http://dx.doi.org/10.3745/KIPSTC.2011.18C.5.303

무선 센서 네트워크에서 Compromised Aggregator에 대응을 위한 모니터링 기반 시큐어 데이터 병합 프로토콜

Boonsongsrikul, Anuparp† · 이 경 석†† · 박 승 규†††

요 약

데이터 병합은 무선 센서 네트워크의 주요 기술이지만 다수의 보안 문제를 유발할 수 있으며, 이들 가운데 하나가 데이터 병합 과정 중의 compromised node에 의한 허위 데이터 삽입이나 메시지 누락이다. 이 문제에 대한 대부분의 기존 해결책은 암호화에 의존하고 있는데, 이들은 많은 연산량과 통신 부하를 필요로 한다. 이와 같은 요구 조건에도 불구하고 기존 방법들은 공격 노드의 탐지가 아닌 공격의 확인 기능만을 가진다. 이러한 제약 사항은 공격이 반복적으로 발생하는 경우, 센서 노드의 에너지 낭비를 유발한다. 비록 기존의 한 연구에서 허위 데이터 삽입 점을 식별하는 기능을 제공하고 있지만, 이 방법은 전체 노드 중 최대 50%의 노드만이 데이터 전송에 참여하는 단점을 가진다. 따라서 공격자 확인이 가능할 뿐만 아니라, 동시에 데이터 전송에 참여하는 노드의 수를 증가시킬 수 있는 새로운 접근 방법이 요구된다. 본 논문에서는 허위 데이터 삽입 혹은 메시지 누락을 수행하는 compromised aggregator에 대한 대응을 위해 모니터링 기반 시큐어 데이터 병합 프로토콜을 제안한다. 제안된 프로토콜은 크게 병합 트리 작성과 시큐어 데이터 병합으로 구성되는데, 시큐어 데이터 병합에는 모니터링을 기반으로 하는 비정상 데이터 삽입 탐지와 최소한의 암호화 기법이 사용된다. 시뮬레이션 결과에 따르면 제안된 프로토콜은 데이터 전송에 참여하는 노드의 수를 95% 수준으로 증가시키는 동시에, 허위 데이터 삽입 혹은 메시지 누락을 수행하는 compromised node의 탐지가 가능한 것으로 확인되었다. 한편 제안된 프로토콜에서 compromised node를 추적하는데 요구하는 통신 오버헤드는 전체 노드의 수가 n일 때, $O(n)$으로 기존 연구보다 우수하거나 유사한 수준을 가진다.

키워드 : 무선 센서 네트워크, 데이터 병합, 보안, 허위 데이터 삽입, 서비스 거부 공격

Monitoring-Based Secure Data Aggregation Protocol against a Compromised Aggregator in Wireless Sensor Networks

Boonsongsrikul, Anuparp† · Lhee, Kyung-Suk†† · Park, Seung-Kyu†††

ABSTRACT

Data aggregation is important in wireless sensor networks. However, it also introduces many security problems, one of which is that a compromised node may inject false data or drop a message during data aggregation. Most existing solutions rely on encryption, which however requires high computation and communication cost. But they can only detect the occurrence of an attack without finding the attacking node. This makes sensor nodes waste their energy in sending false data if attacks occur repeatedly. Even an existing work can identify the location of a false data injection attack but it has a limitation that at most 50% of total sensor nodes can participate in data transmission. Therefore, a novel approach is required such that it can identify an attacker and also increase the number of nodes which participate in data transmission. In this paper, we propose a monitoring-based secure data aggregation protocol to prevent against a compromised aggregator which injects false data or drops a message. The proposed protocol consists of aggregation tree construction and secure data aggregation. In secure data aggregation, we use integration of abnormal data detection with monitoring and a minimal cryptographic technique. The simulation results show the proposed protocol increases the number of participating nodes in data transmission to 95% of the total nodes. The proposed protocol also can identify the location of a compromised node which injects false data or drops a message. A communication overhead for tracing back a location of a compromised node is $O(n)$ where n is the total number of nodes and the cost is the same or better than other existing solutions.

Keywords : Wireless Sensor Networks, Data Aggregation, Security, False Data Injection, Energy Consumption

† 준 회 원 : 아주대학교 정보통신공학과 박사과정
†† 정 회 원 : 전 아주대학교 정보컴퓨터공학부 조교수
††† 정 회 원 : 아주대학교 정보통신대학 교수(교신저자)

논문접수 : 2011년 5월 11일
수 정 일 : 1차 2011년 7월 29일
심사완료 : 2011년 8월 8일

1. Introduction

Data aggregation is important in wireless sensor networks because it substantially reduces the communication cost. However it also introduces many security problems, one of which is that a compromised node may inject false data during aggregation. This may be difficult to detect because the data source is unknown during the in-network processing.

Most existing schemes for detecting false data injection use expensive encryption, which require high computation and communication costs due to the complexity of the encryption algorithm itself and the necessary key distribution and management algorithm. They might be too expensive for some sensors with limited resources (slow CPU, small memory, weak transceiver, and small battery capacity), substantially reducing the sensor network's application functionality.

Most solutions can detect the occurrence of an attack, but solutions merely discard an injected false aggregate without finding the attacking node. As long as the attacker hidden in the network keeps injecting false data, the Base Station (BS) stops receiving correct data and sensor nodes also waste their energy in sending false data.

Recently researchers [1] proposed the lightweight protocol in finding the false data injection attacker. However, the solution has limitations: 1) the protocol requires a specific aggregation tree that many external nodes only monitors an internal node but cannot participate in data aggregation and 2) the protocol does not deal with an attacker which drops an aggregated value.

To counter such problems, in this paper we propose a monitoring-based secure data aggregation protocol to prevent compromised aggregators from injecting false data or dropping messages. Compared with other existing schemes, the main advantages of the proposed protocol are the following. The proposed protocol is applicable to many sensors with limited resource. It also can locate the attacking node so that stop occurring the attack. Note that once the offending nodes are identified, they can be easily handled. For example, the BS can broadcast the ID of the offending nodes so that they are ignored by other nodes in the next aggregation session. The proposed protocol also improves over [1] by increasing the number of nodes that transmit their reading values during the aggregation session, therefore achieving better utilization of the deployed sensors.

We assume that the attacker initially performs passive attacks by eavesdropping on communications and learning the formats of the sensor nodes' data packets. After a particular amount of time, the attacker performs active attacks through a compromised node by either injecting false data in data aggregation or dropping an aggregated value.

2. Related work

Przydatek [10] proposed a secure Information aggregation, which uses a statistical security property to prevent false data injection attacks. However, there is only one aggregator, so it does not scale well to large, multi-hop sensor deployments. In addition, internal nodes do not process sensing raw data but merely compute a hash value using data from their two children. Therefore this scheme requires deployment of many more sensors than other schemes.

Yang [14] proposed a secure hop-by-hop data aggregation protocol. This protocol divides sensor nodes into multiple groups. The aggregated value of each group is sent to the BS. The BS then identifies the suspicious groups based on the set of group aggregates. However, the protocol cannot pinpoint the location of the attacking node. Therefore the attacker may launch an attack by repeatedly injecting false data (or giving an inconsistent Message Authentication Code, MAC).

Vu [13] proposed threshold security for information aggregation in sensor networks, under the assumption of a single aggregator. A sensor node does not provide its signature if aggregation data is not close to its reading value. The BS discards aggregation data, if the number of sensors' signatures is less than a given threshold. The cost of establishing and maintaining cryptographic keys is expensive. In particular, sensors require a cluster key shared by all nodes in a cluster, which is used by the cluster head to encrypt an aggregated value and broadcast it to all nodes in the cluster.

Mlaih [8] proposed a secure hop-by-hop aggregation and end-to-end authentication scheme. The ID list of the sensors involved in a pair of aggregated values is produced, which is used to regenerate a MAC to check the integrity of the aggregated values. Using the ID list, this scheme may identify the attacker. However the ID list is concatenated to the aggregation messages and therefore increases the communication overhead. Moreover, an aggregator who is closer to the BS will have to send longer packets (due to the longer ID list) and therefore suffers a higher communication overhead.

Da Silva [5] proposed a decentralized intrusion detection scheme. In this scheme, monitor nodes issue an alert when the number of occurrences of abnormal behavior is greater than a threshold value, which is computed from all network failures observed by monitors. This scheme is lightweight in that it does not rely on expensive cryptographic techniques. However, this scheme does not focus on securing data aggregation and has other limitations including the following: 1) it needs special monitoring nodes, 2) no security mechanism is provided to protect monitoring nodes themselves, and 3) it cannot pinpoint the attacker. In contrast, our proposed protocol can handle all these problems.

Boonsongsrikul [1] proposed securing data aggregation by identifying the location of false data injection attacks. This scheme is divided into three phases: 1) the query dissemination phase, where the base station initiate the aggregation; 2) the aggregation phase, where all nodes perform the aggregation; and 3) the attestation phase, where suspecting nodes send verification messages to the BS to find suspicious nodes and verify them. However, there are limitations. First, external sensor nodes are only dedicated for monitoring the data sensing nodes and cannot participate in aggregation data because they are left unmonitored. Second, the scheme does not deal with a compromised node which is able to drop an aggregation message. This motivates us to develop our protocol.

3. Requirements of the Proposed Protocol

3.1 Physical phenomena and cryptographic requirements

We assume that the sensor nodes are responsible for sampling physical phenomena such as temperature and humidity. Sensor nodes have overlapping sensing ranges.

The BS shares a unique key with each node and also knows the pairwise key shared between a parent and a child. Unique keys between a node and the BS are pre-installed before the network is deployed. In exchanging a pairwise key between a parent and child, we may use a Needham-Shroeder style key exchange. The BS broadcasts messages to all nodes using μTESLA [9] for the security mechanism.

3.2 Requirements of the aggregation tree

(1) External nodes and the aggregation graph

One of the two main ideas of our detection strategy is monitoring by children. That is, the activities of a node are monitored by its children nodes. However, in this strategy the external nodes themselves are left unmonitored and therefore should not be allowed to provide data during an aggregation session (since they can freely lie). This is a serious problem, because the number of external nodes (who do not contribute to aggregation except by monitoring their parents' activity) may be larger than the number of internal nodes. Thus, when we build the aggregation tree we minimize the number of external nodes by allowing them to monitor each other. Therefore the aggregation tree can be represented as a graph. We will explain this in Section 4.1 in the details.

(2) Minimum number of children per node

To ensure monitoring by children is effective, a node needs to have a sufficient number of children (who can snoop on the messages sent by the node). The minimum number of children is determined a priori, depending on the desired level of accuracy in detecting falsely injected data (since the BS uses a statistical technique for detection in [1], [11] and [14]).

4. Proposed protocol

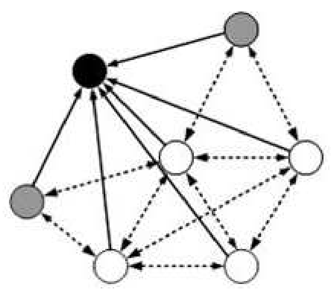
The proposed protocol consists of the following phases.

1. Tree construction phase
2. Tree post-processing phase
3. Query dissemination phase
4. Aggregation phase
5. Attestation phase
6. Testification phase

Aggregation tree construction consists of phases 1 – 2. Secure data aggregation consists of phases 3 – 6. In phase 5, nodes who suspect an injection of false data send verification messages to the BS. In phase 6, nodes who did not receive aggregation messages from their children send a report to the BS in order to deal with dropping data attacks. The message formats for these phases are described in <Table 3 and 4>.

4.1 Tree construction phase

In the tree construction phase, we build the aggregation tree that is suitable for our protocol. Basically, we ensure that each node has a minimum number of children. We propose two algorithms to build such a tree, as described in <Table 1 and 2>. The first algorithm produces a shortest-path spanning tree using breadth-first search as in [15], where the degree of an internal node is equal to



(Fig.1) Classification of external contributing nodes (white) and external non-contributing nodes (gray)

or greater than the minimum number of children (Section 5.1 later illustrates this algorithm in more detail). Subsequently, the second algorithm reshapes the tree to allow multiple parents for an external node (therefore the tree is represented as a graph). This maximizes the number of nodes being monitored. Those nodes will be eligible to provide data during an aggregation session. After this algorithm, each node will be one of the three types below (refer to Fig. 1).

- Internal node (black node)
- External contributing nodes (white nodes)
- External non-contributing nodes (gray nodes)

An external node who can find a minimum number of children will be an external contributing node. An external contributing node has 1) a primary parent, whom it monitors and sends its reading value to, and 2) a number of secondary parents whom it monitors but does not send its reading value to. Note that an external contributing node himself becomes a secondary parent of its siblings.

An external node who cannot find a minimum number of children will be an external non-contributing node. Such a node only monitors its parent(s) but does not provide data during an aggregation session.

In (Fig. 1), external nodes whose in-degree is equal to or greater than three are contributing nodes (white nodes). The solid lines denote the primary parent-children relationship, and dotted lines denote secondary parent-children relationship.

Note that a memory overhead to build an aggregation tree is equal to the number of IDs of neighboring nodes which rebroadcast the BS's flooding message.

4.2 Tree post-processing phase

After building the aggregation tree (actually a graph, but we will henceforth refer to it as a tree for

〈Table 1〉 Building the aggregation tree algorithm

```
1) In the flooding stage,
  a) The BS broadcasts the discovery message.
  b) Each node receiving the flooding message records it
     in the parent list. The node rebroadcasts it, if it
     received the flooding message for the first time.
2) After flooding, each node
  a) Picks a candidate parent from its parent list and
     requests childship to the candidate.
  b) Picks its children from the received requests by
     choosing t children (where t is the minimum number
     of children), and acknowledges parentship to the
     chosen children.
  If the node receives less than t requests, it will be an
  external node.
3) To reduce orphans (nodes who do not have parents)
  a) Each orphan broadcasts an adopt-me message.
  b) Each internal node that received an adopt-me
     message replies by sending an adoption message to
     the orphan.
  c) The orphan chooses its parent from the adoption
     messages and acknowledges the parent.
```

〈Table 2〉 Reducing the number of external nodes algorithm

```
1) Each orphan broadcasts an adopt-orphan message.
2) Each external node broadcasts an adopt-external
   message.
3) If an external node receives more than t such
   messages (where t is the minimum number of
   children) then it will be an external contributing node.
  a) It picks t nodes and replies by sending an adoption
     message to them (chooses orphans first and then
     considers external nodes).
  b) Each external node who receives an adoption
     messages marks the senders as its secondary
     parents (note that an external node sends an
     aggregation message only to its primary parent, not
     to its secondary parents).
4) Otherwise it will be an external non-contributing node.
```

simplicity), each node sends a report to the BS. The report contains the following information.

- The node's primary parent and the list of secondary parents

  The purpose of this information is to enable the BS to find out the topology of the aggregation tree in order to verify the integrity of the tree.
- The type of node (internal, external contributing or external non-contributing)

  The purpose of this information is to let the (primary) parent of this node snoop on this message in order to discover whether or not this node is allowed to provide data during aggregation.

If node A falsely claims to be an internal or external contributing node, the BS can detect that node A is a liar because the number of reports (confirming that node A is a parent) is less than a given threshold. The BS lets the parent of node A know that node A is a liar. Then node A is not allowed to provide data.

4.3 Query dissemination phase

Data aggregation may be regularly requested (this depends on applications). The BS starts data aggregation by flooding a query message. We do not provide an authenticated broadcast mechanism but instead assume a technique of μTESLA [9].

4.4 Aggregation phase

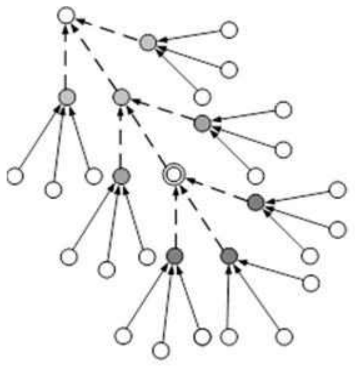
Different from query dissemination, data aggregation starts from external nodes towards the BS. Since an external contributing node does not need to do aggregation, so it sends its reading value to its primary parent. The primary parent first verifies the AMAC (MAC type A, explained in Table 3). If the AMAC is legitimate, then the primary parent aggregate its own reading value and a child's reading value; otherwise it discards the message. External non-contributing nodes sense data values but do not send aggregation messages. Their roles are to monitor their parents and participate in the attestation phase. If an external non-contributing node sends an aggregation message to its parent, the message will be discarded because its parent knows the types of their children (after post-processing the aggregation tree). If there are no suspicious values after a primary parent and all secondary parents send a data value, a node goes into the sleep mode. The node has done monitoring.

4.5 Attestation phase

During the aggregation phase, each child also overhears the aggregation message sent by its parent (to the grand parent). In detecting an abnormal value, we can consider two cases.

1. If the difference between the readings of a child and that of its secondary parent is greater than a given threshold then the secondary parent is considered suspicious. Since a child is close to its secondary parent, so their readings should be similar.
2. If an aggregated value of a primary parent is abruptly changed from aggregated values in the history then the primary parent is considered suspicious. We assume that aggregated values are initially trustworthy (before performing passive attacks). A node compares an aggregated value with the aggregated values given by its primary parent in the history.

Note that we do not consider detecting the case where a compromised node tries to forge an aggregated value


(Fig. 2) The attestation tree. The dashed edges are the attestation tree and the double circle is an attack node. Black, gray and light gray nodes denote high, moderate, and low probability of detecting abnormal values, respectively

<Table 3> Message fields

Notation	Meaning
type	1-bit value specifying the types of aggregation functions (sum or mean).
op	2-bit value specifying the message type (aggregate, attest, absence or testify).
sid	Unique session identifier.
id	Node identifier.
pid	Identifier of a parent node.
gpid	Identifier of a grandparent node.
cid	Identifier of node who did not send an aggregate.
R	A reading or raw data value of a node.
agg	An aggregated value combined by a reading of a node and readings of children.
pagg	An aggregated value of a parent which a child snoops on.
$K_{i,j}$	Shared key between node *i* and *j*.
AMAC	*MAC* of data *A* calculated with key $K_{id,pid}$ where *A* is a concatenation of *id* and *sid*.
BMAC	*MAC* of data *B* calculated with key $K_{id,BS}$ where *B* is a concatenation of *id*, *pid*, *agg* and *sid*.
pBMAC	*BMAC* of a parent which a child snoops on.
CMAC	*MAC* of data *C* calculated with key $K_{id,BS}$ where *C* is a concatenation of *id*, *pid*, *R*, *agg*, *pagg* and *sid*.
DMAC	*MAC* of data *D* calculated with key $K_{id,BS}$ where *D* is the concatenation of *id*, *cid* and *sid*.

<Table 4> Message formats in an aggregation session

Messages	Format
query	*<sid,type>*
aggregate	*<op,id,pid,agg,AMAC,BMAC>*
attest	*<op,id,pid,R,agg,pagg,CMAC,pBMAC>*
absence	*<op,id,cid,DMAC>*
testify	*<op,id,pid,gpid,pagg,pBMAC>*

being a normal data range because it has a little effect and hard to detect. In contrast, we are interested in detecting a compromised node that is trying to forge aggregation data of their non-compromised children.

When a node finds an abnormal value, it sends an alert message to the BS. This alert message is called an attestation message. The attestation message is not sent through the aggregation tree. Instead, the sender picks another node in its parent list to send its attestation message. This is because it may not trust its primary parent to reliably forward its attestation message. Such a routing is possible for many existing routing algorithms. Note that there may be many shortest paths between a node and the BS if the network is sufficiently dense [12].

In order for an attack to be effective, the data value injected by the adversary usually needs to be quite different from the normal reading value. However, in such a case, children of the adversary would send attestation messages to the BS. All the ancestor nodes on the path from the adversary to the BS are affected by the injected value to a diminishing degree, so their siblings would detect an anomaly and send attestation messages with decreasing probability. The BS then uses the attestation messages to identify a subtree consisting of alerting nodes. We call it an attestation tree, which is used to locate the adversary. (Fig. 2) illustrates such an attestation tree.

4.6 Testification phase

If a node sends a normal value with an invalid AMAC in an aggregation message, its parent will discard its message. However, its children who snoop on the aggregation message would falsely assume that it successfully sent its aggregation message.

<Table 5> Finding dropping data attack algorithm

1) Before starting the aggregation phase, each internal node sets a timeout according to the longest aggregation time.
2) After the time has expired, each internal node sends an absence message to the BS, containing the IDs of any of their children who did not send aggregation messages (the suspicious nodes).
3) From these messages, the BS builds a testification tree.
4) The BS finds the suspicious node claimed by the leaf node of the testification tree, and asks the children of that suspicious node to send the aggregation message of the suspicious node (which they snooped during the aggregation phase).
5) If the children supply the valid aggregation message of the suspicious node, then the parent of the suspicious node is the attacker (case 2 in Fig. 6). Otherwise the suspicious node is the attacker (case 1).

In effect, a (compromised) node can drop a message. To deal with this type of attack, nodes who did not receive an aggregation message from one or more of its children (and are thereby unable to complete the aggregation process) participate in the testification phase by sending absence messages to the BS. Then the BS builds the testification tree which is a subtree consisting of the nodes who sent absence messages as illustrated in (Fig. 5).

For this we assume that each node knows the longest time required to complete an aggregation session, which may be inferred from the number of nodes, average node density, collision rate, etc. [7]. The algorithm for this phase is shown in <Table 5>.

5. Detecting the Injected False Aggregated Value

In this section, we discuss a scenario in detecting a false data injection attacker. When the BS receives attestation messages, it computes CMACs (MAC type C) to verify authentication of reporting nodes. If CMACs computed by the BS and CMACs sent by reporting nodes match, the BS uses *id* and *pid* in attestation messages to build an attestation tree. The parent of leaf nodes of the attestation tree is the most suspicious. If this parent has either an inconsistent value which its children snooped or an outlier aggregate, it will be a compromised node. The BS carries out two steps: 1) checking consistency and 2) finding an outlier.

5.1 Checking consistency

The BS retrieves a reading of a parent of leaf nodes in the attestation tree. If the parent gives inconsistent data value between the aggregation phase and the attestation phase then it is the attacker. As illustrated in (Fig. 3), we assume using a sum function for data aggregation. Node S is a parent of nodes r0, r1, r2, r3 and r5. Node r2 is an internal node. Nodes r1 and r3 are an external contributing node. Nodes r0 and r5 are an external non-contributing node. In the aggregation message, node r2 sends (aggregated value) 14 while node r1 and r3 send (reading) 3 and 5 respectively. Node S sends (aggregated value) 38 to its parent. If node r1, r2 and r3 consider that node S sends an abnormal value then they report the attestation message to the BS. Since an aggregated value of node S exists in the attestation messages of children, the BS can retrieve a reading value of node S. The reading of node S (R_S) is 38-3-14-5 = 16. Note that the aggregated value is not included readings of external non-contributing nodes r0 and r5. If node S does not give R_s = 16 to the BS, then node S is the attacker. (Since node S possibly gives a normal range of R_s but an abnormal aggregate of its parent, *pagg*, in its attestation message in order to claim that it does not inject false data but its parent does.)

5.2 Finding the outlier

If node S gives its reading, R_s = 16, then the BS goes into a next step in computing the outlier. If value 16 is the outlier then node S is the attacker. Otherwise, if value 16 is not the outlier, then there is a false positive. The outlier is normally quite different from the other values. Criteria in considering the outlier depend on the allowable value and applications. To compute the outlier, the BS uses a set of readings in the attestation messages. As illustrated in (Fig. 3), the partial formats of the attestation messages given by node r0, r1, r2, r3, r4, r5 and S are <R=2, *agg*=2>, <R=3, *agg*=3>, <R=4, *agg*=14>, <R=5, *agg*=5>, <R=6, *agg*=6> and <R=16, *agg*=38> respectively. The *agg* denotes an aggregated value combined by a node's reading and children's readings. Assume a reading value of node r2 is 4 and after combining the reading and children's reading, the aggregated value is 14. The completed format of the attestation message is showed in <Table 4>. Therefore the set of the readings given by the leaf nodes and their parent in the attestation tree is {2,3,4,5,6,16}. In computing whether or not a suspicious value, Sv, is the outlier the BS computes the sample statistic, $S_s = (s_v - \overline{x})/s$ where $\overline{x}$ and s are the mean and standard deviation of all readings in the set. If the sample statistic, S_s, falls in the rejection range defined by the critical values then the suspicious value, S_v is the outlier. (In our example, S_v = 16) However, we do not propose an algorithm in finding the outlier. We adopt an algorithm [14] in finding the outlier. The other example of computing an outlier is showed in [1].

Note that when a compromised node is found, the BS can broadcast the ID of the offending node so that it will be ignored by other nodes in the next aggregation session. Alternatively, we revoke compromised keys using techniques in [3], [4], and [16]. Therefore, the attacker cannot inject false data after revoking compromised keys.

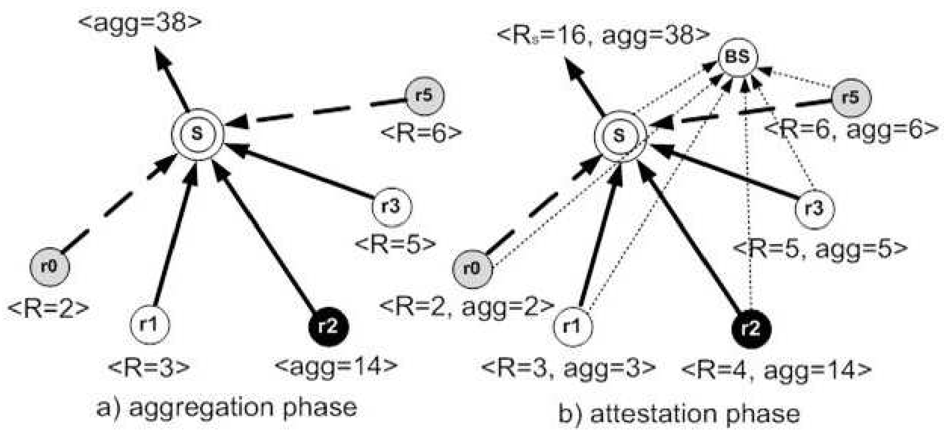
6. Security in the Aggregation Phase

In this section we discuss various attack scenarios that an adversary attempts to inject false id, data, MAC, etc. during the aggregation phase and how our protocol can cope with them.

As illustrated in (Fig. 4), a compromised node A fabricates an aggregation message (containing an abnormal value) from one of its children to himself. Note that such a message is valid, because the attacker knows the AMACs of its children. Then, it may or may not choose to send an abnormal aggregation message to its parent.

Case a. If node A sends a normal value to its parent, then it is simply framing one of its children but there is still no false data injection. That is, it aims to induce the children of the victim to send attestation messages and therefore drain the energy of the children.

Case b. If node A sends an abnormal value to its parent, then it is trying to inject a false data while framing one of its children. In this case, children r1, r2 and r3 of node v send an attestation message to the BS.



(Fig. 3) A suspicious node and leaf nodes in the attestation tree. The solid edges representing data values sent to a parent. The dashed lines are monitoring by external non-contributing nodes. The dotted edges representing data values sent to the BS

The BS will then consider the victim as the attacker if the victim is on the lowest level of the attestation tree. In other words, node A aims to inject a false data while framing the victim. However, the BS can distinguish attestation messages that are generated by a fake aggregation message, because such attestation messages cannot contain the valid BMAC (MAC type B) of the victim (pBMAC, BMAC of a parent) where a BMAC is a Message Authentication Code between a node and the BS.

In Case b, the attacker is detected because it is on the lowest level of the attestation tree (this is because attestation messages from the children of the victim are found to be invalid).

In Case a, the BS only knows 1) the children of the framed node, and 2) the fact that the attacker is within the transmission range of the victim's children. In other words, the BS cannot locate the attacker. However this attack that drains the energy of child nodes is out of the scope in this paper. A viable alternative to avoid this type of attack is to introduce a group key that is shared by a parent and its children (as many other schemes have done). However, this would substantially increase the complexity of the protocol.

7. Security in the Attestation Phase

In this section we discuss various attack scenarios that an adversary may attempt during the attestation phase, and how our proposed protocol can cope with them.

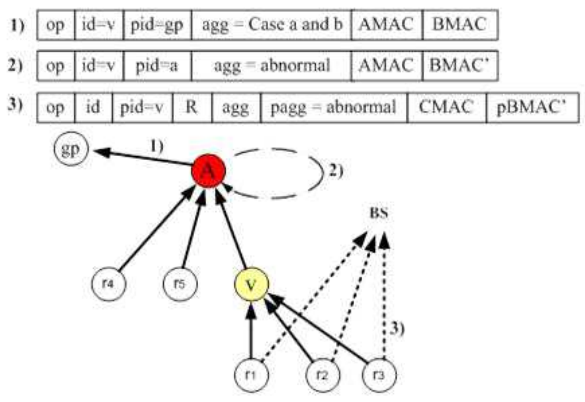
7.1 False information in the attestation message

An attacker might include an abnormal data value in the aggregation message but include a normal data value in the attestation message in order to fool the BS. However, an attestation message contains the data value of its parent (*pagg*) as well as its own data value (*agg*) <Table 4>. If the attacker lies, the many attestation messages of its children will prove him as a liar (monitoring by children and decision by majority).

Also, an attacker might include a false pid and an abnormal value in data in order to frame an innocent node. However, the BS will not regard the framed node as an attacker unless a minimum number of messages had implicated him (decision by majority).

7.2 Not sending an attestation message

An attacker might choose not to send the attestation message because it cannot successfully report false


1) op | id=v | pid=gp | agg = Case a and b | AMAC | BMAC
2) op | id=v | pid=a | agg = abnormal | AMAC | BMAC'
3) op | id | pid=v | R | agg | pagg = abnormal | CMAC | pBMAC'
gp
1)
A
2)
BS
r4
r5
V
3)
r1
r2
r3

(Fig. 4) Examples of sending fake aggregation messages. Node A is the attacker and v is the victim. The dashed edge is fake aggregation message. The dotted edges are attestation messages. BMAC' is invalid

information during the attestation phase. However, the BS will still discover its information, because the attestation messages sent by its children include this <Table 4>.

8. Security in the Testification Phase

In this section we discuss how an adversary attempts to drop a message and how our algorithm detects such attempts.

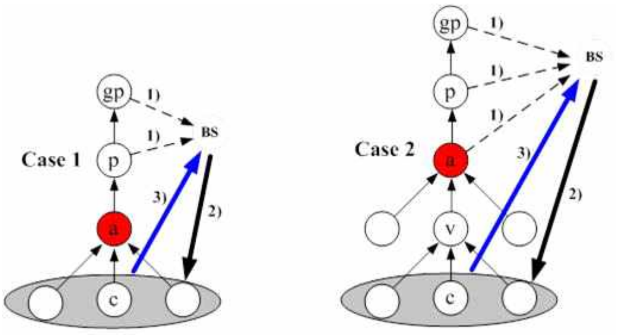
8.1 Not reporting a missing aggregation value

The attacker who did not send its reading value may not report to the BS, for the obvious reason that it wants to avoid detection. That is, the attacker does not send an absence message (Case 1 in Fig. 5) including DMAC (MAC type D). The resulting testification tree would actually be a path unless there are multiple attackers.

According to the testification algorithm (in Table 5), the BS builds a testification tree, which consists of nodes p and gp in Case 1. Subsequently, the BS chases down the testification tree, finds the last suspicious node (it would be node a in Case 1 as indicated by p), and asks the children of the suspicious node to send the aggregation message of the suspicious node (which they snooped during the aggregation phase).

The BS then can verify the AMAC and BMAC of the aggregation message.

In this case, the children either did not observe an aggregation message sent by the suspicious node, or have snooped on aggregation messages that contain an invalid MAC. In either case, the BS knows that the attacker is dropping a message. In other words, the children testify against the attacker.



(Fig. 5) Examples of the testification process. Case 1 shows testification of this dropping data attack. Case 2 shows testification of framing a child. 1), 2) and 3) represent an absence message, a query of the BS and a testification message respectively

8.2 Framing a child

The attacker may choose to send an absence message, this time framing one or more of its children (Case 2 in Fig. 5). The resulting testification tree contains nodes a, p and gp. Again, the children of the last suspicious node (the framed node v) are asked to send the aggregation message of v. Those messages contain a valid MAC, so the BS knows that the parent of v is lying and hence the real attacker. In other words, the children testify for node v.

9. Communication Overhead

In this section we analyze the communication overhead during 1) construction of the tree and 2) an aggregation session. We estimate the communication overhead as the number of hops traveled by all messages.

In summary, the communication overhead for building an aggregation tree takes $O(n\sqrt{n})$, but this may not be a serious concern, because it is not necessary to build the aggregation tree often.

More importantly, the communication overhead of the entire aggregation session (including aggregation, attestation and testification phase) remains $O(n)$.

We assume that the number of hops (or the path length) between any two nodes is, where n is the total number of sensor nodes uniformly deployed in two dimensional space [2].

• Overhead in the tree construction phase

In the aggregation tree algorithm, steps 1 b), 2 a), 2 b), 3 a), 3 b), and 3 c) each take $O(n)$. In step 3 b) we assume the number of neighbors is a constant.

In the aggregation graph algorithm, steps 1, 2, 3 a), and 3 b) each take $O(n)$. In step 3 a) we assume a threshold value is constant. Therefore the overhead in the tree construction phase is $O(n)$.

• Overhead in the tree post-processing phase

Since a distance from a node to the BS is $O(\sqrt{n})$, therefore a communication overhead of information sent by a node via intermediate nodes to the BS takes $O(\sqrt{n})$. The information sent by a node denotes its node's type (illustrated in Fig. 1) and the list of its primary and secondary parents. Since there are n sensor nodes in the network, therefore total communication cost in sending the information is $O(n\sqrt{n})$.

• Overhead in the query dissemination and aggregation phase

A query dissemination message can be flooded to all nodes with $O(n)$ number of hops. The aggregation phase requires $O(n)$ number of hops because a node sends an aggregation message.

• Overhead in the attestation phase

The number of messages generated in the attestation phase depends on the number of attackers and their location. In this section we consider a simple case where there is only one attacker.

As we mentioned in Section 4.5, if the abnormality of a data value is high enough, then all the nodes (and their siblings) on the path from the abnormal node to the BS would send attestation messages. The total number of hops traveled by all the attestation messages can be estimated by equation (1), where d is the maximum node degree (the minimum number of children per node) and pl is the path length (the level of the attestation tree). Each attestation message is unicast and takes the shortest path to the BS, as explained in Section 4.5.

$$\begin{aligned} \mathrm{Cost}_{\mathrm{attest}} &= \mathrm{d} \times (1+2+3+\cdots+\mathrm{pl}) \\ &= \mathrm{d} \times \sum_{i=1}^{pl} i \\ &= O(\mathrm{p}l^2) \\ &= O(n) \end{aligned} \tag{1}$$

The attestation phase thus takes $O(n)$ number of hops.

• Overhead in the testification phase

In the testification algorithm, step 2) takes 1+2+ ... + pl = $O(n)$, where pl is the path length of the testification tree. Step 4) takes $d \times \sqrt{n} = O(\sqrt{n})$ where d is the node degree (the number of leaf nodes of the suspicious node). The overall cost of the testification phase is thus $O(n)$.

10. Simulation Results and Comparison with Existing Work

In this section we present our simulation results that provide the following information.

1. An example of an aggregation tree and an attestation tree
2. The properties of the aggregation tree
3. The communication overhead in the attestation phase
4. The energy consumption
5. Comparison with related work

Below are our assumptions for the simulation in Section 10.1, 10.2 and 10.3.

- The BS is at coordinate (0,0) while sensor nodes are randomly deployed in a square area (1,000 – 18,000 nodes)
- The minimum number of children is 3, 5, 7, 9 and 11
- The adversary node is far from the BS, in order to compute the worst case communication overhead
- Each communication overhead is averaged based on 100 experiments

10.1 An Example of an aggregation tree and an attestation tree

(Fig. 6) shows the result of the aggregation tree after the aggregation tree algorithm <Table 1> and the aggregation graph algorithm <Table 2> are applied. After stages 1) and 2) of the aggregation tree algorithm, the tree is basically the same as a shortest-path spanning tree such as ENCAST [15], except nodes who cannot find a minimum number of children remain as external nodes. As a result, nodes whose level is lower than such external nodes are unconnected (orphans).

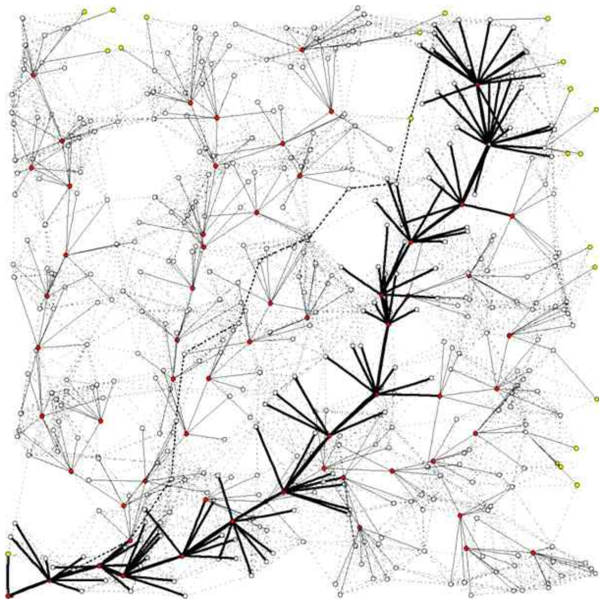
However, after stage 3) of the aggregation tree algorithm, practically all the orphans are adopted by some internal nodes, unless they are not in the transmission range of any of the internal nodes. After the aggregation graph algorithm, the aggregation tree is augmented with more edges among external nodes (shown in dotted line). However, the two nodes, connected by such an edge, do not exchange data (such a node only monitors the other node). Therefore, although the aggregation graph appears to be dense, the communication overhead does not increase.

The thick edges show an attestation tree. In identifying a compromised node, the most suspicious node is a parent of leaf nodes of the attestation tree. The dashed line shows an example of an actual shortest path which a reporting node sends the attestation message to the BS. As explained in Section 4.4, a reporting node who suspects its primary parent chooses an alternate shortest path to the BS by sending the message to one of the nodes in its parent list. If leaf nodes' parent in the attestation tree has either an inconsistent value, which its children snooped, or an outlier aggregate, then this parent is a compromised aggregator.

10.2 Properties of the aggregation tree

(1) Connectivity of the aggregation tree

According to the aggregation tree algorithm (Section 4.1), nodes who cannot find the minimum number of children will be external nodes. As a result, nodes whose level is lower than such nodes are unconnected (orphans). In general, the connectivity is higher if the minimum number of children is less than the average number of neighbors. In our experiment, orphans are practically nonexistent if the average number of neighbors is twice as large as the minimum number of children (or more). For example, when the minimum number of children is 11 we need more than 20 neighbors in order to avoid orphans.

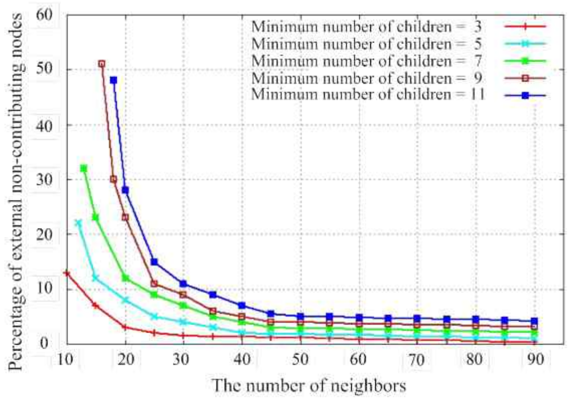


(Fig. 6) The final aggregation tree (graph). The dotted lines among external nodes denote secondary parent-children relationships. An attestation tree is shown as thick edges. The dashed edges show an example of a path, along which an attestation message is routed to the BS

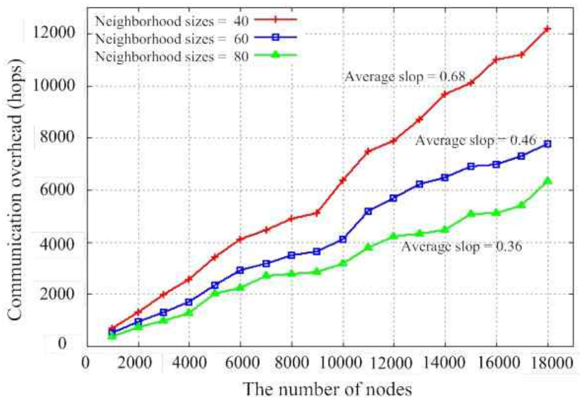
(2) Percentage of non-aggregating nodes

A node is not allowed to provide data during aggregation if it cannot find a minimum number of children (who monitor him), because we cannot believe its data. In general, it is more likely to have a minimum number of children (and hence will be a data-providing node) if it has more neighbors or a lower number of monitoring nodes (children) is required.

(Fig. 7) shows the result of our simulation. The result shows that with the neighbor size of 40, the percentage of non-aggregating nodes is well below 10 for all of the node degrees (minimum number of children) we tested. For a node degree such as 5 or less, even 20 neighbors suffice. This result clearly shows that the number of non-aggregating nodes is negligible. For all the cases of the minimum number of children, our simulation shows that 95% of total nodes can participate in the data transmission when the number of neighboring nodes is 50.



(Fig. 7) Percentage of monitoring-only nodes



(Fig. 8) Communication overhead when the minimum number of children is 11. Note that we excluded the case of 20 neighbors in order to avoid unconnected nodes

10.3 Communication overhead in the attestation phase

The results of our simulation (see Fig. 8) show that the communication overhead is linear with respect to the number of nodes, as analyzed in Section 4.4. Generally speaking, with a given number of nodes, the stronger the radio range (hence the higher the number of neighbors), the lower the number of hops it takes to send a message to the BS. Our simulation result confirms this expectation.

10.4 Energy consumption

We divide this section into two sections. First, we measure energy overhead when a false data injection attack occurs. Second, we compare wasted energy between our work and related work [14].

To measure energy consumption, we use ns-2 simulator [16] and energy models [17]. The energy required in sending a message of a node is s. ($\delta+\Theta\times d^q$), where s is the message size, δ = 50 nJ/b is a distance-independent term, Θ = 100 pJ/b/m is the coefficient for a distance-dependent term, q = 2 is the exponent for the distance-dependent term, and d = 15 m is the transmission distance. The energy required in receiving a message of a node is s×ɣ, where ɣ = 50 nJ/b is a coefficient independent of transmission distance.

The initial energy budget at each sensor node is set at 50 J. Let *op*, *id*, *agg* and MAC be 2, 16, 32 and 64 bits. Therefore, aggregation and attestation data of our work are 194 and 226 bits. Aggregation and attestation data of work [14] are 152 and 136 bits, respectively. We simulate over ten topologies where 100 sensor nodes are randomly deployed in an area of 50 × 50m^2 and the locations of a compromised node are variously changed in order to find an average energy in detecting the attack.

• Energy overhead

We measure the energy overhead when there is a false data injection attack. Average energy consumption of all sensor nodes that send aggregation data is 438mJ and average energy consumption of sensor nodes that send attestation messages to the BS is 102mJ. Therefore, an average energy overhead is 23.3% increase when there is a false data injection attack.

• Wasted energy between our work and related work [14]

In this section, we simulate and compare wasted energy between our work and work [14] when a single compromised node keeps injecting false data in a network.

Let β denote an accumulation of wasted energy.

<Table 6> Accumulation of wasted energy

The number of false data injection attacks	β (mJ)	
	Our work	Work [14]
1	540.6	293.8
2	540.6	587.6
3	540.6	881.4
k ($k > 3$)	540.6	293.8×k

Wasted energy consists of 1) energy of sensor nodes which send false data, 2) energy of sensor nodes which send attestation messages to the BS and 3) energy of sensor nodes which forward flooding messages of the BS to announce the ID of a compromised node.

Except work [14], we cannot find information about the packet size of aggregation data. Therefore, in this section we only compare our work and work [14].

Since the packet size of aggregation and attestation data of our work are longer than that of work [14], energy consumption of our work is 46% higher during the first false data injection attack. However, after a compromised node is found, it will be ignored by other nodes and cannot participate in data transmission. Therefore, sensor nodes do not waste their energy to send false data in the next aggregation session. Wasted energy will be 540.6mJ as illustrated in <Table 6>. Wasted Energy of work [14] will increase as a linear function because their sensor nodes waste their energy to send false data. Energy consumption of work [14] is higher than that of our work at the 2nd round of false data injection where a false data injection is occurring every aggregation session.

10.5 Comparison with existing work

<Table 7> compares the communication overhead, limitations and the percentage of the average number of contributing nodes per total nodes for each scheme. The communication overhead of the proposed scheme in an aggregation session is the same as or better than the overhead of other schemes. But unlike other schemes, our work can identify the adversary node as well. While many existing schemes assume centralized aggregation, our scheme assumes hierarchical aggregation and therefore scales well with a large number of sensor nodes. In [8], [13] and [14] their aggregation message requires encryption and decryption between a sender and a receiver. This increases a computation cost of sensors. Unlike their schemes, our scheme avoids using encryption and decryption where an aggregation message is only plaintext. Our solution is more simple and lightweight than them because we use merely a minimal cryptographic technique that is a MAC. In terms of a computation cost occurred by encryption, our current solution is as efficient as the solution [5] and more efficient than the solutions [8], [13] and [14]. In terms of contributing nodes that can participate in an aggregated value, our proposed protocol has more the number of contributing nodes than that of the solutions [1] and [10]. (Fig. 7) shows that even though only 95% of total

<Table 7> Comparison of security aspects, communication overhead and the number of contributing nodes

Approaches	Security mechanism	Limitation(s)	Communication overhead		The percentage of the avg. num. of contributing nodes per total nodes
			Building a tree	Detecting an attack	
Madden[7]	No use	No security	N/A	$O(n)$	100
Przydated[10]	Hash tree MAC	Internal nodes do not sense data	N/A	$O(l^2)$, $l \leq n$	50
Yang[14]	Encryption MAC	Cannot identify attacker	N/A	$O(n)$	100
Boonsongsrikul[1]	MAC	External nodes do not participate in aggregated values	N/A	$O(n)$	Less than 50
Vu[10]	Encryption MAC	Cannot identify attacker and needs cluster keys	N/A	$O(n)$	100
Mlaih[8]	Encryption MAC	Long aggregation messages	N/A	$O(n\ ln(n))$	100
Da Silva[5]	MAC	No security for monitoring nodes	N/A	$O(n^2)$	100
Our work	MAC	Require our specific aggregation tree		$O(n)$	95

number of nodes participates in data transmission, our proposed protocol detects the attacks and also identifies the location of the attacker.

11. Conclusion

We proposed a protocol for detecting false data injection that uses minimal cryptographic techniques. Our proposed protocol can also find the attacker, whereas to our best knowledge no other schemes can. Our proposed scheme can also handle other kinds of attacks during data aggregation, including framing other nodes and dropping an aggregation message.

In our protocol, the communication overhead of the entire aggregation session (aggregation, attestation, and testification phase) remains $O(n)$, which is better than or equal to other work. However, the communication overhead of building a tree is. Unfortunately we cannot compare the communication overhead in construction of an aggregation tree of our protocol because the related work does not provide a communication overhead in building the aggregation tree or assumes that the aggregation tree already exists.

There is a limitation in the proposed protocol. We can currently pinpoint the attacker for all of the attack types we analyzed, except the framing attack (Section 6 Case a), which leaves the possibility of a attack that can drain the battery of the victim's children. We plan to study this issue in our future work.

References

[1] A. Boonsongsrikul and et al., "Securing Data Aggregation against False Data Injection attacks in Wireless Sensor Networks," ICACT 2010, pp.29-34, 2010.

[2] H. Chan and A. Perrig, "PIKE: Peer Intermediaries for Key Establishment in Sensor Network," IEEE INFOCOM 2005, pp.524-535, 2005.

[3] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," Proc. of IEEE Symposium on Security and Privacy, May, 2003.

[4] H. Chan, V. Gligor, A. Perrig, and G. Muralidharan, "On the distribution and revocation of cryptographic keys in sensor networks," IEEE Transactions on Dependable and Secure Computing, Vol.2, No.3, pp.233-247, July-Sept., 2005.

[5] Da Silva and et al., "Decentralized Intrusion Detection in Wireless Sensor Networks," Q2SWinet '05, pp.16-23, 2005.

[6] W. R. Heinzelman and et al., "Energy efficient communication protocol for wireless microsensor networks," Proc. of the 33rd Hawaii Int. Conf. on System Sciences, pp.3005-3014, 2000.

[7] S. Madden and et al., "TAG: aTiny AGgregation service for ad-hoc sensor networks," OSDI'02, 2002, pp.1-16, 2002.

[8] E. Mlaih and et al., "Secure Hop-by-Hop Aggregation of End-to-End Concealed Data in Wireless Sensor Networks," IEEE INFOCOM, pp.1-6, 2008.

[9] A. Perrig, R. and et al., "SPINS: security protocols for sensor networks," ACM SIGMOBILE, pp.189-199, 2001.

[10] B. Przydatek and et al., "SIA: Secure information Aggregation in Sensor Networks," SenSys' 03, pp.255-265, 2003.

[11] J. R. Taylor, An Introduction to Error Analysis. 2nd edition. Sausolito, California: University Science Books. 1997.

[12] The Network Simulator－ns-2 [Online], Available: http://www.isi.edu/nsnam/ns/

[13] H. Vu and et al., "THIS: THreshold security for Information aggregation in Sensor networks," ITNG'07, pp.89-95, 2007.

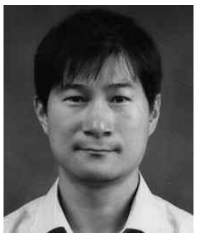
[14] Y. Yang and et al, "SDAP: A Secure Hop-by-Hop Data Aggregation Protocol for Sensor Networks," MobiHoc'06, pp. 356-367, 2006.

[15] S. Zou and et al., "ENCAST: EnergyCritical Node Aware Spanning Tree for Sensor Networks," CNSR, pp.249-254, 2005.

[16] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in CCS '02: Proc. of the 9th ACM conference on Computer and Communications Security, pp.41-47, 2002.

Boonsongsrikul, Anuparp

e-mail : anuparp@ajou.ac.kr
1998년 Mahanakorn 기술대학교 통신공학과(공학사)
2002년 Kasetsart 전자공학과(석사)
2007년～현 재 아주대학교 정보통신공학과 박사과정
관심분야 : 센서 네트워크, 에드혹 네트워크, VANET, IC 디자인 등



이 경 석

e-mail : kyungsuk.lhee@gmail.com
1991년 고려대학교 서어서문학과(학사)
1993년 그리피스대학교 business computing학과(석사)
1996년 보스톤대학교 컴퓨터공학과(석사)
2005년 시라큐스대학교 컴퓨터공학과(박사)
2005년～2010년 아주대학교 정보컴퓨터공학부 조교수
관심분야 : 컴퓨터보안, 네트워크보안 등

박 승 규

e-mail : sparky@ajou.ac.kr
1974년 서울대학교 응용수학과(공학사)
1976년 한국과학원(KAIST) 전산학과 (석사)
1982년 Institut National Polytechnique de Grenoble 전산학과(박사)
1976년~1977년 한국과학기술연구소(KIST) 연구원
1977년~1978년 KIET (현ETRI) 연구원
1978년~1982년 프랑스 그레노블 IMAG 연구원/학생
1982년~1984년 KIET (현ETRI) 실장/선임연구원
1984년~1985년 미국 IBM 왓슨연구소 연구원
1985년~1992년 ETRI 연구위원/책임연구원
1992년~현 재 아주대학교 정보통신대학 교수
관심분야 : 임베디드 테스팅, 자가 컴퓨팅/치료 시스템, 차세대 컴퓨터 구조 등