Research Article

Impersonating-Resilient Dynamic Key Management for Large-Scale Wireless Sensor Networks

Gowun Jeong, Yong-Ho Seo, and Hyun S. Yang

1 AI and Media Laboratory, Department of Computer Science, Korea Advanced Institute of Science and Technology, Daehak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea
2 Department of Intelligent Robot Engineering, Mokwon University, Doanbuk-ro, Seo-gu, Daejeon 302-729, Republic of Korea

Correspondence should be addressed to Yong-Ho Seo; yhseo@mokwon.ac.kr

Received 18 March 2013; Accepted 2 June 2013

Key management in a large portion of ubiquitous sensor networks has been a challenge due to the limited capabilities of their wireless communicating and battery-powered sensors. Moreover, an attacker physically capturing even a few nodes hampers the entire network security by impersonating nodes to inject false data in an undetected manner. To efficiently protect from such impersonating by node capture, we propose a new dynamic key management framework particularly for large-scale clustered sensor networks. In the framework, different keying mechanisms, respectively, secure in-cluster, intercluster, and individual communication by refreshing keys on demand, while adaptively handling node addition and capture. Theoretic analysis and simulation results show that our proposed framework provides higher connectivity and security against impersonating than other existing studies do, for better trade-off with resource overheads.

1. Introduction

Wireless sensor networks (WSNs) of wireless communicating and battery-powered sensors have attracted attention from ubiquitous networking due to such sensors’ cheapness and handy installation. In particular, for unmanned monitoring, these networks are often deployed in unattended and adversarial environments [1]. Here rises providing security against security attacks, which are more likely to incur in such WSNs due to the sensors’ limited capabilities on communication, computation, and storage, as a key issue. Among varied security attacks introduced in [2], we particularly target impersonating by physical node capture because such an attack enables attackers to compromise all the secrets, such as cryptographic keys, of captured nodes and spread malicious data out over the entire network with impersonating the captured nodes by the obtained keys. Thus, as several studies [2–5] have already noted, any security strategies to be proposed should be highly resource-efficient as well as provide the basic security requirements: confidentiality, protection of the content of a packet; authentication, corroboration of the source of a packet; and integrity, ensuring that the content of a packet is unchanged during transmission.

To achieve all of them, a lot of security schemes have been proposed based on symmetric or asymmetric cryptography. Simply speaking, the difference between them is if the same key is employed both by a sender for encryption and by its receiver for decryption or not [4, 6]. Although asymmetric schemes generally provide stronger authentication [2, 4], symmetric key algorithms have been superior in WSNs for their light complexity [5, 7, 8]. For instance, one of typical sensors Tmote has 10 kb RAM, 48 kb flash memory, 1 mb storage, and 250 kbps communication bandwidth, which is insufficient to enable traditional asymmetric cryptography to work [9].

In the paper, we propose a new symmetric key-based security framework for efficiently secure communication in WSNs. To suggest an efficient data aggregation model for large-scale WSNs, we first assume that a WSN consists of a single very powerful base station (BS) and a number of clusters of regular sensors as in [10, 11]. Accordingly, as in
Figure 1, our framework supports three different communication patterns invoked by the following three types of keys: a static individual key shared with BS; a one-time in-cluster key shared within a cluster; and a static intercluster key between two neighbouring nodes across different clusters. We demonstrate how different keying mechanisms for these keys work to cope with impersonation by node capture in the following organisation. Section 2 describes what assumptions we take first and introduces our proposed security framework in cluster-based WSNs of our interest. In Section 3, with designing the impersonating attack model, we provide theoretic analysis of our framework in several performance metrics. Section 4 compares the performances of ours with those of selected conventional key schemes. Finally, we conclude the paper in Section 5.

2. Our Proposed Security Framework

Our framework provides multiple keying mechanisms to support the three communication patterns, while giving confidentiality, authentication, and integrity. We first give several assumptions regarding WSNs of our interest; present the keying schemes to establish and manage each of individual, in-cluster, and intercluster keys with secure transmission employing the keys; and discuss how to handle node addition and eviction.

2.1. Network and Security Assumptions. In the paper, a WSN consists of a single BS and a number of static wireless sensors that are deployed as in [12]. Every time a helicopter stays in a different deployment point, it scatters a sensor subset, called a cluster. In such a cluster, every node knows all of its cluster members’ IDs and directly or indirectly interacts with others in a hop-by-hop manner. One of them is safely announced as the cluster head, a local controller, to the other members. When a node newly joins one cluster after the initial deployment, the head is informed of its ID by BS and lets all the members know the ID as well before it is actually placed. Since any sensor does not know its immediate neighbours in advance, it attempts to find its neighbours and establish required keys shortly after its deployment. This keying phase is assumed to be fairly well protected. However, once a security attacker captures a node, it can obtain all of the node’s cryptographic information. We also assume that the most feasible path from a source to its destination is selected and notified to all the on-path nodes by the cluster-based back-pressure routing algorithm of [13].

We illustrate our security framework using the next notations that appear in the rest of this discussion.

(i) $N$ is the number of nodes in a WSN.
(ii) $N_c$ is the number of clusters in a WSN.
(iii) $D_i$ is the average size of a cluster.
(iv) $D_n$ is the average number of a node’s neighbours.
(v) $D_b$ is the average number of border nodes, which relay messages to the outside of its cluster, in a cluster.
(vi) $i$ and $j$ are principals for clusters.
(vii) $u$ and $v$ are principals for sensor nodes.
(viii) $f_k(s)$ is a pseudorandom function (PRF) based on seed $s$ with key $K$.
(ix) $h(K)$ is a one-way function (OWF) generating a one-way key from key $K$.
(x) $K_u^i$ is the individual key of node $u$.
(xi) $C_i = \{C_j^i\}$ for $0 \leq l < k$ is a one-way key chain shared within cluster $i$.
(xii) $b^u$ is the base of node $u$ based on which $u$ utilises its key chain in its own manner.
(xiii) $r^u$ is a packet sequence number, which increases as time goes by, of node $u$.
(xiv) $O^u$ is a one-time in-cluster key of node $u$.
(xv) $K^{uv}_n$ is the intercluster keys of border node $u$ with border node $v$ of another cluster.
(xvi) $[p]_K$ is the encryption of packet $p$ with key $K$.
(xvii) $[p]_K$ is the collision-free message authentication code (MAC) of packet $p$ with key $K$. Given $p$ and $[p]_K$ with known $K$, the receiver authenticates the sender and ensures $p$’s integrity by confirming that the given $p$ can generate the same MAC value as the arrived $[p]_K$.
(xviii) $d_1 || d_2$ is the concatenation of data $d_1$ and $d_2$ by concatenator $||$.

2.2. Individual Key. Since individual communication with BS contains highly private information, such as new node’s joining announcement by BS or any neighbour’s misbehaviour report by a node, which can be detected by any anomaly-based intrusion detection system discussed in [14], one’s individual key is only shared with BS to authenticate the source of such notification. These keys are assigned by BS prior to deployment as follows.

Key Predistribution. To assign a unique individual key to each node, BS first builds a symmetric key matrix of $N_c \times N_c$, where every key pair of $(i, j)$ and $(j, i)$ such that $i \neq j$ is identical. Every $(i, i)$ key is used to compute a distinct individual key for each member $u$ of cluster $i$ by $K^{u}_i = f_{(i,j)}(u)$. Before
its deployment, every node is preloaded not only such an individual key but also \{\(i, j\)\} for its cluster \(i\) and \(1 \leq j \neq i \leq N_c\) to be utilised in the initial intercluster key establishment.

2.3. In-Cluster Key. We note that in-cluster communication most frequently occurs to exchange successive incoming percepts and to efficiently aggregate data. If a single cluster key is shared within a cluster for economical reasons, an adversary easily endangers the entire cluster by capturing only one node. Thus, we propose that a group of nodes sharing a key chain utilises all of the keys as each one’s one-time encrypting keys in their own manners. One’s current one-time key is derived only by its neighbours in its cluster, called in-neighbours, with its privately known base in advance and the current packet sequence number unless the base is compromised. This idea has, over the conventional key chain studies [1, 15, 16], the following additional advantages: strong key freshness, no need of key disclosure synchronisation, and no message overhead for direct key delivery.

**Key Chain Predistribution.** Before the initial deployment, BS generates and provides a unique one-way key chain of \(k\) keys for every cluster \(i\) based on OWF \(h\) and key (\(i, i\)) of the key matrix. As in \(\mu\)TELSA, \(C_i\) is constructed by \(h(i, i)) = C_{i-1}, h(h((i, i))) = h^2((i, i)) = C_{i-2}, \ldots, h^k((i, i)) = C_0\). To every member \(u\), BS randomly assigns neighbour-distinct base \(b_u\) in \([0, k)\), based on which \(u\) has its own key use order as in Figure 2. Due to the hard-to-reverse nature of OWF, this generation-reversing key-use guarantees that any lower indexed key hardly implies higher indexed keys.

**Neighbour Discovery and Base Exchange.** After its deployment, every node \(u\) first attempts to find its any neighbour \(v\) by broadcasting its id, cluster ID \(i\), and random key index \(r\) in \([0, k)\) in public as well as its base in private by key \(C_i\) as (1). Receiver \(v\) in the same cluster replies as (2). Otherwise, \(v\) in another cluster \(j\) does as (3)

\[
\begin{align*}
\text{(1)} & \quad u \rightarrow * : u, i, r, [b^r]_{C_i} \\
\text{(2)} & \quad v \rightarrow u : v, [b^r]_{C_i} \\
\text{(3)} & \quad v \rightarrow u : v, j.
\end{align*}
\]

Having received packets as (3), \(u\) becomes aware that it is a border node.

**Rechaining and Key Chain Distribution.** When member \(v\) reports its key reference exhaustion on the currently shared key chain to the head of cluster \(i\) or when any member \(v\) is perceived as captured, the head \(u\) generates key \(K' = f_{K^u}(v)\) and new \(C_i\) by \(k\) recursions of \(h(K')\). To propagate this key chain, \(u\) conveys \(K'\) after encrypting it with its current \(O^u\) of the old key chain as follows:

\[
\begin{align*}
\text{(4)} & \quad u \rightarrow * : t^u, [K']_{O^u} [K']_{O^u}.
\end{align*}
\]

Every receiver can generate the same key chain based on the arrived \(K'\) and passes it on to its in-neighbours in the same manner. As soon as a nonborder node forwards \(K'\) to others or a border node generates the new key chain, \(K'\) and the old key chain are immediately erased and every sequence number is reset to 0.

2.4. Intercluster Key. For packets crossing clusters, every pair of border nodes in two adjacent clusters, called inter-neighbours, should share a distinct pairwise key. So far, every node has been loaded \((N_c - 1)\) keys of the key matrix and known if or not it is a border node. Every nonborder node immediately erases the keys because only border nodes make use of them to establish intercluster keys as follows.

**Key Establishment.** Every border node \(u\) of cluster \(i\) is given a series of pairs \((v, j)\) for border node \(v\) of cluster \(j\) after having received messages as (3). It can produce intercluster key \(K'^{uv} = f_{(i, j)}(u \oplus v)\) for each \((v, j)\). Simultaneously, \(v\) also computes the same key, differently named \(K'^{vu} = f_{(i, j)}(v \oplus u)\) for \(u\) of \(i\), for the symmetry as \((i, j) = (j, i)\). As soon as computing all the required intercluster keys, \(u\) erases all the given \((N_c - 1)\) keys.

2.5. Secure Transmission. Now, we present how these established keys practically secure in-cluster, intercluster, and individual communication as in (5) to (7), respectively:

\[
\begin{align*}
\text{(5)} & \quad u \rightarrow v^* : t^u, [p]_{O^u}, [p]_{O^v}, \\
\text{(6)} & \quad u \rightarrow v : [p]_{K^{uv}}, [p]_{K^{vu}}, \\
\text{(7)} & \quad BS \rightarrow v : [u||i]_{K^{uv}}, [u||i]_{K^{vu}}, [p]_{K^{vu}}.
\end{align*}
\]

(i) Within a cluster, node \(u\) sends another node \(v\) or broadcasts packet \(p\) by transmitting the \(p\)'s encryption and MAC by its current \(O^u\) with the current \(t^u\) as (5). Any receiver privately obtains \(p\) after deriving the \(O^v\) with the previously known \(b^u\) and the just arrived \(t^u\).

(ii) Node \(u\) of cluster \(i\) always uses intercluster key \(K'^{uv}\) to encrypt packet \(p\) and produce its MAC when it sends \(p\) to inter-neighbour \(v\) as in (6). As \(v\) also holds \(K'^{vu} = K'^{uv}\), \(p\) is safely restored by \(v\).

(iii) When BS individually informs node \(u\) in cluster \(i\) of packet \(p\), it transmits \(p\) to the first node \(v\) on a given path as in (7). Whereas the former two to address the destination are repeatedly decrypted and reencrypted by the on-path nodes including \(v\) through a deal of in- and intercluster communication until they reach \(u\), the rest two are just carried during the transmission and only decrypted by \(u\). The opposite case of \(u\) to BS reverses the course we have described.

2.6. Handling Node Addition. Before its deployment, new node \(u\) is preloaded its individual key \(K'^u\) only shared with BS, cluster ID \(i\), unique base \(b^u\), and all of \(i\)'s member ids as assumed. BS has enough time to inform \(i\)'s members of \(u\)'s joining and random key \(K'^{iu}\), temporarily used for \(u\)'s key
and information. The notification is started as in (7), where
of $u$ with its own $V$. This also holds for
neighbouring nodes for both in-cluster and intercluster
framework achieves perfect connectivity between any two
nodes. Because of its deterministic nature, our
3.1. Connectivity. Of course, the
joining procedure is finished in
$T_{\text{add}}$, the temporal key $K^m$ is automatically dropped by every member after $T_{\text{add}}$ passes from when it is given.

2.7. Handling Node Eviction. A node is regarded to be evicted when its battery seems to be exhausted; when a large portion of its communication links do not work; or when it is detected as captured. Any of its neighboring nodes perceiving one of the conditions announces it to the entire network. As soon as its uselessness is notified, its cluster members and inter-
neighbours discard every related secret from their memory. In particular, if the node is captured as in the last case, the
rest of its cluster should update the shared key chain as in the rechaining and key chain distribution. Also, its in-neighbours
individually reselect and broadcasts a new base different from the previous one to their in-neighbours as in (1).

3. Analysis of Our Framework

In this section, we, in turn, analyse our framework in the following performance metrics: network connectivity, resiliency against impersonating attacks by node capture, and resource requirements.

3.1. Connectivity. Because of its deterministic nature, our framework achieves perfect connectivity between any two
neighbouring nodes for both in-cluster and intercluster communication at the initialisation phase. This also holds for
newly added nodes due to the prior node ID announcement by BS as in Section 2.6.

3.2. Resiliency. Since we view that active attacks, such as false
data injection, most degrade network performances, we make the following strong attack model by node capture.

(i) The attacker can retrieve all the information stored in
a sensor node once it captures the sensor node.

(ii) The attacker can capture a set of sensor nodes selectively in a WSN.

(iii) The attacker ultimately aims at impersonating legitimate nodes to inject false data with compromised keys.

An attacker can purposely locate and capture sensors having
more secrets as border nodes in our framework by selectively attacking such nodes. Such an attacker can impersonate only
existing nodes whose IDs are compromised because a new
node with a falsified ID is thoroughly excluded due to the
prior node ID announcement by BS. Thus, the resiliency
against this attack is measured by estimating the fraction of
total sensor nodes that properly impersonate ID-known nodes by an attack, modeled as $\{c_i, \alpha_i, \beta_i\}$ for every cluster $i$ in a
network of $N$ nodes and $N_c$ clusters, where $c_i$ is the number of $i$'s captured nodes, $\alpha_i$ is the distinction rate amongst $c_i$'s
in-neighbours in $(0, 1]$, and $\beta_i$ is the bordering ratio of $c_i$ in $[0, D/D_c]$. In this attack model, the greater $\alpha_i$ or $\beta_i$, the
higher the selectiveness of the attack. Then, we formulate the
overall impersonated fractions on in-cluster, intercluster, and
individual communication, $P_m^*, P_{\text{int}}^*$, and $P_{\text{ind}}^*$, respectively, as follows:

$$P_m^* (\{(c_i, \alpha_i)\}) = \frac{1}{N_c} \sum_{i=1}^{N_c} \rho_m (c_i, \alpha_i),$$

$$\rho_m (c_i, \alpha_i) \leq c_i + (\alpha_i c_i D_n (D_n - 1) / (D_n - 1) k) + ((D_n - \alpha_i c_i D_n - 1) D_n / D_c k^2),$$

$$P_{\text{int}}^* (\{(c_i, \beta_i)\}) = \frac{1}{N_c} \sum_{i=1}^{N_c} \rho_{\text{int}} (c_i, \beta_i),$$

$$P_{\text{ind}}^* (\{c_i\}) = \frac{\sum_{i=1}^{N_c} c_i}{N_c}.$$
The overall $p_{in}$ is the normalised sum of impersonated fractions over every cluster, $p_{in}$, given by (10). In cluster $i$, given $(c_i, \alpha_i)$, adversary $a$ behaves as follows:

(i) a completely impersonates every node $u$ of $c_i$ nodes,

(ii) to impersonate each of $u$'s $\Theta(D_u)$ in-neighbours with their compromised bases, $a$ should compute its other $(D_u - 1)$ in-neighbours than $u$ by $(D_u - 1)/(D_c - 1)$ as well as their sequence numbers by $1/k$, for every $u$,

(iii) to impersonate each of the rest $(D_c - \alpha_i, \epsilon \Theta(D_u) - 1)$ members, $a$ should compute its $\Theta(D_u)$ neighbours by $D_u/D_c$, their bases by $1/k$, and sequence numbers by $1/k$.

The three terms of the numerator of (10), respectively, represent each case given above. For intercluster and individual communication, we can simply count and normalise as in (11) and (12) since one captured node reveals only its individual key and additionally intercluster keys if it is a border node. We simulate our resiliency levels for the different communication patterns with those of alternatives given different key chain lengths in Section 4.2.

3.3. Resource Requirements

Storage. Because the key size is usually larger than any other secret as a node ID or a base, we discuss only the number of retained keys for storage overhead. After the initialisation, every node obtains a single individual key, $k$, in-cluster keys of its key chain and additionally $\Theta(D_u)$ intercluster keys if it is a border node. Thus, the required storage of our framework is mainly due to the key chain length $k$ given $D_u$.

Communication. The communication overhead for our security framework occurs between a neighbouring pair during initial keying, rekeying, and keying for a new node. At the initialisation, the pair exchanges only their ids and bases, whereas a seed key for a new key chain and new bases travel for re-keying. For a new node, the pair where one is the new node exchanges at most their ids, bases, a temporal or intercluster key, and the last key of the currently shared key chain as in Section 2.6.

Computation. In the initialisation, every node does $\Theta(D_u)$ times of MAC to securely obtain its in-neighbours’ bases and additionally $\Theta(D_u)$ times of PRF $f$ to generate intercluster keys if it is a border node. To update the key chain, it restores the sent seed key by one MAC operation and generates a new key chain of $k$ recursions of OWF $h$. Given a new node, its neighbours can verify its base or sent intercluster key by one MAC operation. The new node operates one MAC to extract the last key and $(k - 1)$ recursions of $h$ to generate its needed key chain. Letting $r$ be the key size in bit, OWF $h : \{0, 1\}^r \rightarrow \{0, 1\}^r$ consumes $O(r)$ computation, whereas its inverting cost is $O(2^r)$ [17]. PRF $f : \{0, 1\}^r \rightarrow \{0, 1\}^\alpha$ also has a similar overhead [18]. Since MAC is usually regarded as a kind of hash function as OWF [1], MAC has a similarly low computation complexity as well.

4. Comparison with Previous Studies

In this section, we compare the performances of our proposed framework with those of the following three selected conventional studies.

2KP [see [8]]. BS has two key pools of $M_1$ and of $M$ keys and assigns two key sets, $k_1$ keys from the $M_1$-pool and $k$ keys from the $M$-pool, to each node in advance. As soon as every node is deployed, it broadcasts the key ID sequence of its $k_1$-key set. Only pairs sharing one or more common keys can establish a pairwise key somehow computed by a PRF, and then every node drops its $k_1$-key set. For node addition, a new node and its neighbours exchange the key ID sequences of their $k$-key sets to establish their pairwise keys as before.

LOCK [see [19]]. This utilises two layers of keys for clustered WSNs. BS communicates only with weakly trusted cluster key servers (KSSs) with $(k_b + m)$ keys by assigning a unique subset of $k_b$ keys from $(k_b + m)$ combinations to each node before deployment. Every KS distributes its generated $(k + m)$ keys amongst its cluster members in the same manner. In both layers, each member establishes pairwise keys within its included group by exchanging their key ids. Additionally, every regular node shares $k'$ backup keys with BS to report the compromise of its KS.

LEAP+ [see [1]]. Initially, every node $u$ is preloaded the same set of $k$ keys, termed $K^1, K^2, \ldots, K^r$, by which $u$ can derive its own base key in session $I$ by $K^l_u = f_{K^l}(u)$. For $u$ to establish pairwise key with every neighbouring $v$ in session $I$, it first broadcasts its ID and waits for the encrypted $v$’s ID with $K^l_u$ presumed by $v$. Then, $u$ derives $K^l_v$ as well and the pair individually computes the pairwise key by $K^l_{uv} = f_{K^l}(u)$. After the session ends, every node $u$ erases $K^l_v$ and $K^l_{uv}$ from its memory. Every node is also preloaded a unique one-way chain key of $k$ keys and transmits the last key as the current one-time key to its neighbours before it first attempts to broadcast. Then, every time it broadcasts a message, it uses the current one-time key to encrypt both the message and the next one-time key. The current one-time key is discarded after it is used in encryption or decryption.

In the following simulations, we vary only the number of used or stored keys, $k_1, k, k_b, m$, and $m_b$, to see its impact while fixing the network parameters as $N = 10000, N_c = D_c = 100, D_n = 50$, and $D_b = 30$.

4.1. Connectivity. As already stated, the deterministic key establishing methods in ours and LEAP+ guarantee perfect connectivity regardless of the number of keys.

Since 2KP has stably high resiliency with 200 selected keys from 10000 keys [8], we take that $k_b = 200, k = \{50, 200\}$, and $M_1 = M = 10000$. To see the cases with and without new nodes, we also consider $(N_1, N_2) = \{(10000, 0), (9000, 1000)\}$, where $N_1$ and $N_2$ are, respectively, the numbers of initially deployed nodes and of newly added nodes such that $N = N_1 + N_2$. Respectively saying $c_{p1}$ and $c_{p2}$, the connecting
The connectivity of 2KP is highly sensitive with LEAP+ and LOCK for is resulted as in Figure 3. Amongst ours, 2KP, LOCK, and LEAP+, has given our setting in a 100-node cluster.

126, 120, 105, and 120 key combinations for storage efficiency to, respectively, offer \((k, m) = \{(4, 5), (3, 7), (2, 13), (2, 14)\}\).  

In LOCK, the attacker obtains a broadcasting key, it is hard to derive its neighbours’ one-time broadcasting keys. Eventhough the attacker obtains such a broadcasting key, it is hard to derive its future broadcasting keys by the one wayness of OWF. Similarly, our impersonated probability on individual communication is given by \(c/N\) as well.

In LOCK, keys are always kept to establish pairwise keys with new nodes even though the keys are periodically updated. The fraction of total sensor nodes that is impersonated by \(c = \sum_{i=1}^{N_r} c_i\) captured nodes, \(p_{\text{LOCK}}\), is given as follows:

\[
p_{\text{LOCK}}(\{c_i\}) \geq \frac{1}{N_c} \sum_{i=1}^{N_c} \frac{A_i}{(k+m)C_k}. \tag{16}
\]

We say that \(A_i\) is the minimum number of key combinations in cluster \(i\) that \(c_i\) captured nodes can restore. In other words, the attacker obtains at least \(k_i\) distinct keys, which produces \(C_k\) key combinations, from \(kC_k\) compromised keys. This means that at least \(kC_k\) nodes can be impersonated until they are detected as compromised.

Given \(c\) total captured nodes in \([500, 3000]\) for each cluster \(i\), the average resiliency comparison work over 100 simulations is shown as in Figure 4. More specifically, we consider high selective attacks for \(\alpha_i\) in \([0.7, 1]\) and \(\beta_i\) probabilities between any neighbouring pair after \(N_1\) nodes, are deployed and the one after \(N_2\) nodes are added we formulate the overall connectivity of 2KP, \(c_{2KP}\), as follows:

\[
c_{2KP} = \frac{N_1c_1 + N_2c_2}{N_1 + N_2}, \tag{13}
\]

\[
c_1 = 1 - \frac{(M-k)C_k}{MC_k}, \tag{14}
\]

In both (13) and (14), the second term of the right represents the probability that two nodes do not share any keys to form a secure connection.

Regarding that \(k = k_i\) and \(m = m_i\), the connectivity of LOCK, \(c_{\text{LOCK}}\), is generally given in two different cases as follows:

\[
c_{\text{LOCK}} = \begin{cases} 
1 & \text{if } k > m \\
1 - \frac{nC_k}{(m+k)C_k} & \text{if } k \leq m.
\end{cases} \tag{15}
\]

The comparison of direct pairwising connectivity amongst ours, 2KP, LOCK, and LEAP+ has given our setting is resulted as in Figure 3. As illustrated, our framework, LEAP+ and LOCK for \(k > m\), has the perfect connectivity. The connectivity of 2KP is highly sensitive with \(k\) regardless of node addition. LOCK is more likely to share keys as the difference from \(m\) to \(k\) is smaller. Thus, we observe that our framework enhances network connectivity higher than 2KP and LOCK of \(k \leq m\) regardless of the numbers of keys and of newly added nodes.

4.2. Resiliency. If \(c\) nodes are captured in total, no nodes other than the captured nodes can be impersonated in 2KP and LEAP+. Since they do not keep the keys based on which neighbouring pairs establish their pairwise keys using a PRF, the pairwise keys are hardly discovered by unauthorised parties due to the randomness of PRF. For broadcasting in LEAP+, every node owns its one-way key chain as well as its neighbours’ one-time broadcasting keys. Even though the attacker obtains such a broadcasting key, it is hard to derive its future broadcasting keys by the one wayness of OWF. Similarly, our impersonated probability on individual communication is given by \(c/N\) as well.

In LOCK, keys are always kept to establish pairwise keys with new nodes even though the keys are periodically updated. The fraction of total sensor nodes that is impersonated by \(c = \sum_{i=1}^{N_r} c_i\) captured nodes, \(p_{\text{LOCK}}\), is given as follows:

\[
p_{\text{LOCK}}(\{c_i\}) \geq \frac{1}{N_c} \sum_{i=1}^{N_c} \frac{A_i}{(k+m)C_k}. \tag{16}
\]

We say that \(A_i\) is the minimum number of key combinations in cluster \(i\) that \(c_i\) captured nodes can restore. In other words, the attacker obtains at least \(k_i\) distinct keys, which produces \(C_k\) key combinations, from \(kC_k\) compromised keys. This means that at least \(kC_k\) nodes can be impersonated until they are detected as compromised.

Given \(c\) total captured nodes in \([500, 3000]\) for each cluster \(i\), the average resiliency comparison work over 100 simulations is shown as in Figure 4. More specifically, we consider high selective attacks for \(\alpha_i\) in \([0.7, 1]\) and \(\beta_i\)
in [0.15, 0.3] for our framework and the perfectly connected cases of LOCK as \( (k, m) = \{(5, 4), (7, 3), (13, 2), (14, 2)\} \). Whereas the impersonated probabilities of ours, 2KP, and LEAP+ are under or around \( c/N \) for every \( c \), LOCK has weaker resiliency on average with any \( (k, m) \). Thus, we observe that, regardless of the number of retained keys, our proposed framework similarly works as 2KP and LEAP+ known for the strongest resiliency do against impersonating by node capture.

4.3. Resource Requirements. Now, we compare our framework with 2KP, LOCK, and LEAP+ in three resource overhead metrics: storage, communication, and computation. Every analysed overhead is represented by the big \( O \) notation to see its maximum complexity even in the worst case.

Storage. Table 1 provides the storage overheads of the selected keying frameworks. Regarding that \( r \) stands for the size of key in bit, the different storage requirements to nonborder and to border nodes in our framework are given as stated in Section 3.3. By 2KP, every node is preloaded \( k \) keys from the \( M \)-pool for new nodes and shares pairwise keys with \( \Theta(D_n) \) neighbours. In LOCK, every KS keeps \( k \) keys for its cluster communication, and \( k \) keys for its cluster communication, whereas every regular node stores \( k' \) keys to communicate with BS as well as \( k \) keys as its KS does. Any group member establishes pairwise keys with its \( \Theta(D_n) \) neighbours whatever the group is. LEAP+ initially assigns every node \( e \) keys to establish pairwise keys with any nodes added in any session \( l (\leq s) \) and a \( k \)-length key chain to broadcast. Both types of keys are dropped just after they are used. Usually, \( k \) of 2KP and \( e/k \) of LEAP+ take larger values for high connectivity and network longevity, respectively. Thus, on the storage overhead, our framework is superior to 2KP and LEAP+ but is not to LOCK in our simulations, where the cases of LOCK with smaller \( k \) and \( k_b \) achieve better resiliency than those with large \( k \) and \( k_b \) as in Figure 4.

Communication. While assuming that the key size, \( r \), is greater than node ids, key ids, bases, and the MACs of all of these without losing generality, we present the communication overheads required between a neighbouring pair for each of initial keying, rekeying, and keying for a new node by the different studies as in Table 2. For any keying course, our keying framework consumes communication sources with a single key with its MAC, two bases with their MACs, or two node ids as discussed in Section 3.3. Similarly, a neighbouring pair exchanges associated node ids or a broadcasting key with its MAC in LEAP+. By contrast, in 2KP and LOCK, every node broadcasts the sequence of their own key ids as needed to find neighbours having common keys. Since the broadcasting key delivery of LEAP+ more often occurs than our rechaining, our framework reduces the communication overheads of all the keying phases over 2KP, LOCK, and LEAP+.

Computation. All the complexities required to chain keys by an OWF, to produce an MAC by a hash function, and to generate a pairwise key by a PRF can be regarded to be negligible [1, 8]. Thus, the security frameworks without the key ID comparison, ours and LEAP+, have lower computation overheads than 2KP and LOCK do.

Although our framework does not achieve the least storage overhead, it is fairly competitive because the resource consumption of wireless sensor nodes is usually dominated by communication [20].

5. Conclusion

In the paper, we have proposed a new dynamic keying framework for large-scale clustered WSNs, widely employed
to implement ubiquitous sensor networks. In the framework, different keying mechanisms, respectively, not only protect in-cluster, intercluster, and individual communication but also effectively handle node addition and eviction. Our proposed key-use ordering mechanism of a cluster-shared one-way key chain, illustrated in Figure 2, and intercluster key establishment using the preloaded key matrix achieve perfect connectivity as well as well protect wireless sensors from impersonating by node capture with low resource overheads. Such our claims have been discussed by the given theoretic analyses and varied simulations. As one of extensions for this work, we may raise energy efficiency and practicability by utilizing environment energy and considering crosslayer networking as in [21].

Acknowledgments

This work was supported by the Basic Science Research Program through a Grant from the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2012-0005793 and 2011-0013776). This work was also supported by the IT R&D program of MKE/KEIT (10039165, development of learner-participatory and interactive 3D virtual learning contents technology).

References
