SR3: Secure Resilient Reputation-based Routing [*]

Karine Altisen

Stéphane Devismes

Raphaël Jamet

Pascal Lafourcade

Abstract

In this paper, we propose SR3 (which means Secure Resilient Reputation-based Routing), a secure and resilient algorithm for convergecast routing in WSNs (Wireless Sensor Networks). SR3 uses lightweight cryptographic primitives to achieve data confidentiality and unforgeability. Security of SR3 has been proven formally using two verification tools: CryptoVerif and Scyther. We made simulations to show the resiliency of SR3 against various scenarios, where we mixed selective forwarding, blackhole, wormhole, and Sybil attacks. We compared our solution to several routing algorithms of the literature. Our results show that the resiliency accomplished by SR3 is drastically better than the one achieved by those protocols, especially when the network is sparse. Moreover, unlike previous solutions, SR3 self-adapts after compromised nodes suddenly change their behavior.

keywords: Wireless sensor networks, Routing, Security, Resiliency.

1 Introduction

Nowadays, there is a growing interest in WSNs. WSNs are multi-hop mesh networks made of numerous small battery-powered sensors that generate data about the environment (*e.g.*, temperature) and use them for specific services (*e.g.*, emit an alarm when the surrounding temperature is too high). Moreover, they embed wireless communication capabilities allowing them to exchange data. The low capabilities of the sensors, their wireless communications, and the fact that they are deployed in open areas make them prone to attacks.

Routing is a crucial issue in WSNs. Here, we consider a routing scheme called *convergecast* routing. In this problem, a node is distinguished as the *sink* and all non-sink nodes, called here *source nodes*, must be able to transmit data to the sink on request or according to an *a priori* unknown schedule. The sink can be arbitrarily far (in terms of hops) from other nodes. Typically, in WSNs, source nodes are *sensors* and the sink is a *base station* that is linked to another network, like a *gateway.*

A routing protocol in a WSN may have to face many kinds of attacks. Here, we consider the critical scenario, where some sensors are compromised and controlled by an attacker. In particular, such an internal attacker has access to all secret and received information of the compromised nodes.

The attacker can impact the routing protocol at two main levels:

[*]A preliminary version of this paper appeared in [1].

Message Level: First, he can attack the data message to learn secret information, *i.e.*, violate the data *confidentiality*, as this property consists in guaranteeing that data remain secret between the source and destination.

He can also make the sink deliver incorrect information, *i.e.*, violate the *integrity* of data messages. Integrity guarantees that the destination is able to detect whether the data inside a message have been modified.

Moreover, the attacker can act against the *authentication* of the nodes. Authenticity guarantees that the destination is able to detect whether the alleged source in a message is the true one.

Routing Level: Secondly, the attacker can affect the routing scheme itself, *e.g.*, he may prevent data from being delivered by the sink (leading to degrade the quality of service, essentially the delivery rate), or create congestion by increasing the load in all or part of the network (leading to reduce the lifetime of the network). Such an attack can also evolve over time, *e.g.*, the attacker can attract traffic to a given intruder node using various means. Thanks to that, the intruder node (called a *sinkhole*) will have more impact for further malicious actions.

Related Work. Numerous solutions have been introduced to cope with attacks on data. The confidentiality, authenticity, and integrity properties are mainly guaranteed using cryptographic mechanisms. However, the choice of the cryptographic primitives should be led by the inherent constraint of WSNs. WSNs being limited in terms of resource and power, *lightweight* cryptographic mechanisms [2] are mandatory. An example of such a mechanism is *elliptic curve cryptography* [3, 4]. In contrast, classical asymmetric cryptography, *e.g.*, *RSA*, should be excluded due to its computational cost.

Using such lightweight cryptographic primitives, routing protocols implementing some security properties have been proposed, *e.g.*, *μ-Tesla* [5, 6] is a broadcast authentication protocol that enables receivers of the broadcast data to verify that these data really originate from the alleged sources. μ-Tesla has a low communication and computation overhead, scales to a large number of receivers, and tolerates message loss.

Although it is not strictly a routing protocol, *SPINS* [5] is a set of tools for routing, which provides security guarantees without using any costly operations: μ-Tesla is one part of SPINS; the other part is *SNEP*, a message format that guarantees various security properties, like authenticity and confidentiality, using few additional bits per message.

Some other protocols, called *secure route discovery protocols*, have been introduced [7, 8] to help securing routing. Actually, they compute a valid route (*i.e.*, the computed path exists in the network) between the source and destination, and for some of them (*e.g.*, [7]), they guarantee that nodes in the chosen route achieved a certain security level, *e.g.*, the integrity of the discovered route, which means that the computed route has been effectively traversed during the discovery process.

However, all aforementioned solutions do not use specific strategies to combat attacks at the routing level, *e.g.*, selected forwarding, blackhole, *etc.* Specific approaches have been proposed to maintain a good quality of service in presence of insiders, that drop all or part of messages. For example, the notion of *resiliency* has been introduced in [9, 10] as the ability of a network to "continue operating" in presence of compromised nodes, *i.e.*, the capacity of a network to endure and overcome internal attacks. For example, a resilient routing protocol should achieve a "graceful degradation" in the delivery rate with increasing the number of compromised nodes. In [11, 10], authors experimentally analyze the resiliency of several classical routing techniques, *e.g.*, random walk [12], gradient-based routing [13], geographic routing [14]. The experimental results show that these solutions are weak in terms of resiliency. Then, they propose several resilient variants of the gradient-based routing; this latter routes messages following a *Destination Oriented Directed Acyclic Graph* (DODAG). Mainly, they introduce randomization and duplication in that protocol. As a result, the proposed patches drastically increase the delivery rate when the network is subject to selective forwarding or blackhole attacks. However, in their simulations, they always assume that the DODAG is available and not attacked by insiders. Moreover, they mainly consider dense networks in their simulations, *e.g.*, networks with average degree around 30.

Contribution. This paper deals with convergecast routing in WSNs, where all source nodes have several messages to route. We propose a Secure, Resilient, and Reputation-based Routing algorithm, called SR3. This protocol is a reinforced random walk that is partially determinized using a reputation mechanism.

SR3 uses lightweight cryptographic primitives: symmetric cryptography (precisely, *authenticated encryption* [15]), nonces, and hash functions. Thanks to these primitives, it achieves interesting security properties, including data confidentiality and unforgeability, this latter property implies integrity and authenticity of the data. We prove the desirable security properties achieved by SR3 in the computational model using the formal tool *CryptoVerif* [16]. We also prove in the symbolic model the secrecy of data and authentication of nodes using the tool Scyther [17].

Then, we show the resiliency of SR3 against various scenarios, where we mixed selective forwarding, blackhole, wormhole, and Sybil attacks. The resiliency of our algorithm is mainly captured using the delivery rate and some fairness measure. Our simulation results show in particular that unlike previous solutions, SR3 self-adapts when compromised nodes change their behavior (*e.g*, an interesting case is when a compromised node behaves well to attract the traffic and then suddenly decide to drop all received messages). We compare our solution to several routing algorithms of the literature, including resilient ones. Our simulations show that the resiliency accomplished by SR3 is drastically better than the one achieved by those protocols, especially when the network is sparse.

A shortcoming of our solution is the number of hops to reach the destination, as it is usually greater than other solutions of the literature. However, in our experiments, we observed that this complexity remains sublinear in the number of nodes.

Note also that our solution is *reactive* (*i.e.*, in absence of data to route the protocol eventually stops.), has a low overhead in terms of communications, and does not use any underlying infrastructure, such as spanning tree or DODAG. Hence, SR3 is well-suited for WSNs.

Roadmap. The remainder of the paper is organized as follows. In the next section, we present our routing algorithm, SR3. Section 3 deals with the automatic proof of the security properties of SR3 using CryptoVerif [16] and Scyther [17]. In Section 4, we present experimental results that show the resiliency of SR3. Section 5 is dedicated to concluding remarks.

2 SR3

The formal code of our routing protocol, SR3, is given in Algorithms 1 and 2. Below, we identify the assumptions we made about networks. Then, we informally explain the behavior of SR3.

2.1 Assumptions

We consider arbitrary connected networks with bidirectional links, although we will focus on Unit Disk Graphs (UDG) in simulations. Each node p has a unique ID (to simplify, we shall identify any node with its identifier, whenever convenient) and knows the set of its neighbors, $\mathcal{N}eig_p$ — this latter assumption will be relaxed, when considering Sybil attacks.

Networks are made of one sink, which is the data collector, and numerous source nodes. The source nodes are sensors, and consequently are limited in terms of memory, computational power, and battery. Sensors are non-trustworthy since they are vulnerable to physical attacks and an adversary can compromise them. In contrast, the sink is assumed to be robust and powerful in terms of memory, computation, and energy. So, we assume that it cannot be compromised.

All nodes have access to a lightweight cryptography library (hash function, symmetric encryption, and random number generation). Each source node shares a symmetric key with the sink. Moreover, we assume that all source nodes have several data to route; however, the scheduling of the data generation is *a priori* unknown. Finally, there is no time synchronization between nodes.

2.2 Overview

Randomization is interesting to obtain resilient solutions because it generates behaviors unpredictable by an attacker. However, note that the "classical" *uniform* random walk, where a node chooses the next hop uniformly at random among its neighbors, is known to be inefficient even against a small number of compromised nodes [9]. So, we designed SR3 rather as a *reinforced* random walk, based on a reputation mechanism. The idea is to locally increase the probability of a neighbor to be chosen at the next hop, if it behaves well. Such a reputation mechanism is based on acknowledgments. We propose a scheme in which if a process receives a valid acknowledgment, it has the guarantee that the sink actually delivered the corresponding data message. Hence, upon receiving such an acknowledgment, a process can legitimately increase its confidence on the neighbor to which it previously sent the corresponding data message. Therefore, eventually all honest nodes preferably choose their highly-reputed neighbors, and so the data messages tend to follow paths that successfully route data to the sink.

2.3 Reputation Mechanism

To implement our reputation mechanism, we identify each data message (tagged `MSG` in the algorithm) with a nonce, *i.e.*, an unpredictable random number that should remain secret between the source and sink until the delivery of the data message.

Assume that node v initiates the routing of some value $Data$. It first generates a nonce N_v (NEW_NONCE(), Line 1). Then, it encrypts in a ciphertext C the concatenation of $Data$ and N_v using the key k_{vs} it shares with the sink only ($\mathrm{E}_{k_{vs}}(\langle Data, N_v\rangle)$, Line 3). Then, both C and the identifier of v (in plaintext) are routed to the sink, and only the sink is able to decrypt C. So, upon receiving the data message, the sink decrypts C using k_{vs}, delivers $Data$, and sends back to v an acknowledgment `ACK` containing N_v (Lines 36-39). Finally, if v receives this acknowledgment, it has the guarantee that $Data$ has been delivered, thanks to N_v.

Now, during the routing, a compromised relay node can blindly modify the encrypted part of the message. To prevent the sink from delivering erroneous data, we assume a *(symmetric) authenticated encryption* scheme [15], which guarantees confidentiality, authenticity, and integrity of encrypted information. This way, when receiving a message, the sink checks the validity of the message by decrypting the ciphertext using the key $keys[o]$ of the alleged source o of the message ($\mathrm{E}^{-1}_{keys[o]}(C)$, Line 36). If the ciphertext has been modified, then $\mathrm{E}^{-1}_{keys[o]}(C) = \perp$ (with overwhelming probability) and the message is simply discarded. Similarly, if a compromised node has modified the plaintext identifier in the message, then either $keys[o]$ is undefined, or the sink decrypts the ciphertext with a wrong key (in this latter case again, $\mathrm{E}^{-1}_{keys[o]}(C) = \perp$). So, in both cases the message is also discarded.

Upon receiving an acknowledgment, if the receiving node v is the initiator of the corresponding data message m, v can conclude that m has been delivered. In that case, v should reinforce the probability associated to the neighbor to which it previously sent m. To achieve that, we proceed as follows: when v initiates the routing of m, v saves in the list L_{Sent} the nonce stored in m, together with the identifier of the neighbor to which v sends m (L_{Sent} is appended in Line 5 using $\odot$, this latter operator is defined below). Hence, on reception of an acknowledgment, v checks (in Line 20) if it is the destination of the acknowledgment and if the nonce N_o attached to that acknowledgment appears in L_{Sent} (see the test $\langle N_o, \_ \rangle \in L_{Sent}$ in Line 20).[1] In that case, v gets back the corresponding neighbor from the list (GET(L_{Sent}, N_o), Line 21), increases its confidence on that neighbor (Line 22, further details about increasing the confidence are given in Subsection 2.4), and removes the record from L_{Sent} ($L_{Sent} \setminus \langle N_o, \_ \rangle$, Line 23). (If v is the destination of the acknowledgment, but N_o does not appear in L_{Sent}, the acknowledgment is simply discarded.)

Due to the memory limitations, L_{Sent} must have a maximum size,[2] s_Q. If a node v has some new data to route and L_{Sent} is full (that is, it contains s_Q elements), then the oldest element is removed from the list to make room for the new one. A side effect is that records about lost messages or of messages whose acknowledgment has been lost are eventually removed from L_{Sent}.

[1] "_" means "any value". So, $\langle N_o, \_ \rangle$ is any record whose left value is N_o.

[2] All lists used in SR3 are of bounded size. We made several experiments to choose the appropriate bounds, see Subsection 4.2.

Note that it may happen that some data message m comes back to the node v from which it originates because m followed a cycle in the network. In this case (Lines 8-13), the validity of m is checked, and if so, the routing process of m is restarted. Since the old entry in L_{Sent} is not relevant anymore, it is simply replaced by the new one.

Consequently, the concatenation of $\langle x, y\rangle$ to the list L using $\odot$ works as follows: first, if L contains any pair with a left member equal to x, that pair is removed from L; then, if L is (still) full, the rightmost pair is removed; finally, $\langle x, y\rangle$ is inserted on the left side of the list. Note that, using $\odot$, any left member of a pair in the list is unique.

2.4 Compute the Reputation

To choose the next hop of some data message, a node performs a random choice among its neighbors, weighted according to their reputation (see Lines 4 and 7).

The reputation of a neighbor actually corresponds to the number of occurrences of its identifier in the list $L_{Reputation}$: each time a node v wants to reinforce the reputation of some neighbor u, it simply adds an occurrence of u into its list (Line 22).

Our reputation mechanism is implemented using the probability law denoted by $\mathcal{L}^{v}_{SR3}(L_{Reputation})$: Let X be a random variable taking value in $\mathcal{N}eig_v$; the law $\mathcal{L}^{v}_{SR3}(L_{Reputation})$ is defined, $\forall x \in \mathcal{N}eig_v$, by:

$$Pr(X = x) = \frac{|L_{Reputation}|_x + \delta_v^{-1}}{|L_{Reputation}| + 1}$$

Where δ_v is the degree of v, $|L_{Reputation}|$ is the number of elements in $L_{Reputation}$, and $|L_{Reputation}|_x$ is the number of occurrences of x in $L_{Reputation}$. Hence, when v wants to route a data message, it chooses its next destination according to $\mathcal{L}^{v}_{SR3}(L_{Reputation})$ (see $\textsc{Rand}(\mathcal{N}eig_v,\mathcal{L}^{v}_{SR3}(L_{Reputation}))$ in Lines 4 and 7).

Informally, when a node needs to route a message, it draws at random a value from $L_{Reputation}$ plus a blank element. If the blank element is drawn, it selects a neighbor uniformly at random, and sends the message to that neighbor. Otherwise, the message is sent to the neighbor whose identifier has been drawn. This way, the more a neighbor is trusted, the more it will be selected. However, because of the blank element, there is always a positive probability of selecting a neighbor without taking trust into account. Note that, initially $L_{Reputation}$ is empty, and consequently the first selections are made uniformly at random.

To ensure a better resiliency against attackers that change their behavior over time, and to reduce memory consumption, $L_{Reputation}$ is defined as a FIFO list of maximum size, s_R. The insertions in $L_{Reputation}$ use the operator $\bullet$ that satisfies the following condition: when the list is full, the next insertion is preceded by the removing of the oldest (and consequently, less relevant) element.

An example illustrating the probability law is provided in Figure 1. In this example, we focus on the node v and assume a $L_{Reputation}$ of at most 3 elements. From the given configuration, v will route most messages through z, because it has the greatest number of occurrences in the list $L_{Reputation}$ of v. The high reputation of z may come from the fact that at some point the paths from x to the sink through z were faster and more reliable: this is a side effect of our protocol. Such fast and reliable paths indirectly help increasing the reputation of z, since in this case x receives valid acknowledgments from z more frequently than from other neighbors.

Using such a FIFO finite list, a node only stores the freshest information. Interestingly, if a compromised node first behaves well, its reputation increases, resulting in attracting the traffic. Then, it may change its behavior to become a blackhole (a node dropping all messages it receives). Now, thanks to our mechanism, regularly some messages will be routed via other nodes and consequently the reputation of the compromised node will gradually decrease, inducting then a severe reduction of the traffic going through that node.

Consider again Figure 1. If z turns out to be compromised and starts dropping all messages, then all messages going through z and w will either get lost or loop back to v. However, there is still a positive probability that v routes messages through y, which will retransmit them. Some of these messages will be delivered and consequently acknowledged. So, the identifiers currently stored in $L_{Reputation}$ will be

$$L_{Reputation} = \ [w,\, z,\, z]\ ,\, s_R = 3 \text{ for node } v$$

Next hop probabilities for v:

$$P(X = w) = \frac{1 + 1/3}{4} = \frac{4}{12} \approx 33.33\%$$

$$P(X = z) = \frac{2 + 1/3}{4} = \frac{7}{12} \approx 58.33\%$$

$$P(X = y) = \frac{0 + 1/3}{4} = \frac{1}{12} \approx 8.33\%$$

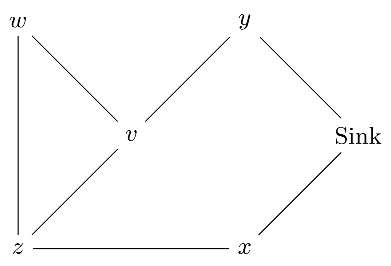


Figure 1: An example of how the reputation affects the routing process

progressively replaced by occurrences of y, increasing its probability (resp. decreasing the probability of w and z) of being chosen.

2.5 Acknowledgment Routing

An acknowledgment message *ack* is emitted because the corresponding data message m has been successfully delivered by the sink. So, we can suppose that the path followed by m was safe. Thus, we can use the bidirectionality of the links to route *ack* (as much as possible) through the reverse path followed by m.

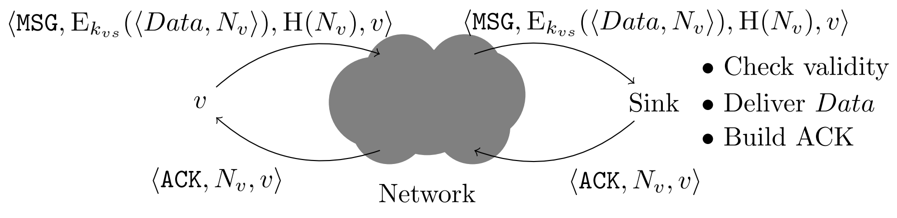
This reverse routing is accomplished by letting a trail along the path followed by m. This trail is actually made using hash $\mathrm{H}(N_v)$ of the nonce N_v identifying the message (see Line 2), this hash being stored in plaintext in the data message (see Line 6). The trail is then is stored thanks to the list $L_{AckRouting}$ maintained at each node: after the reception of each data message, the relaying nodes store the hash of the nonce available in the message, together with the identifier of the neighbor from which they received the message (Lines 14-17). This information will be then used during the return trip of the acknowledgment: when a node v receives an acknowledgment containing the nonce N_x, it checks whether it is the final destination of that acknowledgment (Lines 20 and 25). If this is not the case, v checks if an entry containing $\mathrm{H}(N_x)$ exists in $L_{AckRouting}$ (Lines 26-30). If v finds such an entry, it sends the acknowledgment to the corresponding neighbor and removes the entry from $L_{AckRouting}$ (Line 29). Otherwise, the next hop of the acknowledgment is chosen uniformly at random, in a best-effort mindset ($\mathcal{L}^v_{RW}$ denotes the probability law of the uniform random walk, see Line 31).

If a data message loops back to a node it already visited, the most relevant information regarding acknowledgments for this node is the oldest one. Therefore, before inserting a new trail, the node checks if $L_{AckRouting}$ already contains a trail for that message. If a related entry exists, we do not update $L_{AckRouting}$ (Lines 14-17).

Acknowledgments can be still dropped by compromised nodes. The trail for such lost acknowledgments would unnecessarily clutter the memory of nodes. To avoid this, we manage $L_{AckRouting}$ similarly to $L_{Reputation}$, *i.e.*, $L_{AckRouting}$ is a list of bounded size s_A, appended using operator $\bullet$.

Finally, an intruder may build acknowledgments with false nonces. These fake acknowledgments will increase the load of the network, and impact the energy consumption. Now, some nodes being compromised, a safe node cannot trust information coming from its neighbors to decide whether it should forward or drop an acknowledgment. To circumvent that problem, a relay node decides to drop a received acknowledgment with probability $\frac{1}{N}$, where N is an upper bound on the number of nodes (Lines 33 and 42). So, on the average, an acknowledgment makes N hops in the network before being dropped. An interesting side effect of this method is the following: in a safe network (*i.e.*, a network without attackers), the acknowledgments that follow long routes are often dropped before reaching their final destination. Since the length of the routes followed by the acknowledgments are directly related to the length of the route taken by the corresponding messages, the reputation mechanism ends up favoring shorter routes, thus improving the overall hops complexity.

The overall behavior of SR3 is summarized in Figure 2.



$\langle \texttt{MSG}, \mathrm{E}_{k_{vs}}(\langle Data, N_v\rangle), \mathrm{H}(N_v), v\rangle$
$\langle \texttt{MSG}, \mathrm{E}_{k_{vs}}(\langle Data, N_v\rangle), \mathrm{H}(N_v), v\rangle$
v
Sink
• Check validity
• Deliver *Data*
• Build ACK
$\langle \texttt{ACK}, N_v, v\rangle$
Network
$\langle \texttt{ACK}, N_v, v\rangle$

Figure 2: A message and its acknowledgment.

3 Security Analysis

We evaluate the security of SR3 in two phases. The first phase focuses on the message format, for which we prove the following three properties: *unforgeability*, *confidentiality of the data*, and *confidentiality of the nonce* before message delivery. This analysis uses the tool *CryptoVerif* [16]. We first detail the modeling of SR3 and our intruder model. Then, we model the different security properties we considered. CryptoVerif automatically finds bounds on the security of these properties. We refined these bounds to allow the user to determine the desired trade-off between message sizes and the expected security level. We illustrate our results with an example based on the classical data link protocol *S-MAC* [18][3] which is dedicated to WSNs and allows to transmit up to 250 bytes of data in a message.

The second phase is a symbolic analysis of the protocol, in order to prove its security when running several sessions. This analysis supposes that the cryptographic primitives are perfect. For this, we use the tool *Scyther* [17].

We first describe how we model SR3 and the attacker for these two analyses.

3.1 Modeling of SR3

SR3, as described in the previous section, routes messages through several nodes. There are three distinct roles in this process: the *source* (whose identifier is denoted by *src*) of the considered data message, the *relays*, and the *sink*.

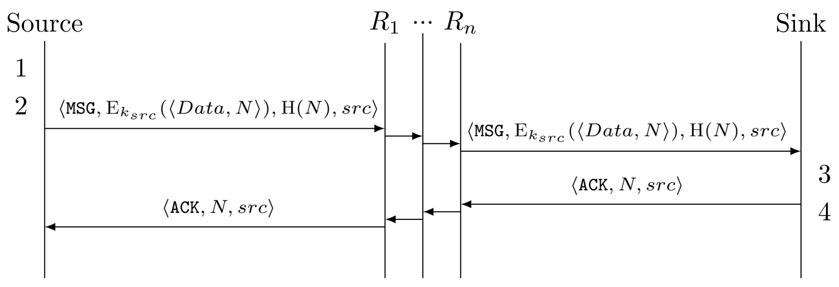
The data message is initially created by the source, which then forwards it either to a relay or the sink. Initially, a data message consists of a ciphertext $C = \mathrm{E}_{k_{src}}(\langle Data, N\rangle)$ containing the (symmetric) authenticated encryption of some data *Data* and a nonce N, and a plaintext part made of the hash of the nonce N and the identifier *src*. Notice that the symmetric key k_{src} is chosen uniformly at random in $\mathcal{K}$, the space of all the possible symmetric keys, before the WSN is deployed. We suppose k_{src} is known only by the source *src* and the sink.

Relays forward the data message without changing it. When a data message reaches the sink, this latter first checks whether the message respects the format $\langle \texttt{MSG}, C, h, s\rangle$. If so, C is decrypted with the key of the node identified by s ($\mathrm{E}^{-1}_{keys[s]}(C)$): if this operation succeeds, then a valid data is extracted and delivered, moreover an acknowledgment $\langle \texttt{ACK}, N, src\rangle$, is generated and routed through relays until reaching the source.

A data message can loop back to its source, and an acknowledgment can loop back to the sink: they act as relays in these cases.

Figure 3 illustrates that view of the protocol. The honest relays do not alter the messages in any way (however, remind that honest nodes may sometimes drop ACK messages), and the protocol works with any number of them, or none at all. Also, our intruder model specifies that any node can be compromised, except the sink. Therefore, all relays R_i are suspicious, and we lump them, whether honest or compromised,

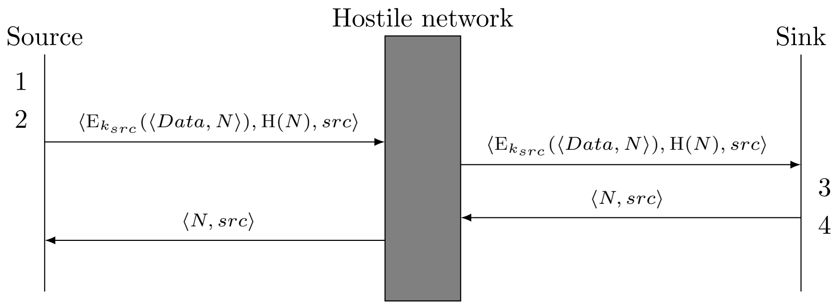
[3]S-MAC stands for Sensor Medium Access Control.


Source
$R_1 \cdots R_n$
Sink
1
2
$\langle \texttt{MSG}, \mathrm{E}_{k_{src}}(\langle Data, N\rangle), \mathrm{H}(N), src\rangle$
$\langle \texttt{MSG}, \mathrm{E}_{k_{src}}(\langle Data, N\rangle), \mathrm{H}(N), src\rangle$
3
$\langle \texttt{ACK}, N, src\rangle$
4
$\langle \texttt{ACK}, N, src\rangle$

1. Generate *Data*, 2. Draw a nonce N,

3. Check validity, 4. Deliver *Data*

Figure 3: One session of the SR3 protocol


Hostile network
Source
Sink
1
2
$\langle \mathrm{E}_{k_{src}}(\langle Data, N\rangle), \mathrm{H}(N), src\rangle$
$\langle \mathrm{E}_{k_{src}}(\langle Data, N\rangle), \mathrm{H}(N), src\rangle$
3
$\langle N, src\rangle$
4
$\langle N, src\rangle$

1. Generate *Data*, 2. Draw a nonce N,

3. Check validity, 4. Deliver *Data*

Figure 4: Modeling of one session of SR3

together in one single entity, called the *hostile network*. All communications between the source and the sink happen through the hostile network, as depicted in Figure 4. The hostile network can modify or drop messages, and can also create messages using informations deduced from previous communications. (As the types MSG and ACK can be deduced from the message format, we omit them in the modeling.)

3.2 Game-Based Proof of the Security Properties in the Computational Model

We now consider the message format.

3.2.1 Background

We model the ability of our protocol to meet a given security property using *games*. A game is a probabilistic algorithm where an adversary, given as a probabilistic polynomial-time Turing machine, faces a challenge, which consists in breaking a specific property modeled by the game. The goal is then to compute the ability of the adversary to win the challenge. This is called the *advantage* of the adversary.

When describing a game, we write $a \overset{\$}{\leftarrow} X$ to denote that a is a random value obtained according the distribution represented by X. If X is a set, a is drawn uniformly at random on X. Similarly, if X is a probabilistic algorithm, a is drawn randomly using the algorithm.

Our analysis uses a common cryptography model, called the *random oracle model* [19], where hash functions are modeled by *random oracles*. A *random oracle* is a theoretical black box $\mathcal{O}$ which satisfies the following property: for every input i, the first time $\mathcal{O}$ is queried with i, $\mathcal{O}$ returns a value v, picked uniformly at random from its output domain; then, each time $\mathcal{O}$ is queried again with i, $\mathcal{O}$ returns the same value v. Moreover, encryptions are modeled using *encryption oracles*. An *encryption oracle* is a theoretical black box which retains the secret encryption key and encrypts arbitrary data at the adversary's request.

3.2.2 Modeling SR3's Primitives

Hash function. Our algorithm uses a hash function of input size η_n and of output size η_h. We model it as a random oracle, and we refer to this modeling using $\mathcal{H} : \{0,1\}^{\eta_n} \rightarrow \{0,1\}^{\eta_h}$.

Nonces. Nonces are modeled as truly random numbers of size η_n.

Encryption scheme. We assume the authenticated encryption is both *IND-CCA2* (*INDistinguishability against adaptive Chosen-Ciphertext Attack*) [20] and *INT-PTXT* [21, 22] (*INTegrity of the PlainTeXTs*).

Intuitively, an encryption primitive is *IND-CCA2*, if it is computationally difficult for an adversary to win a challenge that consists in guessing which of two data is encrypted into a given ciphertext despite it has access to a decryption and an encryption oracle before and after the reception of the challenge. More formally, we recall below the *IND-CCA2* game introduced in [15].

$$\begin{aligned}
&\textit{Experiment } \mathbf{Expt}^{\textit{IND–CCA2}}(\mathcal{A}) :\\
&\qquad K \xleftarrow{\$} \mathcal{K}\\
&\qquad (D_0, D_1, \mathit{state}) \xleftarrow{\$} \mathcal{A}_1^{\mathcal{E}(K,\cdot),\mathcal{E}^{-1}(K,\cdot)}()\\
&\qquad b \xleftarrow{\$} \{0,1\}\\
&\qquad C \xleftarrow{\$} \mathcal{E}(K, D_b)\\
&\qquad \mathbf{Return}\ b = \mathcal{A}_2^{\mathcal{E}(K,\cdot),\mathcal{E}^{-1}(K,\cdot)}(D_0, D_1, C, \mathit{state})
\end{aligned}$$

First, a key K is generated uniformly at random from $\mathcal{K}$, the set of possible (symmetric) keys, of size η_k. Then, the adversary $\mathcal{A}$ runs in two phases: $\mathcal{A}_1$ and $\mathcal{A}_2$. In each of them, the encryption and decryption oracles, respectively denoted by $\mathcal{E}(K,\cdot)$ and $\mathcal{E}^{-1}(K,\cdot)$, can be called a polynomial number of times. $\mathcal{A}_1$ outputs two distinct data D_0 and D_1 of identical size and some information *state* used to link the two attacker phases together. Then, one of the data D_0 and D_1 is selected uniformly at random and encrypted in the ciphertext C. Finally, $\mathcal{A}_2$ receives the challenge (*i.e.*, the two data D_0, D_1, as well as the ciphertext C) and *state*. The adversary $\mathcal{A}_2$ is allowed to call $\mathcal{E}^{-1}(K,\cdot)$ with any ciphertext, except the challenge ciphertext C. The game returns 1 if and only if the adversary correctly guesses the value of the challenge bit b, the index of the data encrypted into C. For any adversary $\mathcal{A}$, the *IND-CCA2* advantage in this game, noted $\mathbf{Adv}^{\textit{IND–CCA2}}(\mathcal{A})$, is defined as:

$$\mathbf{Adv}^{\textit{IND–CCA2}}(\mathcal{A}) = 2 \times Pr[\mathbf{Expt}^{\textit{IND–CCA2}}(\mathcal{A}) = 1] - 1$$

where $Pr[\mathbf{Expt}^{\textit{IND–CCA2}}(\mathcal{A}) = 1]$ is the probability that $\mathcal{A}$ wins the *IND-CCA2* game.

Intuitively, an encryption primitive E is *INT-PTXT*, if it is computationally difficult for an adversary to produce a valid ciphertext decrypting to a data which had never been encrypted using E. Below, we recall the game used in [15] to define this notion for a symmetric encryption scheme $(\mathcal{K}, E, D)$, where $\mathcal{K}$, E, and

D are respectively a set of keys (of size η_k), and encryption and decryption primitives.

$$\begin{aligned}
&Experiment\ \mathbf{Expt}^{\mathit{INT\text{–}PTXT}}(\mathcal{A}):\\
&\quad K \xleftarrow{\$} \mathcal{K}\\
&\quad S \leftarrow \emptyset;\\
&\quad C \xleftarrow{\$} \mathcal{A}^{\mathcal{E}_S(K,\cdot),\mathcal{E}^{-1}(K,\cdot)}()\\
&\quad d \leftarrow \mathcal{D}(K, C)\\
&\quad \mathbf{Return}\ d \neq \perp \text{ and } d \notin S
\end{aligned}$$

Each call to the encryption oracle $\mathcal{E}_S(K, d)$ consists of storing the data d into S, and then returning $E(K, d)$, the encryption of d using key K. The adversary also has access to the decryption oracle $\mathcal{E}^{-1}(K, \cdot)$. The adversary $\mathcal{A}$ wins the *INT-PTXT* game if and only if it can forge a valid ciphertext C whose corresponding decryption has never been queried to the encryption oracle. For any adversary $\mathcal{A}$, the *INT-PTXT* advantage in this game, noted $\mathbf{Adv}^{\mathit{INT\text{–}PTXT}}(\mathcal{A})$, is defined as:

$$\mathbf{Adv}^{\mathit{INT\text{–}PTXT}}(\mathcal{A}) = Pr[\mathbf{Expt}^{\mathit{INT\text{–}PTXT}}(\mathcal{A}) = 1]$$

where $Pr[\mathbf{Expt}^{\mathit{INT\text{–}PTXT}}(\mathcal{A}) = 1]$ is the probability that $\mathcal{A}$ wins the *INT-PTXT* game.

Since we assume the encryption scheme of SR3 is both *IND-CCA2* and *INT-PTXT-secure*, both $\mathbf{Adv}^{\mathit{IND\text{–}CCA2}}(\mathcal{A})$ and $\mathbf{Adv}^{\mathit{INT\text{–}PTXT}}(\mathcal{A})$ become *negligible* in the size of the keys of encryption scheme η_k. An advantage $\mathbf{Adv}(\mathcal{A})$ becomes *negligible* in x, if for every positive polynomial $P(x)$, we have $\exists K, \forall x > K, \mathbf{Adv}(\mathcal{A}) < \frac{1}{P(x)}$.

3.2.3 Modeling SR3

Based on the above presented model, we describe actions performed by the source and sink using two functions, see Figure 5. The attacker has access to some of these functions. For instance, the function $Gen^{\mathrm{E}_{k_{src}}(\cdot)}(\cdot)$ below represents the normal behavior of a node, and an attacker who has access to this function models a chosen-plaintext attack.

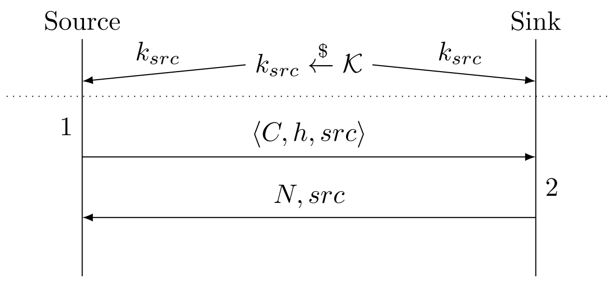
- $Gen^{\mathrm{E}_{k_{src}}(\cdot)}(Data)$ is the function which generates a message produced by src containing $Data$ (whose length is η_d), using the encryption function from $\{0,1\}^{\eta_p}$ to $\{0,1\}^{\eta_c}$ (where $\eta_p = \eta_d + \eta_n$ and $\eta_c \geq \eta_p$). This message is made of $\langle C, h, src\rangle = \langle \mathrm{E}_{k_{src}}(\langle Data, N\rangle), \mathrm{H}(N), src\rangle$, where N is a fresh unpredictable nonce of size η_n, $\mathrm{H}(N)$ is the hash of N, and $\mathrm{E}_{k_{src}}(\langle Data, N\rangle)$ is the encryption of the concatenation of the data and nonce. The function $Gen^{\mathrm{E}_{k_{src}}(\cdot)}(Data)$ returns the pair $\langle C, h, src\rangle, N$. The nonce N is given in cleartext to represent the knowledge of an attacker that listens to traffic in the network. Indeed, such an attacker may have access to both the messages and their acknowledgments; he may then know the nonces contained in the original messages.

  We store all the encrypted ciphertexts in a set called *Queries*, initially empty.

- $Verif^{\mathrm{E}^{-1}_{k_{src}}(\cdot)}(\langle C, h, s\rangle)$ is the function that checks whether the packet $\langle C, h, s\rangle$ is valid or not. Precisely, it checks

  - whether $s = src$,
  - whether $\mathrm{E}^{-1}_{k_{src}}(C) \neq \perp$, which guarantees the integrity of the encrypted data (*e.g.*, $\mathrm{E}^{-1}_{k_{src}}(C)$ fails if some digit has been reversed), and authenticity (since encryption uses a symmetric key, only entities knowing this key, *i.e.*, the source and sink, are able to correctly encrypt and decrypt data, consequently integrity implies authenticity), and
  - whether $h = \mathrm{H}(N)$, where $\langle d, N\rangle = \mathrm{E}^{-1}_{k_{src}}(C)$ (of course, this last check is made only if the two first ones succeed).

  If these three conditions are satisfied, then the function outputs 1 (meaning that the message is valid), 0 otherwise.


Source
Sink
k_{src}
$k_{src} \xleftarrow{\$} \mathcal{K}$
k_{src}
1
$\langle C, h, src\rangle$
N, src
2

1. $(\langle C, h, src\rangle, N) = Gen^{\mathrm{E}_{k_{src}}(\cdot)}(Data)$
2. Message verification phase: $Verif^{\mathrm{E}^{-1}_{k_{src}}(\cdot)}(C)$

Figure 5: The SR3 protocol, using functions

These two functions allow us to model SR3 in CryptoVerif. Notice that, in the following games, we replace $\mathrm{E}_{k_{src}}(\cdot)$, $\mathrm{E}^{-1}_{k_{src}}(\cdot)$, and $\mathrm{H}(\cdot)$ — the encryption, decryption, and hash functions actually used by SR3 — by the encryption, decryption, and random oracles $\mathcal{E}(k_{src}, \cdot)$, $\mathcal{E}^{-1}(k_{src}, \cdot)$, and $\mathcal{H}(\cdot)$ that respectively model them. These oracles represent knowledge accessible to the intruder.

3.3 Properties

The three properties we want to prove are the following:

- *Confidentiality of the (encrypted) data*: the probability of the adversary getting information about the data in a message is negligible, even when the acknowledgment has been sent.
- *Confidentiality of the nonce*: the probability of the adversary getting information about the nonce N in a message, before the message has been delivered, is negligible.
- *Unforgeability*: the probability that the adversary creates a new ciphertext C such that $Verif^{\mathrm{E}^{-1}_{k_{src}}(\cdot)}(C) = 1$ is negligible.

With the help of CryptoVerif, we analyzed those three properties of SR3. Each of these three properties is evaluated thanks to a game. For each game, CryptoVerif outputs a bound on the advantage of any adversary in that game. This bound is obtained automatically after successive game reductions. The complete verification code is available online [23].

3.3.1 Data Confidentiality

The first property we consider is the confidentiality of the data. The game (named FG, for *Find-then-Guess*) is based on the idea that even if the adversary chooses the set of possible data, it cannot guess which of those data is inside a given message. On the other hand, if the attacker was able to win reliably, it would be also effectively able to recover some information about the data contained in messages, without knowledge of the key.

Let $\mathcal{A}$ be an adversary running in two phases: $\mathcal{A}_1$ and $\mathcal{A}_2$. First, $\mathcal{A}_1$ outputs two data, D_0 and D_1, of identical size together with some information *state* used to link the two attacker phases together. One of these two data is selected uniformly at random, and a data message $\langle C, h, src\rangle$ is generated using the selected data and the key k_{src} of *src*, initially generated uniformly at random from $\mathcal{K}$. Then, $\langle C, h, src\rangle$, the nonce N it contains, and *state* are given to $\mathcal{A}_2$. To win, $\mathcal{A}_2$ should guess which of the two data is in C. During this game, $\mathcal{A}$ can query $Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot)$, $\mathcal{H}(\cdot)$, and $Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)$.

$$
\begin{aligned}
&\textit{Experiment } \mathbf{Expt}^{FG}(\mathcal{A}):\\
&\quad k_{src} \overset{\$}{\leftarrow} \mathcal{K}\\
&\quad (D_0, D_1, state) \overset{\$}{\leftarrow} \mathcal{A}_1^{Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot),\mathcal{H}(\cdot),Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)}()\\
&\quad b \overset{\$}{\leftarrow} \{0,1\}\\
&\quad (\langle C, h, src\rangle, N) \overset{\$}{\leftarrow} Gen^{\mathcal{E}(k_{src},\cdot)}(D_b)\\
&\quad \mathbf{Return}\ b = \mathcal{A}_2^{Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot),\mathcal{H}(\cdot),Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)}(\langle C, h, src\rangle, N, state)
\end{aligned}
$$

Let $Pr[\mathbf{Expt}^{FG}(\mathcal{A}) = 1]$ be the probability of winning the find-then-guess game. We define the find-then-guess advantage of $\mathcal{A}$ against FG, noted $\mathbf{Adv}^{FG}(\mathcal{A})$, as follows:

$$\mathbf{Adv}^{FG}(\mathcal{A}) = 2 \times Pr[\mathbf{Expt}^{FG}(\mathcal{A}) = 1] - 1$$

We modeled this game in CryptoVerif to obtain a bound on $\mathbf{Adv}^{FG}(\mathcal{A})$. Actually, this bound depends on the ability of the attacker $\mathcal{A}$ to break the encryption scheme used by SR3, *i.e.*, this game can be reduced to the *IND-CCA2* game. This means that the bound on $\mathbf{Adv}^{FG}(\mathcal{A})$ depends on the advantage $\mathbf{Adv}^{IND\text{–}CCA2}(\mathcal{B})$ of some adversary $\mathcal{B}$ in the *IND-CCA2* game. Precisely, for all adversaries $\mathcal{A}$

- making q_G queries to $Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot)$, q_V queries to $Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)$, and q_H queries to $\mathcal{H}(\cdot)$ in the FG game, and
- running the FG-game in $T_{\mathcal{A}}$ time units,

there exists an adversary $\mathcal{B}$

- making $q_G + 1$ queries to the encryption oracle $\mathcal{E}(k_{src}, \cdot)$ and q_V queries to the decryption oracle $\mathcal{E}^{-1}(k_{src}, \cdot)$ in the *IND-CCA2* game, and
- running the *IND-CCA2* game in $T_{\mathcal{B}}$ time units with $T_{\mathcal{B}} = T_{\mathcal{A}} + P_1(q_G, q_V, s)$ time units, where $P_1(q_G, q_V, s)$ is polynomial in q_G, q_V, and the message size s (see [23] for details)

such that

$$\mathbf{Adv}^{FG}(\mathcal{A}) \leq 2 \times \mathbf{Adv}^{IND\text{–}CCA2}(\mathcal{B})$$

Note that the *IND-CCA2* property of the encryption scheme allows to obtain a security bound which is independent from η_c. Also, q_H does not appear in the result meaning that calls to $\mathcal{H}(\cdot)$ does not help the adversary in anyway.

Moreover, *IND-CCA2* guarantees that for all adversaries $\mathcal{B}$, $\mathbf{Adv}^{IND\text{–}CCA2}(\mathcal{B})$, and consequently $\mathbf{Adv}^{FG}(\mathcal{A})$, becomes *negligible* in η_k, the size of the key. We now refine and instantiate this bound to select the necessary trade-off between the desired level of security and the mandatory minimization of the message overhead.

Illustrative Example. As an example, we now show how to fix parameters when focusing on the *S-MAC* protocol [18] for which the data packet length can be up to 250 bytes.

Recall that we want to achieve the *IND-CCA2* and *INT-PTXT-secure* properties. Such properties can be ensured using an *Encrypt-then-MAC* primitive [15]. *Encrypt-then-MAC* consists of an encryption primitive, *e.g.*, AES-128 [24],[4] and a *Message Authentication Code (MAC)* [25], *e.g.*, HMAC-SHA-1 or HMAC-SHA-256.[5]

Here, we use AES-128 [24], which outputs 128 bits (16 bytes) of ciphertext using an input plaintext of 128 bits. To overcome the size limitation, we use AES-128 together with a block cipher mode of operation

[4]AES stands for *Advanced Encryption Standard*.

[5]HMAC-SHA stands for Keyed-Hash Message Authentication Code and Secure Hash Algorithm.

called *Cipher Block Chaining (CBC)* [26]. This latter allows to securely link together several fixed-length ciphertexts, so-called blocks. Using AES, each block will be of size 128 bits (16 bytes), and the overhead of CBC is one block (the initialization vector), so 16 bytes. We fix here the size of the plaintext to 208 bytes, *i.e.*, 13 blocs of 16 bytes. So, using AES-128 together with CBC, we obtain a ciphertext constituted of 14 blocs of 16 bytes, *i.e.*, 224 bytes.

Then, for the MAC, we use HMAC-SHA-256 which, given an input of up to 2^{64} bits, produces an output *message authentication code* (MAC) of 256 bits (32 bytes). However to reduce the overhead, we truncate this output to 9 bytes, *i.e.* 72 bits. Overall, we obtain a ciphertext of 233 bytes.

Then, using these parameters and several results from [27, 28, 29, 15], we obtain the following bound (see [23] for details):

$$\begin{aligned}\mathbf{Adv}^{FG}(\mathcal{A}) \quad \leq \quad & 4 \times \mathbf{Adv}^{RKA}_{\texttt{comp-SHA-256}^*}(\mathcal{M}) + 4 \times \mathbf{Adv}^{PRF}_{\texttt{comp-SHA-256}}(\mathcal{N}) + \\ & 2 \times (q_V - 1) \times q_V \times \big(8 \times \mathbf{Adv}^{PRF}_{\texttt{comp-SHA-256}}(\mathcal{O}) + \frac{1}{2^{256}}\big) + \\ & \frac{q_V}{2^{70}} + 4 \times \mathbf{Adv}^{PRF}_{\texttt{AES}}(\mathcal{I}) + \frac{196 \times (q_G + 1)^2}{2^{127}}\end{aligned}$$

where

- `comp-SHA-256` (resp. `comp-SHA-256`*) is the compression function of SHA-256 (resp. its dual function);
- the PRF-advantage $\mathbf{Adv}^{PRF}_{F}(\mathcal{X})$ measures the ability of an adversary $\mathcal{X}$ to guess whether a given oracle is a random instance of F, a family of pseudorandom functions, or a truly random function;
- $\mathbf{Adv}^{RKA}_{F}(\mathcal{X})$ is the PRF-advantage of $\mathcal{X}$ under related key attacks;
- $\mathcal{I}$ runs in time $O(T_{\mathcal{B}})$ and makes $14 \times (q_G + 1)$ queries to the encryption oracle, modeling the encryption function of AES;
- $\mathcal{M}$ is a related key adversary that performs two oracle queries and has time $O(T_{\mathcal{B}})$;
- $\mathcal{N}$ makes q_V queries and runs in $O(T_{\mathcal{B}})$ time;
- $\mathcal{O}$ makes 2 queries and runs in $O(T)$, T being the time for one computation of `comp-SHA-256`.

We assume $q_V = 2^{20}$ and $q_G = 2^{30}$. Finally, we bound the strength of the adversaries using estimations based on the current best attacks on `AES` ($2^{126.1}$) and `comp-SHA-256` (2^{256}):

- For AES, if the attacker can make $N_{\texttt{AES}}$ queries, its advantage can be estimated by $\frac{N_{\texttt{AES}}}{2^{126.1}}$.
- For SHA-256, if the attacker can make $N_{\texttt{SHA}}$ queries to the compression function, then the advantage of the attacker can be estimated by $\frac{N_{\texttt{SHA}}}{2^{256}}$.

We assume $N_{\texttt{AES}} \leq 2^{70}$ and $N_{\texttt{SHA}} \leq 2^{100}$. Hence, we obtain $\mathbf{Adv}^{FG}(\mathcal{A}) \leq 2^{-49}$.

3.3.2 Nonce Confidentiality

In the next game, we evaluate whether an adversary can extract a nonce from an undelivered message. Let $\mathcal{A}$ be an adversary running in two phases ($\mathcal{A}_1$ and $\mathcal{A}_2$) that communicate using a variable named *state*. The game consists in giving a challenge data message $\langle C, h, src\rangle$ to an adversary $\mathcal{A}$, who should guess the nonce inside this message in at most nb_A tries. To do this, the adversary is allowed to call $Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot)$, $\mathcal{H}(\cdot)$, and $Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)$. Moreover, it chooses the data that will be contained in the challenge message.

```
Experiment $\mathbf{Expt}^{N\text{–}conf}(\mathcal{A})$ :
    $k_{src} \xleftarrow{\$} \mathcal{K}$
    $(Data, state) \xleftarrow{\$} \mathcal{A}_1^{Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot),\mathcal{H}(\cdot),Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)}()$
    $(\langle C, h, src\rangle, N) \xleftarrow{\$} Gen^{\mathcal{E}(k_{src},\cdot)}(Data)$
    $Answers \leftarrow \mathcal{A}_2^{Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot),\mathcal{H}(\cdot),Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)}(\langle C, h, src\rangle, state))$
    Return $(|Answers| \le nb_A \wedge N \in Answers)$
```

The nonce confidentiality advantage of $\mathcal{A}$ against *N–conf* is defined as the probability of winning the game, *i.e.*, $Pr[\mathbf{Expt}^{N\text{–}conf}(\mathcal{A}) = 1]$:

$$\mathbf{Adv}^{N\text{–}conf}(\mathcal{A}) = Pr[\mathbf{Expt}^{N\text{–}conf}(\mathcal{A}) = 1]$$

CryptoVerif outputs that for all adversaries $\mathcal{A}$:

- making q_G queries to $Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot)$, q_V queries to $Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)$, q_H queries to $\mathcal{H}(\cdot)$, and nb_A tries in the *N–conf* game, and
- running the *N–conf* game in $T_{\mathcal{A}}$ times units,

there exists an adversary $\mathcal{B}$:

- making $q_G + 1$ queries to $\mathcal{E}(k_{src},\cdot)$ and q_V queries to $\mathcal{E}^{-1}(k_{src},\cdot)$ in the *IND-CCA2* game, and
- running the *IND-CCA2* game in $T_{\mathcal{B}}$ time units with $T_{\mathcal{B}} = T_{\mathcal{A}} + P_2(q_G, q_V, s)$ time units, where $P_2(q_G, q_V, s)$ is polynomial in q_G, q_V, and the message size s (see [23] for details),

such that:

$$\mathbf{Adv}^{N\text{–}conf}(\mathcal{A}) \le \frac{nb_A + q_H + q_G}{2^{\eta_n}} + \mathbf{Adv}^{IND\text{–}CCA2}(\mathcal{B})$$

Similarly to the previous property, this bound becomes negligible when increasing η_n and η_k. Again, one can find the right values to obtain a given security bound.

Illustrative Example. Considering the encryption parameter values already fixed in Subsections 3.3.1 and the results from [27, 28, 29, 15], we obtain the following bound (see [23] for details):

$$\begin{aligned}\mathbf{Adv}^{N\text{–}conf}(\mathcal{A}) \quad \le \quad & \frac{nb_A + q_H + q_G}{2^{\eta_n}} + 2 \times \mathbf{Adv}^{RKA}_{\texttt{comp-SHA-256}^*}(\mathcal{M}) + \\ & 2 \times \mathbf{Adv}^{PRF}_{\texttt{comp-SHA-256}}(\mathcal{N}) + \\ & (q_V - 1) \times q_V \times \big(8 \times \mathbf{Adv}^{PRF}_{\texttt{comp-SHA-256}}(\mathcal{O}) + \frac{1}{2^{256}}\big) + \\ & \frac{q_V}{2^{71}} + 2 \times \mathbf{Adv}^{PRF}_{\texttt{AES}}(\mathcal{I}) + \frac{196 \times (q_G + 1)^2}{2^{128}}\end{aligned}$$

($\mathcal{I}$, $\mathcal{M}$, $\mathcal{N}$, $\mathcal{O}$ are the same adversaries as those defined in Subsection 3.3.1.)

To obtain an estimation, we use the same values as in Subsection 3.3.1 and we assume $nb_A = q_V = 2^{20}$ and $q_H = 2^{40}$. Finally, we fix the size η_n of the nonce to 96 (12 bytes). Hence, we obtain $\mathbf{Adv}^{N\text{–}conf}(\mathcal{A}) \le 2^{-49}$, using a ciphertext of length 233 bytes and a nonce of 12 bytes.

3.3.3 Unforgeability

Finally, the last game evaluates the unforgeability, *i.e.*, the ability of an intruder to create a new valid ciphertext. Note that this property implies both *indistinguishability* and *authenticity* of the data. To evaluate unforgeability, the game gives an attacker $\mathcal{A}$ access to both $Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot)$ and $Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)$. To win, $\mathcal{A}$ should return a ciphertext which is valid and which has never been encrypted by $Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot)$. Recall that the set $Queries$ contains the ciphertext encrypted by $Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot)$.

```
$Experiment\ \mathbf{Expt}^{UF\text{–}CMVA}(\mathcal{A})$
    $Queries \leftarrow \emptyset$
    $k_{src} \xleftarrow{\$} \mathcal{K}$
    $\langle C, h, s\rangle \leftarrow \mathcal{A}^{Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot),\mathcal{H}(\cdot),Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)}()$
    $\mathbf{Return}\ C \notin Queries \wedge Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(C)$
```

The unforgeability advantage of $\mathcal{A}$ against $UF\text{–}CMVA$, noted $\mathbf{Adv}^{UF\text{–}CMVA}(\mathcal{A})$, is defined as the probability of $\mathcal{A}$ winning this game, $Pr[\mathbf{Expt}^{UF\text{–}CMVA}(\mathcal{A}) = 1]$:

$$\mathbf{Adv}^{UF\text{–}CMVA}(\mathcal{A}) = Pr[\mathbf{Expt}^{UF\text{–}CMVA}(\mathcal{A}) = 1]$$

Using CryptoVerif, we find that the advantage of $\mathcal{A}$ depends only the strength of the authenticated encryption. Formally, for all adversaries $\mathcal{A}$:

- making q_G queries to $Gen^{\mathcal{E}(k_{src},\cdot)}(\cdot)$, q_V queries to $Verif^{\mathcal{E}^{-1}(k_{src},\cdot)}(\cdot)$, q_H queries to $\mathcal{H}(\cdot)$ in the $UF\text{–}CMVA$ game, and
- running the $UF\text{–}CMVA$ game in $T_{\mathcal{A}}$ time units,

there exists an adversary $\mathcal{B}$:

- making q_G queries to $\mathcal{E}(k_{src},\cdot)$ and $q_V + 1$ queries to $\mathcal{E}^{-1}(k_{src},\cdot)$ in the $INT\text{-}PTXT$ game, and
- running the $INT\text{-}PTXT$ game in $T_{\mathcal{B}}$ time units with $T_{\mathcal{B}} = T_{\mathcal{A}} + P_3(q_G, q_V, s)$ time units, where $P_3(q_G, q_V, s)$ is polynomial in q_G, q_V, and the message size s (see [23] for details)

such that:

$$\mathbf{Adv}^{UF\text{–}CMVA}(\mathcal{A}) \leq \mathbf{Adv}^{INT\text{–}PTXT}(\mathcal{B})$$

Similarly to the previous properties, $\mathbf{Adv}^{INT\text{–}PTXT}(\mathcal{B})$ becomes negligible when increasing η_k.

Illustrative Example. Considering the encryption parameter values already fixed in Subsections 3.3.1-3.3.2 and the results from [27, 28, 29, 15], we obtain the following bound (see [23] for details):

$$\begin{aligned}\mathbf{Adv}^{UF\text{–}CMVA}(\mathcal{A}) \quad \leq \quad & \mathbf{Adv}^{RKA}_{\texttt{comp-SHA-256}^*}(\mathcal{M}') + \mathbf{Adv}^{PRF}_{\texttt{comp-SHA-256}}(\mathcal{N}') + \\ & \frac{(q_V - 1) \times q_V}{2} \times \big(8 \times \mathbf{Adv}^{PRF}_{\texttt{comp-SHA-256}}(\mathcal{O}') + \frac{1}{2^{256}}\big) + \frac{q_V}{2^{72}}\end{aligned}$$

where

- $\mathcal{M}'$ is a related key adversary that performs two oracle queries and has time $O(T_{\mathcal{B}})$;
- $\mathcal{N}'$ makes q_V queries and runs in $O(T_{\mathcal{B}})$ time; and
- $\mathcal{O}'$ makes 2 queries and runs in $O(T)$, T being the time for one computation of `comp-SHA-256`.

To obtain an estimation, we use the same values as in Subsections 3.3.1-3.3.2. We obtain $\mathbf{Adv}^{UF\text{–}CMVA}(\mathcal{A}) \leq 2^{-51}$.

Overall, we obtain a security level of at least 2^{-49} for each of the three aforementioned properties (unforgeability, confidentiality of the data, and confidentiality of the nonce before message delivery) using a ciphertext of length 233 bytes and a nonce of 12 bytes. As the nonce is concatenated to the data in the plaintext to encrypt (208 bytes), the size of the data is then at most 196 bytes. Note also that the size of the hash has to be fixed. Here, we fix it to 12 bytes (the same size as the nonce), to prevent random collisions. In addition to the ciphertext (233 bytes) and hash (12 bytes), a data message also contains the identifier of the source (4 bytes) and one bit for the message type (`MSG`). Hence, the data messages can be encoded using less than 250 bytes, with a low overhead as up to 196 bytes of data can be stored in, and so are supported by *S-MAC*. Similarly, our acknowledgment messages are supported by *S-MAC* since they can be encoded using 12 bytes for the nonces, 4 bytes for the source identifier, and one bit for the message type `ACK` (hence less that 250 bytes).

3.4 Symbolic Analysis of SR3

We conducted a symbolic analysis of SR3, focusing on authentication, using the tool *Scyther* [17]. Overall, the symbolic analysis focuses more on the protocol than on the cryptography, because of few key differences with the previous section.

First, this analysis is done in the *symbolic* model instead of *computational* model. This model assumes the perfect encryption hypothesis, which specifies that cryptographic primitives are perfect black-boxes, and the attacker can only interact with them through the expected properties: for instance, the attacker can decrypt $\mathrm{E}_k(x)$ if and only if he has knowledge of k. This knowledge is built from a Dolev-Yao model [30], which specifies that the attacker only knows what can be built or deduced from the communications he overhears.

The symbolic model is more restrictive for an attacker than the computational model. Here, we still use the description given in Figure 4 (page 8), but we consider an independent source node. Also, instead of proving security properties for a single session of the protocol, we determine whether the attacker can execute a bounded number of sessions of the protocol to achieve its goal. The attacker can alter, delete, or create messages, based on its current knowledge, and it can also initiate new protocol sessions.

We focus on authentication, more precisely non-injective agreement for both participants. This property is defined in the hierarchy given in [31]. Consider two actors, A and B, running a protocol. If the protocol verifies this property, it guarantees that if A completes a run of the protocol, apparently with B, then B has previously been running the protocol, apparently with A, and both A and B agreed on the same data (in our case, this data is both *Data* and N).

We model the authenticated encryption by decomposing an *Encrypt-then-MAC* primitive into two parts: an encryption and a MAC (*Message Authentication Code*) part. Our modeling for *Scyther* is available online [23] and we experimented it setting the bound on the number of sessions to 100. Under this condition, *Scyther* reported that SR3 provides the aforementioned authentication property for both actors A and B.

4 Experimental Results

In this part, we evaluate our protocol with respect to classical measures, namely, delivery rate of the messages, fairness, and number of hops. We also study the resiliency of SR3 against several attack scenarios. For that purposes, we ran simulations using *Sinalgo* [32], an event-driven simulator for WSNs, and we compared the performances of SR3 to those of six other routing protocols.

4.1 Experimental Conditions

We deployed sensors uniformly at random on a square plane. We positioned the sink at the center of the square plane. The compromised nodes are selected uniformly at random among other sensors. Two nodes can communicate if and only if their Euclidean distance is less or equal to a preset fixed range, *i.e.*, the topology is a Unit Disk Graph (UDG). We only considered *connected* topologies. The communication links are asynchronous and FIFO. To enforce asynchronism (*i.e.*, to maximize interleavings of events), the transmission time of each link follows an exponential random distribution of parameter $\lambda = 1$ (so, the average transmission time is 1). Only honest sensors generate data to route. The time between two consecutive data generations at the same sensor also follows an exponential random distribution, whose parameter λ is the same for all sensors and whose value depends on the average degree $\overline{\delta}$ and the number of sensors n in the network, to prevent congestion, namely $\lambda = \frac{\sqrt{\overline{\delta}}}{10n}$.

If we fix the number of nodes n and the range of the UDG, we can tune the size of the simulation area to control the expected average degree $\overline{\delta}$ of the network. In our simulations, n varies from 50 to 400 and $\overline{\delta}$ varies from 8 to 32. The percentage of compromised nodes varies from 0 to 30%. We considered various attack scenarios, where compromised nodes made *selective forwarding*: each compromised node drops received messages with a probability $p \in (0, 1]$ (if $p = 1$, the node is called a *blackhole*). In addition, some compromised nodes may have some additional "bad" skills, *e.g.*, they may be *wormholes* or *Sybil*. A compromised node is said to be Sybil when it pretends to be multiple, distinct nodes in the system. A wormhole is a compromised node, typically far from the sink, which (temporarily) violates the UDG topology by directly communicating (*via* a fast private medium) with the sink in order to attract the traffic.

For each setting (number of nodes, average degree, attack scenario, amount of compromised nodes, and routing algorithm), we ran simulations over 20 UDGs, randomly generated. In each simulation, 500 000 data are generated. The simulation stops once all messages have been routed or lost. We made more than 13 000 simulation runs and the overall number of generated data is greater than 6 billion.

4.2 List Sizes

SR3 uses three lists, whose respective sizes are bounded. We made several experiments to set the size of each list to the appropriate value. Of course, the size of these three lists are influenced by the network load, which in turn is influenced by the transmission time, drop rate, and data generation time intervals (all those parameters have been set in Subsection 4.1).

We consider first the list $L_{Reputation}$. We experiment several possible values for its size, s_R. In these experiments, we implement $L_{AckRouting}$ and L_{Sent} as infinite lists, which correspond to their ideal (yet impractical) behavior, to prevent any side effect.

When the behavior of malicious nodes is homogeneous over the time, the greater s_R is, the better the delivering rate is. However, if s_R is big, this results in increasing the time required to refresh the content of the list. So, in the case of malicious nodes that suddenly change their behavior (*i.e.*, a sinkhole attack), the system spends more time to recover if s_R is big. In other words, if the behavior of malicious nodes is heterogeneous with time, a large list $L_{Reputation}$ would reduce the adaptivity, and consequently the overall delivery rate, of our algorithm.

To create a sinkhole attack, we use *wormholes*: their ability makes them more attractive than other nodes, as they allow delivering messages faster to the sink. During the first third of the simulation, wormholes send all received data messages in their channel directly connected to the sink and route acknowledgments as honest nodes in order to obtain a high reputation. Next, they become blackholes, *i.e.*, they drop all messages they receive.

The necessary tradeoff to obtain when choosing the value of s_R is illustrated in Figure 6, where results are given for a network of 200 nodes, average degree 16, and 10% of blackholes, and 10% of wormholes/blackholes which become blackholes after the first third of the simulation. For each point (x, y) of the curves, y is the delivery rate computed over a window of 50 000 messages, from the $(x-50000)^{th}$ to the x^{th} emitted message.

We test the aforementioned scenario with several values for s_R — from 5 to 40. For each size, we run simulations on networks of sizes 100, 200, and 400 with average degrees 8, 16, and 32, and containing both 10% of blackholes (BH) and 10% of wormholes/blackholes (WH/BH). We made 10 simulations (each with a different topology) for each setting (list size, number of nodes, degree). The overall results are summarized in Table 1. In each cell of the table, we print the value of s_R that offers the best average delivery rate.

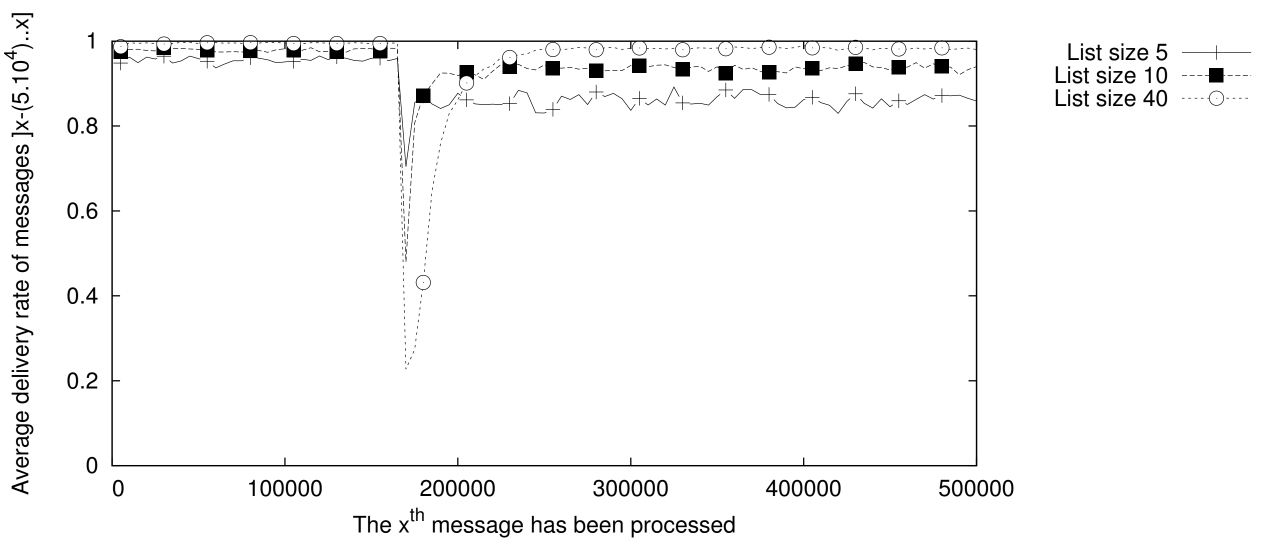


Figure 6: Average delivery rate (10% of WH/BH, 10% of BH, $n = 200$, $\overline{\delta} = 16$)

		Number of nodes		
		100	200	400
Average degree	8	10	5	5
	16	15	10	5
	32	20	15	5

Table 1: Best observed $L_{Reputation}$ size s_R, when facing 10% BH and 10% WH/BH.

An important fact does not appear in Table 1: the average delivery rate is very similar for a large range of list sizes. We chose $s_R = 10$, which appeared to be a good compromise in each setting.

Next, we consider the list L_{Sent}. The size of both L_{Sent} and $L_{AckRouting}$ are bounded for practical reasons only. Indeed, sensors have tight local memories. Now, having infinite sizes for those lists would offer the best behavior. The goal here is to find a reasonable size that achieves the adequate trade-off between performances and resource consumption.

To set the size s_S of L_{Sent}, we led experiments, where $s_R = 10$ (the value we choose previously), but where $L_{AckRouting}$ is still implemented as an infinite list.

In those experiments, our goal is to minimize the proportion of events called *false negatives*. We call false negatives valid acknowledgments that return to their destination, while their corresponding nonce has been removed from L_{Sent}. In that case, the valid acknowledgment is simply discarded.

We made our simulations in safe networks, because this corresponds to the worst case (in terms of number of false negatives), where the routes followed by data messages and acknowledgments are longer.

We tested s_S with values 1, 3, 5, 7, and 9. The results for a particular setting is depicted in Figure 7. Actually we obtained similar results for each setting. We can see value 3 for s_S is sufficient to reach our objective of 99% of accepted acknowledgments. Therefore, we chose to set s_S to 3.

Finally, we repeated the same process to set the size s_A of $L_{AckRouting}$ using the sizes previously selected for $L_{Reputation}$ (10) and L_{Sent} (3). Again, we tried to minimize the proportion of false negatives. If the list is too small, more messages would be randomly routed, which would in turn increase the delay before they reach their destination, and consequently, increase the number of false negatives.

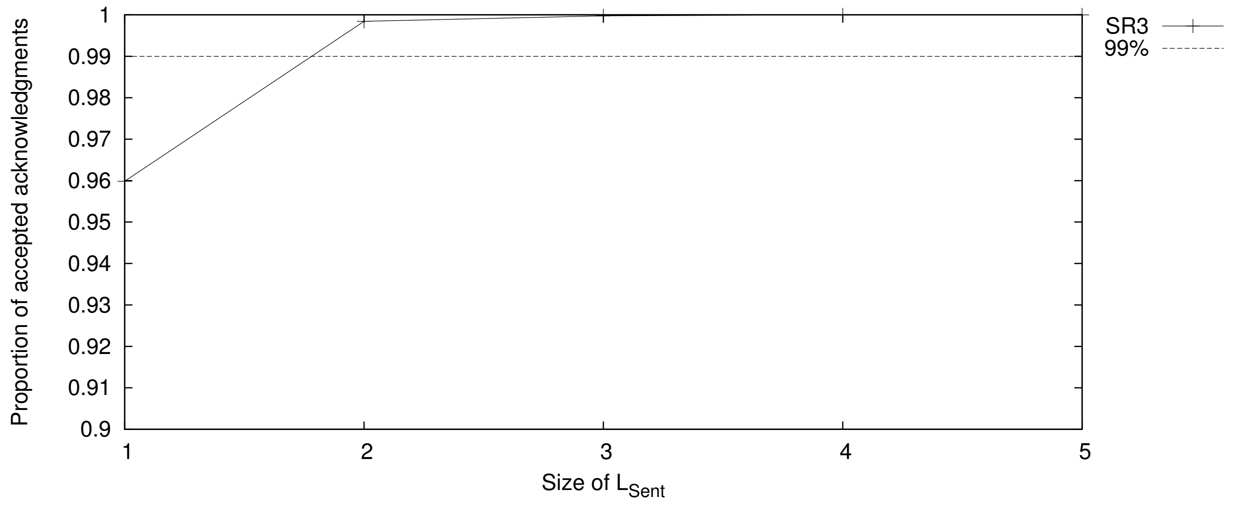


Figure 7: Proportion of accepted acknowledgments, depending on L_{Sent} size ($n = 200$ and $\overline{\delta} = 8$)

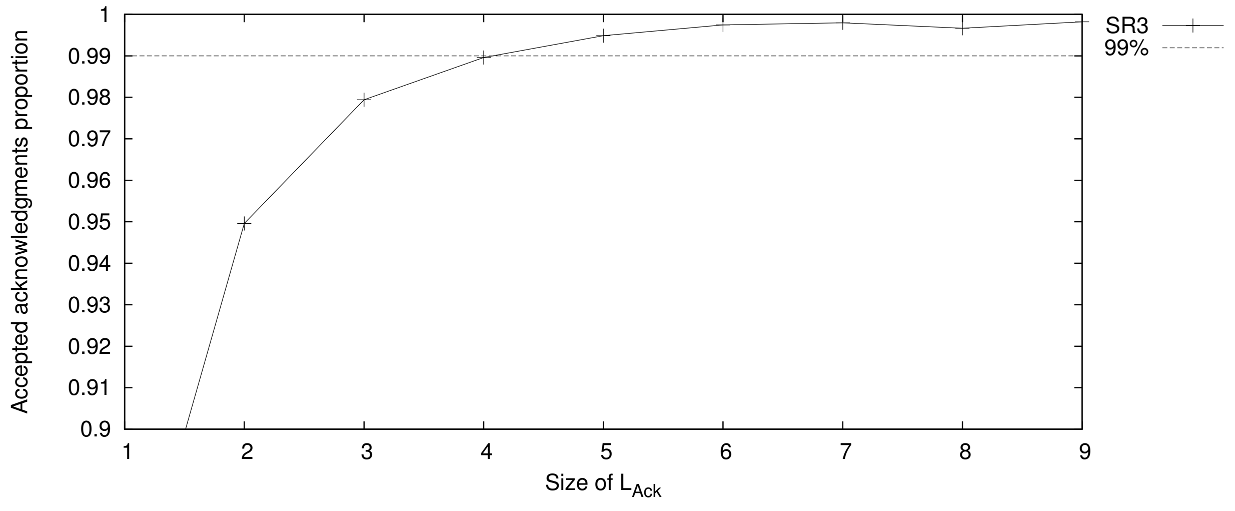


Figure 8: Proportion of valid acknowledgments, depending on $L_{AckRouting}$ size ($n = 200$ and $\overline{\delta} = 8$)

Figure 8 is an example of results we obtained, using the same setting as the previous example (200 nodes, average degree 8). Other settings give similar results. The proportion of accepted acknowledgments stagnates from the size 5 for $L_{AckRouting}$, so we chose that value for s_A.

4.3 Benchmark Protocols

Resiliency is the property targeted by our experiments. It has been introduced by Erdene-Ochir *et al* [10]. In this latter paper, they study resiliency of some classical protocols. They propose new routing solutions dedicated to resiliency in [11]. Hence, in the following, we compare SR3 to a panel of six algorithms proposed and/or studied in these two papers [11, 10].

More precisely, in [10], Erdene-Ochir *et al* propose a classification of routing protocols:

- *Topological-based* protocols that use topological information (*e.g.* hop distances) to deterministically determine the routes.
- *Probabilistic* protocols.
- *Geographic* protocols that determine the routes using GPS information.
- *Hierarchical* protocols, such as RPL [33], that split source nodes into different routing roles.

As in [10], we exclude this latter category from our panel because, in our scenario all source nodes play the same role. Again, similarly to [10], we consider one member of each of the three first categories (actually the same as those studied in [10]):

1. The *Gradient-Based Routing (GBR)* [13] is a topological-based protocol which routes messages along a DODAG, this latter being based on hop distances and rooted at the sink. Precisely, a source node forwards all messages it receives to its (preferred) parent in the DODAG, *i.e.*, a neighbor of lowest level in the DODAG.
2. The *Uniform Random Walk (RW)* [12] is a probabilistic protocol in which the message holder selects the next hop destination uniformly at random among its neighbors until the message reaches the sink.
3. The *Greedy-Face-Greedy protocol (GFG)* [14] is a geographic routing protocol where two modes are alternatively used: Greedy and Face. The Greedy mode (which routes according to the smallest geographic distance) is preferably used, but may lead a message to a dead end. In this case, the Face mode allows the message to escape.

Notice that GBR is close to the hierarchical protocol RPL [33], since RPL is also based on a DODAG rooted at the sink.

We also compare SR3 to the three resilient solutions proposed by Erdene-Ochir *et al* in [11]. These solutions are respectively called *RGBR*, *PRGBR*, and *PRDGBR* in the following. These three protocols are actually variants of the RGB protocol, where some uncertainty is introduced using randomness to make them less predictable by an adversary.

4. *RGBR* uses the levels of neighbors in the DODAG: each sensor chooses next hop for each message uniformly at random among its lowest-level neighbors.
5. In *PRGBR*, each sensor chooses between two modes: (1) with probability 0.4 the message is routed according to RGBR; (2) with probability 0.6 the message is routed to a neighbor of same level (if no such a neighbor exists, the sensor uses mode (1)).
6. *PRDGBR* duplicates the message at each hop and routes the two messages independently using PRGBR. To avoid congestion, each node drops the received copies of messages it already saw.

Notice that an identical approach has been recently used to propose resilient variants of RPL [34].

4.4 Some Scenarios and Results

4.4.1 Average Delivery Rate

Figures 9-11 show the delivery rates observed in networks of average degree $\overline{\delta} = 8$, 16, and 32, facing 30% of blackholes (BH). The size of the networks varies from 50 to 400 nodes. (Note that, with 30% of blackholes, several honest nodes cannot safely reach the sink and consequently have delivery rate zero.) We remark that SR3 always offers a better delivery rate than the other protocols on networks of average degrees 8 and 16. In networks of average degree 32, its delivery rate is approximately the same as PRDGBR, while still better than the other protocols. In particular, the greater the networks are, the greater the gap is.

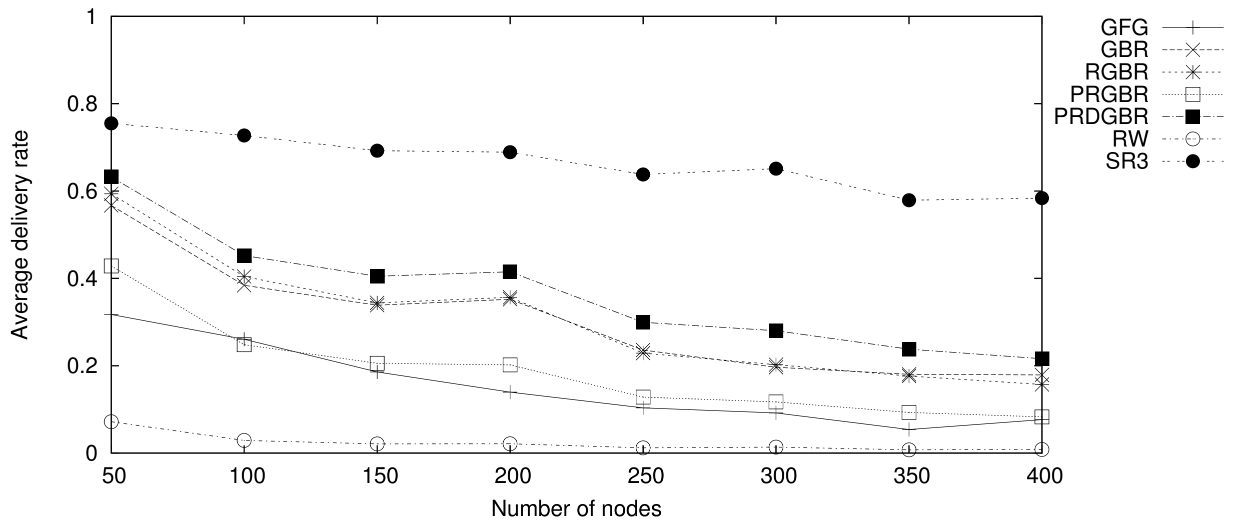


Figure 9: Average delivery rate (30% of BH nodes, $\overline{\delta} = 8$)

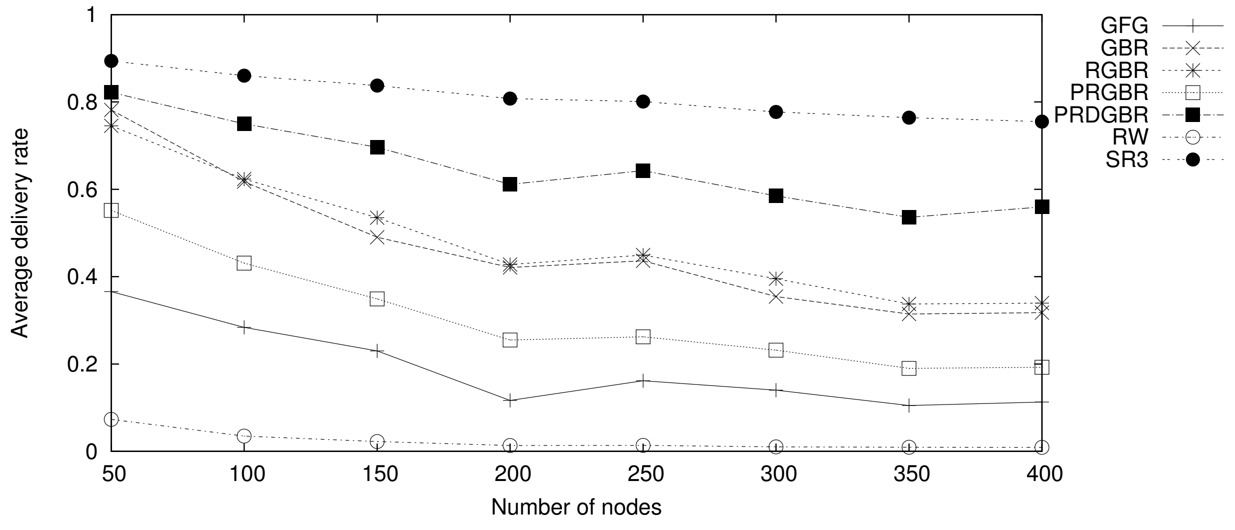


Figure 10: Average delivery rate (30% of BH nodes, $\overline{\delta} = 16$)

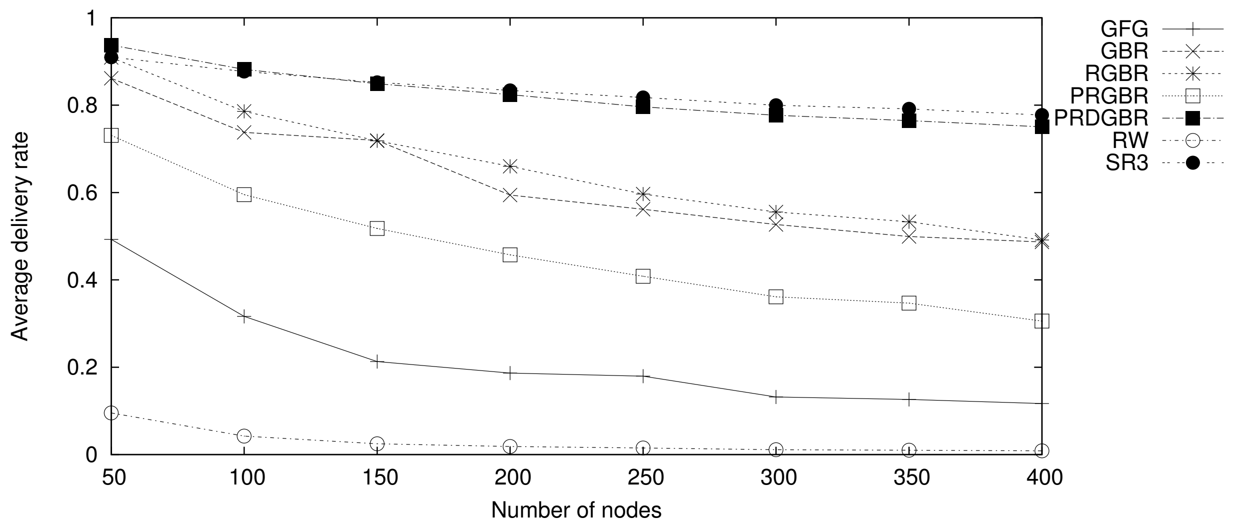


Figure 11: Average delivery rate (30% of BH nodes, $\overline{\delta} = 32$)

Figure 12 shows the delivery rates observed in networks of size $n = 200$ facing 30% of blackholes (BH). The average degree of the networks varies from 8 to 32. Again, we can remark that SR3 always offers the best delivery rate in that case. Moreover, as for RW and GFG, the average delivery rate of SR3 is insensitive to the degree variation. In contrast, the observed delivery rates for gradient-based protocols are low in sparse networks. In high-density networks, the performances of PRDGBR match those of SR3. However, SR3 use only two messages per data, while PRDGBR duplicates the messages at each hop, and consequently heavily increases the load of the network.

SR3 also efficiently combats the selective forwarding (SF) attacks. Figure 13 shows the average delivery rates observed in networks of size $n = 200$ and average degree $\overline{\delta} = 8$ that have to face 20% of compromised nodes, according to the drop rate of these nodes. We can observe that, except RW, all protocols of the panel achieve a graceful degradation in delivery rate when the drop rate increases. Still, SR3 offers one of the best performance. Only PRDGBR has performances close to those of SR3 when the drop rate is 100% (that is, when compromised node are actually blackholes). But again, this performance comes at the price of a high communication overhead.

We also considered networks of size $n = 200$ and average degree $\overline{\delta} = 8$, where 10% of nodes are both blackholes and Sybil (SY). The number of pseudonymous identifiers of these compromised nodes varies from 1 to 10. We can observe in Figure 14, that except for GFG, adding Sybil nodes does not change the relative performances in the panel. Actually, GFG is insensitive to Sybil attacks because it does not use node identifiers. Now, still in that case, SR3 offers the best performances.

4.4.2 Fairness

Fairness among the delivery rates of honest nodes is a desired property in routing protocols. A classical way to capture this property is to compute the standard deviation of the delivery rates of honest nodes [35]. Figure 15 shows the average and standard deviation of delivery rates observed in networks of size $n = 200$ and average degree $\overline{\delta} = 32$, when facing 30% blackholes. The smaller the standard deviation is, the fairer the algorithm is. Now, a shortcoming of this measure is that when the delivery rates are uniformly bad (like for example in RW), the observed fairness is good. So, analyzed alone, this measure is misleading.

Instead, we propose here to visualize the distributions of delivery rates. Figure 16 shows an example of our method. In this figure, we consider the same simulations as in Figure 15. There is one column per

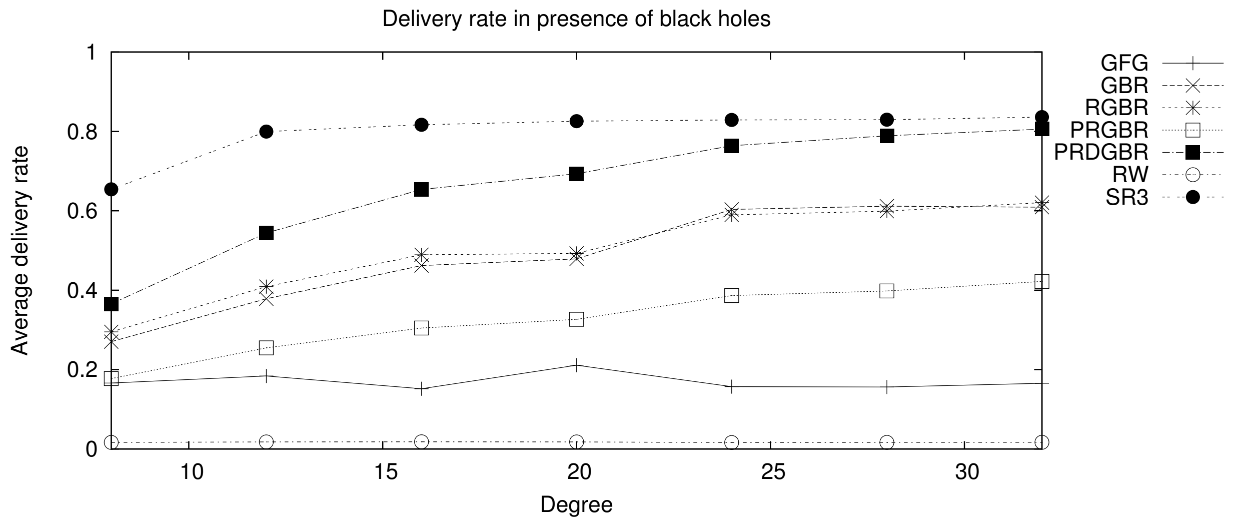


Figure 12: Average delivery rate (30% of BH nodes, $n = 200$)

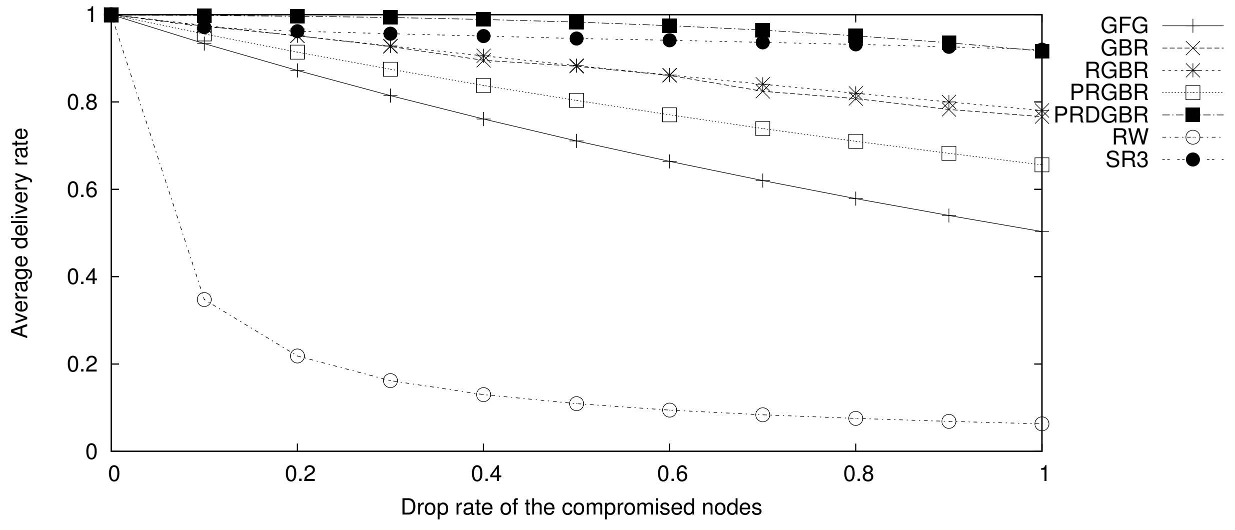


Figure 13: Average delivery rate (20% of SF nodes, $n = 200$, $\overline{\delta} = 8$)

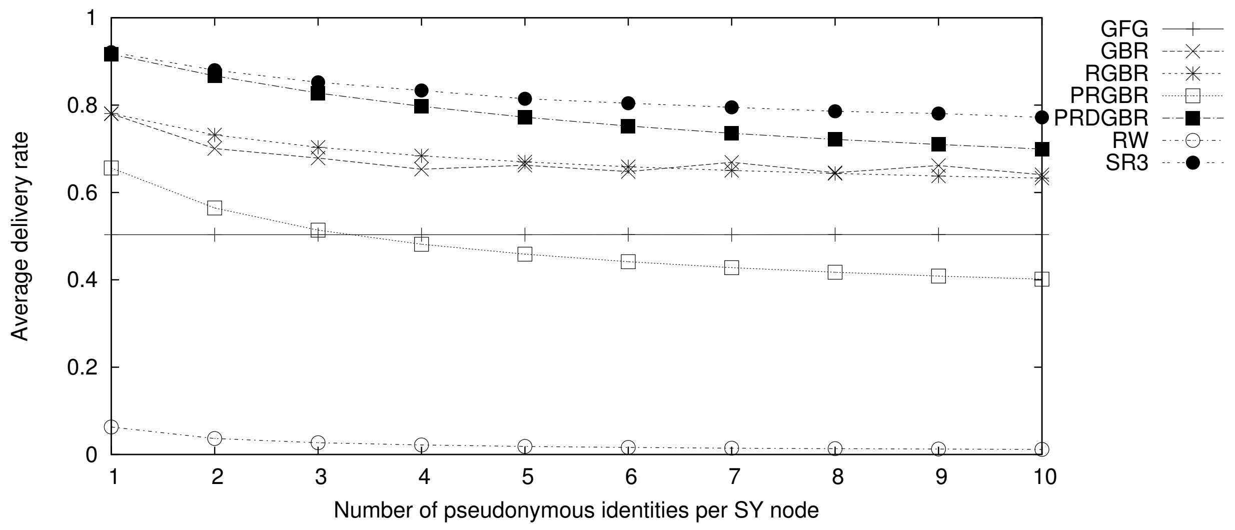


Figure 14: Average delivery rate (10% of SY nodes, $n = 200$, $\overline{\delta} = 8$)

Algorithm	Average delivery rate	Standard deviation
GFG	0.117	0.308
GBR	0.487	0.487
RGBR	0.491	0.307
PRGBR	0.306	0.223
PRDGBR	0.750	0.179
RW	0.008	0.017
SR3	0.777	0.060

Figure 15: Average delivery rate and standard deviation of the delivery rate of nodes (30% of BH, $n = 200$, $\overline{\delta} = 32$)

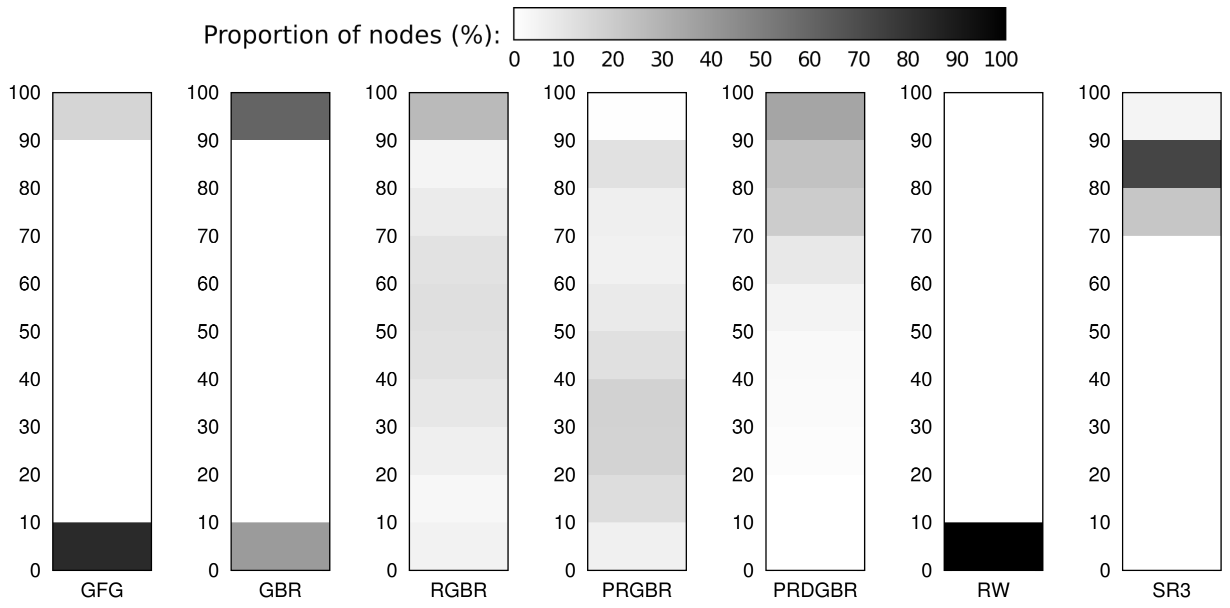


Figure 16: Average delivery rate distribution (30% of BH, $n = 200$, $\overline{\delta} = 32$)

algorithm of the panel. Each column represents the range of possible delivery rates from 0 to 100%, by intervals of 10%. The color shade encodes the proportion of nodes having the corresponding delivery rate. Consider, for example, the RW protocol: almost all nodes have a delivery rate of less than 10%. In contrast, using SR3, almost all nodes have a delivery rate greater or equal to 70%. We can clearly observe two classes of processes when looking at GFG and GBR: nodes have either 0% or 100% of delivery rate; these protocols are unfair. The probabilistic variants of GBR are fairer: the delivery rates are spread on the whole range, but still these results are weaker than those observed for SR3.

We also provide other results in Figure 17. Simulations were run on networks of size $n = 200$ with an average degree $\overline{\delta} = 8$, also when facing 30% blackholes. Overall, we observe results similar to the previous setting, with a few differences. First, GBR and the variants have overall lower deliver rate and fairness, as they behave better in highly connected networks. Also, in this special case, there are more nodes which lost all of their messages, as there is no guarantee for the existence of safe paths for all sources.

4.4.3 Average Number of Hops

Here, we are only interested in the messages that are successfully delivered. So, we consider safe networks. Figures 18-20 show the average number of hops of data messages in networks of average degree respectively $\overline{\delta} = 8$, 16 and 32, where the size n varies from 50 to 400. First, note that we do not show results for RW in the figure because they are drastically worse than other protocols of the panel, *e.g.*, for 50 nodes and $\overline{\delta} = 16$, its average number of hops is 40, and for 400 nodes and $\overline{\delta} = 16$, its average number of hops is 529. Then, by definition, routes followed using GBR or RGBR are optimal. Finally, SR3 generates longer routes than the geographical and gradient-based protocols due to its lack of knowledge about the network. However, this length stays reasonable (*i.e.* we always observed lengths drastically smaller than n), and scales with the number of nodes. Note that Greedy-Face-Greedy does not behave well in some low-degree graphs, this is due to the existence of dead ends in those graphs.

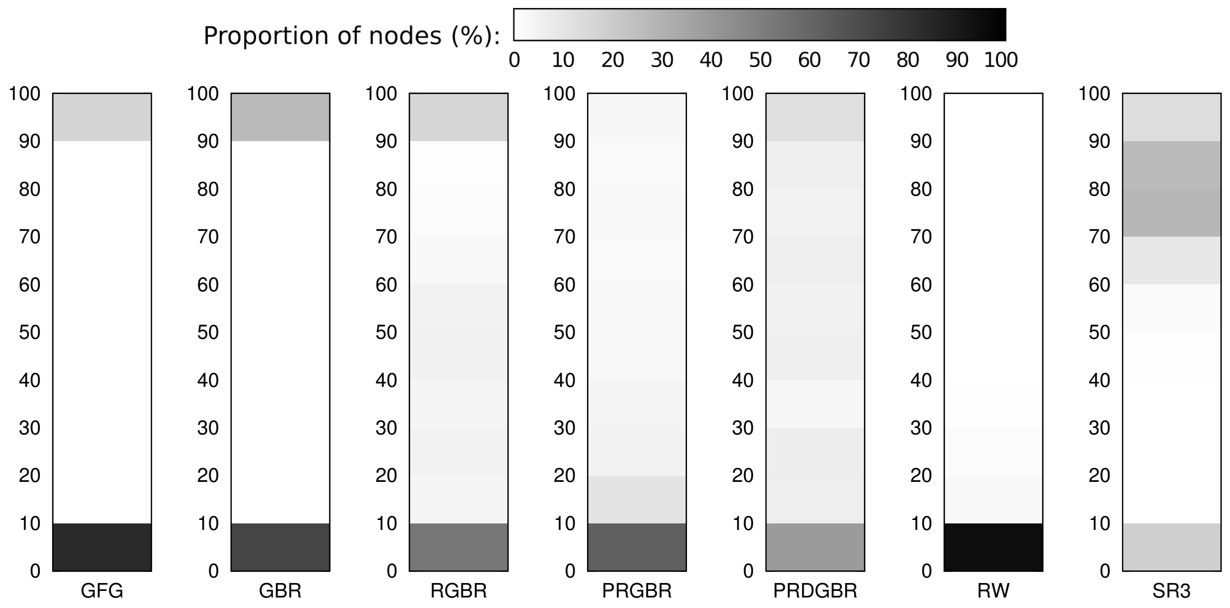


Figure 17: Average delivery rate distribution (30% of BH, $n = 200$, $\overline{\delta} = 8$)

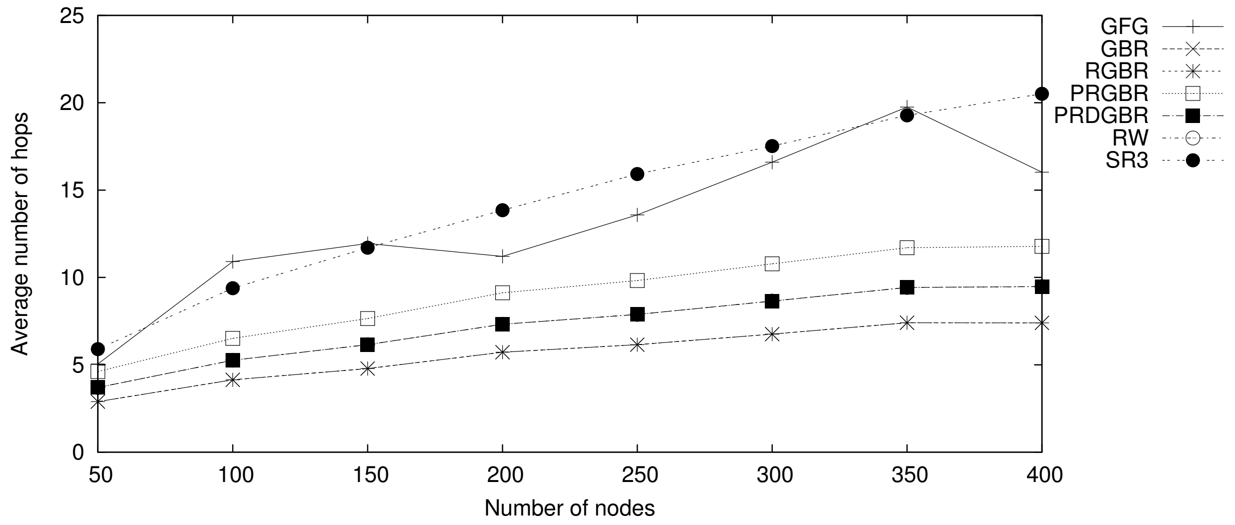


Figure 18: Average number of hops in safe networks ($\overline{\delta} = 8$)

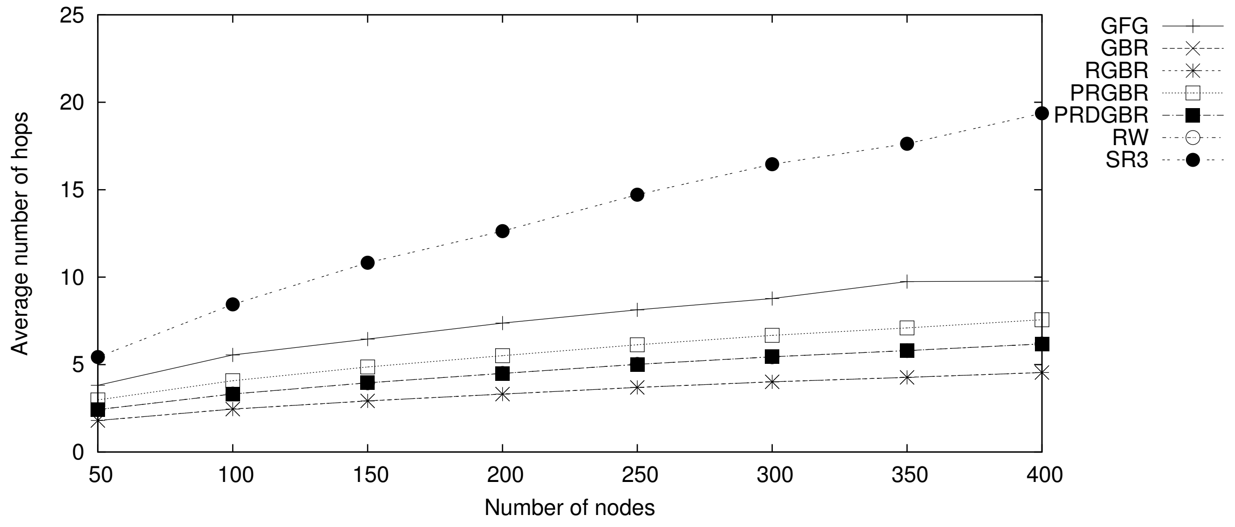


Figure 19: Average number of hops in safe networks ($\overline{\delta} = 16$)

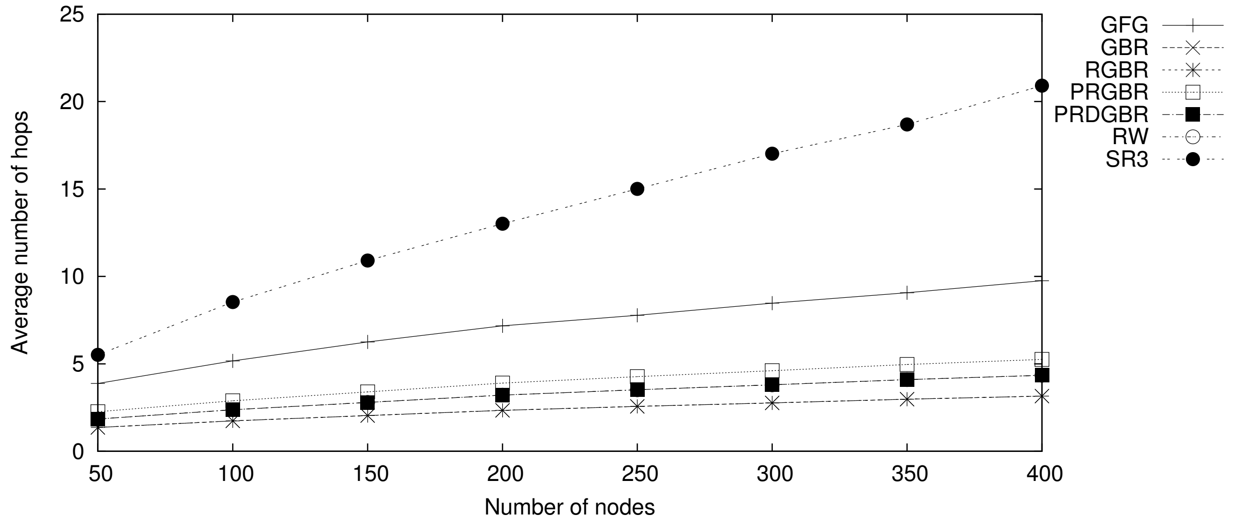


Figure 20: Average number of hops in safe networks ($\overline{\delta} = 32$)

4.4.4 Self-adaptativity

Thanks to its reputation mechanism, SR3 self-adapts to the variations of the hostile environment. To see this, consider the following scenario: in a network of $n = 200$ nodes with average degree $\overline{\delta} = 8$, we assume 5% of blackholes and 5% of wormholes/blackholes (WH/BH), that first behave as wormholes to attract the traffic, and then become blackholes after one third of the simulation. Such nodes appear more attractive to their neighbors because they allow delivering messages faster. Figure 21 shows the evolution of the delivery rates of each protocol: for each point (x, y) of the curves, y is the delivery rate computed over a window of 10 000 messages, from the $(x - 10000)^{th}$ to the x^{th} emitted message. Only SR3 recovers from this attack.

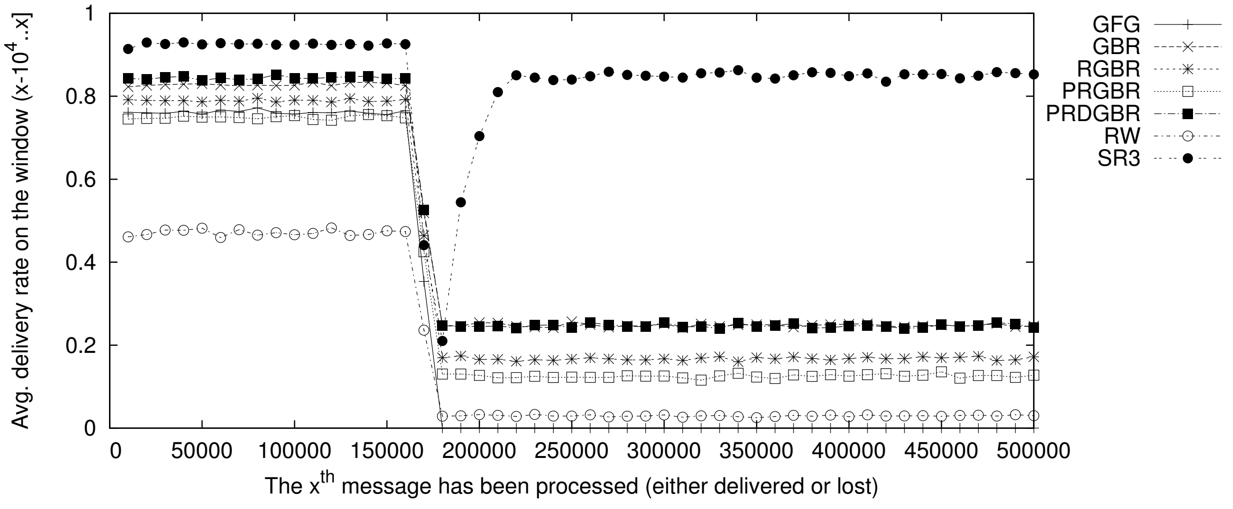


Figure 21: Average delivery rate (5% of WH/BH, 5% of BH, $n = 200$, $\overline{\delta} = 8$)

We show in Figure 22 the average delivery rate distribution in the same setting as previously. We see that the deterministic protocols cause nodes to deliver either none, one third or all their messages. The randomization used in the GBR variants causes a more spread out distribution, but there is still a large concentration of nodes delivering between 30 and 40 percent of their messages. Finally, as observed before, SR3 is fair: most nodes deliver more than 70 percent of all their messages, whereas for the other protocols managing to deliver some data, the delivery rates of the nodes in this scenario strongly depend on their position in the network.

5 Concluding Remarks

We proposed SR3, a secure and resilient algorithm for convergecast routing in wireless sensor networks. Using lightweight cryptographic primitives, SR3 achieves data confidentiality and unforgeability. Using simulations, we showed the resiliency of SR3 in various attack scenarios, including selective forwarding, blackhole, wormhole, and Sybil nodes. The comparative study shows that the resiliency accomplished by SR3 is drastically better than the one achieved by several routing protocols of the literature, even those whose targeted metric is resiliency.

The immediate perspective of this work is to study the performances of SR3 in a more dynamic environment, *e.g.*, networks with mobile nodes or networks where nodes are added/removed on the fly.

An implementation of SR3 in a WSN testbed platform is currently being finalized, the preliminary results are promising, and it should be ready soon for real-world experimentations.

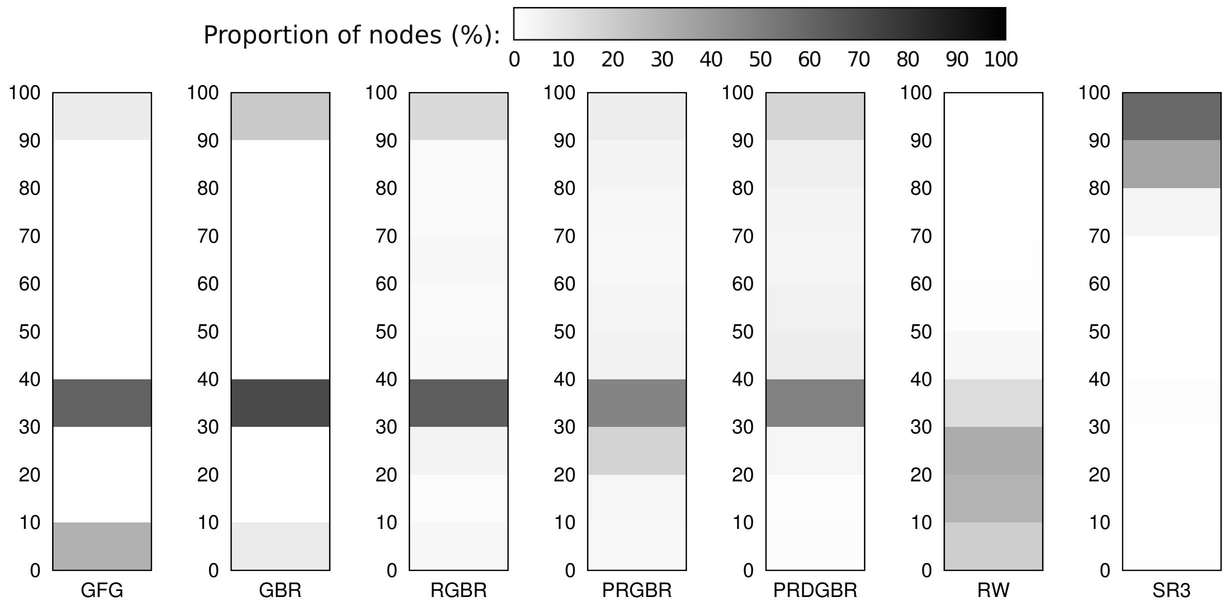


Figure 22: Average delivery rate distribution (5% of WH/BH, 5% of BH, $n = 200$, $\overline{\delta} = 8$)

Acknowledgments

The authors are grateful to Bruno Blanchet for his meticulous reading of the paper and his numerous suggestions.

References

[1] K. Altisen, S. Devismes, R. Jamet, and P. Lafourcade. SR3: Secure resilient reputation-based routing. In *Distributed Computing in Sensor Systems (DCOSS), 2013 IEEE International Conference on*, pages 258–265, 2013.

[2] T. Eisenbarth and S. Kumar. A survey of lightweight-cryptography implementations. *Design & Test of Computers, IEEE*, 24(6):522–533, 2007.

[3] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[4] Victor S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology — CRYPTO '85 Proceedings*, volume 218, pages 417–426, 1986.

[5] A. Perrig, R. Szewczyk, JD Tygar, V. Wen, and D.E. Culler. Spins: Security protocols for sensor networks. *Wireless networks*, 8(5):521–534, 2002.

[6] Donggang Liu and Peng Ning. Multilevel µtesla: Broadcast authentication for distributed sensor networks. *ACM Transactions in Embedded Computing Systems (TECS)*, 3:800–836, 2004.

[7] P. Papadimitratos and Z.J. Haas. Secure Routing for Mobile Ad hoc Networks. In *Proceedings of the SCS Commnication Networks and Distributed Systems Modeling and Simulation Conference (CNDS)*, pages 193–204, 2002.

[8] Y.C. Hu, A. Perrig, and D.B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wireless Networks*, 11(1-2):21–38, 2005.

[9] Ochirkhand Erdene-Ochir, Marine Minier, Fabirce Valois, and Apostolos Kountouris. Resiliency of wireless sensor networks: Definitions and analyses. In *Telecommunications (ICT), 2010 IEEE 17th International Conference on*, pages 828–835, 2010.

[10] Ochirkhand Erdene-Ochir, Marine Minier, Fabrice Valois, and Apostolos Kountouris. Toward resilient routing in wireless sensor networks: Gradient-based routing in focus. In *Proceedings of the 2010 Fourth International Conference on Sensor Technologies and Applications*, SENSORCOMM '10, pages 478–483, 2010.

[11] O. Erdene-Ochir, A. Kountouris, M. Minier, and F. Valois. Enhancing resiliency against routing layer attacks in wireless sensor networks: Gradient-based routing in focus. *International Journal On Advances in Networks and Services*, 4(1 and 2):38–54, 2011.

[12] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science*, pages 218–223, 1979.

[13] C. Schurgers and M. Srivastava. Energy efficient routing in wireless sensor networks. In *Proceedings of MILCOM 2001*, pages 357–361, 2001.

[14] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[15] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptol.*, 21(4):469–491, September 2008.

[16] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008.

[17] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification*, pages 414–418. Springer, 2008.

[18] Wei Ye, John S. Heidemann, and Deborah Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12(3):493–506, 2004.

[19] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73, 1993.

[20] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer, 1991.

[21] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.

[22] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299. Springer Berlin Heidelberg, 2001.

[23] K. Altisen, S. Devismes, R. Jamet, and P. Lafourcade. SR3 supplementary material, December 2013.

[24] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full aes. In *Advances in Cryptology–ASIACRYPT 2011*, pages 344–371. Springer, 2011.

[25] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 1–15, London, UK, UK, 1996. Springer-Verlag.

[26] W.F. Ehrsam, C.H.W. Meyer, J.L. Smith, and W.L. Tuchman. Message verification and transmission error detection by block chaining, February 14 1978. US Patent 4,074,066.

[27] Mihir Bellare. Symmetric encryption. `https://cseweb.ucsd.edu/~mihir/cse207/w-se.pdf`.

[28] Mihir Bellare. New proofs for nmac and hmac: Security without collision-resistance, 2006. An extended abstract of this paper appeared in Advances in Cryptology - Crypto 2006 Proceedings, Lecture Notes in Computer Science Vol. 4117, C. Dwork ed, Springer-Verlag, 2006.

[29] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.

[30] Danny Dolev and Andrew Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[31] Gavin Lowe. A hierarchy of authentication specifications. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 31–43. IEEE, 1997.

[32] Sinalgo. Simulator for network algorithms. http://www.disco.ethz.ch/projects/sinalgo/. Distributed Computing Group at ETH Zurich.

[33] N. Accettura, L.A. Grieco, G. Boggia, and P. Camarda. Performance analysis of the RPL routing protocol. In *Mechatronics (ICM), 2011 IEEE International Conference on*, pages 767 –772, april 2011.

[34] Karel Heurtefeux, Ochirkhand Erdene-Ochir, Nasreen Mohsin, and Hamid Menouar. Enhancing RPL resilience against routing layer insider attacks. In Leonard Barolli, Makoto Takizawa, Fatos Xhafa, Tomoya Enokido, and Jong Hyuk Park, editors, *29th IEEE International Conference on Advanced Information Networking and Applications, AINA 2015, Gwangju, South Korea, March 24-27, 2015*, pages 802–807. IEEE Computer Society, 2015.

[35] Ochirkhand Erdene-Ochir, Apostolos A. Kountouris, Marine Minier, and Fabrice Valois. A new metric to quantify resiliency in networking. *IEEE Communications Letters*, 16(10):1699–1702, 2012.

Algorithm 1 SR3 for any source node v

```
Input: $k_{vs}$: the key of node $v$, shared with the sink $s$
Variables:
  $L_{Sent}$: List of at most $s_S$ pairs, initially empty
  $L_{AckRouting}$: List of at most $s_A$ pairs, initially empty
  $L_{Reputation}$: List of at most $s_R$ elements, initially empty
On generation of $Data$
 1: $N_v \leftarrow$ New_Nonce()
 2: $h \leftarrow \mathrm{H}(N_v)$
 3: $C \leftarrow \mathrm{E}_{k_{vs}}(\langle Data, N_v\rangle)$
 4: $next \leftarrow \textsc{Rand}(\mathcal{N}eig_v, \mathcal{L}^v_{SR3}(L_{Reputation}))$
 5: $L_{Sent} \leftarrow L_{Sent} \odot \langle N_v, next\rangle$
 6: Send $\langle \texttt{MSG}, C, h, v\rangle$ to $next$
On reception of $\langle \texttt{MSG}, C, h, o\rangle$ from $f$
 7: $next \leftarrow \textsc{Rand}(\mathcal{N}eig_v, \mathcal{L}^v_{SR3}(L_{Reputation}))$
 8: if $v = o$ then
 9:     if $\mathrm{E}^{-1}_{k_{vs}}(C) \neq \perp$ then
10:         $\langle Data, N_o\rangle \leftarrow \mathrm{E}^{-1}_{k_{vs}}(C)$
11:         $L_{Sent} \leftarrow L_{Sent} \odot \langle N_o, next\rangle$
12:         Send $\langle \texttt{MSG}, C, h, o\rangle$ to $next$
13:     end if
14: else
15:     if $\langle h, \_ \rangle \notin L_{AckRouting}$ then
16:         $L_{AckRouting} \leftarrow L_{AckRouting} \bullet \langle h, f\rangle$
17:     end if
18:     Send $\langle \texttt{MSG}, C, h, o\rangle$ to $next$
19: end if
On reception of $\langle \texttt{ACK}, N_o, o\rangle$ from $f$
20: if $v = o \wedge \langle N_o, \_ \rangle \in L_{Sent}$ then
21:     $first\_hop \leftarrow \textsc{Get}(L_{Sent}, N_o)$
22:     $L_{Reputation} \leftarrow L_{Reputation} \bullet first\_hop$
23:     $L_{Sent} \leftarrow L_{Sent} \setminus \langle N_o, \_ \rangle$
24: else
25:     if $v \neq o$ then
26:         $h \leftarrow \mathrm{H}(N_o)$
27:         if $\langle h, \_ \rangle \in L_{AckRouting}$ then
28:             $next \leftarrow \textsc{Get}(L_{AckRouting}, h)$
29:             $L_{AckRouting} \leftarrow L_{AckRouting} \setminus \langle h, \_ \rangle$
30:         else
31:             $next \leftarrow \textsc{Rand}(\mathcal{N}eig_v, \mathcal{L}^v_{RW})$
32:         end if
33:         Send $\langle \texttt{ACK}, N_o, o\rangle$ to $next$ with probability $\frac{N-1}{N}$
34:     end if
35: end if
```

Algorithm 2 SR3 for the sink s

```
Input: $keys[]$: array of shared keys, indexed on node identifiers
On reception of $\langle \texttt{MSG}, C, h, o \rangle$ from $f$
36: if $\mathrm{E}^{-1}_{keys[o]}(C) \neq \perp$ then
37:     $\langle Data, N_o \rangle \leftarrow \mathrm{E}^{-1}_{keys[o]}(C)$
38:     Deliver $Data$ to the application
39:     Send $\langle \texttt{ACK}, N_o, o \rangle$ to $f$
40: end if
On reception of $\langle \texttt{ACK}, N_o, o \rangle$ from $f$
41: $next \leftarrow \textsc{Rand}(\mathcal{N}eig_s, \mathcal{L}^{s}_{RW})$
42: Send $\langle \texttt{ACK}, N_o, o \rangle$ to $next$ with probability $\frac{N-1}{N}$
```