A Sinkhole Resilient Protocol for Wireless Sensor Networks: Performance and Security Analysis

Fabrice Le Fessant[a], Anthonis Papadimitriou[b], Aline Carneiro Viana[c], Cigdem Sengul[d], Esther Palomar[e]

[a] *INRIA, France*
[b] *University of Athens, Greece*
[c] *INRIA, France/TU-Berlin, Germany*
[d] *Deutsche Telekom Labs/TU-Berlin, Germany*
[e] *University Carlos III of Madrid, Spain*

Abstract

This work focuses on: (1) understanding the impact of selective forwarding attacks on tree-based routing topologies in Wireless Sensor Networks (WSNs), and (2) investigating cryptography-based strategies to limit network degradation caused by sinkhole attacks. The main motivation of our research stems from the following observations. First, WSN protocols that construct a fixed routing topology may be significantly affected by malicious attacks. Second, considering networks deployed in a difficult to access geographical region, building up resilience against such attacks rather than detection is expected to be more beneficial. We thus first provide a simulation study on the impact of malicious attacks based on a diverse set of parameters, such as the network scale and the position and number of malicious nodes. Based on this study, we propose a single but very representative metric for describing this impact. Second, we present the novel design and evaluation of two *simple and resilient* topology-based reconfiguration protocols that broadcast cryptographic values. The results of our simulation study together with a detailed analysis of the cryptographic overhead (communication, memory,

and computational costs) show that our reconfiguration protocols are practical and effective in improving resilience against sinkhole attacks, even in the presence of collusion.
Key words: Wireless sensor network, selective-forwarding and sinkhole attacks, resilience, tree-based routing protocols.

1. Introduction

The deployment of a wireless sensor network (WSN), in general, is governed by its application. In this paper, we focus on applications, such as data collection, where a large number of static nodes need to be deployed in a difficult to access geographical region. The general communication pattern is many-to-one: the sensors collect and send data to sink nodes, which in turn relay the data directly to a base station outside the network. Due to the difficulty in accessing the geographic location, the network is expected to operate for a satisfactory period of time without any intervention. A WSN provides a lightweight infrastructure to monitor changes remotely in hostile environments. Unfortunately, and precisely because of the nature of such environments, a sensor network is particularly prone to failures and, it is necessary to cope with various forms of disruptions, ranging from battery outages to malicious attacks. Furthermore, these malicious attacks can be as simple as propagating false information and still significantly impact network operation, especially routing. Therefore, it is essential to quantify the risk a network is under different type of attacks. Tackling this challenging problem constitutes the first goal of this paper.

For many applications, security (i.e., confidentiality, integrity and availability of information) is vital to the acceptance and use of sensor networks. For in-

stance, a large set of routing protocols in WSNs are based on the construction of a tree-based routing topology initiated by a sink [1, 2, 3, 4, 5, 6, 7]. In particular, these protocols use advertised information (e.g. hop count from a sink) to build a routing topology. Secure operation of these protocols is essential for the health of the network. Consider the attack, known as the *sinkhole attack* [8], where malicious sensors pretend to be closer to the sinks than all their neighbors. Attracting more traffic, these sensors can either selectively drop the received data (i.e., *selective-forwarding attack*) or collect sensitive information. Clearly, the protocols that construct a routing topology would be significantly affected by these attacks. More specifically, in Directed Diffusion [1] and TinyOS [7], routes are established simply based on the reception of beacon messages initiated by the sink. Hence, sinkholes are easy to create even without any collusion among sensor nodes as there is no mechanism to verify the originator and the contents of the message. Therefore, fighting against these attacks constitutes the second goal of this paper.

To meet aforementioned challenges, this paper first studies the impact of selective-forwarding attacks in tree-based routing topologies. We present a simulation study where we show the impact of these attacks based on a number of key performance parameters (e.g., node distribution, density, positioning, and attacker capability) that influence the impact of these attacks. Our study illustrates the effect of different combinations of these parameters. For instance, a low number of malicious nodes that are one hop away from the sink can affect the network in the same way as a high number of randomly distributed malicious nodes. Thus, based on our simulation study, we propose a single metric named "risk factor", that can span these variations.

Selective-forwarding attacks are usually combined with other attacks. Therefore, next, we consider the case when the compromised nodes combine selective-forwarding attacks with sinkhole attacks. In comparison with the current work [9, 10, 11, 12], this paper focuses on *resilience* against compromised nodes instead of *detection* of compromised nodes. We believe resilience is an important property in WSNs deployed in environments where human intervention is difficult. Furthermore, detection mechanisms often introduce more complexity, and so more weaknesses, into the system, which do not always justify their benefits [9]. To this end, as our second contribution, we propose two *RESIlient and Simple Topology-based reconfiguration protocols*: RESIST-1 and RESIST-0. RESIST-1 prevents a malicious node from modifying its advertised distance to the sink by more than one hop, while RESIST-0 does not allow this at the cost of additional complexity. Via simulations and using our risk factor metric, we studied the performance of RESIST-1 and RESIST-0 for three tree-based routing protocols, on a large set of topologies, and with different levels of adversarial power. We also evaluate the time and energy consumption of security operations of RESIST algorithms to illustrate their feasibility.

In summary, the contributions of this paper are three-fold:

1- We propose a simple but representative metric describing the impact of selective-forwarding attacks in tree-based routing protocols.

2- We introduce two protocols for building up resilience in wireless sensor networks. The simulation results show that our protocols are practical and effective in improving resilience against sinkhole attacks with different levels of adversarial power.

3- We provide an analysis of the feasibility of the proposed protocols (e.g., in terms of time and power consumption). These discussions expose in greater detail our motivation on the viability of implementing the proposed protocols in current sensor devices, such as MICAz and TelosB.

The remainder of this paper is structured as follows. In Section 2, we present the system model. In Section 3, we investigate the representation of the impact of malicious nodes. In Section 4, we lay out our proposal for two simple and resilient topology-based routing protocols. Performance results are presented in Section 5. The Section 6 overviews the current literature. Section 7 concludes with future work. In Appendix, we present further details on cryptographic overhead of the protocols and discuss optimizations to reduce these costs.

2. Problem statement

We focus on sensor networs, where the main application is environmental monitoring scenario and physical access to the monitored region is difficult. Our main goal is to quantify and limit the impact of disruptions caused by compromised/malicious nodes in such networks. (In the rest of the paper, the terms compromised and malicious are used interchangeably.) In the following, the network and threat models are presented in more detail.

2.1. Network model

We consider a connected WSN consisting of S static sensor nodes and one sink node deployed in a remote area. Each node has a unique ID. Nodes do not know any location information. Each node n_i or the sink is able to communicate wirelessly with a subset of nodes N_{n_i} (its *neighbors*) that are in its transmission

range, r_t. We assume that for any two nodes X and Y with similar transmission ranges, if X can communicate with Y, then Y can communicate with X.

We focus on routing protocols that rely on tree-based topology construction [1, 2, 3, 4, 5, 6, 7], where the data is routed from sensor nodes to the sink through a tree rooted at the sink. The routing tree is an aggregation of the shortest paths from each sensor to the sink based on a cost metric, which typically represents any application requirement (e.g., hop count, loss or delay). In this paper, we assume the routing tree is built by using the hop distance to the sink and through periodic routing messages the routing topology is refreshed regularly. It is worth noting that our RESIST protocols are deployed under the routing protocol, and expects correct execution of the tree construction and maintenance.

2.2. Trust and threat models

A straightforward implementation of a secure WSN may consider multiple sinks, each equipped with its own public/private key pair. For simplicity of presentation, we only consider the case of a single sink. Hence, in our model, all sensors know and trust the public key, K_{pub}^{sink}, of the sink. Additionally, each sensor n_i has a pair of public-private keys $(K_{pub}^{n_i}, K_{pri}^{n_i})$ that it uses to prove its identity. These key pairs can be generated and uploaded offline to the sensors before the deployment. Using these key pairs, nodes perform authentication and sign data messages. Finally, in our trust model, sensors never lie about their identities due to the use of cryptographic methods [13]. In fact, we assume that public-key cryptographic primitives are available on all sensors. In Section 5.4 and Appendix A, we discuss the overhead, and hence, the feasibility of our assumptions for current sensor node architectures.

In this paper, we consider two types of threats: selective-forwarding and sink-

hole attacks. We first focus on selective-forwarding attacks launched by the compromised nodes inside the network (Section 3). Compromised nodes are modeled as nodes that drop messages with probability p instead of forwarding them. When probability $p = 1$, compromised nodes drop all the messages (this is usually the case in sinkhole attacks). When $p < 1$, compromised nodes can disrupt the network operation, without being easily detected.

Next, we focus on sinkhole attacks launched by the compromised nodes inside and/or outside the network (Section 4). In this case, the objective of the compromised nodes is to appear attractive to their surrounding nodes in terms of routing. An example scenario could be that a malicious node claiming to reach the sink in a single hop. Hence, the compromised node advertises a single high-quality route to the sink attracting a possibly large volume of traffic. Furthermore, two or more sensors may collude to increase the impact of their attack on the network (e.g., a wormhole attack). Solutions to the above attacks have been generally based on temporal and geographical stamps [14]. We will analyze the impact of collusion on our security protocols in Section 5.3.

We define a common notation, $SA(X, d_1, d_2)$, for a sinkhole attack by node X, advertising a distance d_1 instead of its real distance d_2. Note that in a pure selective-forwarding attack, the malicious node might not lie about its distance. Hence, the attack is $SA(X, d(X), d(X))$, but packets are dropped with a probability p.

3. Impact of Malicious Sensors

When assessing the performance of tree-based routing protocols, it is crucial to characterize the routing topology in terms of its vulnerability to malicious sensors. Typically, "the number of compromised sensors" is used as a metric for

this purpose [9, 15]. However, this metric is not necessarily a good indicator of the hazard that malicious nodes might cause in a WSN: one compromised sensor close to the sink can reduce the data delivery success more than dozens of compromised sensors at the border of the network. Intuitively, when tree-based routing protocols are in use, the impact of a malicious sensor mostly depends on the number of uncompromised sensors in its sub-tree. We thus introduce a new metric, called *Risk Factor*, which is able to represent the interplay among different parameters such as the number of compromised sensors, their position, the density and size of the network. This new metric allows us to evaluate the impact of selective forwarding and sinkhole attacks on tree-based routing protocols by classifying different compromised topologies into a few equivalence classes. Next, we present our metric and show, through simulations (performed using a discrete event-based simulator implemented in Java), how it captures various parameters of compromised topologies.

3.1. Risk Factor computation

We compute the "Risk Factor" of a given topology by first computing a *local risk factor* for each node X, denoted as $LRisk_X$. Essentially, $LRisk_X$ intuitively shows the probability that a message from a node X arrives at a compromised sensor on its way to the sink. Then, the risk factor of the whole topology can be computed as the average of the local risk factors of all nodes in the network.

To compute $LRisk_X$ for all nodes, we first consider the network topology as a graph $G(V, E)$, where V is the set of sensor nodes and the sink, and E is the set of edges, (i.e. links between nodes that can communicate directly within transmission range). Any shortest path algorithm, e.g. Dijkstra or Bellman-Ford, can be run over $G(V, E)$ to compute the *distance to the sink* for each sensor as the

minimum hop count to the sink.

The $LRisk_X$ of a compromised node X is its probability p of dropping a message, while, $LRisk_{sink}$ of the sink is 0, as we assume that the sink cannot be compromised. For all other nodes, $LRisk_X$ is computed as the average of the local risk factors of all neighbors that are strictly closer to the sink. More formally:

$$LRisk_X = \begin{cases} 0 & \text{if } X \text{ is the sink} \\ p & \text{if } X \text{ is malicious} \\ \frac{\sum_{Y \in N_X | d_Y < d_X} LRisk_Y}{|\{Y \in N_X | d_Y < d_X\}|} & \text{otherwise,} \end{cases} \quad (1)$$

where N_X is the neighbor set of X, d_X is its distance to the sink, $\{Y \in N_X \mid d_Y < d_X\}$ is the subset of its neighbors with a shorter distance to the sink, and $|S|$ is the cardinality of S. While Equation 1 does not explicitly represent the attacker capability, except for selective forwarding probability p, the effect of different type of "distance" attacks is captured implicitly through the use of d_X. Note that the "distance attacks", such as the sinkhole attacks considered in this paper, mainly affect how a node perceives its distance to the sink and hence, d_X. We present further detail on risk factor computation under different scenarios in Section 5.

$LRisk_X$ is computed recursively in a distributed way starting from the sink until the leaves of the routing tree. Given $LRisk_X, \forall X \in V$, the risk factor of the entire topology, ***TRisk***, is:

$$TRisk = \frac{\sum_{X \in V} LRisk_X}{|V|} \quad (2)$$

The strength of the proposed risk factor lies in its ability to capture the mean impact of all the possible shortest-path trees that can be created by an arbitrary routing protocol. Essentially, the local risk factor accounts for all neighbors that are closer to the sink, and hence, it is able to represent all the potential parents

(including compromised nodes pretending to be closer to the sink) on any tree-based routing topology.

3.2. Risk Factor pertinence

In this section, we show how our Risk Factor captures the different characteristics of a compromised topology. We assume malicious nodes perform selective-forwarding attacks with $p = 1$. We evaluate Risk Factor with varying:

- **Compromised node distributions**, which represents the distribution of compromised nodes in the geographic area covered by the network.
- **Network scale**, which defines the number of sensor nodes and the area the network covers.
- **Number of compromised sensors**

3.2.1. Distribution of compromised nodes

The distribution of compromised sensors has an important impact on the extent of the damage. As a rule of thumb, if the compromised nodes are closer to the sink, their effect is higher since they are expected to forward more data than nodes that are farther away. To understand how the Risk Factor takes this into account, we evaluate four different distributions of compromised sensors (not necessarily realistic):

- **Uniformly Random (*UR*)**
- **Linear (*L*)**, so that they form an imaginary line that runs through the area of the network.
- **Ring (*R*)**, so that they form a ring surrounding the sink.

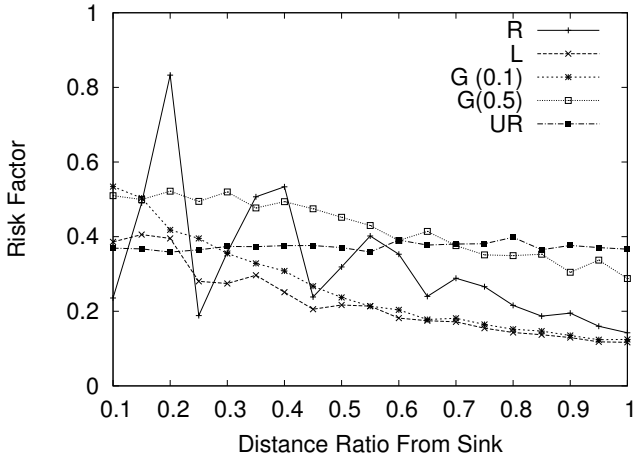


Figure 1: Risk Factor of different malicious node distributions.

- **Gaussian (*G*)**, so that they follow a Gaussian distribution around a *center point* according to a dispersion parameter.

Fig. 1 shows that our risk factor indeed captures the impact of these distributions. All topologies are of fixed size (500 sensors), density ($3 \ln 500$) (as defined in [16]) and number of compromised nodes (50). All nodes are uniformly distributed in a simulation area $a^2 = \frac{\pi r_t N}{3 \ln N}$ (as in [17]), except the sink which is always in the center. For each distribution, we plotted the risk factor as the *distance from the sink* increases. The distance is the distance of the ring for R, and is not a factor for UR. For L, it represents the distance from the sink to the closest (imaginary) point on the line. Finally, for G, it is the distance to the center of the distribution. Note that the distance is normalized by the maximum distance to the side of the network area. For *G*, we use the same normalization for the variance, and restrict ourselves to 0.1 and 0.5 for Fig. 1.

As expected, the risk factor increases as the distance decreases (except for *UR*, where the distance is not a parameter of the distribution). Perhaps less ex-

pectedly, the risk factor oscillates for the R distribution, which can be explained as follows: if we represent all the nodes with the same distance to the sink as a disk, compromised nodes on the border of the disk have a higher chance than the ones inside to be chosen as parents by the nodes outside the disk. Hence, the risk factor is maximum when the ring of compromised nodes is exactly at a multiple of the transmission range (here, the transmission range is equal to 0.21 times the maximum distance to the sink). Nevertheless, the risk factor for R still globally decreases as the distance increases.

3.2.2. Scale of the Sensor Network

In this section, we investigate the effect of network scale. Intuitively, networks with higher number of nodes are expected to experience less danger compared to sparse networks with the same number of malicious nodes. We evaluate the scale of a sensor network as (1) the number of sensors and (2) the geographical area the sensor network covers. Hence, in our simulations, we either kept the area of the network constant (*Area-Constant*/AC deployment) and hence, increased the density by adding more nodes, or increased the area of the network proportionally to the number of sensors (*Density-Constant*/DC). Furthermore, for each case, we first assumed that the number of malicious nodes remained the same (*Malicious-Constant*/MC). Next, we also scaled up the adversary capability and kept the uncompromised to compromised ratio constant (*Malicious-Adapting*/MA). In our simulations, in the *DC* deployment, the network density is $3ln(100)$, whereas in the *AC* scenarios the network spans 95×95 meters. In *MC* scenarios, the number of malicious nodes is 50. Finally, we use two *MA* configurations with the ratio of malicious nodes is 10% and 50%.

Fig. 2 depicts the risk factor for these different cases. For *Area-Constant* and

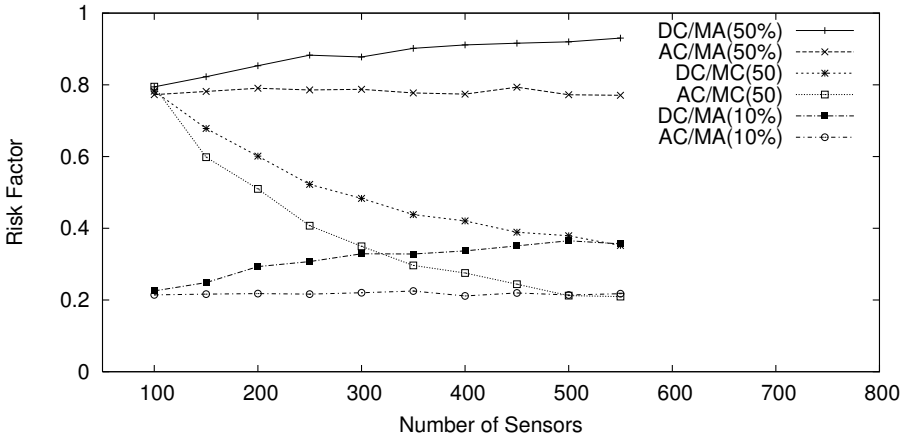


Figure 2: Risk Factor as the scale of the network increases

Malicious-Constant (AC/MC), as expected, the risk factor decreases considerably, as the number of nodes increases and the number of compromised nodes stays constant. The same argument also applies to *Density-Constant* and *Malicious-Constant* (DC/MC). However, in the case of *Malicious-Adapting* (MA), the two different deployments exhibit different behaviors. For *AC*, the increase in the number of nodes is neutralized by the increase in compromised nodes. However, this is not the case for *DC*. Since the transmission range is fixed, a bigger area increases the depth of routing trees that connect nodes to the sink. So, as the number of malicious nodes scales with the number of nodes, each malicious node has a potentially higher impact based on the depth of the tree. The risk factor captures this difference between *AC/MA* and *DC/MA*, as it remains constant for the former and increases for the latter.

3.2.3. Number of compromised nodes

Finally, we present how the risk factor captures the effect of the number of malicious nodes in the network. In Fig. 3, we evaluate the risk factor for differ-

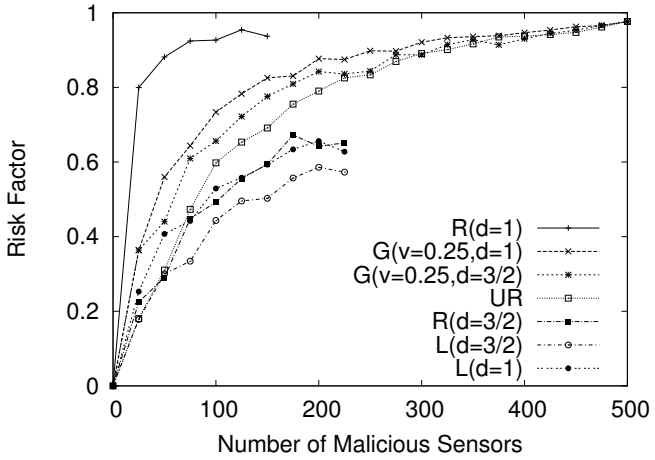


Figure 3: Risk Factor for increasing number of malicious nodes and different positioning: L and R curves are limited by the number of malicious nodes that can be put on the line or the ring.

ent malicious nodes distributions (discussed in Section 3.2.1). All topologies are networks of 512 sensors with moderate density ($3ln(512)$) and the transmission range r_t of each sensor is 20 m and the network is $185 \times 185\ m$.

Fig. 3 shows that the ring case (i.e., *R*) can cause major damage to the network with a relatively small number of malicious nodes, however, only at limited distances from the sink (d). On the other hand, for both *UR* and *G*, there is no limit on the number of malicious nodes. Hence, as the number of malicious nodes increases, their risk factors become higher than the risk factor of *R (d=1)*. Most importantly, Fig. 3 shows that risk factor increases fast until $25-40\%$ of the nodes are compromised and from this point on, the increase is not significant. This is also what would be expected in a real world scenario. Hence, we believe our risk factor metric is able to represent the impact of different parameters. Furthermore, the correctness of our metric is also shown in Section 5, where we show that the receive success always decreases as the risk factor increases.

4. Security protocols

In this section, we describe two reconfiguration protocols aimed at fighting sinkhole attacks on tree-based routing in WSNs.

4.1. Overview and notation

To achieve higher resilience in tree-based routing protocols [1, 2, 3, 4, 5, 6, 7], we propose two schemes, which are executed during the routing tree reconfiguration phase triggered by the sink. The proposed schemes are implemented under the routing protocol and can be adapted to any tree–based protocol. We do not have any constraints on the period between reconfigurations: it is chosen by the routing protocol and, for optimization reasons, can be tuned based on cost or topology vulnerability. We define a class of RESIST-h protocols that prevent malicious nodes from modifying their advertised distance to the sink by more than h hops. Based on this definition, we introduce two protocols, RESIST-1 and RESIST-0, which are presented in the remainder of this section. We also describe here cryptographic operations and message contents of the proposed protocols, but refer the reader to Appendix A for an efficient way of implementing them.

We use the following notation. ID^{n_i} symbolizes the unique identification number of the node n_i and N_{n_i} represents the set of neighbors of node n_i. Moreover, let $K^{n_i}_{pub}, K^{n_i}_{pri}$ be the key pair for node n_i and $\{x\}_K$ be a signature algorithm (e.g. any suitable ECC–DSA algorithm) that signs message x under key K.

4.2. Simple reconfiguration protocol (RESIST-1)

The reconfiguration starts by the sink sending a `Hello`$(epoch, tokens)$ message (Fig. 4–$m_{1.1}$) to all its neighbors (N_{sink}), where $epoch$ is a strictly increasing timestamp, chosen by the sink, and $tokens$ is a list of tokens $[T_1, T_2, ..., T_R]$ (note

Simple reconfiguration protocol (RESIST-1)

1. $\mathcal{S}ink \rightarrow n_i \in N_{sink}$: $m_{1.1} = hello(epoch, [\mathbf{T_1}, \mathbf{T_2}, ..., T_R])$
2.1 $n_i \in N : n_i \rightarrow n_j \in N_{n_i}$: $m_{2.1} = hello(epoch, [\mathbf{T_d}, \mathbf{T_{d+1}}, ..., T_R])$
2.2 $n_j \rightarrow n_k \in N_{n_j}$: $m_{2.2} = hello(epoch, [\mathbf{T_{d+1}}, ..., T_R])$

where $T_x = \langle x, epoch, \{\langle x, epoch\rangle\}_{K_{pri}^{sink}}\rangle$

Complex reconfiguration protocol (RESIST-0)

1. $\mathcal{S}ink \rightarrow n_i \in N_{sink}$: $m_1 = hello(epoch, [\mathbf{T_1}, \mathbf{T_2}, ..., T_R])$
2.1. $n_i \in N : n_i \rightarrow n_j \in N_{n_i}$: $m_{2.1} = hello(epoch, [\mathbf{T_d}, \mathbf{T_{d+1}}, ..., T_R])$
2.2. $n_j \rightarrow n_k \in N_{n_j}$: $m_{2.2} = hello(epoch, [\mathbf{T_{d+1}}, ..., T_R])$
3. $n_k \rightarrow n_j$: $m_3 = Challenge(d, epoch)$
4. $n_j \rightarrow n_k$: $m_4 = ChallengeReply(d, epoch, K_{pub}^d, \{\langle d, epoch, K_{pub}^d\rangle\}_{K_{pri}^{sink}}, \{\langle ID^{n_j}, ID^{n_k}\rangle\}_{K_{pri}^k})$

where

(K_{pub}^x, K_{pri}^x) is a new key pair for token k generated by the sink

and $T_x = \langle x, epoch, K_{pub}^x, \{\langle x, epoch, K_{pub}^x\rangle\}_{K_{pri}^{sink}}, K_{pri}^x\rangle$

Figure 4: RESIST-1 and RESIST-0 schemes.

that this list is created according to the underlying tree-based routing protocol, i.e. R represents the largest hop distance to the sink). Essentially, each token is a *(token number, epoch)* pair signed by the sink:

$$T_x = \langle x, epoch, \{\langle x, epoch\rangle\}_{K_{pri}^{sink}}\rangle \tag{3}$$

where x is the token number.

When a sensor n_j receives a `Hello` message (Fig. 4–$m_{2.1}$), and after verifying that the tokens are correctly signed by the sink (i.e. by using the public key K_{pub}^{sink}), it does the following:

2.1 If the $epoch$ is new, it remembers the identity of the node sending it (his *parent*), and propagates the `Hello` message after removing the token with the shortest hop distance from the list of tokens (Fig. 4–$m_{2.2}$). In other words, it receives `Hello`$(epoch, [\mathbf{T_d}, T_{d+1}, ..., T_R])$ but sends `Hello`$(epoch, [\mathbf{T_{d+1}}, ..., T_R])$.

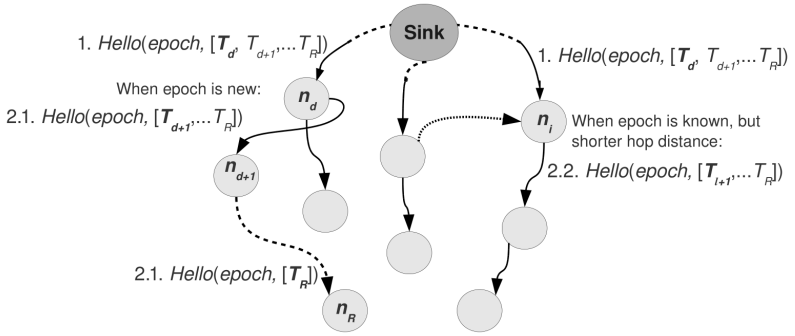


Figure 5: RESIST-1 overview. Tokens are removed from the list while the messages are forwarded up the tree, so that sensors far from the sink have less information than sensors closer to the sink.

2.2 If the $epoch$ is already known, but the `Hello` message advertises a shorter hop distance to the sink (i.e., contains a smaller token), a *selfish approach* would only update the node itself, while a *gossip approach* would also propagate a new `Hello` message to the neighbors. In the rest of the paper, we follow the gossip approach.

Each sensor remembers as its *parent*, from which it received the shortest distance token in the most recent epoch. Alternatively, sensors can also choose to remember *all the nodes* that advertise the shortest distance for a given epoch. In Section 5, we also evaluate this approach. Fig. 5 illustrates an overview of the RESIST-1 scheme.

4.2.1. Sinkhole attack resilience

A compromised node can directly forward the `Hello` message without dropping the first token. Assume that the node is the first compromised node on the branch where the `Hello` message travels. Then, if the compromised node is at distance d from the sink, its neighbors would believe they are at distance d too, and so, they would believe that the compromised node is at distance $d - 1$. Nev-

ertheless, the compromised node cannot pretend to be at a distance smaller than $d-1$, because it would be unable to provide smaller tokens than T_d. Note that as the `Hello` message travels up the tree, it might encounter other malicious nodes that do not drop the token before forwarding the message. In this case, each correct sensor would believe that it is at a shorter distance from the sink depending on how many malicious nodes exist before it (e.g., if the number of malicious nodes between the sensor and the sink is 2, then it will *at most* believe it is 2 hops closer to the sink than the reality). Hence, the deviation from the real distance in RESIST-1 increases with the number of malicious nodes on the path. However, the main impact is still received from the node closes to the sink, and other compromised nodes are to expected to have diminishing effects as the distance to the sink increases.

A compromised node may also make nodes think they are farther away from the sink that in reality, by removing more than one token from the list of tokens. Nevertheless, our focus is on selective forwarding combined with sinkhole attacks, and so, the increase of the distance to the sink will not help compromised nodes attract traffic. Additionally, the gossip approach can limit the impact of this attack, since `Hello` messages advertising the shortest distance to the sink may be received from other neighbors.

4.3. Complex reconfiguration protocol (RESIST-0)

The protocol RESIST-0 is inspired by a protocol used to measure availability in peer-to-peer networks [18], where newly generated pairs of cryptographic keys are diffused in the network at every round.

The sink sends a `Hello`$(epoch, [T_1, ..., T_R])$ message (Fig. 4–m_1), where the

generated tokens are:

$$T_x = \langle x, epoch, K^x_{pub}, \{\langle x, epoch, K^x_{pub}\rangle\}_{K^{sink}_{pri}}, K^x_{pri}\rangle, \quad (4)$$

where (K^x_{pub}, K^x_{pri}) is a newly generated pair of cryptographic keys for token T_x at a given $epoch$. The protocol is similar to RESIST-1, except that, before choosing a sensor Y as its parent, a sensor X first challenges Y by sending a `Challenge`$(d, epoch)$ message (Fig. 4–m_3). Basically, this message asks Y to prove its distance d from the sink (i.e. that it has a copy of the token T_d). Sensor Y replies with a message `ChallengeReply` (Fig. 4–m_4), which contains:

$$\langle d, epoch, K^d_{pub}, \{\langle d, epoch, K^d_{pub}\rangle\}_{K^{sink}_{pri}}, \{\langle ID^Y, ID^X\rangle\}_{K^d_{pri}}\rangle$$

The first part is the first half of the token T_d that Y received. At the reception of the `ChallengeReply` message and using the public key K^{sink}_{pub}, node X can first verify if the token T_d was correctly signed by the sink. In addition, node X recovers the public key of the token T_d, K^d_{pub}. Then, it can verify the second part of the `ChallengeReply` message, i.e. the identities of X and Y, that were signed with the private key of token T_d, K^d_{pri}. If verified, it accepts Y's advertised distance d from the sink. For the sake of illustration, Fig. 6 depicts a sequence chart for RESIST-0 scheme. It is worth noting that, these sets of operations prevent a node, which is a neighbor of both X and Y and that got the `Hello` message, from answering the `Challenge` message.

4.3.1. Sinkhole attack resilience

It is impossible for a compromised sensor (without collusion) to correctly reply to a `Challenge`, since it needs to know the privvate key for the token T_d to sign the second part of the `ChallengeReply` message. Furthermore, compromised nodes cannot even carry out the attack that we described for RESIST-1.

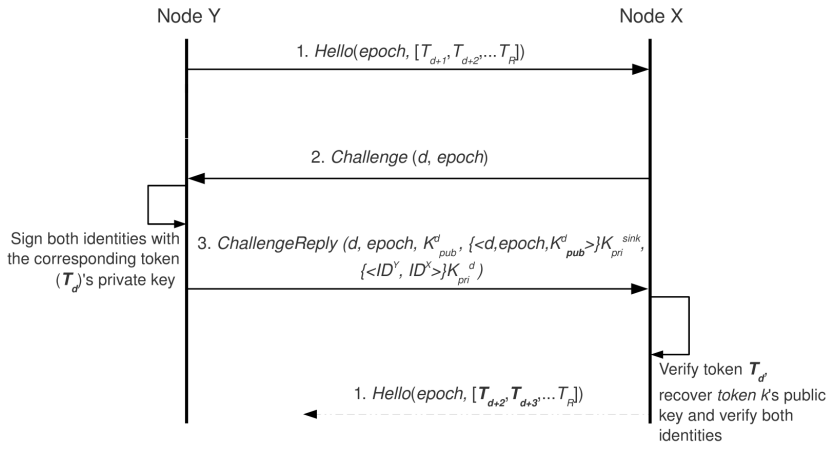


Figure 6: RESIST-0 overview. RESIST-0 improves on RESIST-1 by checking the advertised distance of each sensor.

Essentially, not dropping the token would fail, because they would not be able to respond to the `Challenge` for the shortest hop count. Hence, RESIST-0 provides strong resilience against sinkhole attacks. We discuss the impact of collusion on our protocols in Section 5.

5. Performance Evaluation

The main focus of our evaluation is to understand the amount of resilience obtained by RESIST protocols described in Section 4. We also consider the overhead of RESIST protocols in terms of the time and energy it takes to perform cryptographic operations. Additional implementation requirements and a discussion on the feasibility of the RESIST protocols are presented in the Appendix.

In the following, we first evaluate the effect of three different routing schemes and multi-path routing. Next, we present performance results under different scenarios with both non-colluding and colluding malicious nodes. We finally con-

clude with analytical results on energy and time overhead.

The experiments were run in a discrete event-based simulator implemented in Java. As we are only interested in a RESIST's algorithmic evaluation and on its resilience to message losses due to selective-forwarding and sinkhole attacks, our simulator uses a simplified MAC layer, where wireless message losses are not considered. These losses may only lead to some nodes not learning the real shortest paths to the sink, and does not affect the correctness of RESIST protocols.

5.1. Simulation Setup

This section describes three baseline protocols and our simulation setup. We consider a data collection application, where each sensor periodically sends data (e.g., measurements) to the sink. The routing tree is regularly reconfigured (e.g., according to the specification of the [7]). Malicious nodes do not generate data and they drop every received message with probability $p = 1$.

We implemented three baseline routing protocols: *FTree*, *RRobin* and *RWalk*. We studied the performance of these protocols in networks when resilient reconfiguration schemes are used (RESIST-1 and RESIST-0) and not used (*vulnerable* case). In our simulations, compromised nodes try to attract higher volumes of traffic by advertising shorter paths.

In *FTree*, the routing tree is rebuilt at each reconfiguration phase. Every sensor forwards all its data to its parent until the next reconfiguration. *RRobin* differs from *FTree* as each sensor computes a set of alternative parents during the reconfiguration phase. This set includes the neighbor that sent the first `Hello` message and any neighbor that sent a `Hello` message with a hop count smaller or equal to the first neighbor. Each time a sensor has to send a message, it selects one parent from this set in a round robin way. In *RWalk* protocol, each sensor makes a ran-

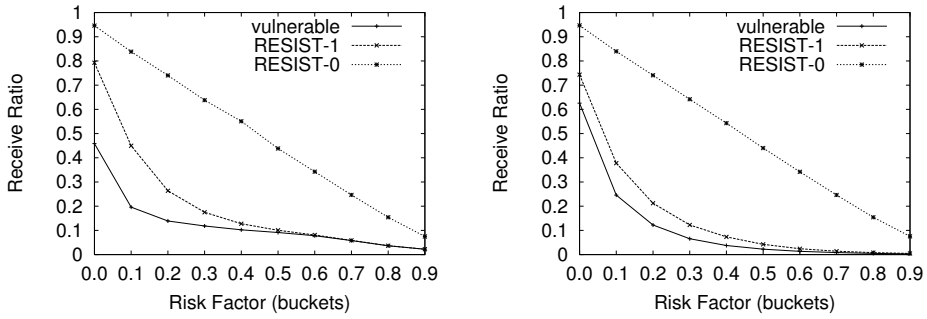
dom decision about forwarding a message either over the routing tree (computed as in *FTree*) or forwarding it to a randomly selected neighbor. If the message is not sent over the tree, it follows a n-hop random walk and after n hops, it is again forwarded over the routing tree. The goal of both *RRobin* and *RWalk* protocols is to avoid regions that may be severely affected by malicious nodes. In all our experiments with *RWalk*, we use $n = 1$.

We generated many random topologies. The space of topologies was divided in 10 *buckets*, where buckets 0, 1, etc. contain the topologies whose risk factor is respectively in [0,0.1), [0.1, 0.2), etc. At each step, the risk factor for the topology was evaluated and then, the topology added to the corresponding bucket, until every bucket had at least 100 topologies. Using these topologies, the performance gain was computed as the ratio of messages that actually reach the sink compared to the number of messages that should reach the sink if no sensor were compromised.

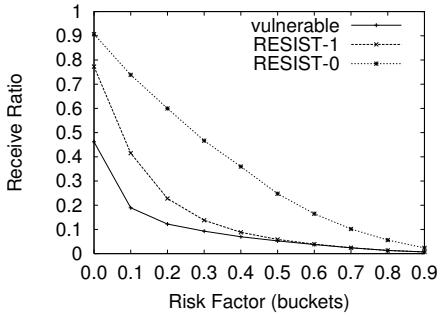
5.2. Evaluation of RESIST protocols with non-colluding attacks

In this section, we evaluate each routing protocol separately. Our results show that RESIST-0 achieves significant performance gain for all routing protocols (see Fig. 7). RESIST-1 improves performance compared to the vulnerable case, but the gain is much smaller than with RESIST-0. In general, as the risk factor increases, the performance of routing protocols decreases. More importantly, for both vulnerable and RESIST-1 cases, this decrease is roughly exponential, whereas for RESIST-0, it has a better, linear decrease, as it does not allow nodes to lie about their distance to the sink.

Fig. 7 confirms that when malicious sensors are able to lie, they can attract more network traffic and thus, incur a much higher impact in the WSN. The linear



(a) FTree routing (b) RRobin routing



(c) RWalk routing

Figure 7: Performance gain for different routing protocols: (a) *FTree*, (b) *RRobin* (c) *RWalk* ($n = 1$)

decrease achieved by RESIST-0 is the upper bound of the performance we can obtain by only addressing the sinkhole attacks.

5.2.1. Effects of using multi-path routing

To get better results, one must also fight selective forwarding attacks. An attractive approach to decrease the impact of selective-forwarding attacks is to send each message through multiple paths to the sink. Fig. 8 shows the improvement gained by using two paths per message (one *FTree* path and another *RRobin* path). In this case, an improvement of 5%-10% is observed. These results confirm the superiority of RESIST-0 in resisting sinkhole attacks compared to RESIST-1 and

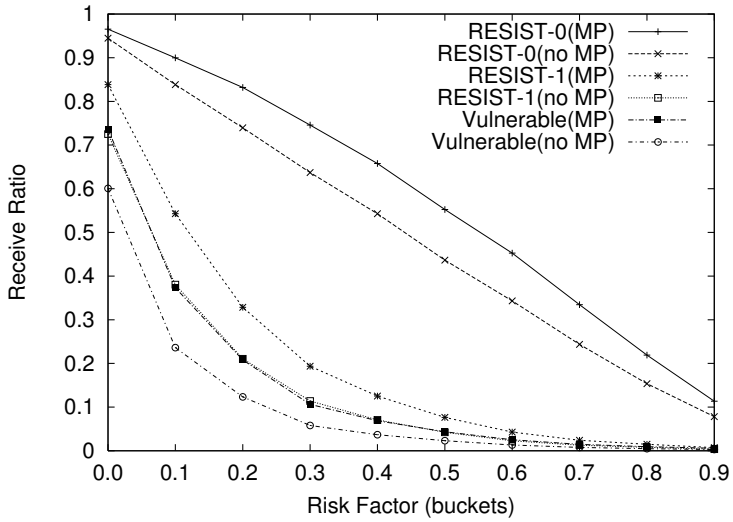


Figure 8: Using two paths (MP) instead of one increases the overall performance of the different strategies.

multi-path routing.

5.2.2. Effects of using different tree-based routing protocols

In this section, we compare the three routing protocols for resilient (i.e., RESIST-0 and RESIST-1) and vulnerable cases. To make such a comparison, we also modified the computation of the risk factor to represent the attacker capability more accurately. The main goal of this study is to understand which routing protocol is more advantageous among the three.

Performance in RESIST-0 Case. The performance results with RESIST-0 are depicted in Fig. 9(a). Note that even if sinkhole attacks are avoided, malicious nodes can still perform the selective forwarding attack. Fig. 9(a) clearly shows that *FTree* and *RRobin* outperform *RWalk*. This is expected as in *RWalk*, the average path length that each message travels to the sink is longer. This consequently

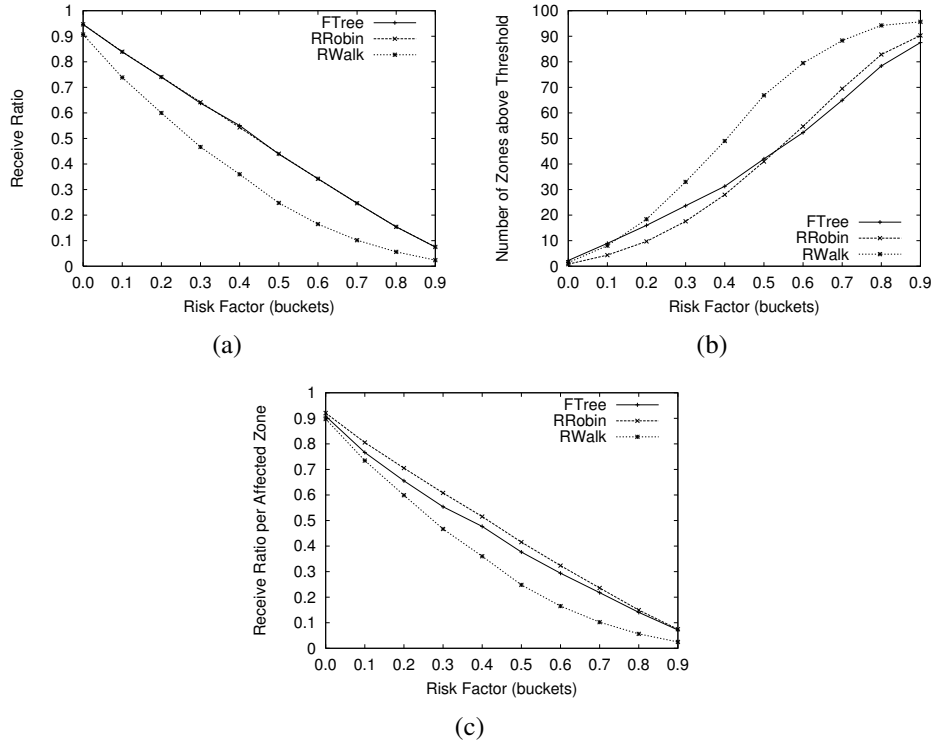


Figure 9: RESIST-0 performance evaluation: (a) Comparative performance of routing protocols. (b) Number of severely affected zones in the network (threshold =60%). (c) Received ratio of affected zones in the network.

increases the probability of meeting a malicious node on the path. Further experimentation on *RWalk* also showed that the protocol performance is inversely proportional to n. This actually means that the best case for n-hop random walk is achieved when $n = 0$, in which case *RWalk* is equivalent to *FTree* routing.

Fig. 9(a) also shows that *FTree* and *RRobin* have similar performance. This is surprising since, intuitively, the performance of *RRobin* should be better compared to *FTree*. Analyzing the results, we observe that, as expected, for sensors, which have malicious parents, *RRobin* improves the performance by letting these nodes periodically send to alternative parents. However, this does not necessarily

improve overall performance as *the reverse case* also holds: sensor nodes with good parents switch to using malicious nodes as parents in a round robin fashion. Consequently, any gain from *RRobin* is neutralized by putting sensor nodes with good parents at risk.

To better understand the effect of malicious nodes on the protocol behavior, we divide the network into 100 equal zones and define the *failure threshold of a zone* as the percentage of data missing from the zone to qualify the zone as poorly monitored. In reality, this threshold would depend on the criticality of the sensor network application. We set the failure threshold as 60% in our experiments. Fig. 9(b) illustrates how many zones fell above the failure threshold for each risk factor bucket and routing protocol. Initially, the number of zones above the failure threshold is higher for *FTree* than *RRobin*. Noticing that both protocols share the same receive ratio (see Fig. 9(c)), this means that *RRobin* just diffuses the effect of malicious nodes to more zones, so that fewer zones actually fail. However, as the risk factor increases, the number of zones above threshold increases beyond *FTree* due to *the reverse case* appearing more often. Increasing the failure threshold moves the shift point to the right. Nevertheless, although the number of affected zones is higher for *RRobin*, the average received data ratio per affected zone still remains higher than *FTree*. On the other hand, the number of affected zones in *RWalk* is always the highest due to its overall poor performance.

Performance in RESIST-1 and vulnerable cases. To better understand the performance of RESIST-1 and vulnerable cases, we slightly modified the computation of the risk factor presented in Section 3. The main reason for this modification is to represent the different adversarial power of compromised nodes in RESIST-1 and vulnerable cases. Note that the only difference between these two cases is

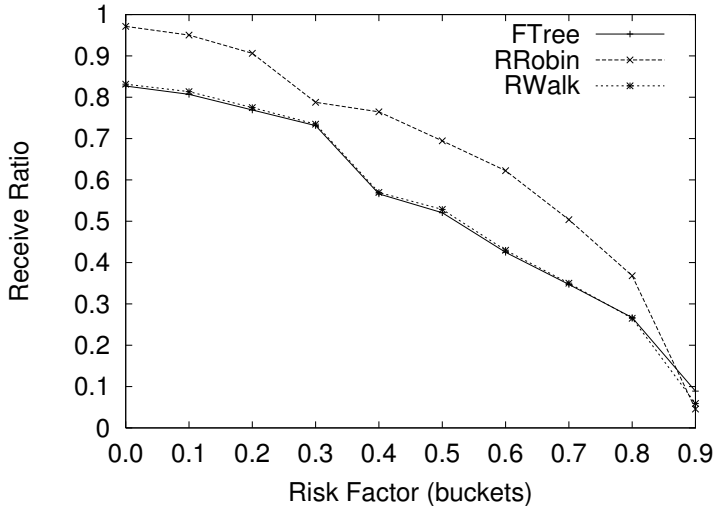


Figure 10: Comparative performance of routing protocols without any RESIST protocols (using *Vulnerable Risk Factor*).

the advertised *distance to the sink*, and hence, the only change in the computation is the way initial distances are calculated for each sensor. In the RESIST-1 case, since a malicious node can lie by one hop, its distance is equal to that of its neighbor with the smallest distance to the sink. For the vulnerable case, malicious nodes can pretend to be the sink, and so, the distance of each malicious node is 0. Hence, the shortest path algorithm needs to be run once for each sink, real and pretend. At the end, each node is assigned the shortest distance to one of the sinks. Hereafter, each version of the risk factor is referred as *RESIST-1 Risk Factor* and *Vulnerable Risk Factor*, respectively.

Figs. 10 and 11 show the performance of *FTree*, *RRobin* and *RWalk* ($n = 1$) under the vulnerable and RESIST-1 cases, respectively. For each graph, we partitioned topologies based on their respective risk factors (i.e., *RESIST-1* and *Vulnerable Risk Factor*). Quite different than the RESIST-0 results (see Fig. 9(a)), *RRobin* performs the best for the vulnerable case (see Fig. 10). This is because,

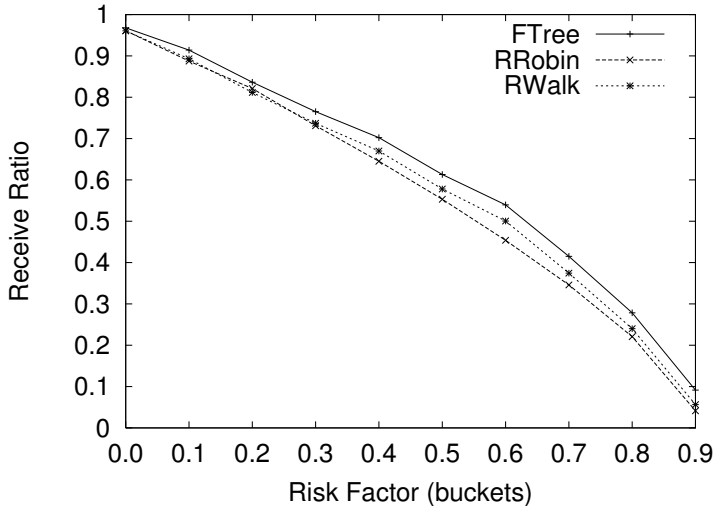


Figure 11: Comparative performance of routing protocols under the RESIST-1 protocol (using *RESIST-1 Risk Factor*).

in this case, the *FTree* algorithm outputs a forest of small routing trees, where the real sink and each malicious node is the root of one of these trees. Obviously, only the nodes that belong to the tree of the real sink can deliver data. In contrast, since *RRobin* allows nodes to follow different routes to the sink, it is able to reduce the effect of these sinkhole attacks. Note that *the reverse case* of *RRobin* (i.e., nodes with good parents using malicious nodes as alternative parents) does still exist. However, in the vulnerable case, the effect of fragmenting the network into several trees with *FTree* is greater than *the reverse case* of *RRobin*. Such fragmentation also occurs in *RWalk*, which explains why its performance is lower than *RRobin* as well. Note that, in *RWalk*, the routing tree is built in the same way as in *FTree*.

Interestingly, *RRobin* cannot sustain the same performance in the RESIST-1 case. In the vulnerable case, the performance of *FTree* and *RWalk* is highly affected by the fragmentation of the network into disconnected trees. Hence, *RRobin* is able to perform better. However, note that, in *RRobin*, if a node re-

ceives the first `Hello` message from a malicious node, then other neighbors may not be able to join the set of alternative parents, if they advertise longer distances. Hence, the set of alternative parents becomes very small and most often, consists of one malicious parent (or one of its descendants). This problem, although it appears in the vulnerable case too, is more obviously seen in RESIST-1 case, since *FTree* and *RWalk* can perform better in this case. Hence, in the RESIST-1 case, all protocols perform comparably, with *FTree* performing slightly better (see Fig. 11).

5.3. Evaluation of RESIST protocols with colluding attacks

To be able to implement sinkhole attacks in the presence of RESIST protocols, malicious sensors have to be designed to collaborate and share good tokens (i.e., a token that can prove a short distance to the sink). More specifically, two malicious nodes need to collude through a private communication channel in order to send tokens to each other. Consider the example of the sensor network in Fig. 12. Node C1, which is close to the sink, helps node C2, which is at the border of the network. Thanks to C1, C2 can attract traffic from nodes Z and possibly Y. However, since C1 is already very close to the sink, it would have captured these messages anyway: so, collusion in this case does not increase the power of an attack much.

Collusion is also limited by the communication capabilities of malicious sensors: in Fig. 13, we simulated the impact of collusion when colluding sensors have normal-range radios and are distributed randomly on the network area. In our simulations, malicious nodes exchange tokens so that they all appear at the same distance from the sink (i.e., the distance of the malicious node that is closest to the sink). We only plot RESIST-0 performance with collusion as this is the

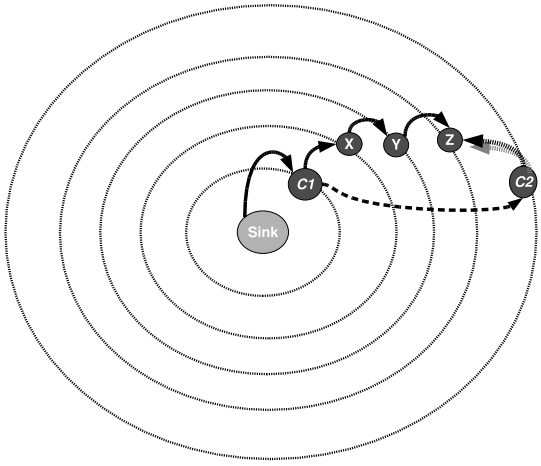


Figure 12: A compromised sensor C1 close to the sink helps another sensor C2 to launch a sinkhole attack. However, C1 has already enough power of disruption, and hence, C2 the increase the power of the attack marginally.

more interesting case. In RESIST-0, the sharing of tokens enables replying challenges for shorter distances and hence, has an effect on performance (see Curves I and II when compared to Curve VI in Fig. 13). Fig. 13–Curve V also shows the default malicious behavior scenario (i.e., without colluders) for RESIST-1 for reference. Note that in the case of RESIST-1, a malicious node close to the sink (e.g. at level k) will manage to attach a very large subtree by just lying by one hop (e.g., all the nodes within his neighborhood at level $k+1$). Although these neighbors do not explicitly collude, the pure existence of them allows non-malicious nodes to think they are closer to the sink: the more there are malicious nodes on the path, the closer the nodes think they are to the sink, hence, significantly affecting the way the tree is built.

The most powerful attack would be a malicious sensor, which is close to the sink and has a long-range radio, propagating tokens to malicious sensors far from the sink (see Fig. 13–Curves III-IV). For example, the Curve III in Fig. 13 assumes

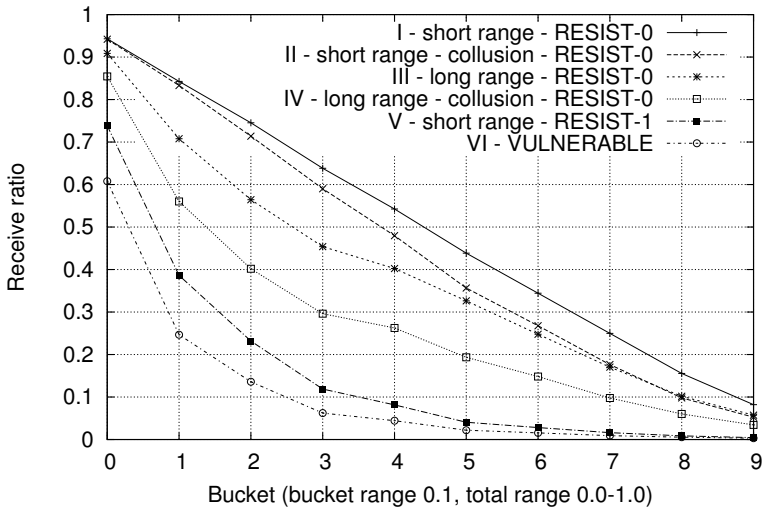


Figure 13: Detailed comparison of collusion implications within RESIST protocols and different radio ranges. In the presence of short range collusion, RESIST-0 still performs much better than vulnerable routing without collusion. For RESIST-0, Curves I and II depict short range malicious behavior with and without collusion respectively (*no collusion* means that malicious nodes communicate on a certain range but do not lie about their position). Curves III and IV display the effect of colluders for long ranges. Curve V shows RESIST-1 short range malicious impact. Finally, Curve VI represents simulations obtained for vulnerable routing at short range malicious behavior without colluders.

that malicious nodes shares tokens with all the malicious nodes in the network, but do not lie about their positions. Again, this case might not cause a significantly higher degradation in performance, as the dominant impact already comes from the malicious node closest to the sink. In fact, RESIST-0 achieves far better performance than the vulnerable case, even in the worst case, where malicious nodes have long-range radios and collude (see Curve IV when compared to Curve VI in Fig. 13). For the scenario shown in Fig. 13 Curve IV, a malicious node at level k will not be able to attract uncompromised nodes at level $k+1$, as it is impossible to lie to them. However, by passing the token to a colluder farther away, a malicious node enables its colluder to respond to challenges. Nevertheless, note that the lie about hop count starts diffusing from the point the malicious node reaches to another malicious node, while with RESIST-1 lies can diffuse much earlier, from the point the first malicious node at k lies.

In summary, if a node C_1 helps a node C_2, with $d(C_1) < d(C_2)$, to perform a sinkhole attack:

- With RESIST-1, two sinkhole attacks are performed: $SA(C_1, d(C_1)-1, d(C_1))$ and $SA(C_2, d(C_1) - 1, d(C_2))$
- With RESIST-0, one sinkhole attack is performed $SA(C_2, d(C_1), d(C_2))$ while the second attack is a pure selective-forwarding one $SA(C_1, d(C_1), d(C_1))$

In both cases, the attack by C_1, which is closer to the sink, would probably be more efficient than the one by C_2 and hence, the benefits from collusion would not justify the cost of implementing it.

	Static	Message	Dynamic	Reconfiguration Cost per Sensor
RESIST-1	64B	$45B \times R$	2KB	V
RESIST-0	64B	$85B \times R$	2KB	$3 \times V + S$

Figure 14: A summary of memory consumption, signature operations, and computational/communication complexity for relaying sensors. Note that sink operations are managed by the base station. Legend: S: Signature generation; V: Signature verification; H: Hash computation; R: Distance hops from the sink; k: Current distance to the sink; EC: ECC point multiplication cost.

	MICAz	TelosB
$S_t(S_p)$	0.89 s (26.96 mWs)	0.52 s (6.26 mWs)
$V_t(V_p)$	1.77 s (53.42 mWs)	1.02 s (12.41 mWs)
$H_t(H_p)$	3636 μs (5.9 μWs/byte)	7272 μs (5.9 μWs/byte)
$EC_t(EC_p)$	1.24 s (26.10 mWs)	1.44 s (6.00 mWs)

Figure 15: Current benchmarks of time and energy (in brackets) estimations for ECDSA–160 signature generation S, verification V, SHA–1 computation H and EC–160 point multiplication operations.

5.4. Analysis of Time and Power Consumption in RESIST

In this section, we briefly discuss the feasibility of implementing the RESIST protocols in WSN platforms. We focus mainly on the time and power needed to perform cryptographic operations. To estimate the time and power, we used the results presented in [19, 20, 21, 22, 23, 24]. For the sake of clarity, Fig. 14–column 4 gathers the cost of specific cryptographic operations for each RESIST version. In our analysis, we assume ECDSA–160 signature generation S, verification V, SHA–1 computation H and EC–160 point multiplication operations. The reasons for these choices are detailed in the Appendix, where we also present additional optimizations to RESIST algorithms to further reduce their costs.

Using power consumption estimations on a MICAz and the more powerful TelosB in [20, 21, 23], we lay out some interesting results regarding the cryptographic operations (see Fig. 15). In particular, ECDSA–160 takes only 0.52 s

(6.26 mWs) and 1.02 s (12.41 mWs) on a TelosB, for the signature generation and the verification operations respectively [24]. Moreover, on MICAz, the same operations consume 0.89 s (26.96 mWs) and 1.77 s (53.42 mWs) respectively. For EC point multiplication time and energy consumption, we used the results from [20]. It takes, on average, 1.24 seconds for a MicaZ sensor mote to compute a fixed point multiplication and 1.44s on TelosB. The power consumed by the sensor nodes to perform ECC-160 is around 26.10mWs on MICAz and around 6.00mWs on TelosB [22]. Furthermore, according to the analysis and further comparison of data sheets for TelosB and MicaZ [20], we approximated the energy consumption of the computation of an SHA-1 hash value to be 5.9μWs/byte. In addition, SHA-1 computation takes 3636μs/byte on a MICAz sensor mote, and an estimated time of 7272μs/byte on TelosB mote. Given the resilience achieved against sinkhole and selective forwarding attacks, we believe it is cost-effective to implement RESIST protocols in WSNs.

6. Related work

Security in wireless networks is attracting the attention of many researchers since it is vital to guarantee correct operation of sensor protocols. This paper focuses particularly on sink-hole and selective forwarding attacks. Most other approaches against these attacks revolves around detection of malicious nodes [12, 11, 15, 25]. For instance, in secure AODV (SAODV) [25], the route discovery mechanism of the AODV routing protocol is protected by signing messages. More specifically, a key management scheme is used where each node is assigned to an asymmetric signature key pair and non-mutable fields of the messages are signed with such digital signatures of nodes. Authentication can be then performed in a point-to-point manner: Any neighbor receiving a routing message

can securely verify the association between the address of a given ad hoc node and the public key of that node and detect if a node is behaving maliciously. In [12], multi-hop acknowledgments are used to detect and blacklist nodes that perform selective forwarding attacks. However, in addition to its cost, the proposed scheme requires geographical location information and strict synchronization. In [10, 11], a learning technique based on neural networks is used to predict the sensor measurements, and a reputation scheme is used to mark nodes as faulty if their reports are too different from predictions. In [15], a protocol similar to RESIST-1 is proposed, but without strong cryptography. As a consequence, it requires a protocol to detect malicious sensors (reports are vulnerable to falsification) and to blacklist nodes (through a complex messaging mechanism).

Most practical approaches for establishing secure channels among sensor in the literature are based on symmetric cryptography, where pairwise keys between every two neighbor nodes are established after network deployment [26, 27, 28]. Such approaches provide data authenticity and/or confidentiality in a hop-by-hop manner. However, they rely on uniform wireless communication patterns in WSNs and consequently, are vulnerable to attacks when this assumption does not hold.

The LEDS approach [29] (i.e., Location-aware End-to-end Data Security) was proposed to deal with such constrains on communication patterns. It uses symmetric secret keys for secure and reliable data delivery and integrates two building blocks: a location-aware key management framework and an end-to-end data security mechanism. The first block considers the construction of a virtual geographic grid and the binding of each node's location (i.e., cell in the grid) into symmetric secret keys owned by that node. The second block guarantees that ev-

ery report in LEDS is endorsed by multiple sensor nodes and is encrypted with a unique secret key shared between the event sensing nodes and the sink. Although a very interesting and robust approach, LEDS requires the deployment of a grid-based localization scheme as well as assumes densely deployed networks, where every event of interest can be detected by multiple sensor nodes.

Similar to LEDS, a few other approaches have been proposed to design key management schemes based on collaborative endorsement: reports that are not properly endorsed are filtered out by intermediate nodes en-route to the sink or by the sink itself [30, 31]. Such schemes are complementary to our proposed RESIST schemes. RESIST increases resilience during routing topology construction (i.e., before data transmission takes place) and can be combined with endorsement schemes to improve resilience during data communication.

An interesting analysis of DDoS attacks in sensor networks, which also takes into account different network parameters and some counter measures, is presented in [32]. While their work covers TCP JellyFish and selective-forwarding attacks, we focus on sinkhole attacks. Moreover, our study with the Risk Factor metric captures more network characteristics. An intuitive approach against selective forwarding attacks is to use multipath routing [8, 33]. However, such a protocol dramatically increases communication overhead as the redundancy of paths increases. In addition, these paths eventually converge to a few nodes surrounding the base station where malicious nodes can have a high impact. Indeed, our simulation results show that the efficiency of this approach is limited, as confirmed by [32].

Trust-based systems [34, 35, 36] are interesting approaches to deal with selective forwarding attacks. In these systems, interactions between sensors are

used for trust level computation. Such systems are, however, often complex. We believe resilience, as provided by our protocols, is a better choice. As in [37], RESIST can also use trust levels to choose the set of alternate parents in *RRobin* routing.

In this paper, we did not focus on counter-measures against wormhole attacks as in [38, 39]. In wormhole attacks, a compromised node records control and data traffic at one location and tunnels it to a colluding nodes, which replays it locally in another part of the network in a timely manner. Wormhole attacks may increase the severity of a sinkhole attack [40] as these attacks can prevent nodes from discovering routes that are more than two hops away. A number of routing protocols have been proposed to protect the network traffic against such attacks. In [38], authors introduce the LITEWORP solution, particularly designed for the detection of the wormhole and the isolation of the compromised nodes. Other proposals make use of cryptographic techniques and applies trust-based schemes [40], neighbor list reconstruction procedures [41], special hardware like a directional antenna and precisely synchronized clocks [42], and also distance bounding protocols (commonly used in RFID systems) [43].

Finally, the use of Public Key Infrastructures in WSNs, and exhaustive comparisons between Elliptic Curves Cryptography (ECC) and RSA on 8-Bit CPUs have been subject of extensive research [19, 21, 44, 45]. Recently, [45] showed that ECC can be implemented at a very low cost in WSN and RFID networks [46]. ECC keys are known to be much smaller than equivalent RSA keys [47], so that signatures and keys shorter than 110 bits would be largely sufficient in most contexts.

7. Conclusion

In this paper, we analyzed two protocols that increase the resilience of the network in the presence of sink-hole attacks: RESIST-1 prevents malicious nodes from lying about their advertised distance to the sink more than one hop; and RESIST-0, which although is more costly, completely stops malicious nodes from lying about their distance. Our performance evaluation confirmed the higher resilience of our protocols, even in the presence of some collusion. Moreover, we introduced a new metric, the Risk Factor, to measure the impact of selective forwarding and sinkhole attacks on sensor networks. We showed that it successfully captures different topology-based parameters, such as the position and number of malicious nodes, the network scale, and attacker capability. The initial study of the overhead of the cryptographic operations of RESIST protocols shows the feasibility of implementing these protocols in WSN platforms. As future work, we plan to perform an experimental evaluation of our schemes using TinyECC Library.

References

[1] C. Intanagonwiwat, R. Govindan, D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in: ACM MOBICOM, Boston, MA, 2000, pp. 56–67.

[2] F. Ye, A. Chen, S. Lu, L. Zhang, Gradient broadcast: A robust, long-live large sensor network, Tech. rep., UCLA (2001).

[3] F. Ye, A. Chen, S. Lu, L. Zhang, A scalable solution to minimum cost forwarding in large sensor networks, in: Conf. on Computer Communications and Networks, Arizona, USA, 2001.

[4] U. Cetintemel, A. Flinders, Y. Sun, Power-efficient data dissemination in wireless sensor networks, in: ACM MobiDE, San Diego, CA, USA, 2003.

[5] B. Krishnamachari, D. Estrin, S. Wicker, The impact of data aggregation in wireless sensor networks, in: IEEE ICDCS, Washington, DC, USA, 2002, pp. 575–578.

[6] Y. J. Zhao, R. Govindan, D. Estrin, Residual energy scans for monitoring wireless sensor networks, in: IEEE WCNC, 2002.

[7] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong, TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks, ACM OSR (2002) 131–146.

[8] C. Karlof, D. Wagner, Secure routing in wireless sensor networks: Attacks and countermeasures, in: IEEE Workshop on Sensor Network Protocols and Applications (SNPA), Anchorage, Alaska, USA, 2003.

[9] W. R. P. Junior, T. H. de P. Figueiredo, H. C. Wong, Malicious node detection in wireless sensor networks, in: IEEE IPDPS, Miami, Florida, USA, 2004.

[10] P. Mukherjee, S. Sen, Detecting malicious sensor nodes from learned data patterns, in: Agent Technology for Sensor Networks, Honolulu, Hawaii, 2007.

[11] P. Mukherjee, S. Sen, Using learned data patterns to detect malicious nodes in sensor networks, in: ICDCN, Kolkata, India, 2008.

[12] B. Yu, B. Xao, Detecting selective forwarding attacks in wireless sensor networks, in: IEEE IPDPS, Rhodes Island, Greece, 2006.

[13] L. B. Oliveira, D. Aranha, E. Morais, F. Daguano, J. Lpez, R. Dahab, Identity-based encryption for sensor networks, in: IEEE PERCOM, 2007.

[14] Y.-C. Hu, A. Perrig, D. B. Johnson, Packet leashes: a defense against wormhole attacks in wireless networks, in: Procs. of the 22nd Annual Joint Conference of the IEEE Computer and Communications, 2003, pp. 1976–1986.

[15] S.-B. Lee, Y.-H. Choi, A secure alternate path routing in sensor networks, Computer Communications, Elsevier 30 (2006) 153–165.

[16] P. Gupta, P. Kumar, Critical power for asymptotic connectivity in wireless networks, in: Stochastic Analysis, Control, Optimization and Applications, Birkhauser, Boston, 1998, pp. 547–566.

[17] Z. Bar-Yossef, R. Friedman, G. Kliot, RaWMS - random walk based lightweight membership service for wireless ad hoc networks, ACM Transactions on Computer Systems (TOCS) 26 (2) (2008) 1–66.

[18] F. Le Fessant, C. Sengul, A.-M. Kermarrec, Pace-maker: Tracking peer availability in large networks, Tech. Rep. RR-6594, INRIA (2008).

[19] N. Gura, A. Patel, A. Wander, H. Eberle, S. C. Shantz, Comparing elliptic curve cryptography and rsa on 8-bit cpus, in: Procs. of the Cryptographic Hardware and Embedded Systems, Springer, 2004, pp. 119–132.

[20] A. S. Wander, N. Gura, H. Eberle, V. Gupta, S. C. Shantz, Energy analysis of public-key cryptography for wireless sensor networks, in: IEEE Int. Conf. on Pervasive Computing and Communications, 2005, pp. 324–328.

[21] K. Piotrowski, P. Langendoerfer, S. Peter, How public key cryptography influences wireless sensor node lifetime, in: Procs. of the 4th ACM workshop on Security of ad hoc and sensor networks, 2006, pp. 169–176.

[22] H. Wang, Q. Li, Efficient implementation of public key cryptosystems on mote sensors, in: Procs. of the 8th Int. Conf. on Information and Communications Security, 2006, pp. 519–528.

[23] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, I. Verbauwhede, Low-cost elliptic curve cryptography for wireless sensor networks, in: Procs. of the 3rd European workshop on security and privacy in ad hoc and sensor networks (ESAS), 2006, pp. 6–17.

[24] G. de Meulenaer, F. Gosset, F.-X. Standaert, O. Pereira, On the energy cost of communication and cryptography in wireless sensor networks, in: IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communication, 2008, pp. 580–585.

[25] M. G. Zapata, Secure ad hoc on-demand distance vector routing, ACM Mobile Computing and Communications Review 6 (2002) 108–114.

[26] K. Ren, K. Zeng, W. Lou, A new approach for random key predistribution in large scale wireless sensor networks, to appear in Journal of Wireless Communication and Mobile Computing.

[27] W. Du, J. Deng, Y. Han, S. Chen, P. Varshney, A key management scheme for wireless sensor networks using deployment knowledge, in: IEEE Infocom, 2004.

[28] S. Zhu, S. Xu, S. Setia, S. Jajodia, Establishing pair-wise keys for secure communication in ad hoc networks: A probabilistic approach, in: IEEE ICNP, 2003.

[29] K. Ren, W. Lou, Y. Zhang, Leds: Providing location-aware end-to-end data security in wireless sensor networks, IEEE Transactions on Mobile Computing (TMC) 7 (5) (2008) 585–598.

[30] Y. Zhang, W. Liu, W. Lou, , Y. Fang, Location based security mechanisms in wireless sensor networks, to appear in IEEE JSAC, Special Issue on Security in Wireless Ad Hoc Networks.

[31] H. Yang, F. Ye, Y. Yuan, S. Lu, W. Arbaugh, Toward resilient security in wireless sensor networks, in: ACM MOBIHOC, 2005.

[32] I. Aad, J.-P. Hubaux, E. W. Knightly, Impact of denial of service attacks on ad hoc networks, IEEE/ACM Transaction on Networking.

[33] S. M. Jing Deng, Richard Han, Insens: Intrusion-tolerant routing in wireless sensor networks, in: IEEE ICDCS, Providence, Rhodes Island, USA, 2003.

[34] D. H. McKnight, A. E. Kamal, The meanings of trust, Tech. rep., Univ. Minnesota (1996).

[35] S. Marti, T. Giuli, K. Lai, M. Baker, Mitigating routing misbehavior in mobile ad hoc networks, in: ACM MOBICOM, Boston, MA, 2000, pp. 255–265.

[36] Y. Sun, Z. Han, W. Yu, K. J. R. Liu, A trust evaluation framework in dis-

tributed networks: Vulnerability analysis and defense against attacks, in: IEEE INFOCOM, Barcelona, Spain, 2007.

[37] Z. Liu, A. Joy, R. Thompson, A dynamic trust model for mobile ad hoc networks, in: Future Trends of Dist. Computing Systems, Suzhou, China, 2004.

[38] I. Khalil, S. Bagchi, N. B. Shroff, LITEWORP: a lightweight countermeasure for the wormhole attack in multihop wireless networks, in: Multihop Wireless Network in the International Conference on Dependable Systems and Networks (DSN.

[39] R. Poovendran, L. Lazos, A graph theoretic framework for preventing the wormhole attack in wireless ad hoc networks, Wireless Network, Springer 13 (2007) 108–114.

[40] A. Pirzada, C. McDonald, Circumventing sinkholes and wormholes in wireless sensor networks, in: Proceedings of the International Conference on Wireless Ad Hoc Networks, 2005.

[41] S. Madria, J. Yin, Serwa: A secure routing protocol against wormhole attacks in sensor networks, Ad Hoc Network 7 (2009) 1051–1063.

[42] R. Poovendran, L. Lazos, A graph theoretic framework for preventing the wormhole attack in wireless ad hoc networks, Wireless Network 13 (2007) 27–59.

[43] S. Čapkun, L. Buttyán, J.-P. Hubaux, Sector: secure tracking of node encounters in multi-hop wireless networks, in: ACM workshop on Security of ad hoc and sensor networks, 2003, pp. 21–32.

[44] R. Roman, C. Alcaraz, Applicability of public key infrastructures in wireless sensor networks, in: Procs. of the 4th EuroPKI workshop, Springer, 2007, pp. 313–320.

[45] D. J. Malan, M. Welsh, M. D. Smith, Implementing public-key infrastructure for sensor networks, ACM Transaction on Sensor Networks 4 (4) (2008) 1–23. doi:http://doi.acm.org/10.1145/1387663.1387668.

[46] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, I. Verbauwhede, Public-key cryptography for rfid-tags, in: Procs. of the IEEE Int. Conf. on Pervasive Computing and Communications Workshops, 2007, pp. 217–222.

[47] A. K. Lenstra, E. R. Verheul, Selecting cryptographic key sizes, in: PKC, Springer-Verlag, London, UK, 2000, pp. 446–465.

[48] J. Lopez, J. Zhou, Wireless Sensor Network Security, IOS Press, 2008, Ch. L. Batina et al. Chapter Public–key Primitives.

[49] R. Roman, C. Alcaraz, J. Lopez, A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes, Mob. Netw. Appl. 12 (4) (2007) 231–244.

[50] Y. W. Law, J. Doumen, P. Hartel, Survey and benchmark of block ciphers for wireless sensor networks, ACM Transaction on Sensor Networks. 2 (1) (2006) 65–93.

[51] N. Fournel, M. Minier, S. Ubéda, Survey and Benchmark of Stream Ciphers for Wireless Sensor Networks, in: Procs. of the Workshop on Information Security Theory and Pratice Information Security Theory and Practices.

Smart Cards, Mobile and Ubiquitous Computing Systems Lecture Notes in Computer Science, Vol. 4462, Springer Berlin / Heidelberg, 2007, pp. 202–214.

[52] A. Francillon, C. Castelluccia, Tinyrng: A cryptographic random number generator for wireless sensors network nodes, in: Procs. of the 5th Int. Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops, 2007, pp. 1–7.

[53] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, D. Culler, Spins: Security protocols for sensor networks, Wireless Networks 8 (2001) 521–534.

A. Implementation Requirements of RESIST Protocols

In this section, we provide an estimation of the cost of using cryptographic primitives in sensors networks, as these primitives are the main barrier to the implementation of RESIST protocols. In particular, we evaluate code size, computational cost, and communication overhead of our security protocols.

A.1. Encryption Parameters

Hardware limitations of sensors determine the feasibility of security solutions. Hence, there has been extensive research on investigating the usability of cryptographic algorithms in wireless sensor networks. These efforts provide tests performed on dedicated platforms in order to establish a ranking of candidates and benchmarking for common cryptosystems according to the energy-efficiency and small-storage requirements of sensors [48, 22, 49, 21]. These performance evaluations typically use the energy models of [22]:

- Mote2: 16-bit microcontroller with a clock frequency of 12 MHz, a Flash of 512 KB, and a RAM of 64 KB
- MICAz: based on the low-power 8-bit microcontroller ATMEGA128L running at 7.37 MHz with 128KB Flash + 4KB RAM
- TelosB: 16-bit MSP430 microcontroller which runs only at 4 MHz and offers 1 MB + 10 KB of memory

Taking into account the resource constraints of such hardware, first, we evaluate the memory requirements of our protocols. Our calculations presented in the rest of the section show that the code, parameters and variables used by cryptographic algorithms should fit the small memory of sensors. Fig. 16 depicts the main features (memory requirements for code, length of keys and outputs) for some well-known cryptographic algorithms [49, 21]. We include most of the common public key cryptosystems (PKC) like RSA, Elliptic Curve (EC) and Ntru for asymmetric operations like signatures; several symmetric algorithms such as some well–known stream ciphers (RC4, Snowv2 and Phelix) and block ciphers (the standard AES, RC5 and Skipjack), and hash functions.

RSA (1024 bits–key) signatures occupy 1024 bits, while the same operation using EC (ECDSA) with 160 bits–key only generates a block of 320 bits. For this reason, we will assume that ECDSA is used in RESIST protocols. Consequently, we need 20KB of static memory for the code of cryptographic primitives. However, we do not discard the option of using other encryption schemes with different overheads that might provide increasing levels of protection vs. efficiency [50, 51]. For example, the cost of symmetric encryption is negligible compared to elliptic curve operations.

	Type (/rounds)	FLASH (bytes)	Key(/IV) (bits)	Block (bits)
RSA-1024	PKC	15832	1024	1024 bits-signature
ECDSA-160	PKC	18800	160	320 bits-signature
NTruSign	PKC	2214		1169 bits-signature
AES	Block/10	4354	128	128
RC5	Block/18	1110	128	64
Skipjack	Block/32	856(CTR)	80	64
RC4	Stream	6064	128/0	8
Snow-v2	Stream	11152	128/128	32
Phelix	Stream	9968	256/128	32
SHA1	1–Way Hash	30000	$<$264	160 bit–digest

Figure 16: Code and encryption parameters for several cryptographic algorithms from the literature, especially from [49, 21]

To generate the $epoch$ values, the RESIST protocols rely on a continuous counter at the base station. This can be implemented using a lightweight pseudo-random number generator (PRNG) using just logical operations. For instance, TinyRNG approach presented in [52] only uses 10KB of static memory consumption and 416 bytes of RAM on a MICAz. Moreover, this method eliminates the need for tight network-wide synchronization, which is typically hard to achieve.

A.2. Estimation of Messages Sizes

The main cost of our RESIST protocols is the size of `Hello` messages, which carry multiple tokens that contain cryptographic values. This cost can be significantly decreased by an efficient way of encoding tokens. For instance, in [15], one-way hash functions are used so that token T_{k+1} can be computed from token T_k by applying a hash-function. The construction of the hash chain (HC) is performed by the base station, who first chooses T_1 and then computes the last token at length R, as follows: $H^R(T_1) = H(H(H(\ldots H(T_1)\ldots)))$ (R times).

Using this method in RESIST-1, `Hello` messages need only to contain the first token (i.e., $H^k(T_1)$) and the last token (i.e., $H^R(T_1)$), signed by the sink , whereas

the "basic" approach would send all the tokens:

Basic RESIST-1:
$\mathcal{S}ink \rightarrow n_i \subset N_{sink}: \quad Hello(epoch, [\mathbf{T_k}, T_{k+1}, ..., T_R])$
Optimized RESIST-1:
$\mathcal{S}ink \rightarrow n_i \subset N_{sink}: \quad Hello(H^k(T_1), epoch, H^R(T_1), \{epoch, H^R(T_1)\}_{K_{pri}^{sink}})$

On receiving this message, a sensor can compute its hop-count distance k by hashing $R-k$ times the token $H^k(T_1)$ until it reaches $H^R(T_1)$. Moreover, as it does not know T_1, and $H(\cdot)$ is a one-way function, it cannot compute $H^{k-1}(T_1)$ and thus, it is not able to lie more than one hop (i.e., as in RESIST-1).

Reconfiguration in RESIST-0 implies additional concerns since the sink generates and transmits new key pairs for each token. Nevertheless, we can apply again the idea of hash chains for reducing the cost of cryptography in RESIST-0: `Hello` messages need only to propagate the first (i.e., $H^k(T_1)$) and the last token (i.e., $H^R(T_1)$), signed by the sink, but these tokens should now be interpreted as public keys. Hence, the basic and optimized RESIST-0 can be formulated as:

Basic RESIST-0:
$\mathcal{S}ink \rightarrow n_i \subset N_{sink}: \quad Hello(epoch, [\mathbf{T_k}, T_{k+1}, ..., T_R])$
Optimized RESIST-0:
$\mathcal{S}ink \rightarrow n_i \subset N_{sink}: \quad Hello(K_{pub}^k, epoch, K_{pub}^R, \{epoch, K_{pub}^R\}_{K_{pri}^{sink}})$

Thus, `Hello` messages just contain the public key generated at level $k-1$. Nodes at level k can use that public key K_{pub}^k to verify replies to challenges based on K_{pri}^k sent by level $k-1$. They also use it to generate the next hop private key (implemented using ECC) K_{pri}^{k+1} (with the cost of a modulo operation), from which the public key K_{pub}^{k+1} is then generated (with the cost of an exponentiation operation). That later public key is sent to level $k+1$ to be used to verify replies sent by level k with the generated private key. Note that the private key at level k is not computed by level $k-1$, but, since nodes are supposed to verify the complete

chain, they should compute all the key–pairs for all higher levels until R (and then discard them). Though we do not present the cost of the above mentioned key generation, note that ECC key generation only involves generating a random number, which becomes the user's private key, and executing an ECDH operation to compute the corresponding public key. On the contrary, RSA key generation is much more time consuming as it requires the generation of large prime numbers. For details, we refer the reader to [20].

On the other hand and as previously mentioned, signature overhead can be alleviated using symmetric algorithms (e.g., as the standard AES, Skipjack or RC4). For instance, Skipjack has been demonstrated to be a powerful candidate of block ciphers suitable for WSNs [50]. The periodic key disclosure can be efficiently tackled using the approach presented in [53]. The use of these schemes, however, would require significant changes in RESIST and we do not further elaborate on this optimization in this paper.

In summary, we have identified here two optimizations concerning the length of `Hello` messages, and we will discuss their transmission and computation cost, in terms of time and delay, further, in the following sections. We refer to the new approaches in the following as:

1. **Basic RESIST-1:** `Hello` messages are lists of simple tokens.
2. **Optimized RESIST-1:** Using the idea of hash chains, all `Hello` messages contain the first and the last token, signed by the sink.
3. **Basic RESIST-0:** `Hello` messages are lists of tokens containing public–private keys.
4. **Optimized RESIST-0:** As Optimized RESIST-1, `Hello` messages are implemented by hash chains, where the hash function is a two-step function

generating both the private and the public key.

A.3. Estimation of Memory Usage

Regarding storage, RESIST protocols do not have special memory constraints. For the four previously described versions of RESIST, we first overview the common attributes that sensors must statically keep in memory:

- Public key of the sink, K_{pub}^{sink} using ECDSA-160 algorithm: 160-bit key (20 bytes).
- Sensor public and private keys: $(K_{pub}^{n_i}, K_{pri}^{n_i})$ (40 bytes). We assume that the keys of the sink are known by the base station, so that instead of the sink, the base station can do the most expensive computations, such as generating new keys or signing data.
- Sensor identifier, ID^{n_i}: at least 4 bytes. Sensors identifiers should be linked with their public key, so that malicious nodes cannot easily forge new identities. For example, the identifier could be a hash of the sensor public key, or a long subpart of it.

In this way, the total static memory needed for our protocols occupies only 176 bits (i.e. 64B in Fig. 17), along to 20KB for ECDSA algorithm code (cryptographic algorithm and variables) and the 4KB OS code space.

On the other hand, the dynamic memory consumption of each version of the RESIST protocol depends on the specification and the cryptographic operations executed. In particular, we study the size of the list of tokens at relaying nodes (see Fig.17 for a summary), as follows:

	Static	Message	Dynamic	Reconfiguration Cost per Sensor
Basic RESIST-1	64B	$45B \times R$	2KB	V
Optimized RESIST-1	64B	84B	3KB	$V + (R - k) \times H$
Basic RESIST-0	64B	$85B \times R$	2KB	$3 \times V + S$
Optimized RESIST-0	64B	84B	2KB	$(R - k + 1) \times S + 2 \times V$

Figure 17: A summary of memory consumption, signature operations, and computational/communication complexity for relaying sensors. Note that sink operations are managed by the base station. Legend: S: Signature generation; V: Signature verification; H: Hash computation; R: Distance hops from the sink; k: Current distance to the sink; EC: ECC point multiplication cost.

- **Basic RESIST-1**, where $T_k = \langle k, epoch, \{\langle k, epoch\rangle\}_{K_{pri}^{sink}}\rangle$. The token numbers k are encoded with 1 byte and the strictly increasing timestamp $epoch$ with 32 bits, resulting in 40 bits in memory for $\langle k, epoch\rangle$. $\langle k, epoch\rangle$ is signed by the sink using a 320 bits-signature, resulting in 360 bits in memory for each token. The system also requires 2KB of free space for signature verification.

  Note that a simple rearrangement of signatures leads to only one signature verification per message, instead of R verifications if all tokens had to be verified. It is obtained by signing each sub-list of tokens instead of the tokens themselves:$X_k = \langle k, epoch, X_{k+1}, \{k, epoch, X_{k+1}\}_{K_{pri}^{sink}}\rangle$.

- **Optimized RESIST-1** is up to 84B for both hash digests (i.e. $H^k(T_1)$ and $H^R(T_1)$) and a signature, and 3KB of free space for hash operations.

- **Basic RESIST-0**, with $T_k = \langle k, epoch, K_{pub}^k, \{\langle k, epoch, K_{pub}^k\rangle\}_{K_{pri}^{sink}}, K_{pri}^k\rangle$, increases the weight of token lists, by including a new pair of public/pri-

	Number of Packets at `Hello` (at worst)					
		Payload(bytes)				
	Hop–distance	30	45	56	80	94
Basic RESIST-1	10	15	10	9	6	5
	20	30	20	17	12	10
Optimized RESIST-1	–	3	2	2	2	1
Basic RESIST-0	10	29	19	16	11	10
	20	57	38	31	22	19
Optimized RESIST-0	–	3	2	2	2	1

Figure 18: Analysis of the number of packets required for *Hello* message in the closest distance to the sink.

vate keys for each token, K^k_{pub}, and K^k_{pri}. A token therefore occupies 85B. Sensor must also keep space for signature operations (2KB). Nevertheless, `Challenge` and `ChallengeReply` messages are shorter, and do not need simultaneous free space.

- **Optimized RESIST-0** uses the same encoding for messages as Optimized RESIST-1. Relaying nodes require additional available memory to generate next hop keys, i.e. 2KB for EC point multiplication. Note that the new generated public key is forwarded to neighbors and it is not further needed.

As shown in Fig. 17, protocol optimizations reduce considerably the length of tokens by means of hashing. Similarly, RESIST-0 estimations at relay nodes represent an upper bound due to the extra free memory required for signature generation (cf. S in Fig. 17) and verification (cf. V in Fig. 17) executed at the *Challenge* stage.

A.4. Estimation of Communication Cost

Using estimations on code size, cryptographic variables, and the performance measurements included above, we now examine the effective data rates and the consumption in the transmission of packets. As it has been shown in the literature, transmission generally consumes more energy than computation. Packets sizes are therefore an important parameter for WSN protocol design.

Since typical packet sizes on WSNs are 30 bytes and 56 bytes, and the highest rate is 250 kbps for current generation of sensor platforms, we can estimate the number of packets required by each protocol, for a network of 20 hops (2000 sensors) and an average of 45 bytes per packet (see Fig. 18 for a summary):

- **Basic RESIST-1:** `Hello` messages occupy $45 \times R$ bytes when they are sent by the sink: I.e., 45×20 bytes, for $R = 20$ according to the example. Since one token is removed at each hop, the size decreases a lot during the propagation of the message.

- **Basic RESIST-0:** `Hello` messages occupy $85 \times R$ bytes when they are sent by the sink: I.e., 85×20 bytes, for $R = 20$ according to the example. Every time one token is removed, the size decreases by two messages.

- **Optimized RESIST-1 and Optimized RESIST-0:** `Hello` messages have a constant size of 84 bytes.The low communication cost is compensated by a higher computational cost.

[1]IEEE 802.15.4 maximum packet size is defined as 127 bytes: MAC header (25 bytes), NWK header (8 bytes) and DATA payload (94 bytes).