N. Shahmehri,and P. Lambrix, "A Hybrid Key Tree Scheme for Multicast to Balance Security and Efficiency Requirements," in the *Proceedings of the 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Austria, June 2003.

[25] C. Karlof and D. Wagner, “Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures,” *Elsevier's AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols*, vol. 1, pp 293-315, 2003.

[26] L. Eschenauer and V. Gligor, “A Key Management Scheme for Distributed Sensor Networks,” in the *Proceedings of the 9th ACM Conference on Computing and Communication Security*, November 2002.

[27] H. Chan, A. Perrig, and D. Song, “Random Key Pre-distribution Schemes for Sensor Networks,” in the *Proceedings of IEEE 2003 Symposium on Security and Privacy*, Berkeley, CA, May 2003.

[28] D. Liu and P. Ning, “Establishing pairwise keys in distributed sensor networks," in the *Proceedings of the ACM Conference on Computer and Communications Security (CCS '03)*, pages 52--61, 2003.

[29] W. Du, J. Deng, Y.S. Han, P.K. Varshney, “A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks,” *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, Washington D.C., October 2003.

[30] J. Tavernier and M.Eltoweissy, “Effects of Keying Protocols on In-Network Processing in Wireless Sensor Networks,” in the *Proceedings of the IEEE Workshop on Algorithms for Wireless and Mobile Networks (A-SWAN 2004)*, Boston, August 2004.

[31] S. Zhu, S. Setia, and S. Jajodia, “LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks,” *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, Washington D.C., October 2003.

[32] S. Yi and R. Kravets, “MOCA: Mobile Certificate Authority for Wireless Ad Hoc Networks,” in the *Proceedings of the 2nd Annual PKI Research Workshop (PKI 03)*, April 2003.

[33] G.Xu and L. Iftode, “Locality Driven Key Management Architecture for Mobile Ad-Hoc Networks,” *Proc.of the 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2004)*, Ft. Lauderdale, FL, October 2004.

[34] S. Zhu, S. Xu, S. Setia, and S. Jajodia, "Establishing pair-wise keys for secure communication in ad hoc networks: A probabilistic approach," George Mason University, Tech. Rep. ISE-TR-03-01, March 2003.

Appendix A

The pseudo-code for the key assignment algorithm is shown in Figures 8 and 9. The "CreateTree" procedure sets the breadth-first parsing and assigns keys according to the methodology described earlier in the paper. The "swap" procedure handles the cases when the hamming distance between the key combinations of two neighbors is not minimal. The following is a definition of the used notation:

N	:	Total number of nodes in the system
AssignComb(node)	:	Assign an unused key combination to the node
Visited(node)	:	Mark node as visited after assigning keys to all its children
neighbor(i, j)	:	Return 1 when nodes i and j are linked
sibling(i, j)	:	Return 1 when nodes i and j are siblings on the breadth first parsing tree
Swapped_before(i, j)	:	Return 1 when nodes i and j are swapped at earlier iteration
Keys[node]	:	Keys assigned to '*node*'
Adjacency[N][N]		Adjacency matrix of the sensor nodes
CollMatrix[N][N]	:	Matrix that tracks the impact of collusion among nodes. It is initialized such that *CollMatrix[i][j]= Keys[i]* $1 \geq j \geq N$

Procedure CreateTree

```
1.  Initialize  root = node with largest number of neighbors;
2.  key_comb(root) = CK_root
3.  Push(root);
4.  While (Stack is not empty)
5.      parent = Pop();
6.      IdentifyChildren(parent);
7.      SortChildren(parent); /* Sorts the child nodes in non-decreasing order
                                 of their distance from  the parent */
8.      Visited(parent);
9.      For every child node i of parent
10.         If (i is not visited)
11.             Push (i);
12.             If (no combination is assigned to i)
13.                 AssignComb(i);
14.             end if
15.             Swap(i);
16.         end if
17.     end for
18. end while
End
```

Fig. 8: Pseudo-code for key assignment algorithm

Procedure Swap(node)

```
1.  min_hd = 2;            /* Minimum possible hamming distance between any two key combinations */
2.  swapflag = false;
3.  current_parent = Parent(node);
4.  current_distance = Ham_dist(current_parent, node);
5.  If (current_distance = min_hd)
6.      return
7.  If (node is visited) // tough case we have to avoid messing up
8.      Try swap with j | (j is visited) & (sibling(j, node) = true) & (neighbor(j, node) = true )
9.      If (swapflag = false) // This is harder since the node is trying a non-neighbor
10.         Try swap with j | (j is not visited) & (sibling(j, node) == true)
11.         If (swapflag = true)
12.             return  // we did it.
13.     else
14.         return  // we did it.
15. else  // node is not visited so we can do more
16.     KC = unused key combinations with hamming distance to current_parent = min_hd
17.     If (KC is null)  // cannot find a free combination that fits
18.         Try swap with j | (j is not visited) & (sibling(j, node) = true) &
                        (neighbor(j, node) = true ) & swapped_before(j, node) = false)
19.         If (swapflag = false)  // let try a harder choice
20.             Try swap with j | (j is visited) & (sibling(j, node) = true) & (neighbor(j, node) = true )
21.             If (swapflag = true)
22.                 return  // we did it.
23.         else
24.             return  // we did it.
25.     else
26.         Key(node) = KC;     // found a better combination, let's use it.
27.         return
28.     end if
29. end if
30. // swapflag = false → cannot swap the child let us try to work it out through the parent
31.  KC = unused key combinations with same hamming distance to parent(current_parent)
32.  If (KC is not null)
33.      If (Ham_dist(KC, node) < current_distance &
              Ham_dist(KC, and other children(current_parent) is not larger)
34.          Key(current_parent) = KC;  // let change it
35.          return
36.  else  // let try to swapping the parent
37.      Try swap with j | (j is not visited) & (sibling(j, current_parent) = true) &
              (neighbor(j, current_parent) = true ) & swapped_before(j, current_parent) = false)
38.      If (swapflag = false)
39.          Try swap with j | (j is visited) & (sibling(j, current_parent) = true) &
                  (neighbor(j, current_parent) = true )
40.          If (swapflag = false)
41.              Try swap with j | (j is non visited) & (sibling(j, current_parent) = true)
42. end if
43. return
End
```

Fig. 9: Pseudo-code for the swap procedure

Appendix B

The pseudo-code for the collusion assessment procedure is shown in Fig. 10. For every node *i*, we check whether the union of the keys of node *i* and each of its neighbors *j* will yield all the keys used in the system. If this is the case, the system is said to be prone to capture via two nodes. If not, we go one step further by finding all the neighbors of node *j* and finding the union of the keys of node *i*, node *j* and a new neighbor *k* ($\neq i$) of *j*. We then investigate whether the network can be captured via three nodes. This procedure is carried out until the network is captured or we have found the union of the keys of *N* nodes, whichever comes first.

```
Procedure AssessCollusion
1.  Collusion = false
2.  NodesToCollude = 2                    /*Minimum of two nodes needed for collusion*/
3.  While (!Collusion)
4.     For (i = 1; i <= N; i++)
5.        For (j = 1; j <= N; j++)
6.           If (Adjacency[i][j] == 1)
7.              For every neighbor k of j
8.                 If (Adjacency[j][k] == 1)
9.                    CollMatrix[i][k] = Union (CollMatrix[i][j] , CollMatrix[j][k]);
10.       end for
11.    end for
/* Now checking if all keys used have been compromised*/
12.    For (i = 1; i <= N; i++)
13.       For (j = 1; j <= N; j++)
14.          NumK = number of 1's in CollMatrix[i][j];
15.          If (NumK == Number of keys used in system)
16.             Collusion = true;
17.          else NodesToCollude++;
18.       end for
19.    end for
20. end while
21.  return (NodesToCollude);
End
```

Fig. 10 Pseudo-code for the collusion assessment procedure