IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 13, 2016, accepted July 10, 2016, date of publication August 3, 2016, date of current version August 26, 2016.

Digital Object Identifier 10.1109/ACCESS.2016.2594478

An Adaptive Early Node Compromise Detection Scheme for Hierarchical WSNs

AHMED AL-RIYAMI, NING ZHANG, AND JOHN KEANE
School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K.
Corresponding author: A. Al-Riyami (ahmed.al-riyami@manchester.ac.uk)

ABSTRACT Node compromise attacks pose a serious threat to wireless sensor networks (WSNs). To launch an attack, an adversary physically captures a node and access data or software stored on the node. Even worse, the adversary may redeploy the captured node back into the network and use it to launch further attacks. To reduce the impact of a node compromise attack on network operations, the network should detect a node compromise as early as possible, ideally soon after a node is being captured, and then isolate the node from future network communications. Solutions for early node compromise detection are based on distributed monitoring of neighboring nodes' aliveness. Nodes regularly send notification (*Heartbeat*) messages to their one-hop neighbors to indicate their aliveness. If no message is received from a node (i.e., if a node is not heard) for a certain period of time, then the unheard node is said to have been compromised. This approach may have a large number of false positive errors when the message loss ratio in the network is high, as missing messages could be caused by message loss during transmission, in addition to node compromises. This paper proposes a novel scheme, called an adaptive early node compromise detection scheme, to facilitate node compromise attack detection in a cluster-based WSN. The scheme is designed to achieve a low false positive ratio in the presence of various levels of message loss ratios. To achieve this feature, two ideas are used in the design. The first is to use cluster-based collective decision making to detect node compromises. The second is to dynamically adjust the rate of notification message transmissions in response to the message loss ratio in the sender's neighborhood. The performance of the scheme, in terms of false positive ratio, false negative ratio, and transmission overheads, is evaluated using simulation. The results are compared against those from the most relevant scheme in the literature. The comparison results show that our scheme can detect all the node compromises in the network more effectively and efficiently, regardless of the message loss ratio in the underlying environment.

INDEX TERMS Node compromise attack, adaptive, detection, wireless sensor networks.

I. INTRODUCTION

Due to the unattended nature of WSNs, sensor nodes (hereafter referred to as nodes) are prone to physical node compromise attacks [1]. To launch an attack, an adversary first locates a node, and then, physically captures it. The adversary may access the data stored on the node, and/or modify its software before redeploying it back into the network. After the redeployment, the adversary may use the node to launch further attacks. To reduce the impact of such attacks or to prevent further attacks on the network from being launched via this node, the network should detect a node compromise as early as possible, ideally soon after a node is being captured, and then isolate the node from future network communications.

Solutions for early node compromise detection are based on distributed monitoring of neighbouring nodes' aliveness. In these solutions, nodes regularly send notification messages (usually called *Heartbeat* [2], *Beacon* [3], or *Hello* [4], [5] messages) to their one-hop neighbors to prove that they are alive (not being captured). If no message is received from a node (i.e., if a node is not heard) for a certain period of time, then the unheard node is said to have been compromised. This approach leverages the intuition that a compromised node would not be able to take part in network communications for a certain period of time (a threshold time). This threshold time represents the minimum length of time required by an adversary to compromise a node before its redeployment. However, this approach may result in detection errors when the message loss ratio (lost messages / transmitted messages) in the network is high, as missing messages could be caused by message loss during transmission (due to transmission errors or collisions in the wireless channel), in addition to

node compromises. The message loss during transmission is related to the issue of message delivery reliability. There are a number of factors that may affect message delivery reliability in a wireless network, even in a single-hop communication scenario [6], [7]. These factors can be classified into two groups, channel-based and traffic-based. Examples of channel-based factors include radio wave propagation effects (large-scale shadowing and/or small-scale fading), interference (caused by concurrent transmissions and/or by electronic devices) and radio transceivers internal noise that may affect the strengths of sent and received signals. Examples of traffic-based factors include message drops by the receiver typically caused by buffer overflows and message collisions caused by simultaneous transmissions. If a notification message is lost due to any of the channel-based and/or traffic-based factors and if this loss triggers a decision being made that the sender (node) has been compromised, then this decision is false, which means that a detection error (a false positive error) has occurred. The higher the message loss ratio in the network, the more the false positive errors will be generated.

To reduce the number of false positive errors, the authors in [3] and [4] have proposed to increase the rate at which notification messages are transmitted by the nodes such that a monitoring node only declares that a monitored node is compromised if it has missed out a specified number (a threshold) of notification messages from the monitored node. In these proposed solutions, the rate at which notification messages are transmitted (i.e., notification message transmission rate) is fixed during the entire lifetime of a node involved in the network operations and also fixed for all the nodes in the network. However, using a fixed notification message transmission rate at all times and by all the nodes in the network may not be appropriate for WSNs. This is because a WSN may be deployed in an indoor, an outdoor or a mixture of these environments. Each such environment may have its own characteristics, which may lead to different message loss ratios. Indeed, it has been shown that radio link quality often fluctuates over space and time [7]. In such an environment, if we use a fixed notification message transmission rate over space and time, then there is an issue as to at what value this rate should be set. If we set the notification message transmission rate to a higher value, then, in an environment with a lower message loss ratio, nodes will transmit more messages than necessary, which will not only lead to a higher level of network traffic, increasing the chance of message collisions, but also cause the nodes to deplete their energy more quickly, reducing the network lifetime. On the other hand, if we set the notification message transmission rate to a lower value and if the environment has a higher message loss ratio, then there will be more compromised node detection errors (i.e., a higher false positive ratio). In other words, there is a trade-off between costs (communication and energy costs) and the detection errors in node compromise detection. Therefore, we hypothesise that, by setting the notification message transmission rate dynamically in response to the message loss ratio in the underlying environment, we may be able to balance this trade-off, i.e., to achieve a low false positive ratio in node compromise detection with as few notification messages as possible. To verify this hypothesis, this paper presents a novel scheme, called an Adaptive Early Node Compromise Detection (AdaptENCD) scheme, that dynamically adjusts the notification message transmission rate in adaptation to the message loss ratio in the underlying communication environment. The AdaptENCD scheme is evaluated through simulation studies, and the evaluation results are compared against the results from the most related solution in literature.

The rest of the paper is structured as follows: related work is discussed in Section II; Section III discusses the properties of AdaptENCD; Section IV presents design preliminaries; the AdaptENCD scheme is described in Section V; Section VI analyses and evaluates the scheme; Section VII suggests measures to further reduce transmission overheads; finally, Section VIII concludes the paper.

II. RELATED WORK

This section gives an overview of the related work and a detailed description of a design published in literature, which is most relevant to our work and will be used to compare our design against.

A. OVERVIEW

A node compromise attack is carried out in three stages [8]: (Stage_1) physically capturing a node, (Stage_2) redeploying the captured node back to the WSN, and (Stage_3) trying to let the node re-join the network to launch further attacks. The attack had been deemed as easy to perform but difficult to detect [9]. However, Bacher et al. [2] have experimentally discovered that node compromise attacks are not as easy to perform as reported previously, provided that some basic precautions are taken, such as disabling interfaces that may be exploited by an adversary to gain access to the node's microcontroller. The effort required by an adversary to successfully launch a node compromise attack may vary from some mechanical work such as (de-)soldering to more advanced invasive attacks on the electronic components of a node using costly equipment. These attacks require the removal of the compromised node from the network for a substantial amount of time.

There have been many solutions proposed in the literature to detect node compromise attacks [3]–[5], [8], [10]–[30]. Depending on the stage of the attack at which node compromise detection is implemented, the solutions can be classified into three groups: (Group_1) node compromise detection is implemented at Stage_1, (Group_2) node compromise detection is implemented at Stage_2, and (Group_3) node compromise detection is implemented at Stage_3.

Earlier solutions have been mostly of Group_2 or Group_3. The Group_2 solutions are based on verifying the physical location of each node. They assume that, if a node has been compromised, it would be hard for an adversary to redeploy

the compromised node back to its exact previous physical location. In other words, if a node's physical location is changed, then the node has probably been compromised. For example, in the solution proposed by Song et al. [8], a compromised node is detected by verifying if a node's physical location has been changed.

The Group_3 solutions use either software attestation methods [10]–[16] or misbehaviour detection methods [17]–[30]. Software attestation methods are based on the observation that an adversary, once successfully compromised and redeployed a node back into the network, will launch further attacks via this node. For doing so, the adversary needs to modify the software on the node. So these methods verify the integrity of the software installed on the node. The misbehaviour detection methods assume that if a node has been compromised, then it may behave maliciously. Therefore, by monitoring the behaviour of the nodes at runtime, malicious nodes could be detected.

Recently, some Group_1 solutions have been proposed. These solutions detect compromised nodes based on the assumption that, if a node is compromised, then it would not be heard by other nodes in the neighbourhood for a period of time. Though there are different variants of these solutions, they largely rely on the use of notification messages as discussed in Section I.

Lin [3] proposed the first such scheme that uses notification (*Beacon*) messages. The scheme is called Couple-based Node Compromise detection (CAT). CAT allows any two nodes to form a couple in an ad hoc manner and the nodes of the same couple monitor each other to detect any node compromise. CAT also assumes that each node can self-detect if it is connected to a programming board. If a node has detected that it is connected to a programming board, then the node announces itself as compromised and sends a message to its partner to inform it about the attack. To address *Beacon* message loss, each monitoring node allows three *Beacon* messages to be missed before declaring that the monitored node is compromised. Although this method reduces false positive errors, if both nodes in a couple are compromised at the same time, then no detection can be achieved.

Ding et al. [4] proposed two schemes, First Stage Detection (FSD) and Sink Enhanced FSD (SEFSD). Both schemes rely on the use of *Hello* and probe messages. Each node sends *Hello* messages periodically to its neighbours to indicate that it is alive. If a (monitoring) node does not receive three consecutive *Hello* messages from a neighbour (monitored node), the monitoring node sends two probe messages successively. If no reply to the probe messages is received, then the monitoring node concludes that the neighbour is compromised. The two schemes differ in how they respond once a compromised node has been detected. In FSD, the monitoring node broadcasts a message to the entire network to notify the ID of the compromised node. In SEFSD, the message is only sent to the base station (BS). In this latter scheme, upon the receipt of multiple notifications about the compromise from multiple monitoring nodes, the BS notifies all the nodes in the network. It does so periodically using an integrated message that contains a list of reported compromised node IDs. SEFSD is more efficient than FSD as the former uses a centralised and resource-rich BS to facilitate the revocation of a detected compromised node from the network.

Megahed et al. [5] proposed an efficient Group_1 scheme to detect node compromises in cluster-based WSNs. In this scheme, a WSN is divided into a set of interconnected clusters. The clusters form a chain of interconnected rings, and each ring is a cluster. Each node in a ring sends one *Hello* message to its two neighbours in the ring and receives one *Hello* message from each of the neighbours at every threshold period. If a node in the ring is compromised (i.e., if it has not sent a *Hello* message within the threshold period), the chain breaks and other interconnected rings will discover the compromise. Although this scheme is more efficient than the schemes discussed earlier, it does not address issues caused by message loss during transmission. The scheme may suffer a large number of false positive errors in an environment with a high message loss ratio.

To summarise, among the solutions discussed above, only CAT, FSD and SEFSD are designed to detect node compromises in Stage_1 of the attack and in an environment with message loss. As SEFSD is more effective than CAT and more efficient than FSD, here we choose SEFSD as the reference solution to evaluate our solution (AdaptENCD). In other words, we will compare the performance of AdaptENCD against that of SEFSD to demonstrate the effectiveness of AdaptENCD. In the next section, we describe SEFSD in more detail.

B. SINK ENHANCED FIRST STAGE DETECTION (SESFD)

The SESFD scheme assumes that each node in a network has a unique ID and that all nodes are static and are preloaded with a common key list that is known to, and shared by, all nodes. SEFSD has two functions: (1) to detect node compromises and (2) to revoke the detected compromised node from the network. The detection function is carried out in Stage_1 of the attack and is carried out in a distributed manner by using *Hello* and probe messages. The revocation function is carried out centrally with the involvement of the BS, i.e., the BS manages the revocation of any compromised nodes from the network. These functions are implemented by using eight types of messages (*Hello*, *Setup*, *MyPath*, Are you there? *AYT*, I am fine *IMF*, *Captured*, I am captured *IMC* and *AlertUpdate*). The operations of SEFSD are in two phases, a *Network Setup* phase and an *Operational* phase as described below.

1) NETWORK SETUP PHASE

In this phase, each node discovers its respective neighbours and establishes a path to the BS. For doing so, each node maintains a *Neighbour Table* recording the IDs of its one-hop neighbours. To discover neighbours, each node broadcasts *Hello* messages periodically. Each *Hello* message contains the ID of its sender. Upon the receipt of a *Hello* message,

the receiver extracts the ID of the sender and stores it in its Neighbour Table. Once the neighbours are discovered, each node establishes a path to the BS. This path discovery process is initiated by the BS by broadcasting a *Setup* message to the entire network. Each node, upon the receipt of the *Setup* message, adds its own ID into the received message and then rebroadcasts it to its downstream neighbours. During this process, each node may receive multiple copies of the *Setup* message, and the different copies may contain different sets of node IDs, as they may traverse different paths. Based on the path information contained in the copies of the *Setup* message, each node selects the shortest path to the BS and sends a *MyPath* message to its selected parent on the shortest path. The parent node then forwards the message on to its parent until the message reaches the BS. By the end of the Network Setup phase, each node should have learnt its parent and children.

2) OPERATIONAL PHASE

In this phase, each node performs the two functions, the detection of node compromises and revocation of the detected compromised nodes. The two functions are described below in detail.

With the detection function, each node in the network detects if any node in its neighbourhood (one-hop neighbours) or itself has been compromised. To achieve this, each node periodically sends *Hello* messages to its one-hop neighbours to indicate that it is alive (i.e., has not been compromised). The *Hello* messages sent by all the nodes in the network during a specific time period are encrypted using the same key drawn from the common key list stored on each node. To reduce false positive errors that may be caused by message loss, the scheme requires that a monitoring node should miss out three consecutive *Hello* messages from a monitored node before taking any further action on the monitored node. That is, if a monitoring node misses out three consecutive *Hello* messages from a monitored node, the monitoring node should send two successive *AYT* probe messages to the monitored node. Any node, upon the receipt of an *AYT* message, should reply with an *IMF* message. If the monitoring node does not receive any reply after sending two consecutive *AYT* probe messages, it will conclude that the monitored node is compromised and send a *Captured* unicast message (containing the ID of the compromised node) to the BS using the path established in the Network Setup phase. SESFD also allows a node to detect if the node itself has been compromised, i.e., self-detection. For example, if it detects any tampering with its hardware, then it should send an *IMC* message to report the compromise to its neighbours. A neighbour receiving the *IMC* message should report the compromise to the BS using a *Captured* message.

For the revocation function, the compromised nodes notified in the *Captured* messages sent by the monitoring nodes are revoked with the assistance of the BS. This is accomplished by the following operations. Firstly, upon the receipt of *Captured* messages, the BS periodically (every T period) floods the network with an *AlertUpdate* message. The *AlertUpdate* message contains the IDs of all the compromised nodes indicated in the *Captured* messages received in the last T period. Secondly, in each period, the BS should also send an updated key list to all the nodes in the network. This is to thwart further possible attacks by making use of any exposed cryptographic materials from a compromised node. Upon the receipt of an *AlertUpdate* message and the updated key list, each node adds the IDs of the compromised nodes into a *Captured Node List* maintained by the node and updates its key list. From this point on, each node should discard any messages sent from a compromised node with an ID matching with any listed in the Captured Node List.

SEFSD is constrained in terms of the timing of the detection and revocation of any compromised nodes. To describe the timing constraint, we here define a detection and revocation time duration (td_{dr}) which is the duration from when the first of the three consecutive *Hello* messages is sent to when an *AlertUpdate* message is received. This duration covers the total time taken for detecting a compromised node (sending three *Hello* and two *AYT* messages), for reporting the compromised node (sending a *Captured* message) to the BS and for revoking the compromised node (sending an *AlertUpdate* broadcast by the BS). For SEFSD to effectively detect and revoke a compromised node, the value of td_{dr} should not be greater than the minimum time required by an adversary to compromise a node. Otherwise, it is possible that a compromised node is redeployed back into the network and used as a springboard for further attacks.

Although SEFSD provides a solution to address the false positive issue caused by message loss commonly seen in a WSN environment, it suffers from a number of limitations. Firstly, it is not suitable for environments characterised by a high message loss ratio as it is likely that the nodes will miss the *Hello* and probe messages resulting in false positives. Secondly, the scheme is not scalable as the larger the network, the longer the td_{dr} value, and if a network is sufficiently large, the td_{dr} value may exceed the time required to compromise a node. Thirdly, the scheme requires updating the key list installed on each node upon the detection of each node compromise. This is a costly process if it has to be updated through the BS.

III. AdaptENCD PROPERTIES

AdaptENCD is designed to address the shortcomings of the SEFSD scheme described in Section II-B. It leverages a cluster-based hierarchical topology and provides distributed early node compromise detection and revocation functions. These functions allow member nodes in each cluster to monitor each other to detect node compromises and to revoke compromised nodes from the cluster immediately after detection. The compromised nodes will later be revoked from the entire network in a centralised manner with the involvement of the BS. Similar to other early node compromised detection schemes, AdaptENCD uses periodic notification (*Heartbeat*, or in short, *HB*) messages.

The fundamental difference between AdaptENCD and SEFSD is that AdaptENCD allows the use of different *HB* message transmission rates for different clusters. A local *message loss ratio* (MLR) value is estimated based on the channel conditions in each cluster and this MLR value is used to adjust the *HB* message transmission rate used by the members of the cluster. So if a cluster has a weaker channel condition, the members in the cluster should use a higher *HB* message transmission rate. Similarly, if a cluster has a stronger channel condition, the members in the cluster should use a lower *HB* message transmission rate. In this way, we can reduce false positive detection errors, while, at the same time, keep communication overheads caused by *HB* message transmissions as low as possible. In detail, AdaptENCD has the following features:

1) A reliable and efficient node compromise decision process. AdaptENCD uses a *cluster-based collective decision making approach* to node compromise detection. Unlike SEFSD in which a monitoring node makes a node compromise decision based on a single piece of evidence, i.e., *Hello* and probe messages received from the monitored node, the decision in AdaptENCD is made based on a collection of evidence received from all in-cluster neighbours of the monitored node. Message (or transmission) redundancy can help reduce false positive errors, and the use of clusters allows harvesting redundant messages without requiring the monitored node to repeat *HB* message transmissions. This can make the decision process more reliable and efficient.
2) An *adaptive approach to HB message transmission rate selection*. This allows different clusters to use different *HB* message transmission rates, determined based on the underlying channel conditions in their respective clusters. In this way, the trade-off between false positive errors and transmission overheads introduced in detecting node compromises can be balanced. The underlying channel conditions in each cluster is measured in terms of MLRs. A computationally efficient method has been devised to estimate an MLR value in a cluster; each cluster head (CH) uses message counter values carried in each transmitted message to estimate an MLR value.
3) To effectively revoke compromised nodes, we separate duties in AdaptENCD's key establishment structure, such that different types of communications (unicast, broadcast, intra-cluster, and inter-cluster) are secured using different keys. AdaptENCD takes advantage of this key structure to revoke compromised nodes locally from the cluster first, and then from the entire network.
4) A built-in method to rank cluster members (CMs) based on the *Received Signal Strength Indicator (RSSI)*. The top-ranked CM will serve as the deputy CH (DCH) for the cluster. A DCH can detect the compromise of the CH in a cluster, and in an event when the CH is detected as compromised, the DCH will assume the role of the CH. This allows the revocation of any detected compromised node in a cluster, including the CH, without affecting the cluster's operations.

In the rest of the paper, we give detailed descriptions of the design, implementation and evaluation of the AdaptENCD scheme.

IV. DESIGN PRELIMINARIES

This section provides definitions, system and threat models, assumptions and requirement specifications used in the design of AdaptENCD.

A. DEFINITIONS AND NOTATIONS

The design of AdaptENCD uses the following detection, efficiency and time parameters.

- Detection parameters:
  - *False positive*: A non-compromised node identified as compromised.
  - *False negative*: A compromised node that is undetected.
  - *False positive ratio (FPR)*: Proportion of nodes identified as compromised while they are not, i.e., FPR = number of false positives / (total number of nodes - number of compromised nodes)
  - *False negative ratio (FNR)*: Proportion of compromised nodes that are undetected, i.e., FNR = number of false negatives / number of compromised nodes.
- Efficiency parameters:
  - *Monitoring overheads (MO)*: The total number of messages generated in the WSN for monitoring node compromises during a specific period of time.
  - *Compromise reporting and revocation overheads (CRRO)*: The total number of messages generated in the WSN for reporting and revocation of compromised nodes during a specific period of time.
- Time parameters:
  - *Node compromise minimum duration* td_{min}: This is the minimum time duration required by an adversary to compromise a node. This duration also represents the length of time the node is not present in the network, from when the node is captured by the adversary to when the node is redeployed after the compromise.
  - *Node compromise detection duration* td_{det}: This is the time duration within which a cluster should detect the compromise of a CM. This duration represents the length of time from when the node is captured to when it is declared as compromised. td_{det} is set as half td_{min}, i.e., $td_{det} = td_{min}/2$.
  - *Compromised node local revocation duration* td_{loc}: This is the maximum time duration within which a cluster should locally revoke a CM after being captured. This duration represents the length of time from when the CM is captured to when it is locally revoked from the cluster.

- *Compromised node global revocation duration* (td_{glo}): This is the maximum time duration within which the entire network should revoke a node after being captured. This duration represents the length of time from when the node is captured to when it is globally revoked from the entire network.
- *Clustering interval*: This interval starts from the end of a clustering process to the beginning of the next clustering process. A clustering process involves forming clusters and electing a CH for each cluster.
- *Clustering duration* (td_{clu}): This is the time duration of a clustering interval.
- *Heartbeat duration* (td_{HB}): This is the time duration of a *HB round*. In a HB round, every CM in a cluster broadcasts a *HB* message to other in-cluster neighbours.

The notations used in this paper are summarised in Table 1.

B. SYSTEM MODEL

The WSN, under our investigation, consists of a large number (around 500) of static resource-constrained nodes and a resource-rich BS. The nodes are organised into a hierarchical clustered topology in which the nodes are partitioned into clusters and each cluster has an elected CH and a set of CMs. CMs in a cluster are only allowed to communicate with their respective CH and other CMs in the same cluster. Communication outside a cluster is done through the CHs of the communicating nodes. CHs should be able to reach the BS through established routes involving other CHs that are located closer to the BS. Once clustering is established, no further cluster joining requests are processed.

To allow for immediate local revocation of a detected compromised node and to strengthen security, each node *i* has five cryptographic keys, detailed as follows:

- A *Network Key* (k_N): This is the network-wide key shared between the BS and all nodes in the network. It is used to secure messages broadcast by the BS.
- An *Individual Key* (k_{Bi}) shared with the BS: This key is used to secure pairwise unicasts between node *i* and the BS.
- A *Broadcast Key* (k_i^*) shared with each of its one-hop neighbours: This key is used to secure local broadcasts by node *i* to its neighbours, particularly during clustering/re-clustering.
- A *Pairwise Key* (k_{ij}) shared with each of its one-hop neighbours, *j*: This key is used by node *i* to secure unicast messages to node *j*.
- A *Cluster Broadcast Key* ($k_i^{\odot}$) shared with all the CMs in the same cluster: This key is used to secure local broadcasts to in-cluster neighbours in intra-cluster communications.

k_N and k_{Bi} are preloaded into each node's memory before deployment whereas k_{ij} and k_i^* are established using a secure key establishment method such as the one used in the Energy-efficient Distributed Deterministic Key management scheme (EDDK) [31]. $k_i^{\odot}$ is established during a cluster setup process (see Section V-A3).

TABLE 1. Notations.

Notation	Definition
BS	Base station.
CH	Cluster head.
CM	Cluster member.
DCH	Deputy cluster head, a role given to a backup CM that takes over the role of CH when the current CH is compromised.
MLR	Message loss ratio.
FPR	False positive ratio.
FNR	False negative ratio.
$\mathbb{N}$	Set of all the nodes in the WSN where the total number of nodes $\|\mathbb{N}\| = N$.
T_i	Neighbour Table of node i.
V_i	Monitoring Data Vector of node i. It contains monitoring data records. Each record represents the monitoring data gathered in a specific HB round.
L_i	Compromised Node List of node i. It stores the IDs of compromised nodes.
$\mathbb{C}_c$	Set of all the nodes that are members of a cluster c.
$Enc(k, D)$	Encrypting data D using key k.
$Dec(k, D)$	Decrypting data D using key k.
$\|\|$	Concatenation operator.
$\oplus$	Bitwise XOR operator.
$\wedge$	Bitwise OR operator.
ID_i	Identity of node i.
k_N	Network Key shared by BS and all nodes in the network.
k_{Bi}	Individual Key shared between node i and BS.
k_i^*	Broadcast Key shared between node i and all its single-hop neighbours.
k_{ij}	Pairwise Key shared between node i and node j.
$k_i^{\odot}$	Cluster Broadcast Key shared between CM i and all the CMs in a cluster.
$RSSI_i$	Received Signal Strength Indicator of a message received from a neighbouring node i.
$C_{i\to j}$	Unicast counter. This is the counter of unicast messages sent by node i to node j.
σ_i^U	Number of successfully received unicasts from node i.
$C_{i\to *}$	Broadcast counter of node i. This is the counter of messages broadcast by node i to its neighbours.
σ_i^*	Number of successfully received broadcasts from node i.
$C_{i\to \odot}$	Cluster broadcast counter of node i. This is the counter of messages broadcast by node i to its in-cluster neighbours.
$\sigma_i^{\odot}$	Number of successfully received cluster broadcasts from node i.
td_{init}	An initial time duration required by nodes to accomplish node discovery, Pairwise and Broadcast Key establishment and initial clustering.
td_{min}	Node compromise minimum duration.
td_{det}	Node compromise detection duration.
td_{loc}	Compromised node local revocation duration.
td_{glo}	Compromised node global revocation duration.
td_{clu}	Clustering duration.
td_{HB}	HB duration.
b_i	A bit related to CM i in a sequence of bits forming a monitoring data record.
x	*HB* message transmission rate. It is the number of *HB* messages transmitted by a node within the compromised node detection duration td_{det}, i.e., $x = td_{det}/td_{HB}$.
CH_i^R	A CH-rank given to CM i to serve as a backup CH. CM i is said to have a higher CH-rank than CM j, (where $i \neq j$) if $CH_i^R < CH_j^R$.

As k_N and k_i^* may be exposed to adversaries in a node compromise attack, they have to be updated regularly. To update k_N, the BS generates a new key and unicasts it to every CH located one-hop away from the BS. Upon the receipt of the BS unicast, each CH should (1) broadcast the key to the CMs in the cluster and (2) unicast the key to other CHs downstream. This process continues until all nodes receive the updated key. k_i^* can be updated using EDDK methods [31]. The update is performed as follows. Node i generates a new k_i^* and sends it to each non-compromised neighbour via a unicast message. The message is secured using the pairwise key that node i shares with the respective neighbour.

C. THREAT MODEL

The adversary is assumed to be an active adversary with the aim of capturing nodes to uncover private data stored by the captured nodes. The adversary may also modify the software of the nodes and/or redeploy the nodes back to the network to launch further attacks such as routing attacks (e.g., black hole, gray hole, sybil, wormhole and Hello flood [32]), false data injection, identity replication and passive data gathering. The adversary may also compromise more than one node to launch collusion attacks to breach the security and/or privacy of the network. Therefore, by detecting node compromises at an early stage, such attacks may be impeded. Similar to the assumption used in other early node compromise detection schemes [3], [4], we assume that the adversary is capable of compromising a fraction of the nodes in the WSN. In particular, we assume that the adversary can compromise a maximum of 25% of the nodes in the network. We impose this 25% upper bound on the ground that usually losing a larger proportion of nodes can trigger more serious actions by the underlying intrusion detection system. This is because, when a network loses many nodes, it will become more fragmented and less effective in transmitting data to the BS, which can be easily noted by the support team.

D. ASSUMPTIONS

The following assumptions have been used in the design of AdaptENCD:

(A1) The BS is trustworthy, well protected against physical attacks and always available.

(A2) A node compromise attack is always characterised by a physical node capture and removal from the network for a minimum time duration of td_{min}.

(A3) The network initial operation of node discovery, Pairwise and Broadcast Key establishment and initial clustering processes can be accomplished in a time duration td_{init} that is shorter than the duration required by an adversary to compromise a node (i.e., $td_{init} < td_{min}$). Further re-clustering processes can also be accomplished in a time duration that is much shorter than td_{min}.

(A4) The BS and all nodes in the WSN are loosely time synchronized. By loosely we mean that there exists an upper bound on a synchronisation error that can be tolerated and this bound is known to the BS and every node in the network. For a survey of time synchronization mechanisms, refer to [33].

(A5) Each transmitted unicast or broadcast message contains a unique counter value (i.e., sequence number) that is assigned by the sending node.

(A6) There exists a clustering mechanism whereby nodes are partitioned into clusters and each cluster has an elected CH. It is also assumed that CHs can establish or update routes to the BS whenever required.

(A7) Each node i maintains a *Neighbour Table* T_i. T_i stores the attribute values associated with each of the node's neighbours as shown in Table 2.

TABLE 2. Neighbour table, T_i, maintained by node i.

Node	BS	j	u	...
Pairwise Key (k)	k_{Bi}	k_{ij}	k_{iu}	...
Broadcast Key (k)	k_N	k_j^*	k_u^*	...
Cluster Broadcast Key (k)	–	$k_j^{\odot}$	$k_u^{\odot}$	...
Outbound unicast counter (C)	$C_{i\to B}$	$C_{i\to j}$	$C_{i\to u}$	...
Inbound unicast counter (C)	$C_{B\to i}$	$C_{j\to i}$	$C_{u\to i}$	...
No. of received unicasts (σ^U)	σ_B^U	σ_j^U	σ_u^U	...
Inbound broadcast counter (C)	$C_{B\to *}$	$C_{j\to *}$	$C_{u\to *}$	...
No. of received broadcasts (σ^U)	σ_B^*	σ_j^*	σ_u^*	...
Inbound cluster broadcast counter (C)	-	$C_{j\to\odot}$	$C_{u\to\odot}$	...
No. of received cluster broadcasts (σ^U)	-	$\sigma_j^{\odot}$	$\sigma_u^{\odot}$	...
Received signal strength indicator ($RSSI$) dBm	$RSSI_B$	$RSSI_j$	$RSSI_u$	...
CH-rank (CH^R)	-	CH_j^R	CH_u^R	...

(A8) Each node i maintains a *Monitoring Data Vector* V_i. V_i is used to store and update the monitoring data collected from other CMs. The monitoring data is used in compromise detection decision making. V_i is described in detail in Section V-A1.

(A9) Each node i maintains a *Compromised Node List* L_i. L_i stores the IDs of compromised nodes. The BS maintains a similar list. The list is initially empty.

E. DESIGN REQUIREMENT SPECIFICATIONS

The design requirements for AdaptENCD can be divided into functional, security, node compromise detection performance and efficiency requirements as follows.

- Functional:
  - FUN01: It should take no more than td_{min} to locally revoke a detected compromised node from its cluster (i.e., $td_{loc} \leq td_{min}$).
  - FUN02: It should take no more than td_{clu}, to globally revoke a detected compromised node from the entire network (i.e., $td_{glo} \leq td_{clu}$).
- Security:
  - SEC01: A compromised node should be unable to re-join the network and participate in further

communications once it has been revoked from the network.

- Node compromise detection performance:
  - NCD01: False positive ratio (FPR) should be as low as possible.
  - NCD02: False negative ratio (FNR) should be as low as possible.
- Efficiency :
  - EFF01: Monitoring overheads (MO) should be as low as possible.
  - EFF02: Compromise reporting and revocation overheads (CRRO) should be as low as possible.
  - EFF03: Memory requirements should be as low as possible.

V. THE AdaptENCD SCHEME

This section describes the detailed design of AdaptENCD, including the methods, algorithms and protocols it uses and its operations. The operations of the scheme are in three phases: (1) the *Node Initialization* phase which is a pre-deployment phase, (2) the *Network Setup* phase in which nodes perform neighbour discovery, Pairwise and Broadcast Key establishment and initial clustering processes after their deployment and, (3) the *Operational* phase in which nodes carry out compromise detection and revocation. Section V-A describes the methods, algorithms and protocols, Section V-B describes AdaptENCD operations and Section V-C describes AdaptENCD timing structure.

A. METHODS, ALGORITHMS AND PROTOCOLS

1) HB MESSAGE TRANSMISSION

The *HB* message transmission process used in AdaptENCD is adapted from [34], which is proposed for detecting node failure in cluster-based WSNs.

This process is performed by all CMs periodically every HB round. The duration of a HB round is td_{HB}. In each HB round, every CM in the cluster, including the CH, broadcasts its *HB* message to other CMs in the cluster. A *HB* message, which is sent by a CM in the current HB round, includes the ID of the source CM and the monitoring data the CM gathers in the previous HB round. The monitoring data includes a series of bits that identify the aliveness status of the CMs in the previous HB round. One bit for one CM and the value of each bit indicates the aliveness status of the corresponding CM (i.e., 1 indicates that the monitoring CM has received a *HB* message from the corresponding monitored CM and 0 otherwise). The bits are ordered according to a *CH-rank* which is assigned to each CM to serve as backup CH. In each clustering process, the elected CH is assigned a CH-rank, $CH^R = 1$ and the CM with the highest RSSI is assigned $CH^R = 2$ and called a *deputy CH* (DCH). The rest of the CMs are ranked based on their RSSI values where the CM with the highest RSSI is ranked $CH^R = 3$ and so on (the CH ranking process is detailed in Section V-A.3). So, the first bit in the monitoring data represents the status of the current CH, the second bit represents the status of the DCH, the third bit represents the status of the CM with $CH^R = 3$, and so on. Monitoring data records are stored in a Monitoring Data Vector, V_i, that each node maintains (Fig. 1). The *HB* message transmission process is performed by using the HB protocol (see Algorithm 1).

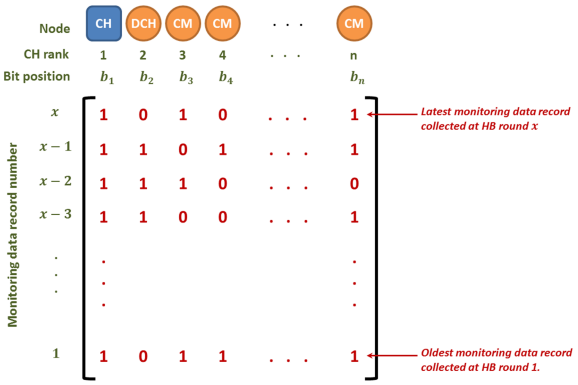

FIGURE 1. Monitoring Data Vector V_i maintained by node i.

Algorithm 1 HB Protocol Algorithm

1: **procedure HB.send**(ID_i)
▷ Executed by the sender CM i every HB round.
2: $D \leftarrow$ **read** monitoring data record related to previous HB round from V_i
3: $Payload_{i\rightarrow *} \leftarrow Enc(k_i^{\odot}, C_{i\rightarrow\odot} \parallel D)$
4: **send** $m_{i\rightarrow *} : ID_i \parallel Payload_{i\rightarrow *}$ ▷ $m_{i\rightarrow *}$ is a local broadcast message
5: **end procedure**

6: **procedure HB.receive**($m_{i\rightarrow *}$)
▷ Executed by the receiver CM j.
7: $C_{i\rightarrow\odot} \parallel D \leftarrow Dec(k_i^{\odot}, Payload_{i\rightarrow *})$
8: **update** T_j **set** $C_{i\rightarrow\odot} =$ received $C_{i\rightarrow\odot}$
9: **update** T_j **set** $\sigma_i^{\odot} = \sigma_i^{\odot} + 1$
10: **update** T_j **set** $RSSI_i = (RSSI_{received-message} + RSSI_i)/2$
11: **update** V_j **set** $b_i = 1$ in monitoring data record related to current HB round
12: **for** each $b_n \in D$ **do**
13: **if** $b_n = 1$ **then**
14: **update** V_j **set** $b_n = 1$ in monitoring data record related to previous HB round
15: **end if**
16: **end for**
17: **end procedure**

The HB protocol has two procedures, HB.send() to broadcast a *HB* message to the in-cluster neighbours and HB.receive() to receive a *HB* message sent by an in-cluster neighbour. In HB.send(), each CM i, reads V_i and extracts the monitoring data record collected in the previous HB round. It then composes a *HB* message that contains a message

counter value and the extracted monitoring data. The message is encrypted using the Cluster Broadcast Key, $k_i^{\odot}$. Next, CM i broadcasts the *HB* message to its in-cluster neighbours.

When an in-cluster neighbour, j, receives the *HB* message sent by CM i, CM j performs HB.receive() procedure. In this procedure, CM j decrypts the received message using the shared key $k_i^{\odot}$ to obtain the message counter value and the monitoring data. Then, CM j updates its Neighbour Table T_j by setting the inbound broadcast counter value of CM i to be equal to the received counter value and the $RSSI_i$ value to be the average of the RSSI value measured by CM j's radio and the value maintained in the table. CM j also updates its Monitoring Data Vector V_j by:

- Updating the monitoring data record of the current HB round: setting the respective position bit related to CM i i.e., $b_i = 1$.
- Updating the monitoring data record of the previous HB round: setting the respective position bit = 1 for any CM that the received monitoring data confirms that it was alive in the previous HB round.

If no *HB* message is received from an in-cluster neighbour during the current HB round, the respective bit of the neighbour is set to 0. Note that a CM can only hear from other CMs that are located within its transmission range. So the bits related to the CMs that are located outside its transmission range are always set to 0.

2) HB MESSAGE TRANSMISSION RATE ESTIMATION

A major benefit of using a cluster-based topology in WSNs is that each cluster may define its own rules locally without affecting the operation of the entire network. Based on this observation, node compromise detection rules can be defined locally within each cluster. In AdaptENCD, each cluster measures its own average message loss ratio, MLR, which reflects the channel conditions in the environment in which each cluster is deployed. Based on the MLR value, the cluster defines the *HB* message transmission rate that is used by all the CMs in the cluster. We use this cluster-dependent approach to determining the *HB* message transmission rate with the intention of reducing false positives in detecting node compromises.

Let us consider a single cluster that consists of n nodes where node i is the CH (Fig. 2). Suppose that the cluster is deployed in an environment where the average MLR = M_L.

A CH decides if a CM in the cluster has been compromised if the following conditions are met:

Condition_1: The CH has not heard the CM in the past x HB rounds.

Condition_2: The *HB* messages received by the CH in the past $(x-1)$ HB rounds confirm that the CM has not been heard by its in-cluster neighbours.

Given M_L, our aim is to determine the number of HB rounds, x, such that the total duration $x \times td_{HB}$ is not greater than the node compromise detection duration, td_{det}, and the CM is not identified as a false positive. Based on this setting,

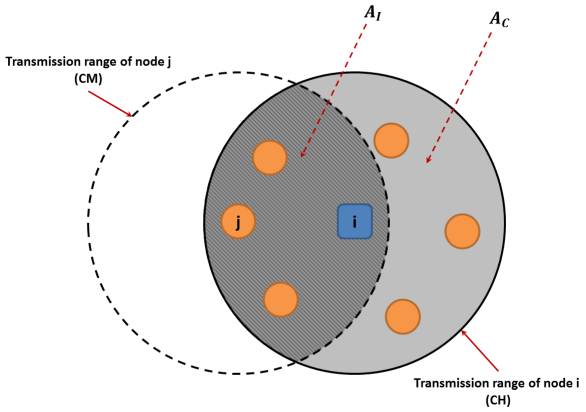

FIGURE 2. Illustration of a cluster with node *i* is a CH.

the probability of a CM being a false positive, P_{FP}, is given by (1).

$$P_{FP} = P_{C1} \times P_{C2} \tag{1}$$

P_{C1} is the probability of fulfilling Condition_1, which can be computed as $P_{C1} = (M_L)^x$, and P_{C2} is the probability of fulfilling Condition_2, computed as $P_{C2} = (M_{L*})^{x-1}$, where M_{L*} is the probability of missing the monitoring data sent by all the neighbours in a single HB round and it can be computed as:

$$M_{L*} = \sum_{k=0}^{n-2} \left(\binom{n-2}{k} \times \left(1 - \frac{A_I}{A_C}\right)^{(n-2)-k} \times \left(\frac{A_I}{A_C}\right)^k \times \sum_{j=0}^{k} \binom{k}{j} \times (1 - M_L)^j \times (M_L)^{k-j} \times (M_L)^j \right) \tag{2}$$

where A_C is the area of the transmission range of the CH, A_I is the area of intersection of the transmission ranges of the CH and another CM (Fig. 2), $\sum_{k=0}^{n-2} \binom{n-2}{k} \times \left(1 - \frac{A_I}{A_C}\right)^{(n-2)-k} \times \left(\frac{A_I}{A_C}\right)^k$ evaluates the probability of having k single-hop neighbours assuming that the nodes are randomly distributed within the cluster, $k \leq n-2$, and $\sum_{j=0}^{k} \binom{k}{j} \times (1 - M_L)^j \times (M_L)^{k-j} \times (M_L)^j$ evaluates the probability of missing the *HB* messages containing the monitoring data sent by the k neighbours.

From (1), P_{FP} can be computed as

$$P_{FP} = (M_L)^x \times (M_{L*})^{x-1} \tag{3}$$

solving for x in (3) yields:

$$x = \frac{\ln\left(P_{FP}\right) + \ln\left(M_{L*}\right)}{\ln\left(M_L\right) + \ln\left(M_{L*}\right)} \tag{4}$$

In the worst case scenario, CM j is at the circumference of the transmission range of the CH as shown in Fig 3.

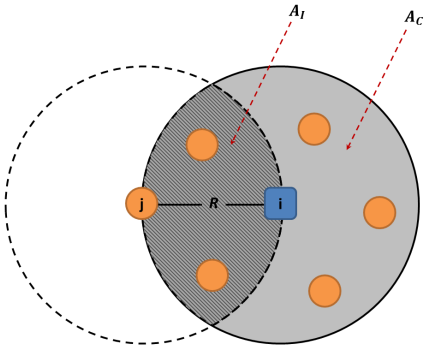

FIGURE 3. The worst-case scenario where CM *j* is at the circumference of the transmission range of the CH.

In this case, based on geometry, we have:

$$\frac{A_I}{A_C} = \frac{2 \times R^2 \times \left(\frac{\pi}{3} - \frac{\sqrt{3}}{4}\right)}{\pi \times R^2} = 0.391 \tag{5}$$

where R is the radius of the transmission range. Therefore,

$$M_{L*} = \sum_{k=0}^{n-2} \left(\binom{n-2}{k} \times (0.609)^{(n-2)-k} \times (0.391)^k \times \sum_{j=0}^{k} \times \binom{k}{j} \times (1-M_L)^j \times (M_L)^{k-j} \times (M_L)^j \right) \tag{6}$$

From this probabilistic analysis, we see that if a CH can measure M_L, in the underlying environment, it will be able to estimate x given a certain level of P_{FP}.

M_L can be measured using message counters that are carried in transmitted messages. This is done as follows. During the neighbour discovery process, nodes send discovery messages at a certain rate for a specific discovery period. Suppose that node i has b neighbours. Upon receipt of the first discovery message from one of the b neighbours, say j, node i adds a new record in its Neighbour Table, T_i, and sets the inbound broadcast counter value $C_{j\to*}$ to be equal to the received counter value carried in the discovery message sent by node j, the number of received broadcasts from node j, σ_j^*, to be equal to 1 and $RSSI_j$ to be equal to the RSSI measured by node i's radio. Upon the receipt of subsequent messages from node j, node i updates node j's record in T_i by updating $C_{j\to*}$, incrementing σ_j^* and averaging the RSSI value of the received message with the value stored in T_i. In a similar way, node i should also update the attributes: inbound unicast counter value $C_{j\to i}$, the number of received unicasts σ_j^U and $RSSI_j$ upon receipt of a unicast from node j, and the attributes: inbound cluster broadcast counter value $C_{j\to\odot}$, the number of received cluster broadcasts $\sigma_j^\odot$ and $RSSI_j$ upon receipt of a cluster broadcast from node j (i.e., during intra-cluster communication).

At the end of the neighbour discovery and key establishment processes, each node should know its neighbours and have populated its Neighbour Table. To measure M_L, each CH elected during the initial clustering process computes M_L using (7) which makes use of the counter values maintained in the Neighbour Table and the number of broadcasts and unicasts successfully received. Then, the CH uses the computed M_L to compute M_{L*} and x using (6) and (4), respectively. The pseudocode of the HB message transmission rate estimation method is detailed in the HB-Rate algorithm (Algorithm 2).

Algorithm 2 HB-Rate Algorithm

procedure HB-Rate(P_{FP})
▷ Executed by the elected CH.
read $\left\{\sigma_i^U, C_{i\to j}, \sigma_i^*, C_{i\to*}, \sigma_i^\odot, C_{i\to\odot}\right\}_{i\in N}$ from T_j
compute $M_L = \frac{1}{3\times b} \times \left(\sum_{i=1}^{b} \frac{C_{i\to*}-\sigma_i^*}{C_{i\to*}} + \sum_{i=1}^{b} \frac{C_{i\to j}-\sigma_i^U}{C_{i\to j}} + \sum_{i=1}^{b} \frac{C_{i\to\odot}-\sigma_i^\odot}{C_{i\to\odot}} \right)$, where σ_i^*, σ_i^U and $\sigma_i^\odot > 0$
compute $M_{L*} = \sum_{k=0}^{n-2} \left(\binom{n-2}{k} \times (0.609)^{(n-2)-k} \times (0.391)^k \times \sum_{j=0}^{k} \binom{k}{j} \times (1-M_L)^j \times (M_L)^{k-j} \times (M_L)^j \right)$
compute $x = \frac{\ln\left(P_{FP}\right)+\ln\left(M_{L*}\right)}{\ln\left(M_L\right)+\ln\left(M_{L*}\right)}$
return x
end procedure

$$M_L = \frac{1}{3 \times b} \times \left(\sum_{i=1}^{b} \frac{C_{i\to*} - \sigma_i^*}{C_{i\to*}} + \sum_{i=1}^{b} \frac{C_{i\to j} - \sigma_i^U}{C_{i\to j}} + \sum_{i=1}^{b} \frac{C_{i\to\odot} - \sigma_i^\odot}{C_{i\to\odot}} \right) \tag{7}$$

where σ_i^*, σ_i^U and $\sigma_i^\odot > 0$.

Each node in a cluster is required to update its Neighbour Table (updating the counter, number of received messages and RSSI attributes) for each unicast or broadcast the node receives from its in-cluster neighbours. If any node becomes a CH during the next clustering process, it uses the collected data to compute x for the new cluster using the HB-Rate algorithm.

3) CLUSTER SETUP AND CH RANKING

Upon forming a cluster and electing a CH, the elected CH together with other CMs execute the cluster setup (CSU) protocol (see Algorithm 4) to (1) establish Cluster Broadcast Keys, (2) estimate a *HB* message transmission rate for the cluster, and (3) rank the CMs to serve as backup CHs. There are two procedures in the CSU protocol, CSU.ch() which is performed by the CH, and CSU.cm() which is performed by other CMs in the cluster. Let us consider a cluster where the elected CH is node i. In CSU.ch(), the CH generates a new Cluster Broadcast Key, $k_i^\odot$, for itself using a pseudorandom generator function. Then, it uses the generated key to compute a new Cluster Broadcast Key, $k_j^\odot$ for every CM in the cluster, where $k_j^\odot = k_i^\odot \oplus k_j^*$. The CH stores the new keys in its Neighbour Table, T_i. Next, the CH estimates the *HB* message transmission rate, x, using the HB-Rate algorithm. After that, the CH ranks the CMs for backup CH selection purpose,

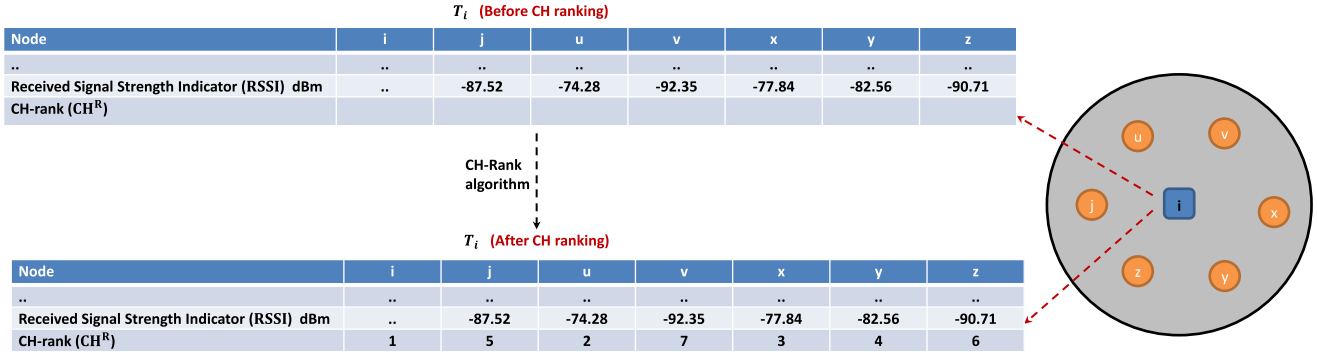

FIGURE 4. An example of a CH ranking process.

i.e., the next CH will be selected from the top ranked CM in this list.

Algorithm 3 CH-Rank Algorithm

procedure CH-Rank( )
▷ Executed by the CH.
  read $\left\{RSSI_i\right\}_{i\in\mathbb{C}_c}$ from T_j
  if CM i is a current CH **then**
    $CH_i^R \leftarrow 1$
  end if
  $rank \leftarrow 2$
  for remaining unranked CMs **do**
    if $RSSI_i$ is maximum **then**
      $CH_i^R \leftarrow rank$
    end if
    $rank \leftarrow rank + 1$
  end for
  return $\left\{CH_i^R\right\}_{i\in\mathbb{C}_c}$
end procedure

The CH ranking is performed using the CH-Rank algorithm (Algorithm 3), where the ranking is based on the RSSI attribute stored in the CH's Neighbour Table. RSSI is chosen as the basis for the ranking method in order to reduce the energy consumption. The higher the RSSI value, the closer the node may be located to the current CH. Closer nodes are preferred to serve as backup CHs because they could connect to the rest of the CMs in the cluster using as low transmission power as possible. The ranking process is performed as follows. The CH assigns itself a CH^R of 1. Other CMs are ranked as follows. A CM which has the highest RSSI value is ranked as number 2 followed by the CM that has the next highest value and so on. An example of the ranking process is illustrated in Fig. 4.

After completing the CH ranking process, the CH sends a unicast message to each CM. The message contains $k_i^{\odot}$, x and CH^R of all CMs and the message is encrypted using the Pairwise Key (e.g., k_{ij} when the CM is j).

When CM j receives the unicast message from the CH, it performs CSU.cm() in which CM j firstly decrypts the received message to obtain $k_i^{\odot}$, x and CH^R of all CMs. Then, CM j updates its Neighbour Table T_j by setting the CH^R value of each member based on the received CH^R. Using $k_i^{\odot}$, CM j computes its Cluster Broadcast Key $k_j^{\odot} = k_i^{\odot} \oplus k_j^*$. It also computes $k_u^{\odot}$ for all other CMs using $k_u^{\odot} = k_i^{\odot} \oplus k_u^*$ and stores the computed keys in its Neighbour Table, T_j. Once the CSU protocol is completed, the cluster is ready for the

Algorithm 4 CSU Protocol Algorithm

procedure CSU.ch( )
▷ Executed by CH i to setup the cluster.
  generate new $k_i^{\odot}$
  for each $j \in \mathbb{C}_c$ **do**
    update T_i **set** $k_j^{\odot} = k_i^{\odot} \oplus k_j^*$
  end for
  $x \leftarrow$ HB-Rate(P_{FP})
  $\left\{CH_i^R\right\}_{i\in\mathbb{C}_c} \leftarrow$ CH-Rank( )
  for each $j \in \mathbb{C}_c$ **do**
    $Payload_{i\rightarrow j} \leftarrow Enc\left(k_{ij}, C_{i\rightarrow j} \parallel k_i^{\odot} \parallel x \parallel \left\{CH_i^R\right\}_{i\in\mathbb{C}_c}\right)$
    send $m_{i\rightarrow j} : ID_i \parallel Payload_{i\rightarrow j}$
  end for
end procedure

procedure CSU.cm($m_{i\rightarrow j}$)
▷ Executed by CM j upon receipt of a cluster setup unicast from the CH.
  $C_{i\rightarrow j} \parallel k_i^{\odot} \parallel x \parallel \left\{CH_i^R\right\}_{i\in\mathbb{C}_c} \leftarrow Dec(k_{ij}, Payload_{i\rightarrow j})$
  set $k_j^{\odot} = k_i^{\odot} \oplus k_j^*$
  set $CH_j^R = \{CH_j^R\}_{received}$
  for each $u \in \mathbb{C}_c$ **do**
    update T_j **set** $k_u^{\odot} = \begin{cases} k_i^{\odot} & \text{if } u = i \\ k_i^{\odot} \oplus k_u^* & \text{if } u \neq i \end{cases}$
    update T_j **set** $CH_u^R = \{CH_u^R\}_{received}$
  end for
end procedure

Operational phase which involves detection and revocation of compromised nodes. It is also worth noting that during the Operational phase, the clusters are closed and should not accept any join request. This is necessary to prevent compromised nodes from participating in the network operations after redeployment.

4) NODE COMPROMISE DETECTION

Node compromise detection is performed within each cluster with the involvement of every CM in the cluster at the end of each HB round. The detection is done using the NCD algorithm (Algorithm 5) which has three cases depending on the role of the node in the cluster. These cases are detailed below:

NCD-Case_1: The CH detects the compromise of other CMs. The CH declares that a CM is compromised if and only if the following two conditions are met:

1) The CH has not heard the CM during the past x HB rounds.
2) The *HB* messages received by the CH in the past $(x-1)$ HB rounds confirm that the CM has not been heard by its in-cluster neighbours.

To verify these conditions, the CH applies a bitwise OR operation on all monitoring data records in its Monitoring Data Vector. If any of the bits in the result is 0, then the CH concludes that the CM associated with that bit is compromised.

NCD-Case_2: The DCH detects the compromise of the CH and other CMs. The DCH declares that the CH and other CMs are compromised if and only if the following two conditions are met:

1) The DCH has not heard the CH and other CMs during the past x HB rounds.
2) The *HB* messages received by the DCH in the past $(x-1)$ HB rounds confirm that the CH and other CMs have not been heard by their in-cluster neighbours.

To verify these conditions, the DCH applies a bitwise OR operation on all monitoring data records in its Monitoring Data Vector. If the first bit of the result is 0 and any other bit is 0, then the DCH concludes that the CH and the CMs associated with the 0 bits are compromised. Note that the DCH concludes that a CM is compromised if and only if the CH is compromised. This is because the CH is responsible for reporting the compromises of the CMs if the CH itself is alive (i.e., not being compromised itself).

NCD-Case_3: A lower CH-rank CM detects simultaneous compromises of the current CH, DCH and other higher CH-rank CMs. A CM declares that simultaneous node compromises have occurred if and only if the following two conditions are met:

1) The CH has not heard any of the suspected nodes (i.e., CH, DCH and other higher CH-rank CMs) during the past x HB rounds.
2) The HB messages received by the CM in the past $(x-1)$ HB rounds confirm that none of the suspected nodes (i.e., CH, DCH and

Algorithm 5 NCD Algorithm

```
1: procedure NCD( )
   ▷ Executed by CM i at the end of each HB round.
2:     result ← 0
3:     ID_det ← φ                ▷ ID_det is the list of detected compromised node IDs.
4:     if CM i is a CH then
5:         for each record R in V_i do
6:             result ← result ∧ R        ▷ ∧ is a bitwise OR operator.
7:         end for
8:         for each bit b_j in result do
9:             if b_j = 0 then
10:                add ID_j to ID_det
11:            end if
12:        end for
13:    else if CM i is a DCH then
14:        for each record R in V_i do
15:            result ← result ∧ R
16:        end for
17:        if b_1 = 0 then ▷ b_1 is the bit associated with the CH in result.
18:            add ID_CH to ID_det
19:            for each other bit b_j in result do
20:                if b_j = 0 then
21:                    add ID_j to ID_det
22:                end if
23:            end for
24:        end if
25:    else                          ▷ CM i is lower CH-rank CM.
26:        for each record R in V_i do
27:            S ← subset of R that includes bits associated with CMs that have CH^R < CH_i^R
28:            result ← result ∧ S
29:        end for
30:        if result = 0 then
31:            for each bit b_j in result do
32:                add ID_j to ID_det
33:            end for
34:        end if
35:    end if
36:    if ID_det is not empty then
37:        LCNR.report(ID_det)
38:        GCNR.report(ID_det)
39:    end if
40:    delete oldest record in V_i
41:    newRecord ← 0
42:    add newRecord to V_i
43: end procedure
```

other higher CH-rank CMs) have been heard by their in-cluster neighbours.

To verify these conditions, the CM selects a subset of the bits from each of the monitoring data records in its Monitoring Data Vector (i.e., the bits associated with the CH, DCH and other higher CH-rank CMs). Then, the CM applies a bitwise OR operation on these subsets. If the result is all zeros, then the CM concludes that the CH, DCH and other higher CH-rank CMs are all compromised. However, if there is at least a single bit in the result with a value of 1, then there are no simultaneous compromises.

At the end of the execution of the NCD algorithm, each node i deletes the oldest monitoring data record from V_i and inserts a new record that is initialised with zeros to prepare for the next HB round. So the size of V_i is always kept at x to save memory.

If one or more node compromises are detected, then the monitoring node performs a compromised node revocation. The revocation is carried out in two steps. The first is to revoke the compromised nodes locally (i.e., within the cluster) by executing the local compromised node revocation (LCNR) protocol. The second step is to revoke the compromised nodes globally (i.e., from the entire network) by executing the global compromised node revocation (GCNR) protocol. LCNR and GCNR are explained in Sections V-A.5 and V-A.6, respectively.

5) LOCAL COMPROMISED NODE REVOCATION (LCNR)

The local compromised node revocation is achieved using the LCNR protocol (Algorithm 6) which has two procedures: LCNR.report() to report a CM compromise locally within the cluster and LCNR.revoke() to revoke the compromised CM from the cluster. In LCNR.report(), the node that has detected the compromise, say i, deletes the records related to the compromised nodes from T_i and adds the IDs of the compromised nodes to its Compromised Node List, L_i. The rest of the procedure varies depending on the role of the reporting node. There are two cases as follows:

LCNR-Case_1: If the reporting node is the CH, it performs the following steps:

1) Broadcasts a message to the cluster notifying the CMs that one or more CM compromises have been detected. The message contains the ID(s) of the compromised CM(s) encrypted using $k_i^{\odot}$.
2) Performs a cluster setup using the CSU protocol.

LCNR-Case_2: If the reporting node is the DCH or any other CM, it performs the following steps:

1) Adjusts its transmission power to connect to all remaining CMs.
2) Broadcasts a message to the cluster to declare that it becomes the current CH and reports the IDs of the detected compromised CMs. The message is encrypted using $k_i^{\odot}$.
3) Performs a cluster setup using the CSU protocol.

Algorithm 6 LCNR Protocol Algorithm

procedure LCNR.report(ID_{det})
▷ Executed by CM i upon detection of a node compromise; ID_{det} is the list of detected compromised node IDs.
  delete ID_{det} records from T_i
  update L_i add ID_{det}
  if CM i is a CH **then**
    $Payload_{i\to *} \leftarrow Enc(k_i^{\odot}, C_{i\to\odot} \parallel ID_{det})$
    send $m_{i\to *} : ID_i \parallel Payload_{i\to *}$
    CSU.ch( )
  else
    adjust transmission power to connect to all remaining CMs
    $Payload_{i\to *} \leftarrow Enc(k_i^{\odot}, C_{i\to\odot} \parallel declareCH \parallel ID_{det})$
    send $m_{i\to *} : ID_i \parallel Payload_{i\to *}$
    CSU.ch( )
  end if
end procedure

procedure LCNR.revoke($m_{i\to *}$)
▷ Executed by CM j upon receipt of a local revocation broadcast from CH.
  if the sender of $m_{i\to *}$ is the a previously declared CH **then**
    $C_{i\to\odot} \parallel ID_{det} \leftarrow Dec(k_i^{\odot}, Payload_{i\to *})$
  else
    $C_{i\to\odot} \parallel declareCH \parallel ID_{det} \leftarrow Dec(k_i^{\odot}, Payload_{i\to *})$
  end if
  delete ID_{det} records from T_j
  update L_j add ID_{det}
end procedure

Upon receipt of the compromise report broadcast, CM j performs the LCNR.revoke() procedure. Firstly, CM j decrypts the received message using $k_i^{\odot}$ to obtain the list of detected compromised node IDs. Then, it deletes the records of the compromised nodes from T_j and adds the IDs of the detected compromised nodes to L_j. After that, CM j waits for further instructions to setup the cluster based on the CSU protocol.

6) GLOBAL COMPROMISED NODE REVOCATION (GCNR)

The global compromised node revocation is done using the GCNR protocol (Algorithm 7). The protocol has four procedures: GCNR.report() performed by a CH to report detected node compromises to the BS, GCNR.receive() performed by the BS to receive node compromise reports from CHs, GCNR.notify() performed by the BS to notify all nodes about

Algorithm 7 GCNR Protocol Algorithm

procedure GCNR.report(ID_{det})
▷ Executed by the CH upon detection of a node compromise.
    $Payload_{i\to B} \leftarrow Enc(k_{Bi}, C_{i\to B} \parallel ID_{det})$
    send $m_{i\to B} : ID_i \parallel Payload_{i\to B}$
end procedure

procedure GCNR.receive($m_{i\to B}$)
▷ Executed by the BS upon receipt of a reported node compromise.
    $C_{i\to B} \parallel ID_{det} \leftarrow Dec(k_{Bi}, Payload_{i\to B})$
    update L_B add ID_{det}
end procedure

procedure GCNR.notify( )
▷ Executed by the BS before the end of the current clustering interval.
    update k_N
    $Payload_{B\to *} \leftarrow Enc(k_N, C_{B\to *} \parallel ID_{det})$
    send $m_{B\to *} : ID_B \parallel Payload_{B\to *}$
end procedure

procedure GCNR.revoke($m_{B\to *}$)
▷ Executed by any node i upon receipt of a revocation broadcast from BS.
    $C_{B\to *} \parallel ID_{det} \leftarrow Dec(k_N, Payload_{B\to *})$
    if $ID_{det} \in T_i$ **then**
        delete ID_{det} records from T_i
        update k_i^*
    end if
    update L_i add ID_{det}
end procedure

the node compromises and GCNR.revoke() performed by all nodes to revoke the compromised nodes notified by the BS.

In GCNR.report(), a current or declared CH composes a unicast message to be sent to the BS. This message contains the ID(s) of the detected compromised CM(s) and is encrypted using the Individual Key that the CH shares with the BS, k_{Bi}. When the BS receives the message, it executes GCNR.receive(). The BS decrypts the message using k_{Bi} and obtains the list of detected compromised node IDs. Then, it adds the IDs of the detected compromised nodes to the Compromised Node List that it maintains.

Before expiry of the current clustering interval, the BS executes the GCNR.notify() procedure. The procedure starts by updating the Network Key, k_N, that may be exposed to adversaries through the node compromises. The update of k_N is performed using the method discussed in Section IV-B. Then, the BS composes a global broadcast message. The message contains the list of all compromised nodes that have been detected during the current clustering interval. The message is encrypted using the updated Network Key k_N. A node receiving the BS global broadcast message should execute GCNR.revoke(). The node decrypts the message using k_N and obtains the list of compromised node IDs. Then, the node deletes the records of the compromised nodes from its Neighbour Table if they exist and updates its Broadcast Key using the method discussed in Section IV-B. Next, the node adds the IDs of the compromised nodes to its Compromised Node List. Any future joining requests initiated by a compromised node that has an ID matching with any of the IDs in the Compromised Node List will not be processed.

B. AdaptENCD IN ACTION

The AdaptENCD three phases (i.e., Node Initialization, Network Setup and Operational) are detailed below.

1) NODE INITIALIZATION PHASE

This is a phase executed prior to the deployment of a network. In this phase, each node i is preloaded with the parameter values: ID_i, k_N and k_{Bi}.

2) NETWORK SETUP PHASE

In this phase, nodes discover their neighbours, establish Pairwise and Broadcast Keys and perform the initial clustering. In the neighbour discovery process, each node broadcasts discovery messages to its neighbours at a certain rate. The discovery messages are also used to collect evidence on the message loss in the underlying environment. This is achieved by using the counter values carried in the transmitted messages as discussed in Section V-A2. Then, the nodes establish the Pairwise and Broadcast Keys using EDDK methods [31]. Once the neighbour discovery and the Pairwise and Broadcast Key establishment processes are executed, each node should have identified its neighbours and populated its Neighbour Table. Next, the clustering process starts. AdaptENCD is independent of any clustering mechanism. However, we recommend using an anonymous clustering protocol such as the Private Cluster Head Election (PCHE) protocol [35]. This is to make it harder for an adversary to identify CHs by observing the election of CHs during a clustering process. Obviously, CHs are more attractive to a compromising adversary as (1) they may contain more private data and/or secret keys related to other neighbouring nodes, and (2) they may be used to launch more effective privacy attacks after redeploying them back to the network owing to their important role compared to ordinary nodes. Upon completion of the clustering process, the CH together with the CMs execute the CSU protocol. This protocol establishes the Cluster Broadcast Keys for the CMs, estimates the *HB* message transmission rate and performs the CH ranking process. This prepares the clusters for the Operational phase.

3) OPERATIONAL PHASE

In the Operational phase, the CMs within each cluster monitor each other using the HB protocol. To detect node compromises, each CM performs the NCD algorithm at the end of each HB round. If a node compromise is detected, the

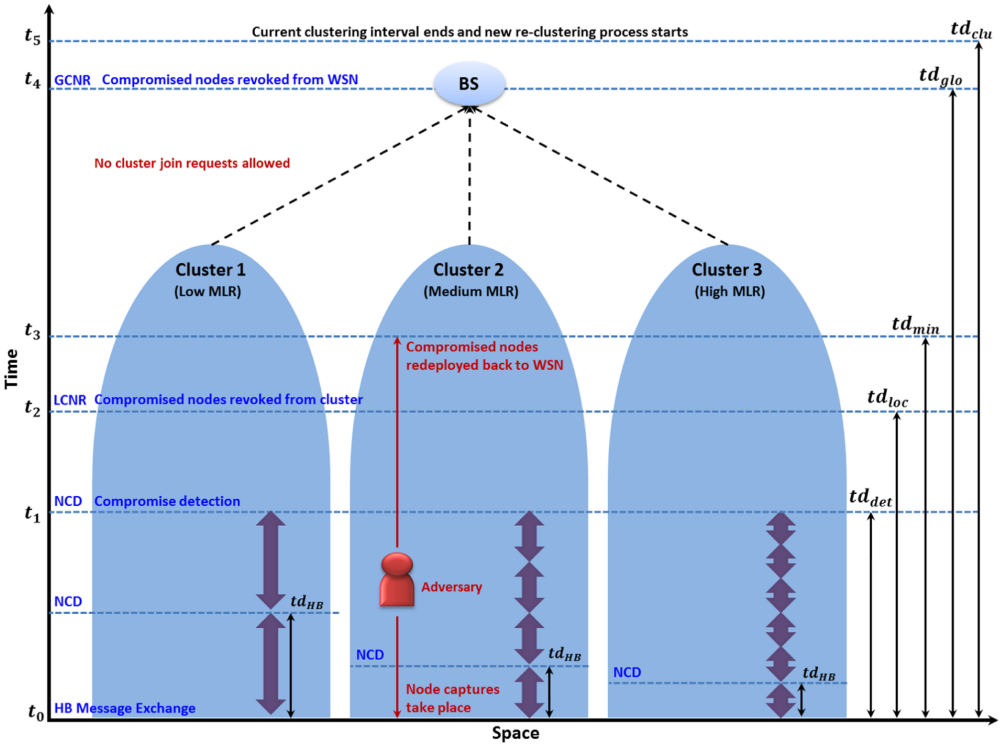

FIGURE 5. Scenario describing the timing structure of detecting node compromises in AdaptENCD.

CMs execute the LCNR protocol to first revoke the compromised node locally from the cluster. The compromised node will later be revoked globally by performing the GCNR protocol before the end of the current clustering interval. By the end of the clustering interval, a new clustering process starts where the nodes may reorganize into different clusters. In each newly formed cluster, the elected CH together with the CMs perform the CSU protocol to setup the cluster and the monitoring process repeats.

C. AdaptENCD TIMINGS

Fig. 5 shows the AdaptENCD timing structure based on a scenario of a WSN consisting of a single BS and three clusters (i.e., Clusters 1, 2 and 3) in one clustering interval. The clusters are deployed in an environment with varying levels of MLR (i.e., low, medium and high, respectively). To ensure the effectiveness of the node compromise detection while minimizing false positives, each CH in a cluster uses the HB-Rate algorithm to compute the *HB* message transmission rate, x, that should be used by the CMs. This means that each CM in the cluster should transmit x *HB* messages to its in-cluster neighbours within the td_{det} duration. Based on the HB-Rate algorithm and given the levels of MLR in each cluster, Cluster 1 imposes the lowest rate of transmitting the *HB* messages ($x = 2$), Cluster 2 imposes a medium rate ($x = 4$) and Cluster 3 imposes the highest rate ($x = 6$). The members of each cluster transmit *HB* messages by executing the HB protocol.

Let us assume that at time instance t_0, an adversary performs node compromises in each cluster. Using the aliveness evidence obtained from the *HB* messages, each cluster should detect the compromises by performing the NCD algorithm at time $t_1 = t_0 + td_{det}$. At time $t_2 = t_0 + td_{loc}$, each cluster should revoke the compromised nodes locally using the LCNR protocol. Note that the duration td_{loc} is no longer than td_{min}, so by the time when the adversary redeploys the compromised nodes back to the WSN, i.e., at $t_3 = t_0 + td_{min}$, the compromised nodes should have already been revoked locally from each cluster, thus satisfying requirement FUN01. Hence, a redeployed compromised node cannot participate in further communication in its respective cluster as it has already been revoked. In addition, the compromised node will be unable to join another cluster as all clusters are closed immediately after completing the clustering process.

In addition to the local revocations of the compromised nodes, the CHs in each cluster report the node compromises to the BS. So, when td_{glo} has passed at time $t_4 = t_0 + td_{glo}$, the BS should have received the IDs of all compromised nodes in all clusters. Based on the GCNR protocol, the BS should be ready to notify all nodes to revoke the compromised nodes by adding the IDs of the compromised nodes to the Compromised Node List that each node maintains. This ensures that global revocation is achieved in a time duration no longer than the clustering duration, i.e., $td_{glo} \leq td_{clu}$, thus, satisfying requirement FUN02. At the end of the clustering interval (at time t_5), nodes should know the compromised

node IDs and thus be able to reject any join request initiated by a compromised node in the next clustering process.

VI. ANALYSIS AND EVALUATIONS

In this section, the security of AdaptENCD is analysed with respect to the SEC01 requirement, and its performance is evaluated with respect to the node compromise detection (NCD01 and NCD02) and efficiency (EFF01, EFF02 and EFF03) requirements.

A. SECURITY ANALYSIS

AdaptENCD revokes a detected compromised node from its respective cluster within td_{loc}. This is done by a key updating method implemented in the key establishment process. With this method, a CH, after detecting a node compromise, generates a new Cluster Broadcast Key. This key is sent to each CM through a unicast message. The unicast message is encrypted using a Pairwise Key that is only known to the CH and the respective CM. Hence, it is difficult for an eavesdropping adversary to decrypt the message without knowing the encryption key. In addition, reading the compromised node's memory does not leak information regarding the new Cluster Broadcast Key as this key is generated independently by the CH using a pseudorandom function generator. All non-compromised CMs will use the new Cluster Broadcast Key received from the CH to compute new Cluster Broadcast Keys. Consequently, future communication among the CMs will be based on the newly computed keys. As the compromised node has no access to these new keys, it cannot participate in future communication in the cluster. Further, it is hard for a compromised node to join any other cluster when it is being redeployed during the same clustering interval as the clusters are already closed; they will not accept any joining request outside the clustering process period. During the next clustering process, each node should have received the list of compromised node IDs from the BS. Hence, it is also difficult for the compromised node to participate in the clustering process.

In AdaptENCD, a CH is responsible for monitoring and detecting any compromise of the CMs in its cluster, whereas the responsibility for detecting any compromise of the CH lies with the DCH in the cluster. In addition, a CM with a $CH^R = 3$ in a cluster can detect simultaneous compromises of the CH and DCH in the cluster. However, an adversary may use timed compromises of the CH and DCH rather than simultaneous compromises, hoping to bypass the detection of the compromises by the CM with a $CH^R = 3$. The timed compromises mean that an adversary may capture the CH and DCH in sequence at two different time instances where the difference between the two instances is carefully chosen in order to bypass compromise detection. For example, the adversary may capture the CH at time t_0, and then, DCH at time $t_1 = t_0 + (td_{det} - \Delta t)$, where $\Delta t < td_{det}$. By time t_1, the DCH has not yet detected the compromise of the CH. According to the NCD algorithm, a CM with a $CH^R = 3$ will need at least td_{det} from when the DCH is compromised (i.e., $t_2 = t_0 + td_{det} - \Delta t + t_{det}$) to detect the compromises of both the CH and DCH. However, at time $t_3 = t_0 + td_{min}$, the adversary may redeploy the CH back into the network which is followed by the redeployment of the DCH at time $t_4 = t_1 + td_{min}$. Thus, the two compromises will be undetected if $t_3 < t_2$. Given that AdaptENCD is designed such that $td_{det} = td_{min}/2$, we can see that $t_3 = t_0 + 2 \times td_{det}$ and $t_2 = t_0 + 2 \times td_{det} - \Delta t$. This means that the time when the adversary redeploys the CH (i.e., t_3) comes after the time when the CM with a $CH^R = 3$ can detect the compromise of both the CH and DCH (i.e., t_2). Therefore, AdaptENCD can effectively thwart the risk of such a timed compromise attack.

Let us also analyse the case when an adversary captures a node during a clustering process. In such a case, the captured node will not be able to complete the clustering process as it will be removed from the network for at least td_{min} where td_{min} is longer than the duration of the clustering process (see assumption (A3) in Section IV-D). Therefore, the node will not be part of any cluster during the current clustering interval. However, the node may be able to participate in future clustering processes, thus bypassing the compromise detection process. This can be addressed by allowing the CHs to send the clustering structure (i.e., the IDs of all the CMs of their respective clusters) to the BS immediately after the clustering process. As the BS has knowledge of all nodes in the WSN, it can easily spot missing nodes after receiving the clustering information from every CH. Therefore, the BS can add missing nodes into its Compromised Node List and notify all the nodes to revoke the missing (compromised) nodes before the start of the next clustering process.

The confidentiality of all transmitted messages is always protected while in transit. This is achieved using encryption. The key used for encryption is only known and shared by the communication entities (i.e., the sender and receiver(s) of a message). By selecting a standard symmetric encryption algorithm such as AES and by using sufficiently large keys (e.g., 128 bits), it is computationally hard for an eavesdropping adversary to decipher any eavesdropped message. However, we note that an adversary may be able to attack the integrity of the transmitted messages which may lead to wrong node compromise detection decisions. This problem can be thwarted by appending a message authentication code (MAC) to each transmitted message to protect the message integrity and authenticity. The MAC can be generated using a keyed hash function such as HMAC. However, this may be provided at an additional computational and transmission costs. To reduce transmission costs, truncated MAC values can be used.

B. SIMULATION RESULTS

This section provides performance investigation of AdaptENCD in relation to requirements NCD01, NCD02 and EFF01. The investigation is done using the Castalia simulator [36]. Castalia is specially designed for simulating WSNs and is based on the discrete event simulator OMNET++ [37]. The simulated WSN consists of 576 nodes deployed in a field

$200\,m \times 200\,m$. We have implemented both AdaptENCD and SEFSD schemes in the simulator, and the implementation is done at the application level. We have analysed the results produced from the AdaptENCD scheme and compared them with those from the SEFSD scheme in terms of compromised node detection effectiveness (i.e., FPR and FNR) and the efficiency metric MO (i.e., total number of messages generated for the monitoring process). Node compromises are randomised in terms of node selections as well as the time of their compromise. Simultaneous compromises are also possible. All nodes have equal probability of being compromised regardless of their roles (CM, DCH or CH). The maximum proportion of nodes that can be compromised during a simulation run is 25%. A compromised node is simulated by stopping the node's resource manager, simulating the node being removed from the network. The minimum time required to compromise a node, td_{min}, is set to 30 s. To obtain statistically significant results, each simulation result is the average value of 30 results produced from 30 repeated simulation runs. The parameter values used in the simulation are given in Table 3. We have considered two scenarios: (1) the entire WSN suffers the same level of MLR, and (2) different parts of the network suffer different levels of MLR. In the following sections, the simulation results are discussed.

TABLE 3. Simulation parameters.

Parameter	Value
Simulator Version	CASTALIA 3.3 OMNET++ 4.6
Number of nodes	576 sensor nodes and a single BS
Field size	$200\,m \times 200\,m$
Node distribution	24×24 grid
BS location	Centre
Simulation time	1800 s
Node startup delay	Between 0 and 1 s
Maximum number of compromised nodes	25% of the nodes.
td_{min}	30 s
Cluster size (AdaptENCD)	9
P_{FP} (AdaptENCD)	0.01
Number of repeated runs	30

1) SCENARIO 1 - SAME LEVEL OF MLR ACROSS THE ENTIRE WSN

This section reports the performance of AdaptENCD measured in terms of FPR, FNR and MO with varying MLR values using Scenario 1. It also compares the performance results with those from SEFSD. To simulate a given MLR value, we set channel conditions as ideal (i.e., no error), but assign a probability of successful message delivery to every transmitted message accordingly. For example, to simulate MLR=0.1, the probability of successful message delivery assigned to each message is set to 0.9 and for MLR=0.2, the probability of delivery is set to 0.8, and so on. In this way, we facilitate a fairer comparison between the two schemes, AdaptENCD and SEFSD, than by altering the error rates in the channels. The performance results are collected with MLR values ranging from 0 to 0.6 with an increment of 0.1.

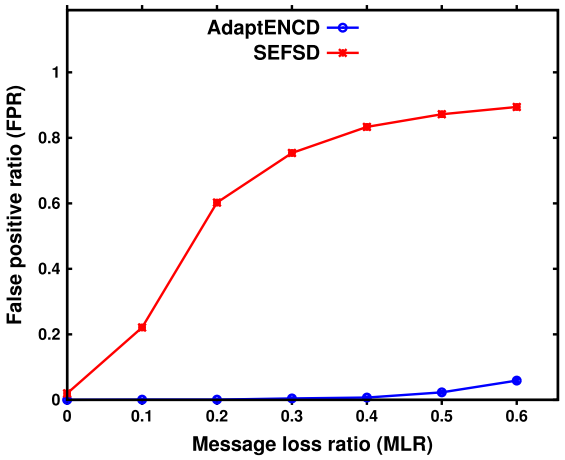

FIGURE 6. False positive ratio (FPR) against varied levels of message loss ratio (MLR) in Scenario 1.

a: False positive ratio (FPR)

Fig. 6 shows the FPR results of the two schemes against MLR values ranging from 0 to 0.6. From the figure, it can be seen that the FPR for SEFSD increases steadily from 0 to 0.9 when the MLR value increases from 0 to 0.6. These results indicate that, when the MLR value is 0, i.e., when the network does not suffer from message loss, SEFSD performs well. However, as the MLR value increases, the network quickly becomes dis-functioning as a majority of the nodes in the network are revoked either because they have been compromised or have been wrongly detected as such.

In contrast, when the MLR value increases from 0 to 0.4, the FPR for AdaptENCD hardly increases, maintaining a value of nearly 0. Only when the MLR value increases beyond 0.4, does the FPR show slight increase, reaching to 0.06 at MLR=0.6. This slight increase in FPR at the MLR value of 0.6 can be explained as follows. When the MLR value increases, each cluster would generate more traffic as the CMs would send *HB* messages at a higher rate to compensate for the message loss. As a result, message collisions are more likely. So, the actual MLR is, in fact, higher than the average message loss M_L estimated by a CH at the beginning of the clustering interval. This issue can be addressed by allowing the CH to perform more frequent estimation of M_L (and consequently the x parameter) within the same clustering interval. If there is a change in x, the CH may inform the CMs to increase the rate of sending the *HB* messages accordingly.

Further, from the figure, we see that AdaptENCD significantly outperforms SEFSD across all MLR values except when MLR=0 (i.e., the case when there is no message loss in the network). This is because the decisions made about node compromises in SEFSD are based on a single piece

of evidence, *Hello* and probe messages sent by a monitored node. However, in AdaptENCD, the decision on a node compromise is based on a collective set of evidence gathered by the in-cluster neighbours of a compromised node.

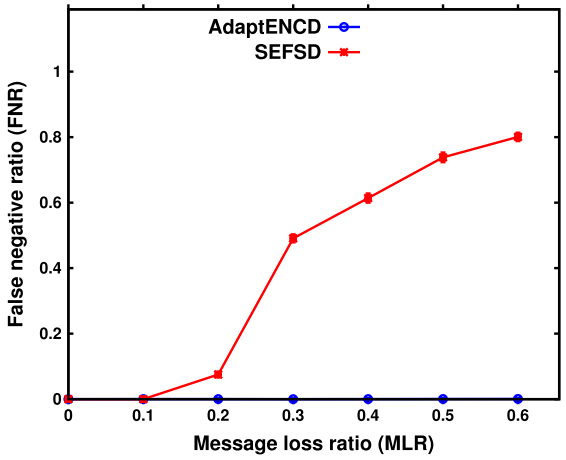

FIGURE 7. False negative ratio (FNR) against varied levels of message loss ratio (MLR) in Scenario 1.

b: False negative ratio (FNR)

The results related to FNR against MLR are presented in Fig. 7. As can be seen, SEFSD reports an FNR value of 0 when MLR is set to 0 and 0.1. However, when the MLR value goes higher than 0.1, FNR increases steadily as MLR increases. This is because, with SEFSD, as MLR increases more false positives are reported (see Fig. 6) and consequently more nodes are revoked. As a result, the number of monitoring nodes decreases. When the number of monitoring nodes decreases, it is likely that the real node compromises go undetected. In contrast, AdaptENCD has achieved an FNR value of 0 for all the MLR settings. This means that, with AdaptENCD, all the node compromises have been detected regardless of the MLR values under investigation.

c: Monitoring overheads (MO)

The MO refers to the total number of messages generated in the WSN for monitoring node compromises during a specific period of time. It constitutes the major communication costs incurred in an early node compromise detection process. In our experiment, the simulator reports the total number of monitoring messages generated in each scheme (AdaptENCD and SEFSD) within the whole length of simulation time. In AdaptENCD, the monitoring messages are *HB* messages, so the MO covers the total number of *HB* messages transmitted by all nodes. In SEFSD, the monitoring messages are *Hello*, *AYT* and *IMF* messages, so the MO covers the total number of these three messages. Revoked nodes due to being compromised or mistakenly detected as compromised (i.e., false positives) are not considered as part of the network and therefore they do not contribute to the overheads (as they do not produce further monitoring messages).

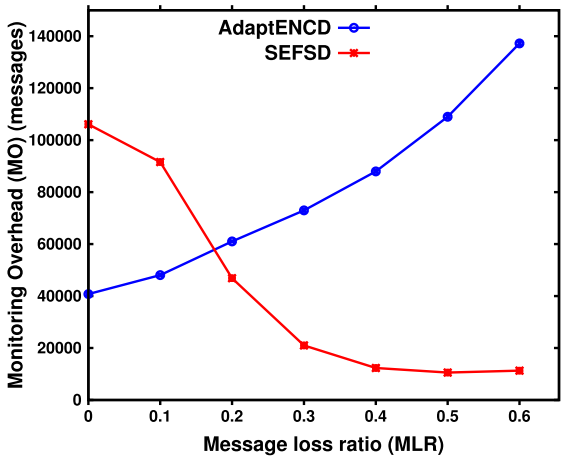

FIGURE 8. Monitoring overheads (MO) against varied levels of message loss ratio (MLR) in Scenario 1.

The results of MO against the MLR value for both schemes are shown in Fig. 8. From the figure, we can make two observations. Firstly, the monitoring overheads for SEFSD decreases steadily, whereas the overheads for AdaptENCD increases steadily, as the MLR value increases. These results are consistent with the results shown in Fig. 6 and Fig. 7, i.e., when the MLR value increases, SEFSD makes more errors in node compromise detection. More such errors would lead to more nodes being revoked by mistake and fewer nodes remain in the network, thus fewer monitoring messages being transmitted. In contrast, AdaptENCD is designed to adjust the monitoring overheads in response to MLR, therefore, the higher the MLR value, the more *HB* messages will be transmitted, thus the higher the monitoring overheads.

Secondly, as shown in Fig. 8, when the MLR is small (approximately below 0.2), AdaptENCD generates fewer monitoring messages than SEFSD; the largest difference between the two is when the MLR value is 0 (40,000 vs 106,000 messages, approximately). This difference disappears when the MLR value increases beyond 0.1. When the MLR value is 0.2 or higher, AdaptENCD generates more monitoring messages than SEFSD, and the gap between the two increases as MLR value increases. The reason is that when the MLR values are higher than 0.2, more nodes in SEFSD get revoked because of being mistakenly reported as false positives and therefore the total number of generated monitoring messages decreases. As the MLR increases, the false positives in SEFSD are reported at an earlier stage in the simulation run causing such dramatic decrease in the number of monitoring messages. This further justifies the increased gap between the two schemes.

These results, combined with our observations made in Fig. 6 and Fig. 7, tell us that AdaptENCD is more effective and efficient than SEFSD in detecting node compromises. This is because the two ideas adopted in AdaptENCD, i.e., the collective decision making approach to node compromise detection and adaptive approach to monitoring message

transmission rate selection, can reduce false positive errors when the MLR value is high and prevent unnecessary monitoring message transmissions in conditions when the MLR value is low.

2) SCENARIO 2 - VARYING LEVELS OF MESSAGE LOSS ACROSS THE WSN

In this scenario, the WSN is assumed to be deployed in an area with varying levels of MLR (e.g., mixing outdoor and indoor deployments or mixing open and obstructed environments). So, the WSN is divided into several groups of nodes and each group is assigned with a certain level of MLR that is randomly selected to be between 0 and 0.6. The scenario is studied with a number of groups ranging between 4 and 24 with an increment of 4 every time. The results are discussed below.

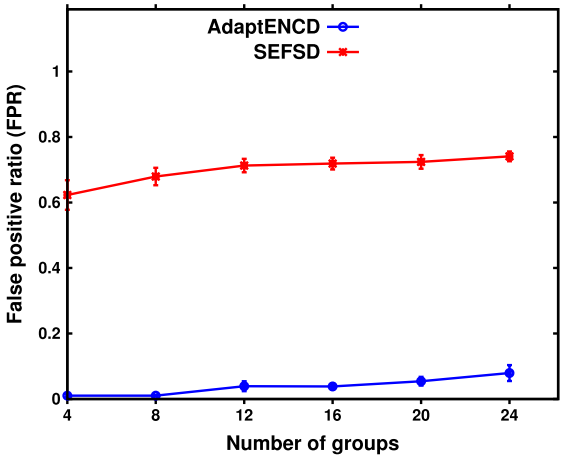

FIGURE 9. False positive ratio (FPR) against number of groups in Scenario 2.

a: FALSE POSITIVE RATIO (FPR)

Fig. 9 shows the FPR against the number of groups for the two schemes, AdaptENCD and SEFSD. The results show that the reported FPR in SEFSD exceeds 0.6 in all the simulated number of groups. In addition, the FPR increases slightly as the number of groups increases. This behaviour is in line with the results obtained in Scenario 1 in which SEFSD performs perfectly only when there is no message loss in the environment. In the mixed environment deployment, half of the groups will probabilistically suffer an MLR of more than 0.3. As SEFSD is not resilient to message loss especially when the MLR exceeds 0.1, the detection process is likely to report a higher number of false positives.

AdaptENCD maintains a much lower level of FPR than SEFSD (0.04 for AdaptENCD and 0.7 for SEFSD on average). This confirms that AdaptENCD is more suited to WSNs deployed in environments with varying levels of MLR. The results also show that there is a slight increase of FPR in AdaptENCD as the number of groups increases. This can be explained as follows. The higher the number of groups, the more clusters are formed with nodes that belong to two different groups (i.e., clusters formed at the border of the two groups), where those two groups have different levels of MLR. Given that the CH computes the average MLR based on the messages received from its neighbours, when the majority of the CMs belong to a group assigned with a lower level of MLR, it is likely that the estimated average *HB* message transmission rate within the cluster may not be enough to prove the aliveness of the nodes that belong to the group experiencing a higher level of MLR. Thus, such nodes may be revoked due to detection errors. This issue may be resolved by changing the method of estimating the M_L value by a CH (refer to the HB-Rate algorithm). Instead of estimating M_L using the average value over all the neighbouring nodes as computed by (7), the average message loss from each neighbour is estimated independently and then the maximum value among all estimated values is used to compute the *HB* message transmission rate within the cluster. However, this may increase the *HB* message transmission rate considerably. Alternatively, the issue may be addressed during the clustering process where each node, before the start of the clustering process, computes M_L based on the information stored in its Neighbour Table using (7). If a node announces itself as a CH, it should broadcast its computed M_L value to its neighbours. Each node in the neighbourhood can then choose to join the CH which provides the M_L value that is the closest to the M_L value computed by the node itself. In this way, we can limit forming clusters using nodes from different groups.

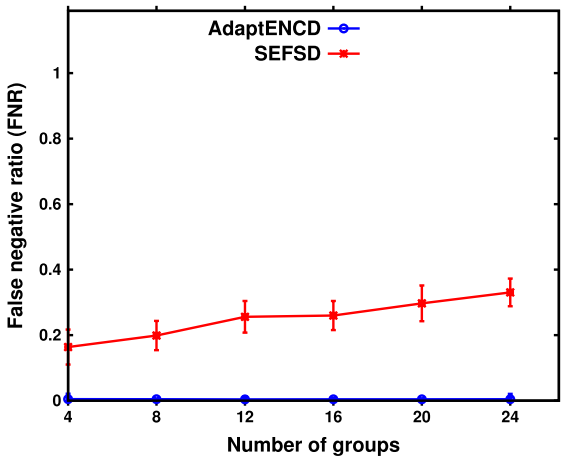

FIGURE 10. False negative ratio (FNR) against number of groups in Scenario 2.

b: FALSE NEGATIVE RATIO (FNR)

Fig.10 shows the results related to FNR against the number of groups. It can be seen that SEFSD suffers a moderate level of FNR. Again these results are due to the inability of SEFSD to maintain a sufficient number of monitoring nodes especially in areas where the MLR value is high. In contrast, AdaptENCD has achieved an FNR value of 0 for all settings. This confirms that AdaptENCD can maintain

a high detection capability even in areas with varying levels of MLR.

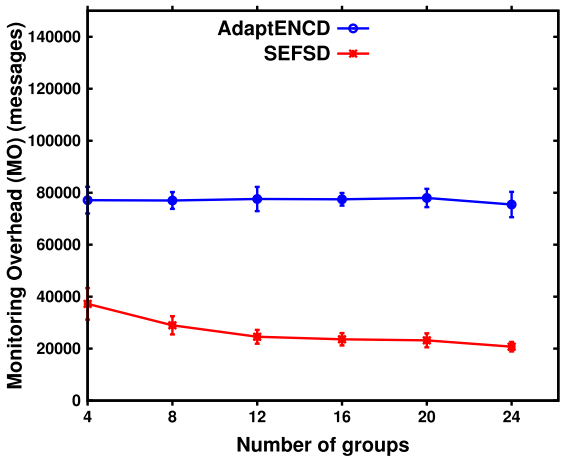

FIGURE 11. Monitoring overheads (MO) against number of groups in Scenario 2.

c: MONITORING OVERHEADS (MO)

The results related to the MO against the number of groups are shown in Fig. 11. SEFSD results show that the number of monitoring messages decreases slightly as the number of groups increases. This is because FPR increases slightly when the number of groups increases, as shown in Fig. 9, and when FPR increases, the number of monitoring messages decreases as more nodes get revoked.

The results related to AdaptENCD show that it exhibits higher monitoring overheads than SEFSD. This is the cost incurred in being able to detect node compromises more effectively as demonstrated in Fig. 9 and Fig.10. In addition, AdaptENCD maintains the same performance regardless of the number of groups with a slight decrease in the number of monitoring messages when the number of groups is set to 24. This is because AdaptENCD has reported a slightly higher level of FPR when the number of groups is set to 24 (see Fig.9). In other words, the total number of monitoring messages is slightly reduced as more nodes get revoked.

C. COMPROMISE REPORTING AND REVOCATION OVERHEADS (CRRO)

This section analyses the CRRO (design requirement EFF02) incurred by AdaptENCD and SEFSD in the process of reporting and revocation of compromised nodes. CRRO are investigated analytically for both schemes by estimating the total number of reporting and revocation messages generated in the network within a time duration equivalent to td_{clu} given a certain number of node compromises. The estimation is based on two scenarios: (1) for a single node compromise, and (2) for multiple node compromises.

1) SCENARIO 1: A SINGLE NODE COMPROMISE

In AdaptENCD, upon the detection of a node compromise, the revocation is carried out in two steps: (1) locally from the cluster within a time duration $\leq td_{min}$, and (2) globally from the entire network within a time duration $\leq td_{clu}$. Let the average number of nodes in a cluster be n and the average number of single-hop neighbours of each node be b. For (1), the CH, based on the LCNR protocol, sends one local broadcast message (LB) to the CMs in the cluster to revoke the compromised node. Then, the CH executes the CSU protocol to setup the cluster by sending one single-hop unicast message (SU) to each CM in the cluster. So, the total number of messages sent to revoke a single node from a cluster within $td_{min} = LB+(n-1)\times SU$. For (2), the CH sends a multi-hop unicast message (MU) to the BS soon after each local compromise detection. The BS, in turn, will firstly need to update the Network Key. The cost of updating the Network Key is approximately equivalent to a global broadcast (GB). Then, the BS will send another GB to notify the nodes about the list of compromised node IDs. In addition, each node that has a compromised neighbour will also need to send one SU to each non-compromised neighbour to update the Broadcast Key. The affected number of nodes is $b-1$ and each node has to sent $b-1$ unicasts to update its Broadcast Key. So, the total number of messages sent in revoking the compromised node from the entire network within $td_{clu} = MU + (b-1)^2 \times SU + 2 \times GB$.

In SEFSD, upon the detection of a node compromise, the monitoring node sends a multi-hop unicast (MU) message (i.e., *Captured* message) to the BS to report the compromise. Then, the BS should send a global broadcast (GB) message (i.e., *AlertUpdate* message) that contains the ID of the comprised node to the entire network to revoke the compromised node within td_{min}. In addition, the BS sends another global broadcast to update the key list. Therefore, the total number of messages sent in revoking a node from the entire network within $td_{min} = MU + 2 \times GB$.

2) SCENARIO 2: MULTIPLE NODE COMPROMISES

For this scenario, we estimate the minimum and maximum CRRO incurred by each scheme in case there are c node compromises in the network. The minimum CRRO is incurred when all the c node compromises are performed simultaneously and reported by the same node (Case_A). The maximum CRRO is incurred when the c node compromises are performed at different time instances where the duration between each node compromise and the next one is at least td_{min}, and each compromise is reported by a different monitoring node in the network (Case_B). We assume that $td_{clu} >> (c \times td_{min})$ to give a sufficient time to revoke all the c compromised nodes from the entire network.

Let us first analyse Case_A. Assuming that the IDs of the compromised nodes can be contained in one message whilst notifying the BS or other nodes about the compromises, the CRRO incurred by each scheme is nearly equivalent to the CRRO incurred in Scenario 1. The only noted difference is that in AdaptENCD, upon locally revoking the c compromised nodes from the cluster, the remaining alive nodes in the cluster will be $n-c$. Therefore, the CH should send

TABLE 4. CRRO incurred by AdaptENCD and SEFSD to report and revoke node compromises based on a single and multiple node compromises.

Scenario	Scheme	CRRO messages*	
(1) A single node compromise	AdaptENCD	$LB + (n-1) \times SU$ within td_{min} $MU + (b-1)^2 \times SU + 2 \times GB$ within td_{clu}	
	SEFSD	$MU + 2 \times GB$ within td_{min}	
		Case_A: minimum CRRO	**Case_B: maximum CRRO**
(2) Multiple node compromises	AdaptENCD	$LB + (n-c) \times SU$ within td_{min} $MU + (b-c)^2 \times SU + 2 \times GB$ within td_{clu}	$c \times (LB + (n-1) \times SU)$ within td_{min} $c \times MU + (N-c) \times (b-1) \times SU + 2 \times GB$ within td_{clu}
	SEFSD	$MU + 2 \times GB$ within td_{min}	$c \times MU + c \times 2 \times GB$ within td_{clu}

* LB: Local broadcast, SU: Single-hop unicast, GB: Global broadcast, MU: Multi-hop unicast.

fewer unicast messages to setup the cluster based on the CSU protocol, i.e., $(n-c) \times SU$ messages compared to $(n-1) \times SU$ messages in Scenario 1. Similarly, the number of SUs required to update the Broadcast Keys is reduced to $(b-c)^2 \times SU$.

For Case_B, in AdaptENCD, each reporting node will send $LB + (n-1) \times SU$ messages within td_{min} to revoke each compromised node locally from the respective cluster. For c node compromises, the overheads would be $c \times (LB + (n-1) \times SU)$ messages. To globally revoke all the reported compromised nodes from the entire network within td_{clu}, the overheads would be $c \times MU + (N-c) \times (b-1) \times SU + 2 \times GB$ messages (assuming that every remaining node in the network will need to update its Broadcast Key). SEFSD would incur $MU + 2 \times GB$ messages within td_{min} to report and revoke each compromised node. For c node compromises, the CRRO is $c \times (MU + 2 \times GB)$ messages.

Table 4 summarises the CRRO incurred by the two schemes in both scenarios. It can be seen that in Scenario 1, AdaptENCD incurs slightly higher CRRO than SEFSD. This is because the revocation in AdaptENCD is done in two steps, locally from the cluster and globally from the entire network. So, the additionally incurred CRRO in AdaptENCD are due to the local revocation of the compromised node from the cluster and also due to the need to update the Broadcast Key of the affected nodes.

In Scenario 2, we make two observations. The first is that in Case_A, each scheme incurs nearly the same CRRO it incurs in Scenario 1, indicating that AdaptENCD still incurs slightly higher CRRO than SEFSD. Again this is due to implementing a two-step revocation in adaptENCD (local and global), whereas SEFSD revocation is done in one step.

The second observation is that in Case_B, AdaptENCD incurs less CRRO than SEFSD. This is in contrast to the observations discussed above in Scenario 1 and Scenario 2-(Case_A). The reason for this change is that in SEFSD, the BS must revoke a compromised node from the entire network within td_{min} (i.e., before an adversary can redeploy the node to join the network). For c node compromises, the BS sends a considerable number of global broadcast messages ($c \times 2 \times GB$ messages). On the contrary, the BS in AdaptENCD only sends $2 \times GB$ messages within td_{clu} to globally revoke all reported compromised nodes. Given that global broadcast messages are much more expensive than single-hop unicast and local broadcast messages, we can see that AdaptENCD is more efficient than SEFSD in this case.

To summarise, the CRRO in SEFSD increases steadily as more compromised nodes are detected by different monitoring nodes at dispersed time instances. This is because SEFSD depends highly on global broadcast messages (the most expensive) to revoke the compromised nodes. In contrast, CRRO in AdaptENCD increases only slightly as all the compromised nodes get revoked from the cluster using local broadcast and unicast messages whereas, the number of global broadcast messages used to revoke the compromised nodes from the entire network is kept constant at a minimum level.

D. MEMORY REQUIREMENTS

This section investigates the memory requirements for the parameters used in AdaptENCD and compares them to those of SEFSD. Each node i in AdaptENCD needs to store the following parameters:

- Parameters related to the the node itself: One ID (ID_i), two keys (k_i^*, $k_i^{\odot}$), two counters ($C_{i\to *}$, $C_{i\to \odot}$) and one CH-rank (CH_i^R).
- Parameters related to the BS: One ID (ID_B), two keys (k_{Bi}, k_N), three counters ($C_{i\to B}$, $C_{B\to i}$, $C_{B\to *}$), two tallies of received messages (σ_B^U, σ_B^*) and one RSSI value ($RSSI_B$).
- Parameters related to each one-hop neighbour j: One ID (ID_j''), two keys (k_{ij}, k_j^*), three counters ($C_{i\to j}$, $C_{j\to i}$, $C_{j\to *}$), two tallies of received messages (σ_j^U, σ_j^*) and one RSSI value ($RSSI_j$).
- Additional parameters related to each in-cluster neighbour u: one key ($k_u^{\odot}$), one counter ($C_{u\to \odot}$), one tally of received messages ($\sigma_u^{\odot}$) and one CH-rank (CH_u^R).

Let us assume a dense network where node i has 100 single-hop neighbours and 10 in-cluster neighbours. Given that ID size = 8 bytes, key size = 16 bytes, counter size = 4 bytes, tally of received messages size = 4 bytes, RSSI size = 8 bytes and CH-rank size = 1 byte, the total memory requirement of AdaptENCD is 7.00 KB. A typical sensor node such as TelosB mote [38] has an architecture where the entire

memory (48KB of flash memory, 10KB of RAM and 1MB of external EEPROM) is accessible for code and data. Therefore, the memory requirements of AdaptENCD can be easily accommodated in the memory of TelosB.

In SEFSD, each node has to store the IDs of its neighbours and a common key list. Given that *Hello* messages are sent at a specified rate and the messages sent in each time period is encrypted using the same key drawn from the key list, then each node has to store a large key list to avoid the costs of frequent re-keying. Using the same assumptions used for AdaptENCD above and additionally assuming that the size of the key list is 1000 keys, the total memory requirement of SEFSD is 16.40 KB. Therefore, SEFSD incurs high memory requirements compared to AdaptENCD.

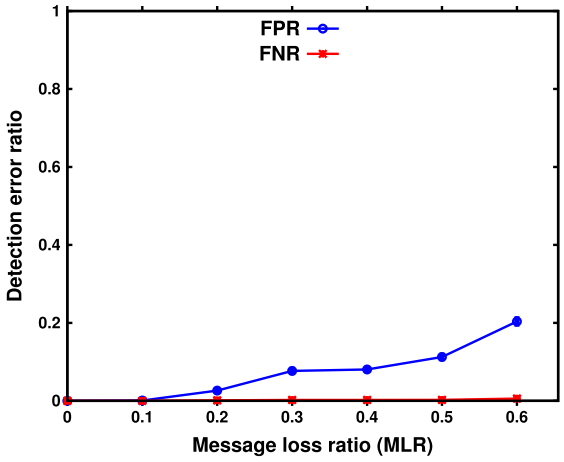

FIGURE 12. Detection error ratios (FPR and FNR) in AdaptENCD when the HB message transmission rate is reduced by 1.

VII. FURTHER DISCUSSIONS

In this section, we discuss minimising monitoring overheads. AdaptENCD is designed to detect node compromises effectively and efficiently in a WSN environment. By effective, we mean the scheme should keep detection errors as low as possible, and by efficient, we mean the scheme should cost as low transmission overheads as possible. The overheads include the monitoring and revocation overheads. In Section I, we claimed that, by using a cluster-based collective decision making approach to node compromise detection and an adaptive approach to monitoring message transmission rate selection, we can reduce monitoring overheads as much as possible without compromising effectiveness in compromised node detection. One way to achieve this is to use a *HB* message transmission rate that is no more than necessary for each cluster. To provide further proof that we cannot reduce the *HB* message transmission rate any further without increasing detection errors (FPR and FNR), we have repeated the experiment carried out for Scenario 1 but with a reduced *HB* message transmission rate estimated for each cluster. More specifically, we reduce x by 1 (i.e., $x = x - 1$), and examine the effects of this reduction on FPR and FNR with varying values of MLR. Fig. 12 shows the results of this experiment. From the figure, it can be seen that the FPR has increased to a level that is much higher than the set target of false positive threshold ($P_{FP} = 0.01$) especially for higher levels of MLR. Nevertheless, FNR results have not experienced any deterioration as the scheme provides a sufficient number of monitoring nodes even when the reported level of FPR reaches 0.2 at MLR=0.6. These results confirm that AdaptENCD uses minimum *HB* message transmission rates within each cluster without impeding the effectiveness of node compromise detection.

Using *HB* messages to detect node compromises introduces considerable transmission overheads in the network as discussed in Section VI-B. Therefore, efforts should be made to minimize these overheads, thus minimising energy consumption, which, in turn, could prolong network lifetime. In the following, we provide ways that such overheads could be further reduced:

1) Increase the time duration required by an adversary to compromise a node (td_{min}): If we could increase this time, we could afford to increase the Heartbeat duration (td_{HB}). Therefore, the total number of generated *HB* messages by a CM in a specific period of time is reduced. Increasing the node compromise duration (td_{min}) can be accomplished by implementing extra security measures in the nodes. In addition to the standard precautions for protecting the nodes from unauthorised access (e.g., disabling interfaces that may be exploited by adversaries to gain access to the node's microcontroller), Bacher et al. [2] recommend other protection measures such as choosing a hardware platform appropriate for the desired security protection level. They also propose a method to protect the bootstrap loader password against adversaries who have detailed knowledge about the node's software. These protection measures could increase the effort and time required by an adversary to launch a successful compromise attack.
2) Integrate monitoring operations with periodic data collection operations: In other words, data messages generated during a data collection operation can be used to carry out the monitoring operation. Such a technique can save energy, especially in applications where data is collected at high rates.
3) Increase cluster sizes: As can be inferred from the computations of x and M_{L*} (refer to (4) and (6), respectively), increasing the size of a cluster can increase the probability that a node can be heard by one of its in-cluster neighbours. Therefore, to obtain the same level of false positive threshold (P_{FP}), and by using larger clusters, the *HB* message transmission rate can be reduced. To confirm the effectiveness of this measure, we have carried out a simulation experiment to investigate the average number of *HB* messages sent by each node in a cluster using different cluster sizes and varying levels of MLR in the network. To measure the effect more accurately, we did not implement node

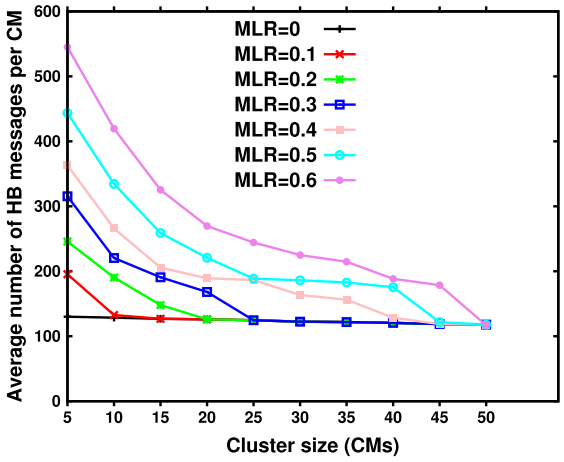

FIGURE 13. Average number of HB messages generated by each CM when varying cluster size and message loss ratio (MLR).

compromises in this experiment. The results are shown in Fig. 13. It can be seen that for each value of MLR, there exists a minimum size of a cluster where the average number of *HB* messages per node is minimal. In addition, the minimum size of a cluster increases as the MLR increases. For example, for a MLR=0.1, the minimum cluster size is 10 and for a MLR=0.2, the minimum cluster size is 20 and so on. However, this measure can only be implemented in dense WSNs where the use of a large cluster size is feasible.

VIII. CONCLUSION

This paper has presented the design and evaluation of a novel adaptive early node compromise detection scheme, AdaptENCD. The scheme is designed for cluster-based WSNs to effectively detect node compromises in a distributed manner in the presence of unreliable communication channels while maintaining false positives below a certain threshold level. This is achieved by allowing each cluster to measure the message loss ratio in the underlying environment and use the measured value to dynamically adjust the transmission rate of *Heartbeat* (i.e., monitoring) messages among the cluster members. Security analysis of the scheme demonstrates its effectiveness in revoking compromised nodes, thus preventing them from participating in future network communications. This can reduce the need for implementing expensive measures to protect the network against further attacks launched by re-deployed compromised nodes. Performance evaluations by means of simulations have been conducted to evaluate the effectiveness and efficiency of the scheme in terms of false positive ratio, false negative ratio and monitoring overheads and the results have been compared against those from the most relevant scheme in the literature. The comparisons have shown that our scheme significantly outperforms the related scheme in terms of compromised node detection reliability under both low and high message loss ratio conditions. When message loss ratio is low, our scheme outperforms the related scheme both in terms of compromised node detection reliability and overhead cost. When the message loss ratio is high, our scheme can detect node compromises much more reliably than the related scheme. These results demonstrate that AdaptENCD is effective in facilitating compromised node detection in the presence of channel unreliability in WSNs. The paper has also discussed other measures to further minimise the monitoring overheads; future work will study these measures.

REFERENCES

[1] C. Hartung, J. Balasalle, and R. Han, "Node compromise in sensor networks: The need for secure systems," Dept. Comput. Sci., Univ. Colorado Boulder, Boulder, CO, USA, Tech. Rep. CU-CS-990-05, 2005. [Online]. Available: http://scholar.colorado.edu/csci_techreports/926

[2] A. Becher, Z. Benenson, and M. Dornseif, *Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks*. Berlin, Germany: Springer, 2006, pp. 104–118. [Online]. Available: http://dx.doi.org/10.1007/11734666_9

[3] X. Lin, "CAT: Building couples to early detect node compromise attack in wireless sensor networks," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov./Dec. 2009, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/GLOCOM.2009.5425922

[4] W. Ding, Y. Yu, and S. Yenduri, "Distributed first stage detection for node capture," in *Proc. IEEE Globecom Workshops*, Dec. 2010, pp. 1566–1570. [Online]. Available: http://dx.doi.org/10.1109/GLOCOMW.2010.5700202

[5] M. H. Megahed, D. Makrakis, and H. Dahshan, "Distributed compromised nodes detection scheme at first stage for SurvSec security architecture," in *Proc. 6th Int. Conf. Sensor Technol. Appl. (SENSORCOMM)*, 2012, pp. 26–35.

[6] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *Proc. 1st Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, New York, NY, USA, 2003, pp. 1–13. [Online]. Available: http://doi.acm.org/10.1145/958491.958493

[7] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis, "Understanding the causes of packet delivery success and failure in dense wireless sensor networks," in *Proc. 4th Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, New York, NY, USA, 2006, pp. 419–420. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182885

[8] H. Song, L. Xie, S. Zhu, and G. Cao, "Sensor node compromise detection: The location perspective," in *Proc. Int. Conf. Wireless Commun. Mobile Comput. (IWCMC)*, New York, NY, USA, 2007, pp. 242–247. [Online]. Available: http://doi.acm.org/10.1145/1280940.1280993

[9] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Proc. Symp. Secur. Privacy*, May 2003, pp. 197–213. [Online]. Available: http://dx.doi.org/10.1109/SECPRI.2003.1199337

[10] T. Park and K. G. Shin, "Soft tamper-proofing via program integrity verification in wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 4, no. 3, pp. 297–309, May 2005. [Online]. Available: http://dx.doi.org/10.1109/TMC.2005.44

[11] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: SoftWare-based attestation for embedded devices," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 272–282. [Online]. Available: http://dx.doi.org/10.1109/SECPRI.2004.1301329

[12] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim, "Remote software-based attestation for wireless sensors," in *Security and Privacy in Ad-hoc and Sensor Networks*. Berlin, Germany: Springer, pp. 27–41. [Online]. Available: http://dx.doi.org/10.1007/11601494_3

[13] Y. Yang, X. Wang, S. Zhu, and G. Cao, "Distributed software-based attestation for node compromise detection in sensor networks," in *Proc. 26th IEEE Int. Symp. Reliable Distrib. Syst. (SRDS)*, Oct. 2007, pp. 219–230. [Online]. Available: http://dx.doi.org/10.1109/SRDS.2007.31

[14] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, "SCUBA: Secure code update by attestation in sensor networks," in *Proc. 5th ACM Workshop Wireless Secur. (WiSe)*, New York, NY, USA, 2006, pp. 85–94. [Online]. Available: http://doi.acm.org/10.1145/1161289.1161306

[15] A. Seshadri, M. Luk, and A. Perrig, "SAKE: Software attestation for key establishment in sensor networks," in *Proc. 4th IEEE Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, Santorini Island, Greece, Jun. 2008, pp. 372–385. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69170-9_25

[16] K. Chang and K. G. Shin, "Distributed authentication of program integrity verification in wireless sensor networks," *ACM Trans. Inf. Syst. Secur.*, vol. 11, no. 3, pp. 14:1–14:35, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1341731.1341735
[17] R. Mitchell and I.-R. Chen, "A survey of intrusion detection in wireless network applications," *Comput. Commun.*, vol. 42, pp. 1–23, Apr. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2014.01.012
[18] A. P. R. da Silva, M. H. T. Martins, B. P. S. Rocha, A. A. Loureiro, L. B. Ruiz, and H. C. Wong, "Decentralized intrusion detection in wireless sensor networks," in *Proc. 1st ACM Int. Workshop Quality Service Secur. Wireless Mobile Netw. (Q2SWinet)*, New York, NY, USA, 2005, pp. 16–23. [Online]. Available: http://doi.acm.org/10.1145/1089761.1089765
[19] S. Rajasegarar, J. C. Bezdek, C. Leckie, and M. Palaniswami, "Elliptical anomalies in wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 6, no. 1, pp. 7:1–7:28, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1653760.1653767
[20] M. Drozda, I. Bate, and J. Timmis, "Bio-inspired error detection for complex systems," in *Proc. IEEE 17th Pacific Rim Int. Symp. Dependable Comput. (PRDC)*, Dec. 2011, pp. 154–163. [Online]. Available: http://dx.doi.org/10.1109/PRDC.2011.27
[21] Z. Xiao, C. Liu, and C. Chen, "An anomaly detection scheme based on machine learning for WSN," in *Proc. 1st Int. Conf. Inf. Sci. Eng. (ICISE)*, Dec. 2009, pp. 3959–3962. [Online]. Available: http://dx.doi.org/10.1109/ICISE.2009.235
[22] J. Hur, Y. Lee, S.-M. Hong, and H. Yoon, "Trust management for resilient wireless sensor networks," in *Proc. 8th Int. Conf. Inf. Secur. Cryptol. (ICISC)*, Seoul, South Korea, Dec. 2005, pp. 56–68. [Online]. Available: http://dx.doi.org/10.1007/11734727_7
[23] S. Ganeriwal, L. K. Balzano, and M. B. Srivastava, "Reputation-based framework for high integrity sensor networks," *ACM Trans. Sensor Netw.*, vol. 4, no. 3, pp. 15:1–15:37, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1362542.1362546
[24] H. Chen, "Task-based trust management for wireless sensor networks," *Int. J. Secur. Appl.*, vol. 3, no. 2, pp. 21–26, 2009.
[25] A. Boukerch, L. Xu, and K. EL-Khatib, "Trust-based security for wireless ad hoc and sensor networks," *Comput. Commun.*, vol. 30, nos. 11–12, pp. 2413–2427, Sep. 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366407001661
[26] I. Onat and A. Miri, "An intrusion detection system for wireless sensor networks," in *Proc. IEEE Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, vol. 3. Aug. 2005, pp. 253–259. [Online]. Available: http://dx.doi.org/10.1109/WIMOB.2005.1512911
[27] M. Zamani, M. Movahedi, M. Ebadzadeh, and H. Pedram, "A DDoS-aware IDS model based on danger theory and mobile agents," in *Proc. Int. Conf. Comput. Intell. Secur. (CIS)*, vol. 1. Dec. 2009, pp. 516–520. [Online]. Available: http://dx.doi.org/10.1109/CIS.2009.215
[28] K. Ioannis, T. Dimitriou, and F. C. Freiling, "Towards intrusion detection in wireless sensor networks," in *Proc. 13th Eur. Wireless Conf.*, 2007, pp. 1–10. [Online]. Available: http://www.ensta-paristech.fr/~sibille/EW2007/papers/1569014848.pdf
[29] Y. Ma, H. Cao, and J. Ma, "The intrusion detection method based on game theory in wireless sensor network," in *Proc. 1st IEEE Int. Conf. Ubi-Media Comput.*, Jul./Aug. 2008, pp. 326–331. [Online]. Available: http://dx.doi.org/10.1109/UMEDIA.2008.4570911
[30] S. Misra, P. V. Krishna, and K. I. Abraham, "Energy efficient learning solution for intrusion detection in wireless sensor networks," in *Proc. 2nd Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2010, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/COMSNETS.2010.5431976
[31] X. Zhang, J. He, and Q. Wei, "EDDK: Energy-efficient distributed deterministic key management for wireless sensor networks," *EURASIP J. Wireless Commun. Netw.*, vol. 2011, no. 1, pp. 765143-1–765143-11, 2011. [Online]. Available: http://dx.doi.org/10.1155/2011/765143
[32] G. Padmavathi and M. D. Shanmugapriya, "A survey of attacks, security mechanisms and challenges in wireless sensor networks," *Int. J. Comput. Sci.*, vol. 4, no. 1, pp. 1–9, Aug. 2009.
[33] S. M. Lasassmeh and J. M. Conrad, "Time synchronization in wireless sensor networks: A survey," in *Proc. IEEE SoutheastCon (SoutheastCon)*, Mar. 2010, pp. 242–245. [Online]. Available: http://dx.doi.org/10.1109/SECON.2010.5453878
[34] A. T. Tai, K. S. Tso, and W. H. Sanders, "Cluster-based failure detection service for large-scale ad hoc wireless network applications," in *Proc. Int. Conf. Dependable Syst. Netw.*, Jun./Jul. 2004, pp. 805–814.
[35] L. Buttyan and T. Holczer, "Private cluster head election in wireless sensor networks," in *Proc. IEEE 6th Int. Conf. Mobile Adhoc Sensor Syst.*, Oct. 2009, pp. 1048–1053. [Online]. Available: http://dx.doi.org/10.1109/MOBHOC.2009.5337013
[36] National ICT Australia (NICTA), *Castalia Wireless Sensor Network Simulator*, accessed on Jul. 20, 2016. [Online]. Available: https://castalia.forge.nicta.com.au/index.php/en/
[37] A. Varga, "The OMNeT++ discrete event simulation system," in *Proc. Eur. Simulation Multi Conf. (ESM)*, vol. 9. 2001, p. 65.
[38] Crossbow. (Oct. 2015), *TelosB Datasheet*. accessed on Jan. 13, 2016. [Online]. Available: http://memsic.com/userfiles/files/Datasheets/WSN/TelosB_Datasheet.pdf

AHMED AL-RIYAMI received the B.Eng. and M.Sc. degrees in electrical and computer engineering from Sultan Qaboos University, Oman, in 1997 and 2010, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science, The University of Manchester, U.K. His research interests include security, privacy, protocol design and performance analysis in wireless sensor, and ad hoc networks.

NING ZHANG received the B.Sc. degree (Hons.) in electronic engineering from Dalian Maritime University, China, and the Ph.D. degree in electronic engineering from the University of Kent, Canterbury. Since 2000, she has been with the School of Computer Science, The University of Manchester, U.K., where she is currently a Senior Lecturer. Her research interests are in security and privacy in networked and distributed systems, such as ubiquitous computing, electronic commerce, wireless sensor networks, and cloud computing with a focus on security protocol designs, risk-based authentication and access control, and trust management.

JOHN KEANE received the B.Sc. degree in computation from UMIST, and the M.Phil. (M.Sc.) degree in computer science from The University of Manchester. He has worked in industry with Lloyds Bank, Fujitsu, and Phillips. He was part of the team that developed the SUDA family of statistical disclosure software and has worked on privacy data and text mining on projects funded by the U.K. Ministry of Defence and Innovate U.K. He is currently the M.G. Singh Chair of Data Engineering with the School of Computer Science, The University of Manchester, and an Honorary Professor of Data Analytics with the Alliance Manchester Business School. He is an Associate Editor of the IEEE TRANSACTIONS ON FUZZY SYSTEMS.