Journal of Network and Computer Applications 33 (2010) 50–62

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

An authentication scheme for locating compromised sensor nodes in WSNs

Youtao Zhang [a,*], Jun Yang [b], Weijia Li [a], Linzhang Wang [c], Lingling Jin [d]

[a] Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260, USA
[b] Electrical and Computer Engineering Department, University of Pittsburgh, Pittsburgh, PA 15261, USA
[c] Computer Science Department, Nanjing University, Nanjing 210000, China
[d] NVIDIA Inc., Santa Clara, CA 95050, USA

ARTICLE INFO

Article history:
Received 29 April 2008
Received in revised form
28 January 2009
Accepted 17 June 2009

Keywords:
Sensor network
Security
Authentication
Incremental hashing
Selective report forwarding
False report injection

ABSTRACT

Wireless sensor networks have recently emerged as a promising computing model for many civilian and military applications. Sensor nodes in such a network are subject to varying forms of attacks since they are left unattended after deployment. Compromised nodes can, for example, tamper with legitimate reports or inject false reports in order to either distract the user from reaching the right decision or deplete the precious energy of relay nodes. Most of the current designs take the en-network detection approach: misbehaved nodes are detected by their neighboring *watchdog* nodes; false reports are detected and dropped by trusted en-route relay nodes, etc. However en-network designs are insufficient to defend collaborative attacks when many compromised nodes collude with each other in the network.

In this paper we propose COOL, a COmpromised nOde Locator for detecting and *locating* compromised nodes once they misbehave in the network. It is based on the observation that for a well-behaved sensor node, the set of outgoing messages should be equal to the set of incoming and locally generated or dropped messages. However, comparing the message sets for different nodes is not enough to identify attacks as their sanity is unknown. We exploit a proven collision-resilient hashing scheme, termed *incremental hashing*, to sign the incoming, outgoing and locally generated/dropped message sets. The hash values are then sent to the sink for trusted comparisons. We discuss how to securely collect these hash values and then confidently locate compromised nodes. The scheme can also be combined with existing en-route false report filtering schemes to achieve both early false report dropping and accurate compromised nodes isolation. Through identifying and excluding compromised nodes, the COOL protocol prevents further damages from these nodes and forms a reliable and energy-conserving sensor network.

1. Introduction

The sensor network has recently emerged as a promising computing model for many civilian and military applications such as patient status monitoring in a hospital, and target tracking in a battlefield. It usually consists of a large number of low-cost, battery-powered sensor nodes that are of limited computation and communication capability. While these nodes are left unattended after deployment, they can adaptively form a routing graph and continuously collect data for events of interests and deliver the data to a designated sink node, a node that is usually resource-abundant and trustworthy.

The unattended nature of a sensor network makes it vulnerable to varying forms of security attacks such as a compromised node injecting false data reports (Karlof and Wagner, 2003; Ye et al., 2004; Zhu et al., 2004). Without identifying false reports, the sink node may reach a sub-optimal or even wrong decision. In addition, routing false reports to the sink wastes the energy of nodes along the routing path, which reduces the lifetime of the network. Identifying compromised nodes is as critical since these nodes can exhaust their upstream nodes even if the false reports are dropped en-route in just a few hops (Ye et al., 2004; Zhu et al., 2004).

Schemes have been proposed to locate misbehaved nodes with en-network detection approaches. Marti et al. proposed to monitor each node by a neighboring *watchdog* node. Wang et al. (2003) improves the scheme through the collaborative decision of neighbors around a suspicious node. Both schemes have limitations (Marti et al., 2000) as the watchdog node may be compromised as well. Thus compromised nodes may not faithfully isolated.

A less ambitious approach is to filter injected false reports en-route, each data report is attached with several MACs (message authentication code) generated from different keys

* Corresponding author. Tel.: +1 412 624 8837.
E-mail address: zhangyt@cs.pitt.edu (Y. Zhang).

1084-8045/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2009.06.003

(Zhu et al., 2004; Ye et al., 2004; Yang et al., 2005). A careful key sharing mechanism can ensure that an en-route node can verify if a received report has been tampered with when an attached MAC is generated from one of the stored authentication key. The key assignment to the nodes can be distributed probabilistically (Ye et al., 2004), setup before routing (Zhu et al., 2004), combined with location information (Yang et al., 2005), or refreshed periodically (Zhang and Cao, 2005). Dropping false reports early en-route saves the routing energy and prolongs the lifetime of the network. However, those schemes also have limitations. First, the compromised nodes are difficult to locate. Even if we know which key has been compromised, we cannot isolate the compromised node that exposes this key since multiple nodes may have the key. Second, compromised nodes continuously force their upstream nodes to waste energy in authentication and routing before the packets are dropped, depriving the energy resource in the long run.

In this paper we propose COOL, a COmpromised nOde Locator for locating malicious nodes if they send out false data reports or drop real reports. Our design is based on an intuitive observation—for any well-behaved node in the sensor network, the set of outgoing messages should be equal to the set of incoming and locally generated or dropped messages.[1] We incrementally extend the testing sets to other nodes so as to capture an inconsistency when a bad link is included. A bad link is a hop between two nodes in which at least one is compromised. For such links, we drop both nodes achieving an upper bound of $2m$ excluded nodes if there are m malicious ones.

However, directly comparing the message sets cannot locate malicious nodes as the sanity of the messages themselves is unknown. We exploit a provably secure incremental hash authentication scheme—AdHASH (Bellare and Micciancio, 1997)—to overcome the difficulty. Two AdHASH hash values are the same if they are computed from the same set of items, irrespective of the item order. It is computationally infeasible to forge another set of items that yields the same hash value. With such a scheme, all sensor nodes compute the AdHASH values for outgoing, incoming and dropped messages. The AdHASH for locally generated reports is computed by the sink for the reason we will explain. We then show how to securely collect the AdHASH values and confidently locate compromised nodes. When the error rate of the network is high, our approach can also be combined with the en-route filtering scheme to achieve early dropping of false reports as well as isolation of the compromised nodes. By identifying and excluding malicious nodes, our scheme prevents future damages from these nodes, resulting in a reliable network which conserves the energy and prolongs the network lifetime through fewer false report forwarding and less authentication in the long run. We simulated and evaluated the proposed schemes. Our results show that the COOL protocol and its optimizations can effectively locate compromised nodes with very modest computation and communication overhead introduced to the network.

The remainder of the paper is organized as follows. We describe the problem, and the network and attack models in Section 2. The COOL protocol is then presented in Section 3 with three optimizations presented in Section 4. We evaluate proposed schemes and show the results in Section 5. Section 6 discusses the related work. Section 7 concludes the paper.

[1] Some messages may be lost due to weak connection in the sensor network. It is also considered as one type of fault. We detect such links and let the sink decide if the involved nodes should be excluded.

2. Problem statement

2.1. The network model

In this paper, we consider a sensor network that consists of a number of battery-powered sensor nodes and a sink node with abundant resources, e.g., energy and computation power. Each sensor is assigned with a unique ID and a secret key before deployment. Both the ID and the key are known to the sink node. Sensor nodes are left unattended after deployment. They monitor events of interests and send the data reports to the sink. When an event happens in the network, it will be detected by multiple sensing nodes. We assume that majority of these sensing nodes for any single event are trustworthy.

We use a cluster-basedmultihop routing scheme since recent research showed that clustering techniques are energy efficient for the sensor network (Heinzelman et al., 2002; Younis and Fahmy, 2004; Kawadia and Kumar, 2003). Sensors in a small range are grouped as a cluster with one cluster head (CH) elected for one epoch. Sensing readings (including the time when the event happens (Ye et al., 2004)) are first sent to the CH which aggregates them to a report to be sent to the sink. By taking the majority of the readings, the CH includes the selected sensing node IDs and their MACs (discussed next) in the content of the aggregated report. The CH also appends its own ID and MAC to the report. After generating the report, the CH forwards it along the routing path to the sink. Messages from the sink are first sent to CH and then broadcasted within the cluster.

At the cluster head level, the routing graph is built using the directed diffusion routing protocol (Intanagonwiwat et al., 1999). The sink reinforces paths for monitoring different *interests* in the field (Fig. 1). As shown in Intanagonwiwat et al. (1999) reports are tagged with interest types while there is an interest cache on each node which saves these interests if that node is on the corresponding paths. In our algorithm reports are forwarded according to the routing path in one epoch. Each node, e.g., s5 checks the received report and drops it if it is not for a cached interest (e.g., from s4). If multiple source nodes, e.g., s1 and s4 report the same interest, some relay nodes, e.g., s2 can distinguish its downstream source IDs and drops the report if it is from s4.

In our basic scheme, each sensing node has a different key which is used to generate a MAC for each of its outgoing sensing readings. We assume that the reading is left in plain text. If this is a concern, it may be encrypted to disable any eavesdropping. The trade-off is, since the secret key is only known to the sink, a CH node may not be able to aggregate the sensing readings in its cluster and therefore has to forward all encrypted readings to the sink, creating more traffic to the network.

2.2. The attack model

We assume that once a sensor node is compromised, the adversary can retrieve all embedded security information including the

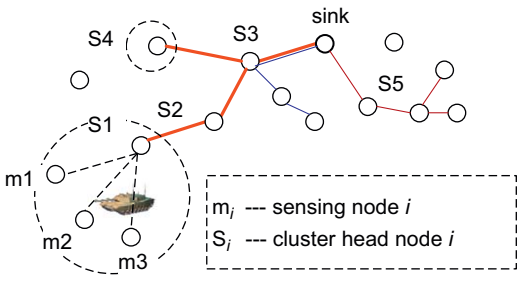

Fig. 1. The sensor network.

secret key. Therefore, a compromised node may inject false data reports as shown in Zhu et al. (2004) and Ye et al. (2004), or may selectively forward real data reports received from its downstream nodes. We further assume the adversary knows the COOL protocol or other security algorithms used in the network. He can modify the status maintained on compromised nodes and send the data that the sink may wish to see.

Either the injection attack or the dropping attack may occur at a sensing node; at a source CH node; or at a relay CH node. In this paper, we address all these types but with an emphasis on the injection attacks and the dropping attack at sensing and relay *CH* nodes. The dropping attack at a source CH node is more difficult to defend since a compromised CH may refuse to form a report even after receiving several sensing readings. A CH can legally drop some readings since it is usually given with the power to construct the report from the majority of the readings (to shield some random error readings). If it is really a concern, then each sensing reading should be sent to more than one source CH nodes resulting in increased routing overhead as we will discuss in the algorithm.

A compromised node is located if and only if its node id is known to the sink who can then securely notify other sensors (using broadcast authentication (Perrig et al., 2001)). Without being located, the compromised node can be elected as a CH node and continuously inject or drop reports. After being located, the network is free from its injection or dropping attacks since others know it is excluded. Of course additional mechanism might be needed to prevent it from malicious signal collision or changing its id.

Reports may be lost due to some weak connections in the network. This is one type of faults that should also be identified. Identifying a weak connection is beneficial since it gives the accurate location where a problem occurs. Based on the frequency of a faulty link, the sink can always make the decision whether or not to exclude the involved nodes. Since it is straightforward to detect/eliminate such links, we will focus on the report loss due to security attacks in the rest of the paper.

2.3. The design objectives

For a sensor network with above settings and model, our design goal is to effectively identify those compromised nodes and then exclude them from the network. The proposed algorithm meets the following requirements:

- the sink has the ability to discriminate the false reports;
- the scheme can defend both true report dropping at the relay nodes and all types of injection attacks;
- the algorithm can locate compromised nodes;
- the algorithm is effective with small overhead introduced to existing clustering and routing algorithms.

3. The COOL protocol

In this section, we present the basic design of the COOL protocol. We first discuss the incremental hash function, and then describe the high-level idea of malicious node detection using a simple example. The details of the systematic protocol operations are then discussed, followed by security analyzes of the protocol.

3.1. The incremental hash function

Fig. 2 illustrates the concept of the incremental hash (Bellare and Micciancio, 1997). It computes a cryptographic hash value for a finite set of elements. Each element is first concatenated with a unique *id* and then hashed by a standard cryptographic hash function, e.g., MD5 or SHA (Schneier, 1996). Those intermediate hash values are then combined by a combining operator to get the incremental hash value.

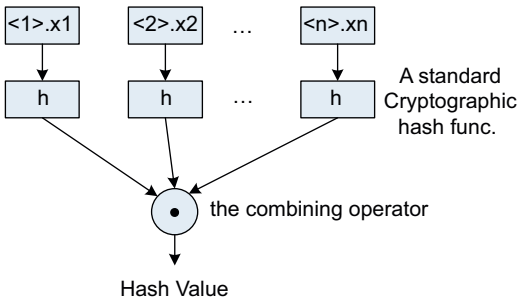

Fig. 2. An incremental hash function.

In this paper, we use the AdHASH introduced by Bellare and Micciancio (1997) (abbreviated as AH(…) in the rest of the paper)

$$AH_M^h(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} h(\langle i \rangle \cdot x_i) \bmod M$$

where h is a standard cryptographic hash function and M is a very large integer value of k bits. The $\langle i \rangle$ is an *id* assigned to each message such that the concatenation of them is unique in the entire set. When we apply the AdHash in our sensor network, each report is assigned with the sensor's ID and a local report sequence number. Therefore, each report received by the forwarding cluster head is unique. As we can see, the AH computed by the cluster head is independent of the order at which reports are received. This incremental hash function has the following properties that are useful in our design.

- *Compression*. It compresses inputs of larger size into k bits such that each incremental hash value can be stored using small number of bits in each node.
- *Incrementality*. The AH of a larger set can be computed incrementally from the AH of its subset. In particular, when a new item is inserted to the set, the new AH can be computed from the old value and the $h()$ value of the new item. That is,

$$AH_M^h(x_1, \ldots, x_{n+1}) = (AH_M^h(x_1, \ldots, x_n) + h(\langle n+1 \rangle \cdot x_{n+1})) \bmod M$$

- *Efficiency*. The computation of an AH hash value just needs several additions and one modulation in addition to the standard hashing. Particularly, for the insertion of a new item, the computation overhead is one addition and one modulation only (the width of the h and AH is of the same order). This is important as most hash values are to be maintained by resource constrained relay nodes in our design.
- *Proven collision-resilience*. It is computationally infeasible to find (or forge) another set of items that can result in a same hash value. The hardness has been proven at the same or higher level of the weighted knapsack problem (Bellare and Micciancio, 1997), a generalized NP-hard problem. This gives us a solid security ground for designing security enhancement schemes for sensor networks.

We selected h to be MD5 (Schneier, 1996) and k to be 128 in the design, i.e., M is a 128-bit integer value. Our scheme is different from secret key based authentication schemes. In our scheme, M is known to all sensors and the base station. The security and correctness of our scheme are ensured by the properties of AdHASH, as we will illustrate in Section 3.4.

The choice of parameters h and k, while being sufficient for most applications, is independent of the COOL protocol design. For example, if MD5, SHA cannot match the requested security (Wang et al., 2004, 2005), they can be substituted by stronger or even future hashing functions. To ensure higher level security, the number of reports should be greater than k (Bellare and Micciancio, 1997), that is, we should accumulate more than 128 reports before authentication. This is generally not a restriction once the network has been warmed up (meaning that at least 128 reports have been generated).

3.2. The basic design—a simple example

We next show how an incremental hash function can be applied to authenticate messages and in particular how to locate malicious nodes in a sensor network.

The design is based on an intuitive observation, i.e., the set of outgoing (forwarded) messages of a well-behaved node in the sensor network should equal the set of the incoming (received) and locally generated/dropped messages. Unfortunately, these message sets are maintained on different sensor nodes across the network making it impractical to pass them around and compare. Luckily with the incremental hash function, we only need to compare the hash values of different sets while keeping sufficient confidence to claim that *their hash values match indicates that the message sets also match*, i.e.,

$$\begin{gathered}\text{No node being compromised}\\ \Leftrightarrow\\ \{\text{msg}_{out}\}=\{\text{msg}_{in}\}\cup\{\text{msg}_{local}\}\\ \Leftrightarrow(\text{with sufficient confidence in the harsh strength})\\ AH(\{\text{msg}_{out}\})=(AH(\{\text{msg}_{in}\})+AH(\{\text{msg}_{local}\}))\bmod M\end{gathered}$$

To see how this principle is applied in our sensor network, let us look at a simple example as illustrated in Fig. 3. Here we show four sensors (s1–s4) and one sink node (s0). Assume that s1–s4 are cluster heads already. The messages are labeled in letter a, b, etc. Suppose node s2 is compromised and it injects a false message X and pretends that X is sent by s4 (in this section, we will also discuss what if X is forged as if it is sent by s2). All messages are forwarded to the sink, but the AH values are calculated and kept locally. Specifically, s4/s3 calculates outgoing AHs for $(a, b)/(d, e)$; s1 calculates two incoming AHs for (d, e) and (a, b, X), respectively, and one outgoing AH for (a, b, c, d, e, X); s2, as a compromised node, can fake the incoming or outgoing AHs for either (a, b) or (a, b, X). Note that s2 will not produce another incoming AH for X as s2 tries to hide itself from being detected immediately (X is "originated" from s4). As we will elaborate later that the AHs for locally generated legitimate messages are computed at the sink.

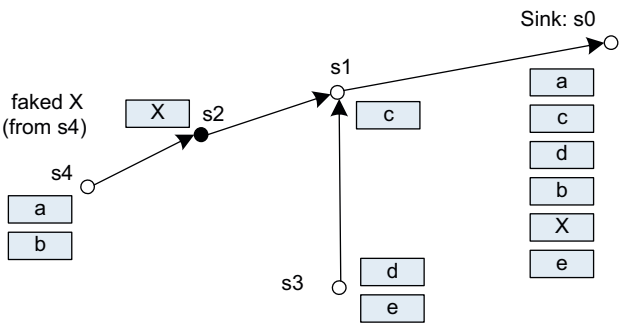

Fig. 3. Locate compromised nodes using incremental hashing.

The sink receives all messages including X. It can immediately identify that X is false since s2 does not have the secret key of s4 and could not generate the right MAC for X. Next, the sink tries to locate the sender of X, i.e., the node that has been compromised. At this time, the sink collects all the AH values from the sensor nodes. We can assume that they all arrive correctly as simple endorsement using secret keys can assure this, and there are fixed number of them for a given routing so that no AHs are dropped. The sink then starts to check the "node consistency", i.e., if

$$AH(incoming\ \cup\ generated)=AH(outgoing\ \cup\ dropped) \tag{1}$$

holds for every node. Note that due to the additivity of the AH function, $AH(incoming \cup generated) = (AH(incoming) + AH(generated)) \bmod M$ (and the same for the right-hand side, RHS). It is easy to see those conditions for s1, s3 and s4 satisfy. For s2, as we mentioned, there are two options—$AH(a, b)$ or $AH(a, b, X)$—for both the incoming and outgoing AH values, forming four possibilities. If the s2 chose the different values for incoming and outgoing AHs, it would immediately be identified as compromised as equality (1) would not satisfy. Let us assume s2 is intelligent enough not to expose itself too easily, and thus chose a consistent incoming and outgoing AH pair. Thus, it will pass at least the "node consistency" check.

Next the sink starts a "link consistency" check in which the AH of the outgoing message on a link should equal the AH of the upstream incoming message. That is, what has been sent out should be what has been received on each link. This can be easily checked for links without the node s2. Let us assume that the s2's incoming and outgoing hashes are both $AH(a, b, X)$. Then, as an outgoing AH, $AH(a, b, X)$ is consistent with one of the incoming AHs for s1. However, as an incoming AH, $AH(a, b, X)$ is inconsistent with the outgoing AH for s4 which is $AH(a, b)$. In other words, s2 chose that value to lie that s4 had given it a false incoming message. The sink now cannot distinguish who is the real compromised node, but can at least conclude that one of them is flawed. A new routing graph will be generated excluding both s4 and s2 after detecting the link inconsistency.

Notice that node s2 can destroy s1 in the same way by producing $AH(a, b)$ for link consistency checking, or even destroy all the adjacent nodes by forging an arbitrary AH making none of the links consistent. It would be unnecessarily conservative if all nodes on the inconsistent links in such a scenario are eliminated from the routing graph since the faults are due to only one evil axial node. Instead, our protocol removes one link at a time, removing the axial node in the first place and saving the other nodes being circumvented.

Let us discuss what if X is forged as if it is sent by s2. The sink can still identify X as a false report since each legal one should have multiple sensing node MACs—s2 cannot construct these sensing node MACs as it does not have these secret keys. Old readings cannot be replayed since a timestamp is included in the reading. Notice that the sink computes local AH values only from legitimate reports, that is, it excludes X from generating the local AH value for s2. Hence, no matter which incoming or outgoing AH values s2 chooses (either $AH(a, b)$ or $AH(a, b, X)$), there bound to be a node or link inconsistency.

Before presenting the COOL protocol, we summarize the benefits from excluding compromised nodes from the network.

- *Communication energy savings*. Once the compromised nodes have been excluded, no false reports can be injected into the network. As a result, the energy drained by forwarding false reports can be saved. This is different from en-route filtering schemes in which false reports are still forwarded several hops (Ye et al., 2004; Zhu et al., 2004) before being detected and dropped.

- *Computation energy savings*. In the basic COOL scheme, we do not perform en-route packet authentication but rather only update incremental hash values. We can afford less frequent authentication by excluding compromised nodes and form a network with trustworthy nodes.

3.3. The COOL protocol

The goal of the COOL protocol is straightforward: we securely collect AH hash values from the network and send them to the sink; we drop the identified node if the node inconsistency is found, and drop both nodes if an inconsistent link is found. The detailed protocol is slightly more complicated than the example above and contains the following phases:

(1) In the initialization phase, we assign a unique ID and a secret symmetric key to each sensor node. The sensor nodes are deployed thereafter.
(2) In the routing graph discovery phase, we broadcast *hello* messages along the downstream routing path and collect the current routing graph from replies.
(3) In the report forwarding phase, we endorse each report by the secret key of the sender and have it forward along the routing path. Each node maintains the AH hash values for each of its incoming and outgoing links.
(4) In the hash value collection phase, the sink sends out a request to collect hash values from the path where the false message belongs to.
(5) In the compromised node detection phase, the sink finds inconsistent nodes and links using the incremental hash function AH.
(6) In the routing graph fix phase, the sink replaces the excluded cluster heads with newly selected ones.

Next, we elaborate each of the phases with more details.

Phase 1: *Initialization*. Each sensor is assigned a unique integer ID and a secret symmetric key before being deployed into the field. Both the ID and the key are known to the sink. This is similar to Zhu et al. (2004) but we do not request any key sharing among sensor nodes. The node ID occupies 2 bytes (16 bits) which can distinguish 64K sensors in a network. Larger networks can adaptively adjust this bit width to accommodate their needs. The key is used to generate a MAC by a sensing node for the sensing reading sent to the source CH node, and by a source CH node for the report sent to the sink. By checking the MACs using the secret keys, the sink can detect tampered reports or injected false reports from compromised source CH or relay CH nodes.

Phase 2: *Routing graph discovery*. The discovery starts at the beginning of an epoch, e.g., cluster heads are reselected and a new routing graph is constructed according to Intanagonwiwat et al. (1999). The sink forms an entire routing graph through collecting information from distributed cluster heads. It sends out a timestamped "hello" message to all its adjacent cluster heads who then forward this message downstream until it reaches the leaf nodes. All the nodes then respond with their node IDs as well as their adjacent node IDs. For those messages, a MAC using the local secret key is also attached to ensure their integrity.

Each node only collect replies from its downstream nodes. To prevent malicious report dropping, the reply is collected in order—each node first collects the replies from all its children nodes and then sends out its own reply. The sink finally assembles the complete routing graph from all replies.

To ensure "hello" messages are not abused, a broadcast authentication (Perrig et al., 2001) is applied. In addition, a selected cluster head may try to find a different routing path if it does not receive a "hello" message within a certain time interval after its election to avoid being isolated from the network.

Phase 3: *Report endorsement and forwarding*. Fig. 4 illustrates the report generation, endorsement and forwarding in the network. We also list the related authentication actions and discuss why injected false reports and dropped legitimate reports can be detected.

To check Eq. (1), each node maintains its incoming AHs and outgoing AHs. The local AH (generated from locally generated reports) is computed at the sink rather than the individual node. This is because false reports should not be used to compute the local AH (otherwise there is no inconsistency) and the sink knows what those false reports are. The sink generates a local AH for each node using only the legitimate reports whose source IDs are that node, discarding all false reports discriminated. *For example*, a node m may forge a false report using its own ID as the source ID. This false report can be identified by the sink since it does not have enough legal sensing node MACs. After identifying X as a false report, the sink will exclude it from updating the local AH of m, which creates a node inconsistency if node m updated the false report into its outgoing AH, or a link inconsistency otherwise.

The drop AH (generated from locally dropped reports) is not used in the basic scheme, i.e., we assume a non-compromised node does not drop reports intentionally. A report is dropped only by compromised nodes who always try to conceal themselves as much as possible. Consequently, the AHs for dropped reports are never created.

Either false report injection or legitimate report dropping creates AH inconsistency for some nodes or links. The difficulty is how to expose the inconsistency. The technique presented in phase 4 handles this problem.

Phase 4: *Hash value collection*. Hash value collection is triggered by any of the following two conditions: (i) the sink has detected one or multiple false reports; (ii) a preset timer has elapsed. The former is to detect injected false report attack while the latter is to detect report dropping attack. In the first case, AH values are collected only from the path where the erroneous report belongs to. Such a path can be identified correctly as we explained earlier that a spoofed report can reach the sink only if it is generated from a downstream node. Collecting the hash values along a path greatly reduces the number of messages introduced to the network. In the second case, however, all the hash values in the network are queried as the sink has no clue of where reports could be dropped. Next we show how to collect AHs from the erroneous path. It is trivial to extend the scheme to all nodes.

To collect the hash values, the sink sends out an inquiry message onto the erroneous path. For example, in Fig. 3, to detect the compromised node on path s4–s2–s1–s0, we only collect hash values on this path but not from the link s1–s3 (but in phase 5 the sink still needs to compute the AH values for these reports injecting to the path from s3). We must be very careful in this collection process as the forwarded AHs may be altered by compromised nodes as well. Thus, we treat all the AH values as *normal data reports* and send them upstream starting from the leaf cluster nodes. This will resulting in the normal updates for the AH values at every node on the path as well. The only constraint is: the outgoing AH on a link does not update the incoming AH on the same link since this would result in link inconsistency and make the hash collection process and later checking too convoluted. As a result, the node consistency checking would be adjusted to accommodate this exception.

Our hash values collection phase effectively removes the slander problem (Gu et al., 2006) that many authentication schemes have. A slander problem happens when a compromised sensor maliciously modifies some hash values such that the detection mechanism slanders well-behaved sensors instead of malicious ones. Our scheme is free from slander problem since hash values are collected in bottom–up order, hash values from

	Sensing node	Cluster head (CH) node	
		Data aggregation	Report relay
Report generation and endorsement	A stimulus in the network is detected by at least M surrounding sensing nodes. Each of them e.g. m sends the sensing reading and the MAC (generated using m's secret key) to C CH nodes.	By taking the majority of received readings, the source CH constructs a report containing the sensing value and received MACs. It also appends its own node ID and a unique sequence number. A unique MAC is generated for the report using its own secret key. Both the report and the MAC are forwarded along a multi-hop route to the sink.	Since the keys to attached MACs are only known to the sink, relay nodes do not perform any en-route checks in the basic scheme. A relay node receives the report from one downstream link and forwards it along one upstream link.
Authentication	An AH value is maintained for the outgoing link of m. The value is updated with the sensing reading and the MAC each time when m sends them out.	A source CH node maintains one outgoing AH value and in the case it is also a relay node, it maintains several AH values with one for each of its incoming and outgoing links respectively. After sending the report generated by itself, it updates the outgoing AH value with the report and the MAC.	A relay CH node maintains a different AH value for each of its incoming and outgoing links respectively. For the forwarded report, it updates two AH values — the incoming AH value for the link from which the report is received and the outgoing AH value for the outgoing link.
Detecting injection attack	A source CH node should receive readings from at least M sensing nodes. If some ($< M/2$) are compromised and send back false readings, these readings will not affect the report generation at the CH node.	A compromised source CH node may forge false reports. It cannot accumulate $M/2$ legal sensing node MACs for an arbitrary false reading. A report with such a value can be detected at the sink. Old readings and MACs cannot be replayed since the sensing reading has a timestamp indicating when the event happens [20]. Different sensing node MACs are expected even for the same sensing reading but at a different time.	If a relay node forges a report with the source node as itself, the false report can be detected as the data aggregation case. If a relay node forges a report with the source node id as one of its downstream nodes, it does not have the secret key of the faked sender to generate a matching MAC. The report will be detected by the sink. The compromised node is then located using our algorithm.
Detecting dropping attack	Dropping sensing readings at some compromised nodes ($< M/2$) does not affect the generation of the legitimate report at a source CH node.	[Discussion only: not the focus in the paper] We can elect more than one CH to perform data aggregation i.e. set C bigger than one. If some but not majority of them refuse to generate reports from received readings, the sink can still receive the legitimate report and thus detect packet dropping in the network. In the rest of the paper we assume that for each event one CH node is elected for data aggregation.	If a relay node drops some reports, the sink can check the sequence number from a source CH. A non-contiguous number from the previous report indicates report dropping along the path. If the compromised relay node chooses to drop all following reports from a cluster, the sink may not get any clue. Instead the sink periodically collects all AH values to detect the dropping attack and relies on the COOL protocol to defend them.

Fig. 4. Actions taken by different nodes.

lower levels are treated as data packets, and any change of these packets result in the inconsistency identified at that node. We will give formal derivation in Section 3.4.

Phase 5: *Identify compromised nodes*. The sink node performs two types of tests: the node consistency and the link consistency test. The first type is to test the matching of incoming and outgoing AH values for each node on the erroneous paths. The AHs are extracted from the returned hash value reports; the AHs for the incoming links not on erroneous paths, e.g., the s1–s3 path in our example, and for locally generated reports are calculated by the sink directly. If there is a mismatch, the node is tagged as a compromised node. For example, in Fig. 3, the sink tests s1 using

$$\left(\underbrace{AH_{2\to 1}}_{\text{collected}} + \underbrace{AH(c)+AH(d,e)}_{\text{calculated by the sink}}\right) \bmod M = ?\underbrace{AH_{1\to 0}}_{\text{collected}}$$

The second type is to test if the outgoing and incoming AHs are consistent on all links. Each hash value should match with the one reported by the other end of the link. If any inconsistency is found, the sink tags both nodes as problematic as it is now hard to flag one node as a bad node for 100% confidence. For example, we test

$$\left(\underbrace{AH_{1\to 0}}_{\text{collected}} = ? \quad \underbrace{AH_{0\leftarrow 1}}_{\text{computed at the sink}}\right) \text{ and } \left(\underbrace{AH_{1\to 2}}_{\text{collected}} = ?\underbrace{AH_{2\leftarrow 1}}_{\text{collected}}\right)$$

Phase 6: *Excluding compromised nodes and routing graph fix*. Once any nodes are tagged as suspicious, they should be excluded from the sensor network immediately. To do so, the sink broadcasts the IDs of the tagged nodes across the network, particularly to those nodes around the compromised ones. This can be done using broadcast authentication algorithms, e.g., μTESLA (Perrig et al., 2001). This packet also initiates the selection

of new cluster heads to replace the excluded ones, and then incorporates new heads into the routing graph. The newly joined nodes send back their IDs to the sink to check if they are allowed to join the network. The sink acknowledges back with the most up-to-date AH values for the new cluster heads.

3.4. The security analyzes

Earlier we have explained using an example in Fig. 3 to explain why attacks can be identified by the sink. In this section, we give systematic proofs for various security properties of our COOL protocol.

Theorem 1. *Any injection attacks can be detected by the COOL protocol.*

Proof. We first assume that there is only a single injection attack launched from a node S_i. Then, we generalize it to multiple injection attacks scattered in the network. In both cases, there could be one or more compromised nodes that can conclude to hide as much as possible the node(s) performing the injections. We will enumerate all possible cases and show how the COOL can detect them.

Single injection attack. Assume that a false report X is injected by a compromised node S_i as shown in the solid dot in scenario 1 of Fig. 5. X will eventually reach the sink. If X is tagged with S_i's ID as its source, then according to Ye et al. (2004), S_i as a cluster head has to forward certain number of received local reports (e.g., five) together with MACs (involving secret keys of the en-cluster sensors) to the sink. Since S_i does not have secret keys of other healthy sensors, X will be eventually detected by the sink as a forged report. Similarly, if X is tagged with some other node S_p's ID as its source, the sink can detect it as S_i does not have S_p's secret key. The sink then starts a hash collection phase as we described previously.

Let S_j and S_k be S_i's downstream and upstream node. Both are healthy nodes and shown as circles in the figure. S_j sends report(s) M to S_i which then forwards it with the injected X to S_k. This is exactly the scenario as we saw in Fig. 3 for s2. There are five possibilities for S_i to produce its incoming and outgoing AHs, as shown in the table in Fig. 5. Case (a) reveals S_i as a compromised node instantly as the node consistency check will fail. Cases (b)–(e) all manifest as link inconsistency with the nodes shown in the last column. When a link inconsistency occurs, the sink removes both nodes on the link.

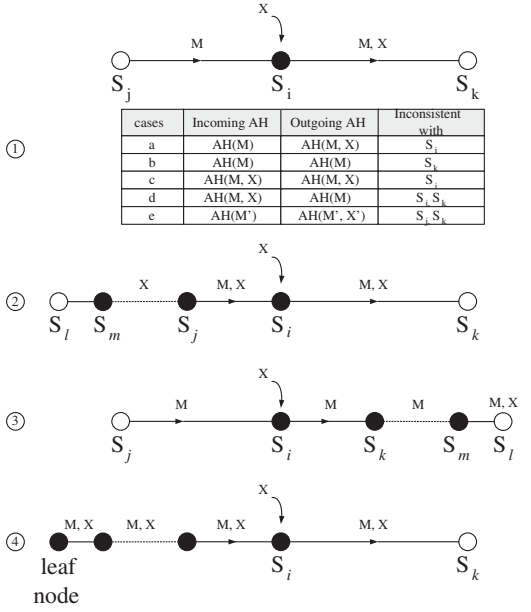

Fig. 5. Possible inconsistencies due to injected false reports.

However in cases (d) and (e), S_i generates some random AHs to mess up both upstream and downstream links. This can be generalized to all links touching S_i. Such an attack can result in many link inconsistencies and may force the sink to exclude too many nodes from the system. In COOL protocol, the sink always remove only one link in the above scenario, e.g., the S_i–S_k link, which assures that the evil node is excluded in the first place, saving the healthy nodes from being removed too conservatively.

Let us now assume that S_i's neighbor nodes are also compromised to collude with S_i and fool the sink, as illustrated in the second scenario in Fig. 5. Here we assume that S_j is compromised but there is at least one healthy node downstream in the path. Let S_l be the first such node. S_j's task is to collude with S_i such that the AHs satisfy both node and link consistency as much as possible. Hence, S_j would pretend that it had sent X to S_i, and such a conspiracy is carried along the path downstream until S_m, the upstream neighbor of S_l. S_m has to lie that it had sent out X as well. However, either the node inconsistency or link inconsistency will happen between these two nodes since S_l will not claim it has sent out X. The third scenario is symmetric with the second one with the only difference of faking upstream AHs without X.

Finally, if all nodes on the path downstream are compromised (the fourth scenario in Fig. 5). All AHs are forged and the inconsistency is pushed to the leaf cluster head. This situation is the same as X being tagged with source S_i, i.e., S_i is the leaf node that produced X. We have discussed this earlier. That is, since the sink will not include X in computing the local AH for S_i, there is a node inconsistency if S_i includes X in its outgoing AH; there is a link inconsistency otherwise.

In summary, we have enumerated all possible cases for different number of compromised nodes located at different places when there is a single injection attack in the network. As we can see that the COOL protocol can defend them successfully.

Multiple injections attack. With the proof for defending single injection attacks, multiple injections attacks can be derived naturally. If multiple injections are launched on different paths to the sink, each injection can be detected in the same way as single injection attacks.

If multiple injections are launched on the same path to the sink, from the aforementioned second and third analyzes, an inconsistency will definitely occur as long as there is at least one healthy node on this path. If, however, all nodes on the path are compromised except for the sink, it will first find out the compromised node from its immediate neighbors, and then replace it with a healthy cluster head, and then continue to detect the rest nodes step by step. In this situation, multiple hash collections are necessary in order to locate all of the compromised nodes.

To conclude, as long as there are injection attacks in the network, the COOL protocol will detect them successfully. □

Corollary 1. *The report dropping attack at the relay nodes can also be detected by the COOL protocol.*

Proof. We make the assumption that a healthy node does not drop reports intentionally in our base COOL protocol. Therefore, a report can only be dropped by a compromised node. Observe that dropping and injection are inverse functions to each other. When it comes to computing the AHs, dropped messages are subtracted from the incoming message set while injected messages are added to the outgoing message set. Therefore, the same principles used for Theorem 1 can be directly applied except that the compromised node will pretend it from dropping any messages but the sink cannot receive them. As we have described in our protocol phases, the sink periodically sends out hash collecting requests to check if there have been message drops.

We focus on the dropping attack at the relay node. If the messages are dropped at the leaf node (the cluster head who is supposed to generate the message), then the sink cannot detect it as it cannot obtain any information if the node does not report everything. This is because we do not extend further to the sensor nodes beyond the cluster head to keep the overhead of our protocol reasonable. □

Corollary 2. *When an inconsistent link is identified, at least one out of two involved nodes is a compromised node. If there are m compromised nodes, our scheme removes at most 2m nodes including those m compromised nodes.*

Proof. The first statement is obviously true. If there are m compromised nodes in the network, the number of node inconsistencies manifested is less than m, say n ($n<m$). And hence, n nodes are removed from the network. The number of link inconsistencies may go unbounded as one compromised node may create many link inconsistencies as we discussed in the proof of Theorem 1. However, remember that the protocol removes only one link at a time and reevaluate the hashes once nodes are removed and new cluster heads are elected. Hence, if there are $m-n$ nodes causing link inconsistencies, at most $m-n$ links are removed, or $2\times(m-n)$ nodes are removed from the network. In conclusion, the total number of nodes removed is $n+2\times(m-n)$ which is $\leq 2m$. □

In our COOL protocol, correctly detecting and locating of compromised nodes depends on the correct hash values collected from distributed sensor nodes across the network. Next, we will go through the detailed hash collection process and show how testing conditions are derived and satisfied.

Theorem 2. *Suppose two sensor nodes i and j are adjacent as shown below. i is a downstream node of j. Both are forwarding AH values to the sink for integrity checks. The incoming (outgoing) AH for i/j at time t is* $IN_i^t/IN_j^t(OUT_i^t/OUT_j^t)$.

If all the INs and OUTs are forwarded according to our protocol without being tampered with, the node consistency checking is equivalent to

$$OUT_j^{t+1} = (IN_j^{t+1} + h(\langle . \rangle . OUT_i^{t+1}) + h(\langle . \rangle . IN_j^{t+1})) \bmod M$$

where $\langle . \rangle$ *represents the index required by the function* $h()$. *The right-hand sides is the incremental AH computation (refer to Section 3.1).*

Proof. In the hash value collection phase, the sink requests all the AH values from the nodes on a erroneous path. However, when the AH values are sent on the path to the sink, they themselves may be tampered with by compromised nodes. Therefore, we developed a scheme to treat those AHs as data reports and update them into new AHs selectively. When the sink receives all the new AH values, it can still check for node and link consistency with slightly different conditions and the confidence that the AH values have not been tampered with along the path.

According to our collection policy, the outgoing AH does not update the incoming AH on the same link since it would be too convoluted. Otherwise, the AH values update upstream AH values the same way as normal data reports. Put it more formally using the example in Fig. 6:

$$\begin{cases} IN_j^{t+1} = (IN_j^t + h(\langle . \rangle \cdot IN_i^{t+1})) \bmod M & \text{if node i is a relay node} \\ IN_j^{t+1} = IN_j^t & \text{if node i is a leaf node} \\ OUT_j^{t+1} = (OUT_j^t + h(\langle . \rangle \cdot OUT_i^{t+1}) + h(\langle . \rangle \cdot IN_j^{t+1}) + h(\langle . \rangle \cdot IN_i^{t+1})) \bmod M \end{cases}$$

where superscripts $t+1$ represent updated version of AH values. The AH updates always happen from the leaf nodes to the sink, i.e., from i to j in the figure. To see the correctness of the above equations, we observe that there is only one link in Fig. 6. Thus, OUT_i^{t+1} does not update IN_j^{t+1}, so it is not in the RHS of the first equation. Instead, IN_j^{t+1} is updated by IN_i^{t+1} only. The OUT_j^{t+1} is updated by all the downstream AHs as expected. However, it will not update its upstream *IN* AH which is not shown in the figure.

The sink will receive all *IN* and *OUT* values tagged with $t+1$. It will first test the link consistency since the *OUT* value of any link does not update its *IN* value, and therefore, simple comparisons between them suffices. For node consistency checking, we observe the last equation listed above and obtain

$$\begin{aligned} IN_j^t &= OUT_j^t \Leftrightarrow OUT_j^{t+1} \\ &= (IN_j^t + h(\langle . \rangle . IN_i^{t+1}) + h(\langle . \rangle . OUT_i^{t+1}) \\ &\quad + h(\langle . \rangle . IN_j^{t+1})) \bmod M \Leftrightarrow OUT_j^{t+1} \\ &= ((IN_j^t + h(\langle . \rangle . IN_i^{t+1})) \bmod M \\ &\quad + (h(\langle . \rangle . OUT_i^{t+1}) + h(\langle . \rangle . IN_j^{t+1})) \bmod M) \bmod M \Leftrightarrow OUT_j^{t+1} \\ &= (IN_j^{t+1} + (h(\langle . \rangle . OUT_i^{t+1}) \\ &\quad + h(\langle . \rangle . IN_j^{t+1})) \bmod M) \bmod M \Leftrightarrow OUT_j^{t+1} \\ &= (IN_j^{t+1} + h(\langle . \rangle . OUT_i^{t+1}) + h(\langle . \rangle . IN_j^{t+1})) \bmod M \end{aligned}$$

The above derivation assumed that the corresponding terms are concatenated with the same index in both the relay nodes and the sink which can be easily achieved by assigning each AH a static number beforehand. As we can see, any slight manipulation to the AH values will break the above derivation. Therefore, the conclusion holds. □

4. Optimizations

While the COOL protocol can successfully detect compromised nodes, it suffers from three problems: (1) it could be too late to detect injected false reports at the sink as they have already consumed the routing energy; (2) the hash value collection overhead could be high; (3) up to half of excluded nodes could be actually good sensors. In this section, we develop optimization approaches that address these problems, respectively.

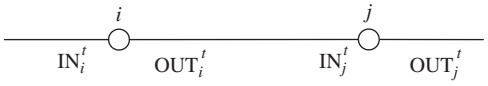

Fig. 6. Node i forwards its IN_i and OUT_i to j.

4.1. Drop-COOL: combining en-route filtering schemes

In the basic COOL protocol false reports are forwarded all the way to the sink. While the sink can detect these false reports, it is just too late since the energy has already been consumed along the routing paths. It may become even worse if a lot of false reports are injected before the COOL protocol is activated to collect AH values to exclude compromised nodes (Fig. 7). We, therefore, propose Drop-COOL, a hybrid scheme that integrates an en-routing filtering scheme (Ye et al., 2004; Zhu et al., 2004).

The Drop-COOL scheme works as follows. The system initializes according to both the basic COOL and the SEF (Ye et al., 2004) protocols. In addition, each node is assigned an integer threshold which is the maximal number of false reports that the node can forward in one round. In the packet forwarding phase, detected false reports up to this threshold are forwarded while following ones are discarded. An additional AH hash value—*drop* hash value, is maintained on each node that drops false reports. It is updated incrementally each time when a false report is detected and dropped. To improve the effectiveness of Drop-COOL, a node may try to forward these false reports with different source CH IDs, which can trigger collecting and detecting more paths and thus expose more compromised nodes in one round.

The Drop-COOL protocol combines the advantages of both COOL and SEF protocols. It detects and excludes compromised nodes while saves the energy from routing less false reports. The energy spent to route a small number of false reports is small compared to the savings after excluding compromised nodes. It removes the worst case overhead that the basic COOL protocol has on routing false reports.

We next illustrate that the Drop-COOL does not affect the ability to locate compromised nodes although random false reports are dropped in the middle. Due to the introduction of the drop hash value, we have *for a well-behaved node in the routing graph, the set of forwarded and dropped messages should be the same as the set of received and locally generated messages*. By collecting and comparing the AH hash values of these message sets, we can adjust the node test to

$$(AH(MESSAGE_{forwarded}) + AH(MESSAGE_{dropped})) \bmod M = (AH(MESSAGE_{received}) + AH(MESSAGE_{local})) \bmod M$$

The link test is unaffected and an inconsistent node or link test result exposes at least one compromised node.

4.2. Hi-COOL: a hierarchical authentication scheme

With Drop-COOL, the worst case energy overhead to route false reports is reduced. However, once we decide to detect compromised nodes, it could still be expensive to collect many hash values as they are distributed on different nodes across the network. To reduce this overhead, we propose Hi-COOL, a hierarchical approach which groups multiple adjacent nodes in the routing graph as a *super node*. We only collect hash values with respect to this super node, and refine the collection if a super node is found problematic.

The Hi-COOL scheme works as follows. First the sink picks up an integer number l and forwards this integer with the "hello" message (for collecting the routing graph). The integer value is decremented for each hop downstream along the routing path and reset after reaching zero. A node sets itself to be the head of a super node if it receives l and be the leaf of a super node if it receives zero. If a node receives two values from two upstream nodes, it picks up the smallest one. In Fig. 8 nodes s1–s5 form a super node in which s2, s4, and s5 are leaf nodes while s1 is the head node. In the phase to collect the AdHASH values, only the incoming hash values to this super node and the outgoing hash values from this super node are collected. For example hash values AH_{23}, AH_{32} are omitted. The link test at the super node level is processed the same as the basic COOL—a failed link test removes two involved nodes. However an inconsistent (super) node test results in one additional round of hash value collection such that we can determine the exact location of the compromised node within the super node. In the second round, we only collect hash values from these inconsistent super nodes.

We next discuss several subtle issues in ensuring security when Hi-COOL is employed. One issue is that in the routing graph collection phase, each node should reply with its received integer number. This is to defend arbitrary super node formation due to compromised nodes. The other issue is the internal security within a super node. A new attack may be developed in some extreme hostile environments as follows. In Fig. 8, compromised nodes s1 and s3 collaboratively attack node s2. s3 generates a lot of packets which are dropped by s1 without being visible to any other nodes outside of this super node. The sink cannot detect this internal attack as it is consistent at the super node level. However, the energy of node s2 may be depleted due to this attack. To defend it, the head of each super node still collects hash values for its inside links. Instead of sending them back to the sink, it calculates an independent AH hash value and send it back to the sink. The super node consistency test can then be augmented to an iterative version with all incoming hash values propagated to compute the outgoing hash and find the inconsistency if they do not match.

4.3. Tag-COOL: recycling tagged sensor nodes

As proved in the proceeding section, it is possible that up to half of excluded sensor nodes are actually healthy ones. Blindly leaving them unused in the field is a waste of resources especially for environments where re-deployment is hard or expensive. We thus propose Tag-COOL, an optimization scheme that recycles these nodes without compromising security. The idea is that the computation overhead is cheaper than the communication

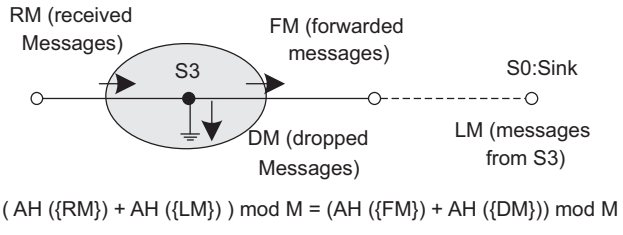

Fig. 7. Reducing routing energy through combined en-route filtering.

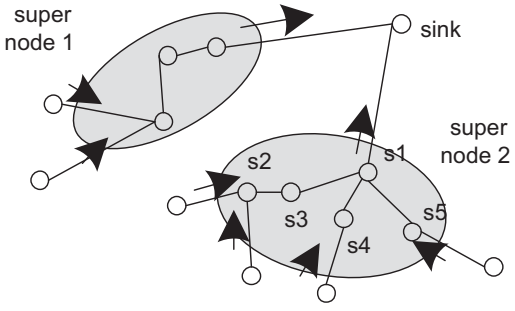

Fig. 8. Hierarchical authentication with super nodes.

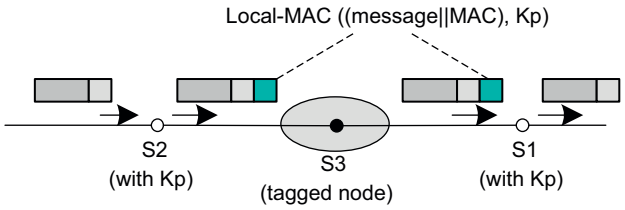

Fig. 9. Reducing routing energy through recycling tagged nodes.

overhead while the latter increases with the increased transmission distance. We thus can save the energy by cutting the transmission distance using a tagged node in the middle of two normal nodes.

The Tag-COOL scheme works as follows. In the construction of a routing graph we allow the tagged node (excluded from previous rounds) be involved as an relay-only node, that is, it can forward packet but is not allowed to generate sensing reports. Another restriction is that both its up and downstream nodes are not be tagged. To recycle this node, a pairwise key is assigned to or negotiated by its up and downstream nodes. Reports forwarded through this node are locally authenticated using this key. For example in Fig. 9, whenever a report is forwarded through s3, a MAC for this report is generated using the pairwise key K_p, which is then checked by s1 or s2. This can effectively disable the injection of false reports at node s3. A packet sequence number can be embedded in the generation of the MAC to further defend possible report dropping at node s3. If an inconsistency is found in the comparison of the outgoing AH value of s2 and the incoming AH value of s1, we first drop s3, and then drop both s2 and s1 if inconsistency persists.

Using Tag-COOL, nodes s2 and s1 spend the extra energy to calculate and check a new MAC but save the energy from transmitting packets half of the distance (e.g., between s2 and s3 instead of between s2 and s1). We gain the benefit if the transmission energy reduction is more than the computation cost.

5. Limitations of the COOL protocols

While the COOL protocol and its optimizations can help to identify compromised nodes in the network, it still has several limitations. We summarize them here. (1) It is possible that a subregion is isolated from the sink. Without having the information about a cluster head in the routing graph collection phase, the sink cannot identify its status and decide if it is compromised or not. (2) Signal blocking or collision is another source of attack, if normal communication cannot be ensured between two nodes. Both of two involved nodes are excluded while the nodes themselves may not be hacked. (3) To ensure security, the number of reports should be greater than k (Bellare and Micciancio, 1997), that is, more than 128 reports. Therefore, the hash value collection should start after the network has warmed up and passed around some reports.

6. Performance evaluation

6.1. Settings

To evaluate the effectiveness of the proposed COOL protocols, we simulate a sensor network with 450 cluster head nodes uniformly distributed in a field of $400 \times 400\,m^2$ area. Each sensor node is Mica2 running TinyOS (Hill et al., 2000) operating at 19.2 Kbps data rate, with battery voltage 3 V. It takes 16.25/12.5 μJ to transmit/receive a byte (Ye et al., 2004). This will be referred as the baseline setting in the rest of the paper. The sink is located at (20,20) and the communication range of each node is 40 m. These sensor nodes form a multihop routing network using the directed diffusion routing algorithm (Intanagonwiwat et al., 1999). A normal packet is of 24 bytes long, a MAC is of 8 bytes (64 bits), and an incremental hash value is of 16 bytes (128 bits). The evaluation is based on false report injection attacks. All results are averaged from 100 different runs.

6.2. The overhead

The protocol overhead comes from four sources: (i) AH hash value computation overhead; (ii) hash value collection overhead; (iii) routing graph discovery overhead; and (iv) routing false reports overhead. Next we study them in more detail.

(i) *Computation overhead*. The computation overhead is for updating incremental hash values at each sensor node. We ignore the authentication overhead at the sink due to its powerful computation capacity. As the incremental hash is maintained per link based, a received report updates two AH values on the relay node. The updates are done incrementally with the overhead mainly from computing the standard hashing MD5() on the input report. MD5() intermediate result is used to update both AH values. Our simulation results show that the incremental hash computation overhead is about twice of the overhead of one RC5 (Schneier, 1996) computation (used in Ye et al., 2004; Hill et al., 2000), that is, 30 μJ per node. It is small and thus omitted in the rest of the discussion.

(ii) *Hash value collection overhead*. The main overhead of the COOL protocol comes from collecting hash values across the network. We first induce a theoretical formula about this overhead. Assume the routing graph is a b-nary balanced routing tree with height $O(\log_b(N))$ where N is the total number of nodes in the graph. Since the hash value collection is per problematic routing path based, the number of node-to-node transmission T for one path is

$$T = (1+2+3+\cdots+H)\cdot(2\cdot k+M) = \frac{H\cdot(H+1)}{2}\cdot(2\cdot k+M)H = \log_b(N) \quad (2)$$

where H is the height of the b-nary tree, N the total number of sensor in the field, k the length of the incremental hash value, and M the length of the MAC value. From the equation, T is in the range of $O((\log N)^2)$. Since we need to exclude all m compromised nodes, we may need to collect from m disjoint paths or detect in m rounds. Therefore, the worst case hash value collection cost is $O(m\,(\log N)^2)$.

We next present the average number of rounds to exclude all compromised nodes. The reason that we need to run multiple rounds is discussed in Section 3. The results are presented in Fig. 10. In the experiment, false reports are randomly injected from the compromised nodes and we start a new detection round if 30 false reports are received. As expected it requires more rounds when there are a larger number of compromised nodes. On average we can detect and exclude more than 10 nodes in one round when more than 30 nodes are compromised.

Fig. 11 illustrates the total energy overhead for collecting hash values in multiple rounds. From Eq. (2), the hash value collection cost is proportional to the number of compromised nodes and to the square of the tree height, these two factors are used as the x and y axes, respectively. To change the tree height, we deploy in a different square field with the same node density, e.g., 1000 nodes are distributed in a field of $600 \times 600\,m^2$. The maximal

routing tree height is increased to 28.5 from 18.5 in the baseline setting.

The trend confirms what we observed from Eq. (2). For example, the energy overhead is about 0.87 J with 20 compromised nodes and the tree height 13.6. It increases about 2 times to 1.56 J if the number of compromised nodes increases 2 times to 40. It increases about 4 times to 3.89 J if the tree height increases about 2 times to 28.4.

We also present the results using the Hi-COOL scheme. It effectively reduces the hash value collection cost. For example, when 20 nodes are compromised, the Hi-COOL overhead is 0.99 J or 61% of the baseline setting (1.61 J).

In collecting the results, we perform a simple optimization. For the two nodes of each link, they may report the same AH() value, e.g., both nodes are healthy nodes. We therefore only need to transmit one AH hash value with two MACs to the sink.

(iii) *Routing graph overhead*. Fig. 11 also illustrates the energy overhead to collect the routing graph. Compared to the hash value collection overhead, it is usually small ranging from 0.5 J when the tree height is 9.2–2.0 J when the tree height is 28.5.

(iv) *The overhead to route false reports*. As we discussed, the sink has the option to start the hash value collection phase after accumulating a number of false reports. Fig. 12 illustrates the wasted routing energy in forwarding these false reports. Clearly if we increase the threshold, the routing overhead increases as well. The benefit of accumulating a reasonable larger number of false reports is that we increase the chance that these false reports are from more problematic paths. They can then detect multiple paths and reduce the total number of rounds. For example, if we detect after receiving one false report, we may need m rounds to exclude all m compromised nodes; on the other hand, we may need only $m/2$ round if we set the threshold to be 2 and these two false reports are always from two different compromised nodes on different routing paths. However, our experiments show that the difference is not significant (with one or two rounds difference). In addition, if the threshold is set larger than 30, the number of rounds does not change much but the wasted energy increases drastically. Therefore we set the threshold to 30 in the paper.

	the number of compromised nodes							
	10	20	30	40	50	60	70	80
Rounds	2.1	2.4	3.5	4.0	4.5	5.3	6.1	6.5

Fig. 10. The number of rounds to exclude all compromised nodes.

6.3. The savings

We next study the benefits from applying COOL protocols and compare it with an en-route false report filtering scheme (Ye et al., 2004). As discussed, the savings comes from two sources: the communication savings and the computation savings. The latter is omitted as it is usual very small.

Let us compare the overhead from different protocols. The overhead in the COOL protocol contains the energy to collect hash values, to discover routing graph, and to route false reports to the sink. The overhead in the SEF protocol is from routing false reports before detecting and dropping them. The exact number of hops varies with the key sharing scheme and the number of

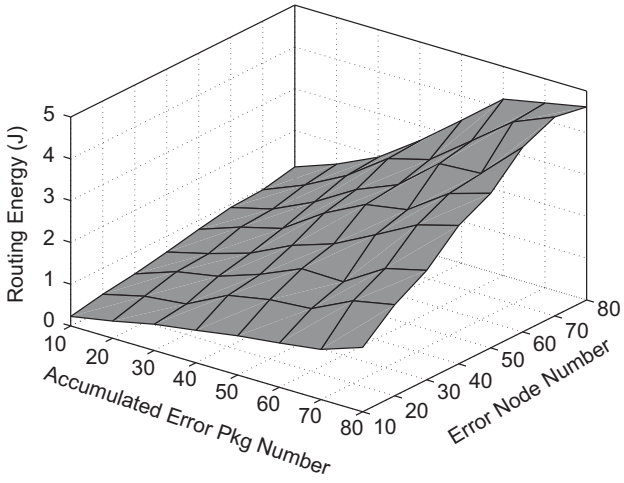

Fig. 12. The overhead to route false reports.

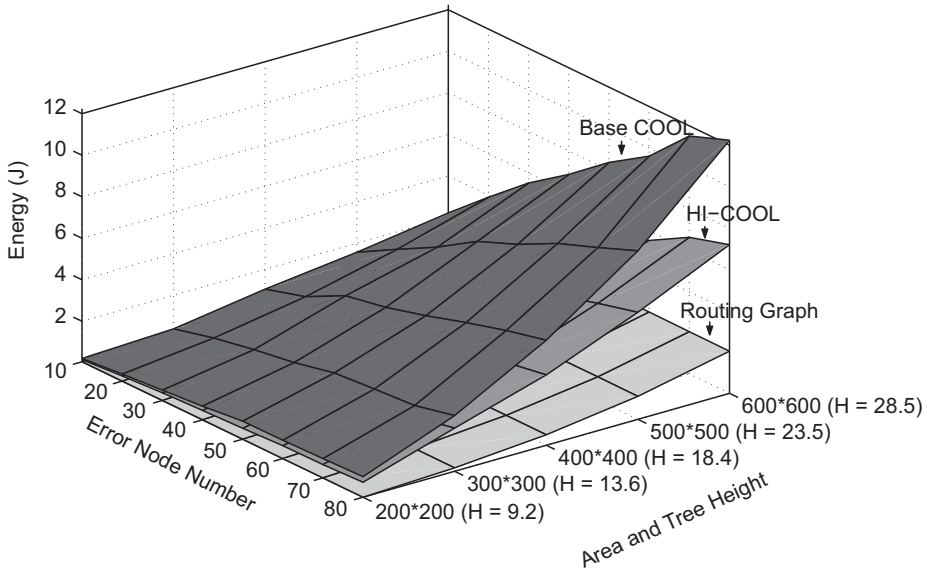

Fig. 11. Hash value collection overhead.

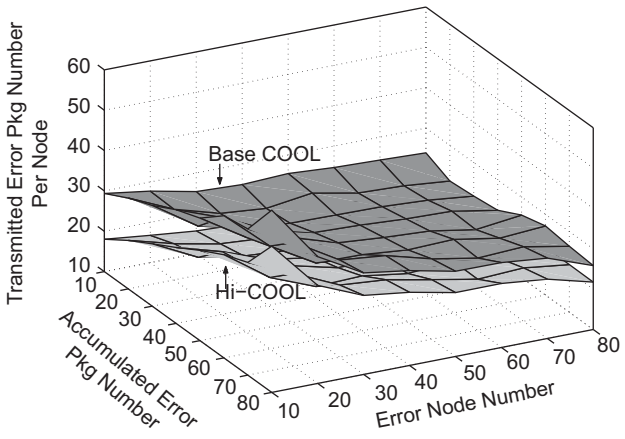

Fig. 13. Comparing the overhead to en-route filtering schemes.

compromised nodes. For comparison purpose, we assume on average each false report is dropped after 5 hops in the paper.

In Fig. 13, x and y axes are the detection trigger threshold and the number of compromised nodes in the network, respectively. Each point in the figure is a number of false reports averaged to each compromised node. It is a break even point at which the overhead to route and drop this amount of false reports in SEF equals the overhead in the COOL protocol. For example, with 20 compromised nodes and trigger threshold set at 30, the number is 25 reports meanings that, if SEF routes and drops 500 reports ($=25$ reports/node $\times$ 20 node) in 5 hops, the energy it wastes is the same as all of the overhead in the COOL protocol.

In addition, consider that the scheme needs to spend 3.3 rounds and a round is triggered at 30 false reports, there are another 100 injected false reports ($=30$ reports/round$\times$ 3.3 rounds). Therefore the COOL protocol outperforms the SEF if each compromised node injects more than 30 reports ($=25$ reports/node $+100$ reports/20 nodes). This is very small. For example, as suggested by Zhang and Cao (2005), a compromised node may inject a faked report every 10 s. At this rate, we outperform SEF in 300 s or 5 min. With the Hi-COOL optimization, it is further reduced to 4 min, a 20% reduction. Of course, depending on the pattern that the false reports are injected, this number may vary but the results show that the COOL overhead is very modest.

7. Related work

Since sensor networks are inherently vulnerable to varying forms of attacks, their wide adoption draws an increasing demand for security designs. Karlof and Wagner (2003) identified several attacks for a multihop routing based sensor network. These attacks include selective forwarding attacks, sinkhole attacks, sybil attacks and many other forms. Approaches have been proposed in the past for defending some of them.

The schemes to locate compromised nodes share some similarity with the approaches to detect and control faults in sensor networks (Jaikaeo et al., 2001; Marzullo, 1990; Chew and Marzullo, 1991). Marzullo (1990) and Chew and Marzullo (1991) proposed to tolerate node failure through redundancy. Jaikaeo et al. (2001) proposed to let the sink periodically sample the raw data from all sensors and therefore find problematic nodes from comparing different reports globally. The significant difference between compromised node detection and fault node detection is that a faulty node always returns a wrong report while a compromised node is smarter, e.g., it may inject false reports but communicate normally with the sink.

Approaches have been proposed to detect misbehaved nodes through their neighboring nodes. Marti et al. (2000) proposed to monitor each node with a watchdog—a node that passively listens to see if its next hop node does forward its messages. The watchdog was be used to improve network throughput by avoiding identified misbehaved nodes in later routing. However, the scheme suffers from several drawbacks as the authors identified (Marti et al., 2000), e.g., a compromised node can report its good neighbor misbehaving; the monitoring largely depends on the transmission range and location of the watchdog and thus is not very accurate. Wang et al. (2003) proposed to improve its accuracy from reaching an agreement collaboratively among neighbors of the suspect node. The basic design, however, is the same as Marti et al. (2000) such that misbehaved nodes can still collude to mark good nodes as compromised ones.

En-route false data filtering schemes (Ye et al., 2004; Zhu et al., 2004; Yang et al., 2005) are proposed to actively detect and drop false reports early in the routing. Ye et al. (2004) designed a probabilistic authentication scheme in which each sensor node selects a subset of keys from a global key pool and has the ability to verify a MAC which is generated using one of shared keys. In the interleaved authentication scheme developed by Zhu et al., the authentication keys are shared through node association along the routing path. These two schemes are t-resilient meaning that an adversary can break the scheme only if he can collect more than t different keys (partitions). Yang et al. (2005) proposed to achieve better resilience by sharing location-based keys. Yang et al. (2005) makes stronger assumption of the routing algorithm. In particular nodes close to the sink are becoming more sensitive.

Many recent research works enhance sensor network security through key management. Eschenauer and Gligor (2002) proposed a key pre-distribution scheme in which each sensor randomly selects a subset of keys before deployment. Protocols were then developed for shared key discovery and path-key establishment. Improvements were later proposed for enhancing security (Chan et al., 2003) and achieving higher probability of key establishment (Liu and Ning, 2003). Zhang and Cao (2005) proposed to represent keys as group key polynomials whose shares are distributed around neighbors. New keys can be regenerated collaboratively by neighboring nodes from these shares achieving better security and resilience to node compromise.

Detecting malicious nodes in WSNs has also been studied through heuristic based instruction detection designs. Yao et al. (2007) proposed a cluster head selection algorithm that improves the resistance to intrusions. Misra et al. (2008) proposed to sample a portion of packets in the network and detect intrusions based on S-model learning automata. Ma et al. (2008) employed a game-theory-based non-cooperative framework to perform intrusion detection and reduce resource consumption. Hai and Huh (2007) proposed algorithms to optimally select and activate intrusion detection agents based on trust heuristics. In other recent work, Wang et al. (2008) studied the detection of external intruders based on the sensing range, node density, and network connectivity. Sun et al. (2007) presented the challenges of constructing intrusion detection systems in WSNs, and surveyed secure localization and secure aggregation techniques in WSNs. These studies and approaches are orthogonal to what we have proposed in this paper.

8. Conclusion

In this paper we introduced the COOL protocol and its optimizations which exploit the provably secure incremental

hash function AdHASH to detect and locate compromised nodes effectively. We first discussed how to securely maintain and collect AdHASH values on sensor nodes across the network, and then use these values to perform node and link tests to expose the compromised nodes. Our experimental results showed that the COOL protocols are very effective and introduce very small overhead to the network.

References

Bellare M, Micciancio D. A new paradigm for collision-free hashing: incrementality at reduced cost. In: Eurocrypt'97, Lecture notes in computer science, vol. 1233, 1997.

Chan H, Perrig A, Song D. Random key predistribution schemes for sensor networks. In: IEEE symposium on security and privacy, 2003.

Chew P, Marzullo K. Masking failures of multidimensional sensors. In: Proceedings of the 10th symposium on reliable distributed systems; 1991. p. 32–41.

Eschenauer L, Gligor VD. A key-management scheme for distributed sensor networks. In: Proceedings of the ninth ACM conference on computer and communication security, November 2002. p. 41–7.

Gu Q, Chu CH, Liu P, Zhu S. Slander-resistant forwarding isolation in ad hoc networks. International Journal of Mobile Network Design and Innovation 2006;1:162–74.

Hai TH, Huh EN. Optimal selection and activation of intrusion detection agents for wireless sensor networks. In: Future generation communication and networking (FGCN), vol. 2, 2007. p. 350–5.

Heinzelman WR, Chandrakasan A, Balakrishnan H. An application-specific protocol architecture for wireless microsensor networks. IEEE Transactions on Wireless Communications 2002;1(4):660–70.

Hill J, Szewczyk R, Woo A, Hollar S, Culler D, Pister K. System architecture directions for networked sensors. In: ASPLOS IX, 2000.

Intanagonwiwat C, Govindan R, Estrin D. Directed diffusion: a scalable and robust communication in wireless sensor networks. In: Fifth IEEE/ACM Mobicom, 1999. p. 174–85.

Jaikaeo C, Srisathapornphat C, Shen C. Diagnosis of sensor networks. In: IEEE international conference on communications, June 2001.

Karlof C, Wagner D. Secure routing in wireless sensor networks: attacks and countermeasures. In: IEEE international workshop on sensor network protocols and applications, 2003. p. 113–27.

Kawadia V, Kumar PR. Power control and clustering in ad hoc networks. In: INFOCOM, 2003.

Liu D, Ning P. Establishing pairwise keys in distributed sensor networks. In: Proceedings of the ACM CCS, 2003.

Ma Y, Cao H, Ma J. The intrusion detection method based on game theory in wireless sensor network. In: IEEE international conference on ubi-media computing, 2008.

Marti S, Giuli TJ, Lai K, Baker M. Mitigating routing misbehavior in mobile ad hoc networks. In: MOBICOM; 2000.

Marzullo K. Tolerating failures of continuous-valued sensors. In: ACM Transactions on Computer Systems, November 1990.

Misra S, Abraham KI, Obaidat MS, Krishna PV. Intrusion detection in wireless sensor networks: the S-model learning automata approach. In: IEEE international conference on wireless and mobile computing (WIMOB), 2008.

Perrig A, Szewczyk R, Wen V, Culler DE, Tygar JD. SPINS: security protocols for sensor networks. In: Proceedings of seventh annual international conference on mobile computing and networks, 2001.

Schneier B. Applied cryptography, 2nd ed New York: Wiley; 1996.

Sun B, Osborne L, Xiao Y, Guizani S. Intrusion detection techniques in mobile ad hoc and wireless sensor networks. Wireless Communications 2007;14(5): 56–63.

Wang X, Feng D, Lai X, Yu H. Collisions for some hash functions MD4, MD5, HAVAL-128, RIPEMD. In: Crypto'04, 2004.

Wang Y, Wang X, Xie B, Wang D, Agrawal DP. Intrusion detection in homogeneous and heterogeneous wireless sensor networks. IEEE Transactions on Mobile Computing 2008;7(6):698–711.

Wang X, Yin Y, Yu H. Finding collisions in the full SHA-1 collision search attacks on SHA1. In: Crypto'05, 2005.

Wang G, Zhang W, Cao G, Porta TL. On supporting distributed collaboration in sensor networks. In: IEEE MILCOM, 2003.

Yang H, Ye F, Yuan Y, Lu S, Arbaugh W. Toward resilient security in wireless sensor networks. In: ACM MOBIHOC'05, 2005.

Yao L, An N, Gao F, Yu G. A clustered routing protocol with distributed intrusion detection for wireless sensor networks. In: Lecture Notes in Computer Science (LNCS), vol. 4505, 2007. p. 395–406.

Ye F, Luo H, Lu S, Zhang L. Statistical en-route detection and filtering of injected false data in sensor networks. In: IEEE INFOCOM 2004, 2004.

Younis O, Fahmy S. Distributed clustering in ad-hoc sensor networks: a hybrid, energy-efficient approach. In: INFOCOM, 2004.

Zhang W, Cao G. Group rekeying for filtering false data in sensor networks: a predistribution and local collaboration-based approach. In: INFOCOM, 2005.

Zhu S, Setia S, Jajodia S, Ning P. An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks. In: Proceedings of IEEE Symposium on Security and Privacy, Oakland, California, May 2004.