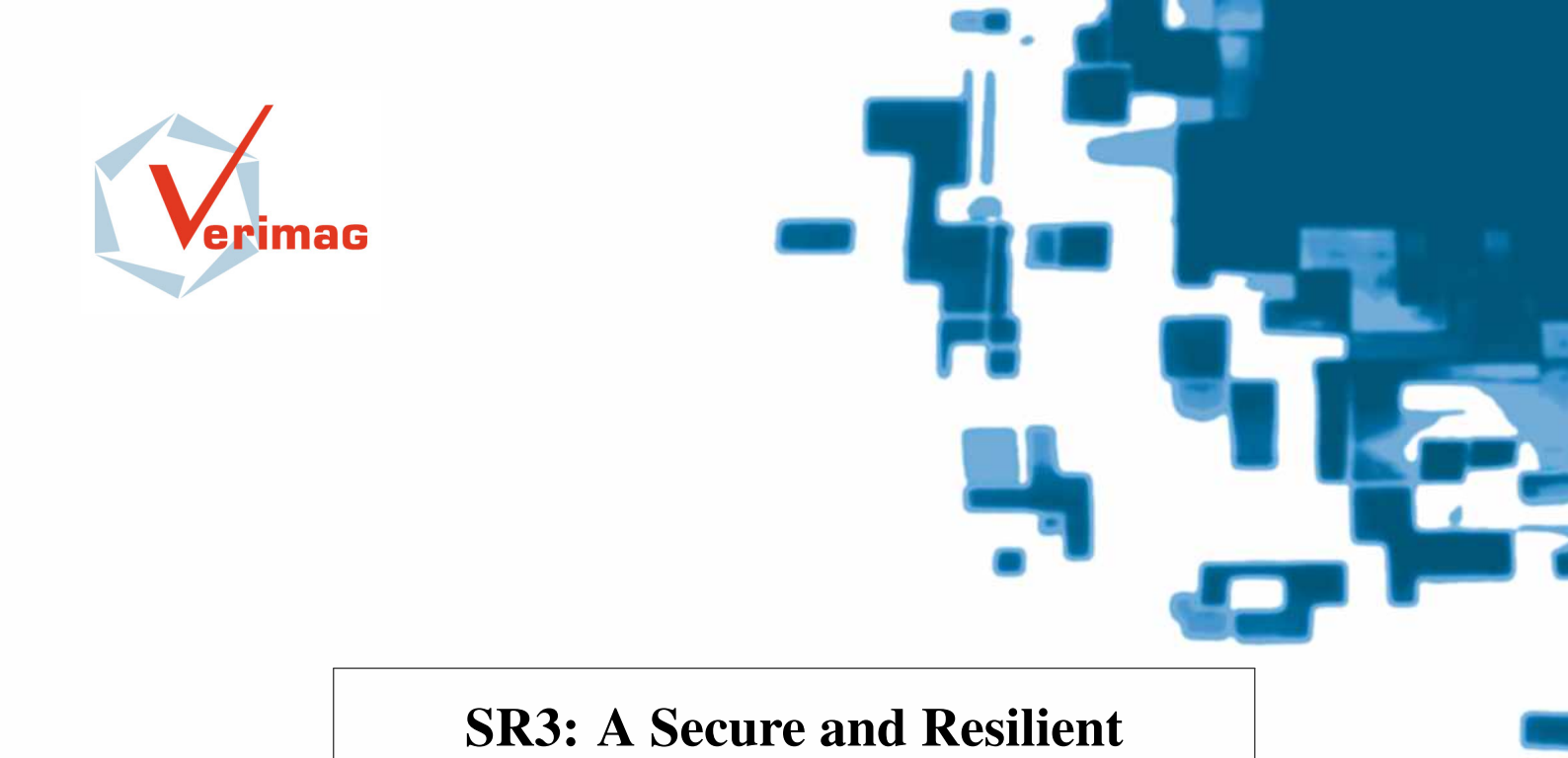

SR3: A Secure and Resilient Reputation-Based Routing Protocol

Karine Altisen, Stéphane Devismes, Raphaël Jamet, Pascal Lafourcade

Verimag Research Report n^o TR-2013-4

April 26, 2013

Reports are downloadable at the following address
http://www-verimag.imag.fr

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

SR3: A Secure and Resilient Reputation-Based Routing Protocol

Karine Altisen, Stéphane Devismes, Raphaël Jamet, Pascal Lafourcade

April 26, 2013

Abstract

We propose SR3, a secure and resilient algorithm for convergecast routing in WSNs. SR3 uses lightweight cryptographic primitives to achieve data confidentiality and data packet unforgeability. SR3 has a security proven by formal tool. 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

Reviewers: Stéphane Devismes

How to cite this report:

```
@techreport {TR-2013-4,
  title = {SR3: A Secure and Resilient Reputation-Based Routing Protocol},
  author = {Karine Altisen, Stéphane Devismes, Raphaël Jamet, Pascal Lafourcade},
  institution = {{Verimag} Research Report},
  number = {TR-2013-4},
  year = {2013}
}
```

Contents

1 Introduction

Nowadays, there is a growing interest in Wireless Sensor Networks (WSNs). WSNs are multi-hop mesh networks made of numerous small battery-powered *sensors* (also called *motes*) 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). Motes communicate using wireless communications and form a large asynchronous connected network. Motes are supplied by batteries, and when those batteries are depleted, nodes die. Hence, the energy consumption of the network is a main concern, since it will determine its lifetime. Also, 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 *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 arbitrary 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. These attacks can rely on a number of techniques, such as forged packets, compromised nodes, cooperating intruder nodes, communications jamming, bogus routing information, and multiple other possibilities [16].

The attacker can impact the routing protocol at two main levels. First, he can attack the data packet to learn secret information — *i.e.*, violate the data *confidentiality*[1] — or make the sink deliver incorrect information — *i.e.*, violate the *authenticity*[2] or *integrity*[3] of the data messages. Secondly, the attacker can affect the routing scheme itself: 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).

Protocols designed for critical applications must be designed to prevent these kinds of malicious actions. The common security tools such as asymmetric cryptography require expensive computations, and so energy. Hence, they are not suitable for WSNs, and so, 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 [11, 22] are mandatory. An example of such a mechanism is *elliptic curve cryptography* [17, 19].

On the other side, some attacks are not critical if the protocol uses mechanisms which mitigates their effects. For instance, when an attacker compromises a few nodes in a large network, the loss of a few packets may be acceptable, as long as most of the information is delivered later on. This characteristic is named *resilience* or *resiliency* in the literature ([18],[12]), and we will use the definition from [12]: resiliency is the capacity of a network to endure and overcome internal attacks. Because of the context, we need to achieve it using power-saving methods.

1.1 Contribution

This report 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 an reinforced random walk that is partially determinized using a reputation mechanism.

SR3 uses lightweight cryptographic primitives — symmetric cryptography, nonces, and hash functions — to achieve several security properties: confidentiality of the data and unforgeability of the data packets, this latter property implies integrity and authenticity of the data packets. We formally prove these properties in the computational model using the tool *CryptoVerif* [7].

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 the fairness (roughly, the standard deviation among the delivery rates of nodes). Our

[1] Confidentiality guarantees that data remain secret between the source and destination.
[2] Authenticity guarantees that the destination is able to detect whether the alleged source in a packet is the truth one.
[3] Integrity guarantees that the destination is able to detect whether the data inside a packet have been modified.

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. 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*,[4] has a low overhead in terms of communications, and does not use any underlying infrastructure, such as spanning tree. Hence, SR3 is well-suited for WSNs.

2 Related work

2.1 Resiliency

Resilience in networks (other than WSNs) is a trending topic, especially for the Internet. Resilinets[5] is a finished project which aimed to better understand and improve the resilience of computer networks, including Internet, wireless networks, SCADA networks, and others. Resumenet [6] is another project, which ended in the beginning of 2012, with a similar focus on resilience, but centered on the Internet. A recent paper from members of both these projects [24] present their work regarding the notion of resiliency, ways to measure it, and how to build networks which are more resilient. Overall, both these projects mainly focused on node or link destructions, from hurricanes to county-wide denial of service. Our study is more focused on ways to resist an active insider attack, which involve malicious behavior.

The notion of *resiliency* has been introduced in [13] as the ability of a network to "continue to operate" 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.

David Wagner considered a different problem: instead of securing the routing, he examined resilient aggregation in WSNs [25]. He evaluated the possible impact of an insider able to insert false data in the network, and then detailed that attacker's influence on the final results depending on the number of compromised nodes and aggregation method.

2.2 Routing in WSNs

There is a lot of routing protocols for wireless sensor networks, which use different techniques. For instance, this survey from 2004 [1] highlights some routing protocols, classified by type. Some of these protocols have been created with security in mind.

Ariadne [15] is an on-demand routing protocol for ad-hoc networks, based on Dynamic Source Routing, where route requests are authenticated and packets are acknowledged. The setting is not exactly similar to ours, as ad-hoc networks are often a many-to-many routing environment. A flaw has been found in this protocol by [9], who provided an amended version named *endairA*, which provably returns existent routes when the network contains at most a single insider.

Although not strictly a routing protocol, SPINS [21] is a set of tools for routing, which provides security guarantees while keeping the resources requirements low enough for WSNs. μTesla, one part of SPINS, is a variant of Tesla[20] using delayed disclosure of symmetric keys in a key chain, which allows authenticated broadcasts without using any asymmetric cryptography. The other part of SPINS, named SNEP, is a packet format that guarantees security properties like authentication and confidentiality using only few additional bits per packets.

However, all aforementioned solutions do not use specific strategies to combat attacks at the routing level, *e.g.*, selected forwarding, blackhole, ... In the same paper where they introduced resiliency, [13, 12]

[4] *I.e.*, in absence of data to route the protocol eventually stops
[5] https://wiki.ittc.ku.edu/resilinets/Main_Page
[6] http://www.resumenet.eu/project/index

proposed three other algorithms based on GBR. 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 breath-first spanning tree is available and not attacked by the insiders. Moreover, they mainly consider dense networks in their simulations, *e.g.*, networks with average degree around 30.

In [2], M. Arnaud *et al* presented a model for ad-hoc routing protocols, which allow to analyze routing protocols on any topology based on constraint solving, and provide decidability and complexity results. Their formal methods were able to find attacks on SRP applied to DSR.

2.3 Roadmap

The remainder of this report is organized as follows. In the next section, we present our routing algorithm, SR3. Section 4 deals with the proof of the security properties of SR3. In section 5, we present the attackers we use in our scenarios. The next section will deal with the other algorithms we compare SR3 against. Section 7 describes the metrics we are going to observe, and Section 8 our experimental protocol. Then, in Section 9, we present experimental results that show the performances and resiliency of SR3. Section 10 is dedicated to concluding remarks.

3 Presentation of our algorithm

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.

3.1 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 secure random number generation). All source nodes share 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.

3.2 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 [13]. 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.

3.3 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 (ENCRYPT($\langle Data, N_v\rangle$,k_{vs}), 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 packet, 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 add a hash of the nonce into the data message (HASH(N_v), Line 2). This way, when receiving a message $\langle \texttt{MSG}, C, H, o\rangle$, the sink can check the integrity of the message by first decrypting C using k_{os} (DECRYPT(C,$keys[o]$), Line 36), and then comparing the hash of the nonce in C to H: if they do not match, the message is simply discarded. Similarly, if a compromised node has modified the plaintext identifier in the message, then the sink will decrypt C with a wrong key, and therefore the hash of the decrypted nonce will not match H.

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_{Queue} the nonce stored in m, together with the identifier of the neighbor to which v sends m (L_{Queue} is appended in Line 5 in 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_{Queue} (see the test $\langle N_o, \_ \rangle \in L_{Queue}$ in Line 20).[7] In that case, v gets back the corresponding neighbor from the list (GET(L_{Queue}, N_o), Line 21), increases its confidence on that neighbor, and removes the record from L_{Queue} ($L_{Queue} \setminus \langle N_o, \_ \rangle$, Line 23). (If v is the destination of the acknowledgment, but N_o does not appear in L_{Queue}, the acknowledgment is simply discarded.)

Due to the memory limitations, L_{Queue} must have a maximum size, s_Q. If a node v has some new data to route and L_{Queue} 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_{Queue}.

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 then the routing process of m is restarted. Since the old entry in L_{Queue} 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.

3.4 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 $L_{Routing}$: 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 $\mathcal{L}^v_{SR3}(L_{Routing})$: Let X be a random variable taking value in $\mathcal{N}eig_v$; $\forall x \in \mathcal{N}eig_v$, the law $\mathcal{L}^v_{SR3}(L_{Routing})$ is defined by:

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

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

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

Informally, when a node needs to route a message, it draws at random a value from $L_{Routing}$ 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 $L_{Routing}$ 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, $L_{Routing}$ is defined as a FIFO list of maximum size, s_R. The insertions in $L_{Routing}$ 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.

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 losing 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.

3.5 Acknowledgment Routing

Let ack be an acknowledgment message. Since ack has been emitted because the corresponding data message m has been successfully delivered by the sink, we can suppose that the path followed by m was safe. Therefore, we can use the bidirectionality of the links to route ack (as much as possible) through the reverse of the path followed by m.

This reverse routing is accomplished by letting a trail along the path followed by m. This trail 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 HASH(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 $L_{Routing}$, *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.

Algorithm 1 SR3 for any source node v

Input: k_{vs}: the key of node v, shared with the sink s

Variables:
L_{Queue}: List of at most s_Q pairs, initially empty
$L_{AckRouting}$: List of at most s_A pairs, initially empty
$L_{Routing}$: List of at most s_R elements, initially empty

On generation of $Data$

$N_v \leftarrow$ NEW_NONCE()
$H \leftarrow$ HASH(N_v)
$C \leftarrow$ ENCRYPT($\langle Data, N_v \rangle$,$k_{vs}$)
$next \leftarrow$ RAND($\mathcal{N}eig_v$,$\mathcal{L}^v_{SR3}(L_{Routing})$)
$L_{Queue} \leftarrow L_{Queue} \odot \langle N_v, next \rangle$
Send $\langle \texttt{MSG}, C, H, v \rangle$ **to** $next$

On reception of $\langle \texttt{MSG}, C, H, o \rangle$ **from** f

$next \leftarrow$ RAND($\mathcal{N}eig_v$,$\mathcal{L}^v_{SR3}(L_{Routing})$)
if $v = o$ **then**
    $\langle Data, N_o \rangle \leftarrow$ DECRYPT(C,k_{vs})
    if HASH(N_o) $= H$ **then**
        $L_{Queue} \leftarrow L_{Queue} \odot \langle N_o, next \rangle$
        Send $\langle \texttt{MSG}, C, H, o \rangle$ **to** $next$
    end if
else
    if $\langle H, \_ \rangle \notin L_{AckRouting}$ **then**
        $L_{AckRouting} \leftarrow L_{AckRouting} \bullet \langle H, f \rangle$
    end if
    Send $\langle \texttt{MSG}, C, H, o \rangle$ **to** $next$
end if

On reception of $\langle \texttt{ACK}, N_o, o \rangle$ **from** f

if $v = o \wedge \langle N_o, \_ \rangle \in L_{Queue}$ **then**
    $first\_hop \leftarrow$ GET(L_{Queue}, N_o)
    $L_{Routing} \leftarrow L_{Routing} \bullet first\_hop$
    $L_{Queue} \leftarrow L_{Queue} \setminus \langle N_o, \_ \rangle$
else
    if $v \neq o$ **then**
        $H \leftarrow$ HASH(N_o)
        if $\langle H, \_ \rangle \in L_{AckRouting}$ **then**
            $next \leftarrow$ GET($L_{AckRouting}, H$)
            $L_{AckRouting} \leftarrow L_{AckRouting} \setminus \langle H, \_ \rangle$
        else
            $next \leftarrow$ RAND($\mathcal{N}eig_v$,$\mathcal{L}^v_{RW}$)
        end if
        Send $\langle \texttt{ACK}, N_o, o \rangle$ **to** $next$ **with probability** $\frac{N-1}{N}$
    end if
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: $\langle Data, N_o\rangle \leftarrow$ DECRYPT(C,$keys[o]$)
37: **if** HASH(N_o)= H **then**
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: $next \leftarrow$ RAND($\mathcal{N}eig_s$,$\mathcal{L}^s_{RW}$)
42: **Send** $\langle \texttt{ACK}, N_o, o\rangle$ **to** $next$ **with probability** $\frac{N-1}{N}$

4 Cryptographic proof of the packet format

4.1 Idea

Our proof works using a simplification of the original protocol. Since legitimate nodes do not change the message nor the acknowledgement while they are routed through the network, we model the protocol with two nodes (the source, and the sink) in an attacker-controled network. We suppose the block cipher is a *PRP-CPA-secure* family of functions, and the hash function is a random oracle.

First, we prove that our protocol is *find-then-guess* (FG) secure. We then prove the *existential unforgeability* (UF) of the packet format, which means that the attacker cannot forge a new valid message without a key. Finally, we prove that an attacker has a negligible probability of being able to extract nonces from a packet which is not delivered.

4.2 Background

We first recall some background on pseudorandom functions, which we use to model the block cipher, and the definition of a random oracle (used to model the hash function).

4.2.1 Pseudorandom Permutations

Let $\mathcal{K}$ be a set of keys, and let $F : \mathcal{K} \times \{0,1\}^{\eta_c} \to \{0,1\}^{\eta_c}$ a family of permutations (and their inverses). For all keys k in $\mathcal{K}$, F_k is a permutation of $\{0,1\}^{\eta_c}$, and F_k^{-1} its inverse. We denote by $Perm$, the set of all possible pairs of a permutation of $\{0,1\}^{\eta_c}$ and their inverses. Since F also contains permutations of $\{0,1\}^{\eta_c}$, we have $\forall k \in \mathcal{K}, (F_k, F_k^{-1}) \in Perm$.

We formalize the attacker as an adversary $\mathcal{A}$, which is a polynomial-time probabilistic Turing machine. We denote $\mathcal{A}^{\mathcal{O}}$ an adversary $\mathcal{A}$ able to query an oracle $\mathcal{O}$. For instance, if $\mathcal{E}_k$ is the encryption oracle, we denote $\mathcal{A}^{\mathcal{E}_k}$ an adversary which can query the oracle $\mathcal{E}_k$ for the encryption of any message m and obtain $\mathcal{E}_k(m)$, without knowing anything about k.

We write $a \xleftarrow{\$} S$ to express the random uniform selection of an element from a set S and its assignation to a.

To measure the ability of an attacker to distinguish between permutations from F and random permutations, we use a *game*, called PRP-CPA (for PseudoRandom Permutation against Chosen Plaintext Attack).

Definition 4.2.1 ($\mathbf{Expt}_F^{PRP\text{–}CPA-1}(\mathcal{D})$ and $\mathbf{Expt}_F^{PRP\text{–}CPA-0}(\mathcal{D})$ [4])**.** *Let $\mathcal{D}$ be an adversary who has access to q oracle queries, $F : \mathcal{K} \times \{0,1\}^{\eta_c} \to \{0,1\}^{\eta_c}$ a function family, and $Perm$ the set of all possible pairs of a permutation of $\{0,1\}^{\eta_c}$ and its inverse. We define $\mathbf{Expt}_F^{PRP\text{–}CPA-1}(\mathcal{D})$ and $\mathbf{Expt}_F^{PRP\text{–}CPA-0}(\mathcal{D})$ as:*

$Experiment\ \mathbf{Expt}_F^{PRP\text{–}CPA-1}(\mathcal{D})$:
$k_{src} \xleftarrow{\$} \mathcal{K}$
$b \xleftarrow{\$} \mathcal{D}^{F_{k_{src}}}()$
Return b

$Experiment\ \mathbf{Expt}_F^{PRP\text{–}CPA-0}(\mathcal{D})$:
$(\mathcal{O}_P, \mathcal{O}_P{}^{-1}) \xleftarrow{\$} Perm$
$b \xleftarrow{\$} \mathcal{D}^{\mathcal{O}_P}()$
Return b

Definition 4.2.1 presents the experiments for the PRP-CPA game. In the first experiment, called $\mathbf{Expt}_F^{PRP\text{–}CPA-1}(\mathcal{D})$, $\mathcal{D}$ is given access to an oracle corresponding to a function drawn from F. In the second experiment, which is called $\mathbf{Expt}_F^{PRP\text{–}CPA-0}(\mathcal{D})$, the adversary has access to an oracle corresponding to a random function drawn from $Perm$.

In this game, the goal of the adversary is to determine from the given oracle which experiment it is currently in. It should answer 1 in $\mathbf{Expt}_F^{PRP\text{–}CPA-1}$, and 0 in $\mathbf{Expt}_F^{PRP\text{–}CPA-0}$. The probability of the adversary being right is called the advantage of $\mathcal{D}$, and we denote it $\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{D})$ (Definition 4.2.2).

Definition 4.2.2 (PRP-CPA advantage $\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{D})$ [4])**.** *Let $\mathcal{D}$ be a polynomial time adversary, $F : \mathcal{K} \times \{0,1\}^{\eta_c} \to \{0,1\}^{\eta_c}$ a function family, and $Perm$ all possible pairs of a permutation of $\{0,1\}^{\eta_c}$ and its inverse. Then the PRP-CPA advantage of $\mathcal{D}$ against F is denoted $\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{D})$, and defined as:*

$$\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{D}) = Pr[\mathbf{Expt}_F^{PRP\text{–}CPA-1}(\mathcal{D}) = 1] - Pr[\mathbf{Expt}_F^{PRP\text{–}CPA-0}(\mathcal{D}) = 1]$$

Definition 4.2.3 (PRP-CPA security of a family of functions)**.** *A family of functions F is PRP-CPA secure if and only if for all polynomial time adversaries $\mathcal{D}$,*
$\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{D})$ is negligible.

There is a variant of this game, called PRP-CCA (for Chosen Ciphertext Attack), which allows attacker access to the decryption oracle. Definition 4.2.4 presents that game, and Definition 4.2.5 the corresponding advantage.

Definition 4.2.4 ($\mathbf{Expt}_F^{PRP\text{–}CCA-1}(\mathcal{D})$ and $\mathbf{Expt}_F^{PRP\text{–}CCA-0}(\mathcal{D})$ [4])**.** *Let $\mathcal{D}$ be an adversary making q oracle queries, $F : \mathcal{K} \times \{0,1\}^{\eta_c} \to \{0,1\}^{\eta_c}$ a function family, and $Perm$ the set of all possible pairs of a permutation of $\{0,1\}^{\eta_c}$ and its inverse. We define the experiments $\mathbf{Expt}_F^{PRP\text{–}CCA-1}(\mathcal{D})$ and $\mathbf{Expt}_F^{PRP\text{–}CCA-0}(\mathcal{D})$ as:*

$Experiment\ \mathbf{Expt}_F^{PRP\text{–}CCA-1}(\mathcal{D})$:
  $k_{src} \xleftarrow{\$} \mathcal{K}$
  $b \xleftarrow{\$} \mathcal{D}^{F_{k_{src}}, F_{k_{src}}^{-1}}()$
  Return b

$Experiment\ \mathbf{Expt}_F^{PRP\text{–}CCA-0}(\mathcal{D})$:
  $(\mathcal{O}_P, {\mathcal{O}_P}^{-1}) \xleftarrow{\$} Perm$
  $b \xleftarrow{\$} \mathcal{D}^{\mathcal{O}_P, {\mathcal{O}_P}^{-1}}()$
  Return b

Definition 4.2.5 (PRP-CCA advantage $\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{D})$ [4])**.** *Let $\mathcal{D}$ be an adversary, $F : \mathcal{K} \times \{0,1\}^{\eta_c} \to \{0,1\}^{\eta_c}$ a function family, and $Perm$ the set of all permutations (and their inverses) of $\{0,1\}^{\eta_c}$. We denote the PRP-CCA advantage of $\mathcal{D}$ against F as $\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{D})$, and it is defined as:*

$$\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{D}) = Pr[\mathbf{Expt}_F^{PRP\text{–}CCA-1}(\mathcal{D}) = 1] - Pr[\mathbf{Expt}_F^{PRP\text{–}CCA-0}(\mathcal{D}) = 1]$$

4.2.2 Hash functions in the Random Oracle Model

Our algorithm uses a hash function of input size η_n, and of output size η_h, denoted $\mathcal{H} : \{0,1\}^{\eta_n} \to \{0,1\}^{\eta_h}$.

We model it as a random oracle, as defined in [5]. A random oracle is an oracle which answers are random, but consistent. To achieve this, couples of parameter and returned value are stored in a table we call $Replies$. If a request has already been done, the oracle will return the same value as before, and otherwise, generate a new one and store it in $Replies$.

$Function\ \mathcal{H}(i)$:
  $If (\exists o\ s.t.\ ((i,o) \in Replies))$
    Return o
  $Else$
    $o \xleftarrow{\$} \{0,1\}^{\eta_h}$
    $Replies \leftarrow Replies \cup (i,o)$
    Return o

4.3 Cryptoverif

Cryptoverif [7] is an automatic prover in the computational model, built by Bruno Blanchet, which uses sequences of games. We used this tool to build the proofs of the security of our protocol for the following games.

4.4 SR3 modelization

For the proof purposes, the algorithm is reduced to a simpler version of the original protocol. We focus on the communication between the source (whose identity is src) and the sink ($dest$), abstracting away all the routing process. These two identities are public. The protocol runs in three phases: initialization of the keys, packet generation, and packet verification.

The original packet format contains a field which indicates the last node who forwarded the message. As we have reduced the protocol to a two-party protocol, this field is not relevant anymore. We omit it for conciseness.

A nonce is a random and unpredictable value uniformly chosen from the set $\{0,1\}^{\eta_n}$, with a fixed size η_n, and denoted by N. We suppose all data have the same length η_d. $\mathcal{O}(\cdot)$ refers to the symmetric encryption oracle (initialized with the node's key), which takes a plaintext for argument. The corresponding decryption oracle is denoted $\mathcal{O}^{-1}(\cdot)$.

First, the setting is initialized using $Init$, then, a packet is generated by Gen, and this packet is transmitted and verified by the sink using $Verif$.

- $Init(\mathcal{K})$ is the function selecting the key the protocol is going to use. It generates a key uniformly at random: $k_{src} \xleftarrow{\$} \mathcal{K}$. This is done before the WSN is deployed.

- $Gen_{src}^{\mathcal{O}(\cdot)}(Data)$ is the function which generates the packet from src which contains $Data$ (required to be of length d), using the encryption oracle $\mathcal{O}$ to build the packet $\langle c, h, s\rangle = \langle \mathcal{O}(Data||N), \mathcal{H}(N), src\rangle$, where N is a fresh unpredictable nonce of size η_n, and $\mathcal{O}(Data||N)$ is the encryption oracle from $\{0,1\}^{\eta_c}$ to $\{0,1\}^{\eta_c}$ (where $\eta_c = \eta_d + \eta_n$). This function returns the pair $\langle c, h, s\rangle, N$.

  Gen is modeled in Cryptoverif by the following process :

```
channel OGenIn, OGenOut.
let OGen =
  !qGen
  in (OGenIn, (DCPA:data));
  new n_oracle_cpa : nonce;
  let p_oracle_cpa =
    ( enc(concat(DCPA, n_oracle_cpa), k), hash(hk,n_oracle_cpa)) in
  out (OGenOut, (p_oracle_cpa,n_oracle_cpa)).
```

- $Verif^{\mathcal{O}^{-1}}(\langle c, h, s\rangle)$ checks whether the packet is considered valid or not. It does three verifications:

  – $c \in \{0,1\}^{\eta_c}$, and $h \in \{0,1\}^{\eta_h}$,
  – s is equal to src,
  – $\mathcal{H}(Right(\mathcal{O}^{-1}(c)))$ is equal to h.[8]

  If all of these conditions are satisfied, then the function outputs 1, and otherwise it outputs 0. This algorithm is modeled in Cryptoverif with the following process :

```
channel OVerifIn, OVerifOut.
let OVerif =
  !qVerif
```

[8] We recall that $Right$ outputs the rightmost part of a concatenation.

```
in (OVerifIn, (cVerif:block,hVerif:hashout));
let concat(dVerif:data,nVerif:nonce) = dec( cVerif, k ) in
let ( verifSuccess:bool ) = ( hash( hk, nVerif ) = hVerif ) in
out(OVerifOut, (verifSuccess)).
```

The packets $\langle c, h, s\rangle$ will be denoted p for simplicity.

As Cryptoverif do not contain the concatenation operation, we model it with a function, which takes a data and a nonce, and outputs a block (the input of the block cipher). This function is declared as a composition, to allow intruder access to its inverse. This construct has a property which is needed for the unforgeability proof: it is impossible to distinguish between the concatenation of a random data and a random nonce, and a random block. We formalized this with the following equivalence relation :

```
equiv
!N new b:block; a() := b
    <=(0)=> [manual]
!N new d:data; new n:nonce; a() := concat(d,n).
```

4.5 Find-then-guess security

The confidentiality of the data should be ensured even after the response of the sink, which discloses the nonce which was in the corresponding packet. To model this, we the nonce used in the packet to the information given to the adversary. However, the adversary does not have control on the nonce generation.

The block cipher is modeled by the family of functions F, given as a parameter of the game. **We assume that F is PRP-CPA-secure, and we do not suppose anything about $\mathcal{H}$.**

Definition 4.5.1 presents two variants of the find then guess game from Bellare and Rogaway [3], where we substituted packets for ciphertexts, packet-building oracles for encryption oracles, and we added the nonce in the information given to the adversary.

Definition 4.5.1 ($\mathbf{Adv}_F^{FG}(\mathcal{A})$ [3]). *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary running in two phases, with s the parameter for communication between these two phases. Let $F : \mathcal{K} \times \{0,1\}^{\eta_c} \to \{0,1\}^{\eta_c}$ a function family.*

$$
\begin{array}{l}
\textit{Experiment } \mathbf{Expt}_F^{FG}(\mathcal{A}) : \\
\quad k_{src} \leftarrow Init(\mathcal{K}) \\
\quad (\mathcal{O}, \mathcal{O}^{-1}) \leftarrow (F_{k_{src}}, F_{k_{src}}^{-1}) \\
\quad (Data_0, Data_1, s) \xleftarrow{\$} \mathcal{A}_1^{Gen_{src}^{\mathcal{O}(\cdot)}(\cdot), \mathcal{H}(\cdot)}() \\
\quad b \xleftarrow{\$} \{0,1\} \\
\quad \langle p, N\rangle \xleftarrow{\$} Gen_{src}^{\mathcal{O}(\cdot)}(Data_b) \\
\quad \textit{If } (b = \mathcal{A}_2^{Gen_{src}^{\mathcal{O}(\cdot)}(\cdot), \mathcal{H}(\cdot)}(p, N, s)) \\
\quad\quad \textbf{Return } 1 \\
\quad \textit{Else} \\
\quad\quad \textbf{Return } 0
\end{array}
$$

The find-then-guess advantage of $\mathcal{A}$ against F is defined as:

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

This modified game measures the ability of the adversary to extract information about the data in a packet. Due to our protocol design, the ACK will reveal its packet's nonce once delivered. To model this, we suppose the attacker has access to that nonce.

The confidentiality of $Data$ is ensured in our scheme if for every polynomial-time adversary $\mathcal{A}$, $\mathbf{Adv}_F^{FG}(\mathcal{A})$ is negligible.

We model this game in Cryptoverif with the previous oracles.

```
channel SetupIn, SetupOut, GameIn, GameOut.
process (
  in(SetupIn, ());
  new hk : hashkey;
  new ks : keyseed;
  new n_challenge : nonce;
  new b : bool;
  let k = kgen(ks) in
  out(SetupOut,());
  (
    in(GameIn,(D0:data,D1:data));
    let cc = concat( (if b then D0 else D1) , n_challenge ) in
    let p_challenge = ( enc(cc, k), hash(hk,n_challenge) ) in
    out( GameOut, (p_challenge, n_challenge) )
  ) | OHash | OGen
)
```

The modelization of this game in Cryptoverif gives us that for all adversaries $\mathcal{A}$ making q_G queries to Gen and q_H queries to $Hash$, there exists a $\mathcal{B}$ making $q_G + 1$ queries to $\mathcal{O}$ such that :

$$\mathbf{Adv}_F^{FG}(\mathcal{A}) \leq \frac{2q_G + 2q_G^2}{2^{\eta_c}} + \frac{2q_G^2 + 4q_G + 2}{2^{\eta_n}} + 2\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{B})$$

Note that the equivalences associated to $Hash$ being a random oracle are not used in the corresponding proof.

4.6 Unforgeability

Unforgeability under *chosen-message and verification attack* is the computational infeasibility for an adversary to create a valid new packet from any data, when this attacker has access to an oracle building packets from data, and another oracle which returns whether a packet is valid or not. This game is formally defined in Definition 4.6.1.

This game models an attacker which tries to inject data in the network, without access to a valid key. This attacker can make nodes send packets containing specific data by tampering with their sensors, as in the previous subsection. This ability corresponds to the Gen oracle access. The attacker can also send forged packets to the sink, which will answer if the packet is valid. This ability is modeled by giving the adversary access to the $Verif$ oracle.

Giving $Gen_{src}^{\mathcal{O}(.)}$ to the adversary does allow it to generate valid packets, which would be an easy way to beat the game. To avoid this, we change the $Gen_{src}^{\mathcal{O}(.)}$ function for this game. When it is called, it adds the returned packet to an array named $Queries$. This allows us to track which packets were returned by the oracle, and ensures the adversary cannot win by using them.

Note also that $Verif$ checks that the third field of a packet is equal to src, so changing this field will always cause a packet to be refused.

Here, **we assume that F is PRP-CCA-secure, and that $\mathcal{H}$ is a random oracle.**

Definition 4.6.1 ($\mathbf{Adv}_F^{UF\text{–}CMVA}$ [6]). *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary running in two phases, with s the state passed from one to the other. Let $F : \mathcal{K} \times \{0,1\}^{\eta_c} \to \{0,1\}^{\eta_c}$ a function family. We consider the following experiment :*

$Experiment\ \mathbf{Expt}_F^{UF\text{–}CMVA}(\mathcal{A})$
$Queries \leftarrow \emptyset$
$k_{src} \leftarrow Init(\mathcal{K})$
$(\mathcal{O}, \mathcal{O}^{-1}) \leftarrow (F_{k_{src}}, F_{k_{src}}^{-1})$
$p \leftarrow \mathcal{A}^{Gen_{src}^{\mathcal{O}(\cdot)}(\cdot), \mathcal{H}(\cdot), Verif^{\mathcal{O}^{-1}(\cdot)}(\cdot)}()$
If $(p \notin Queries \wedge Verif^{\mathcal{O}^{-1}(\cdot)}(p))$
Return 1
Else
Return 0

The unforgeability advantage of $\mathcal{A}$ against F is defined as:

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

Definition 4.6.2 (Unforgeability of the packets)**.** *Packet unforgeability is ensured if and only if for every polynomial-time algorithm $\mathcal{A}$ making q queries to its oracles, $\mathbf{Adv}_F^{UF\text{–}CMVA}(\mathcal{A})$ is negligible.*

We model that game in Cryptoverif using the $Verif$ oracle, and a table $queries$ storing pairs of a data and a nonce, which is updated by Gen. The event bad is set if the attacker is able to output a packet which contains a pair of a data and a nonce never generated before by Gen.

```
process (
  in(SetupIn, ());
  new hk : hashkey;
  new ks : keyseed;
  let k = kgen(ks) in
  out(SetupOut,());
  (
    in (ChallengeIn, (c:block,h:hashout));
    let concat(dV:data,nV:nonce) = dec(c,k) in
    get queries(d2,n2) suchthat (nV=n2 && d2=dV) in
      out(ChallengeOut, ())
    else
      if (h = hash(hk,nV)) then (
        event bad();  (* valid packet, not in queries *)
        out(ChallengeOut, ())
      ) else
        out(ChallengeOut, ())
    ) | OHash | OGen  | OVerif
)
```

The modelization of this game in Cryptoverif gives us that for all adversaries $\mathcal{A}$ making q_G queries to Gen, q_V queries to $Verif$ and q_H queries to $Hash$, there exists an adversary $\mathcal{B}$ making q_G queries to $\mathcal{O}$ and $q_V + 1$ queries to $\mathcal{O}^{-1}$ such that :

$$\mathbf{Adv}_F^{UF\text{–}CMVA}(\mathcal{A}) \leq \frac{q_H + q_H * q_V + q_V * q_G + q_G + q_H * q_G + 2q_G^2}{2^{\eta_n}} + \frac{q_G^2 + q_G + 2(q_V * q_G) + q_V + q_V^2}{2^{\eta_c}} + \frac{1 + q_V}{2^{\eta_h}} + \mathbf{Adv}_F^{PRP\text{–}CCA}(\mathcal{B})$$

4.7 Nonce confidentiality

To determine whether the attacker can generate an ACK from a packet not yet delivered, we use a third game, which focus on the confidentiality of the nonce used in a packet. To win, the adversary, given a packet and access to the Gen oracle, should guess the nonce in the allowed number of queries q_A. **We assume that F is PRP-CPA-secure, and $\mathcal{H}$ is a random oracle.**

Definition 4.7.1 ($\mathbf{Adv}_F^{N\text{–}conf}(\mathcal{A})$)**.** *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary running in two phases, with s the parameter for communication between these two phases. Let $F : \mathcal{K} \times \{0,1\}^{\eta_c} \to \{0,1\}^{\eta_c}$ a function family, and q_A the allowed number of answers for $\mathcal{A}_2$.*

$Experiment\ \mathbf{Expt}_F^{N\text{–}conf}(\mathcal{A})$:
- $k_{src} \leftarrow Init(\mathcal{K})$
- $(\mathcal{O}, \mathcal{O}^{-1}) \leftarrow (F_{k_{src}}, F_{k_{src}}^{-1})$
- $(Data, s) \xleftarrow{\$} \mathcal{A}_1^{Gen_{src}^{\mathcal{O}(\cdot)}(\cdot), \mathcal{H}(\cdot)}()$
- $\langle p, N \rangle \xleftarrow{\$} Gen_{src}^{\mathcal{O}(\cdot)}(Data)$
- $Answers \leftarrow \mathcal{A}_2^{Gen_{src}^{\mathcal{O}(\cdot)}(\cdot), \mathcal{H}(\cdot)}(p, s))$
- *If* $(|Answers| \leq q_A \wedge N \in Answers)$
  - **Return** 1
- *Else*
  - **Return** 0

The nonce confidentiality advantage of $\mathcal{A}$ against F is defined as:

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

We model that game in Cryptoverif using an oracle able to set an event *bad*. This oracle checks whether the adversary-supplied input is equal to the challenge nonce, and sets the event *bad* if this is the case. We run q_A copies of that process, to model the allowed number of tries, and we bound the probability of *bad*.

```
event bad().
channel WIn, WOut.
let OWin = !qAnswer
  in(WIn, (n:nonce));
  if (n = n_challenge) then
    event bad;
  out(WOut, ()).

channel SetupIn, SetupOut, GameIn, GameOut.
process (
  in(SetupIn, ());
```

```
  new hk : hashkey;
  new n_challenge : nonce;
  new ks : keyseed;
  let k = kgen(ks) in
  out(SetupOut,());
  (
    in(GameIn, (D:data));
    let cc = concat( D , n_challenge ) in
    let p_challenge = ( enc(cc, k), hash(hk,n_challenge) ) in
    out( GameOut, (p_challenge) )
  ) | OGen | OHash | OWin
)
```

The modelization of this game in Cryptoverif gives us that for all adversaries $\mathcal{A}$ having q_A answers, making q_G queries to Gen, q_V queries to $Verif$ and q_H queries to $Hash$, there exists an adversary $\mathcal{B}$ making $q_G + 1$ queries to $\mathcal{O}$ such that :

$$\mathbf{Adv}_F^{N\text{–}conf}(\mathcal{A}) \leq \frac{q_A + q_H + q_H * q_G + 2q_G + 2q_G^2}{2^{\eta_n}} + \frac{q_G + q_G^2}{2^{\eta_c}} + \mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{B})$$

4.8 Using these results

These three bounds allow us to select the necessary trade-off between the desired level of security and the mandatory minimization of the message overhead. For instance, we can choose an advantage smaller than 2^{-60} for the three properties against an adversary $\mathcal{A}$ that can query each oracle up to 2^{30} times (around 1 billion queries). To achieve this, we set η_n to 128 bits (16 bytes) and η_h to 96 bits (12 bytes). Then, the advantage of $\mathcal{A}$ would be smaller than $2^{-64} + 2\mathbf{Adv}_F^{PRP\text{–}CCA}(\mathcal{B})$ in the UFCMVA game; the results are similar for the other properties. From this, if we use the AES-192 block cipher (allowing 64 bits of data), the best attack known to this day needs $2^{189.7}$ operations. Therefore, we can expect $\mathbf{Adv}_F^{PRP\text{–}CCA}(\mathcal{B})$ to be much smaller than 2^{-64}, and consequently our security bound of 2^{-60} would be satisfied. Using these sizes, the overhead for each data message would be 36 bytes: 16 bytes for the nonce, 12 bytes for the hash function, and 8 bytes to store a node identifier.

5 Attacker Models

We proved in Section 4 that the packet format guarantees the confidentiality of the data, and the unforgeability of messages. But this proof was focused on the packets, and does not imply that our algorithm will work well when the attacks focus at the routing level. For instance, could attackers stop most messages from being delivered ? We want to evaluate what would be the effect of different attacker behaviors on the network, compared to the algorithms described in the next section.

In the taxonomy of [16], we are looking at active insider attackers, which are intruder-controlled motes with access to the same things as an honest node, such as a valid identity, shared keys, and so on. Aside from their behavior, honest nodes cannot distinguish between honest and dishonest neighbors.

We aim to a fair comparison of algorithms. There are well-known attacks on the protocols we use, but the purpose of their presence here is to be comparison points. To ensure a fair competition between all of them, we assume that attacker do not try to cheat when determining the initial knowledge of protocols, such as localization or gradients.

All the attacks detailed in this section are technically feasible. Since we assume that attacker nodes do not send messages to the sink, and that no algorithm in our selection uses shared keys between nodes, an attacker can buy the same motes as used in the network, and just drop them in range of honest nodes. The attacks themselves do not require anything unrealistic.

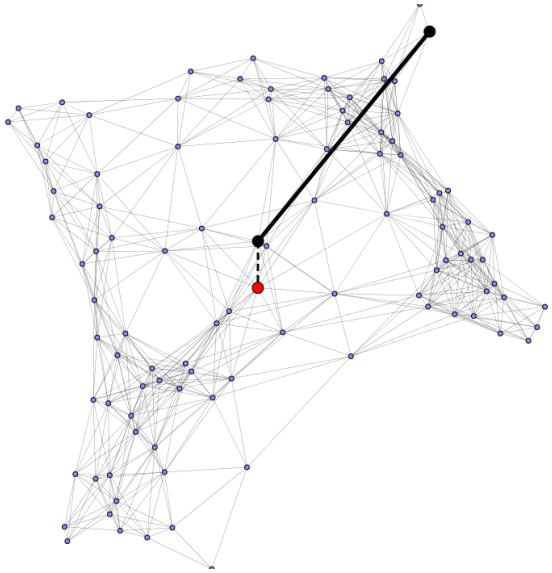

Figure 1: An example wormhole (sink in red)

5.1 Blackholes

The simplest attacker is a blackhole, which participates to the network as any honest node, registers as a neighbor of other nodes, and accepts transmissions, but never forwards data.

Unless the routing protocol supports re-transmissions or duplication of messages, packets that gets routed to one of these nodes are lost. The blackholes we are considering intercept all packets, whether they contain data or are only control information.

5.2 Selective forwarding nodes

Selective forwarding (SF) nodes are a generalization of blackholes, which stop part of the messages they receive. They only do so given certain rules, for instance, only messages with a specific information inside, or messages of a given type.

Since our algorithm uses a reputation scheme, we are interested in SF attackers which try to keep a certain reputation. This way, their neighbors will keep sending them messages, and they can keep routing some of them, and drop the others.

We want to know whether this approach causes more damage than a simple blackhole, and to do this, we will try SF attackers which behave either like a blackhole with a certain probability p, or follow the routing algorithm's rules with probability $1 - p$. The behavior is randomly chosen at each message's reception.

5.3 Wormholes

Wormholes are attackers which act as a relay. They can be either a single repeater node, or two nodes linked by a long-range, out-of-bounds connection. This way, they can transparently build a tunnel between two distant points in the network, with an intruder at each extremity. This effectively distorts the network, and with a good tunnel placement, wormholes can attract a lot of traffic, or totally break protocols using geographical information.

We will consider wormholes where one extremity is in communication range of the sink, and the other is in a remote part of the network, as the other case where the two nodes are at the same distance to the sink does not improve attacks on the protocols we chose. Figure 1 illustrates this. The node close to the sink will only relay communications from and to the remote node, and the remote node will behave like any normal node, but with an added symmetric connection to the sink (through the tunnel).

The remote node builds up reputation in the network, effectively creating a sinkhole, to use the terminology of [16]. Once a specified amount of time has passed, it changes its behavior to become a blackhole, and it uses its reputation to increase the effect of the attack. The node close to the sink is not used anymore after the switch.

When the network runs SR3, wormholes can be seen as a sink by their honest neighbors. When a message gets to its destination, the sink answers with an acknowledgement directly to the last hop of that message (see Algorithm 2). Thus, a wormhole forwarding a message to the sink will automatically receive the corresponding acknowledgement, which can then be forwarded to the corresponding node, which will then trust the wormhole. After enough time, the wormhole has built trust in its neighborhood, and most packets are routed through it. The wormhole will then use that additional routing weight by dropping all messages it receives. The previous trust it built will then gradually fade out because of the reputation mechanism, which will later cause messages to avoid this node.

5.4 Sybil Nodes

Sybil nodes declare multiple identities to their neighbors. This behavior is a way to enhance another attack mechanism: in our case, we use Sybil nodes which are also blackholes, where all the fictive identities of the node will drop packets in the same way.

This attack is usually applied to protocols where some sort of consensus or vote is done, but the multiple identities can also strongly influence protocols where the next hop is randomly selected, as multiple identities will increase the probability of selection of the attacker. For instance, RW is vulnerable to this attack.

6 Related Routing Algorithms

To underline the performance and resiliency of our algorithm, we chose to compare it against some other routing algorithms. These are either well-known comparison points (RW, GFG, GBR, RGBR), or algorithms designed to withstand the same kind of attacks as our algorithm (PRGBR, PRDGBR).

6.1 Uniform Random Walk (RW)

Each packet to route is sent to a random uniformly selected neighbor. As explained before, RW is simple, lightweight, and unpredictable. There is no overhead per message, no memory cost on the nodes and few computations to do at each step. However, messages can take many hops to reach the sink, making this algorithm more suited to small networks where route length is not critical.

In presence of attackers, RW is affected by the number of compromised nodes. If there is a lot of intruders, one should expect that most packets will reach one of them before the sink. However, because of the non-determinism of this algorithm, as long as there is a safe route between a node and the sink, there will be some delivered packets from this node.

6.2 Greedy-Face-Greedy (GFG)

This algorithm is a combination of two other algorithms: Greedy routing, and Face-Based Routing (FBR). They both require that each node knows its position, its neighbors positions, and the sink position. FBR requires the network graph to have no intersecting edges (a requirements which implies the planarity of the network graph). As a consequence, the algorithm needs a transformation of the initial network, described below.

Greedy (or geographical) routing consists in routing each message to a neighbor that is geographically the closest to the sink. However, this algorithm can get stuck in graph components where a node is closer to the sink than all its neighbors, creating a routing loop.

When this happens, the algorithm switches to FBR until the packet reaches a position closer to the sink than the point where the message got stuck. Then, the algorithm switches back to Greedy, and so on.

Face-based routing is described in [8]. It requires first a planarization of the graph, meaning that some connections are ignored, so that no edges cross in the network graph. A line between the origin of the routing and the sink is computed, and the face (a polygon delimited by the edges in the graph) adjacent to the emitter is selected. This face is followed in a certain sense of rotation, until it goes back to the initial vertice. Then, the messages is forwarded to one end of the edge intersected with the previous line which is the closest to the sink. This edge is a delimitation between the face we are currently working with, and the one the message is going to start again with. The algorithm then reiterates with the new intersection point as the origin of the message.

In order to actually have faces, FBR requires the planarization of the graph, which we do through Gabriel Graphs, first defined in [14]. To summarize, an edge (a, b) is kept if and only if there is no node c in the circle of diameter a, b. This algorithm guarantees that if the initial graph is an unit disk graph (a graph where nodes are connected if the distance between them is lower than a constant), the resulting planarized graph will still be connected, as proved in [8].

Overall, GFG routes messages in an efficient way, as long as Greedy runs. The route lengths are not optimal, and the algorithm is completely deterministic. The main weakness is FBR, which is not efficient. As it is only a fall-back for when Greedy gets stuck, a lot of networks (mostly those with a high average degree) will only run Greedy. However, some networks will almost always route messages according to FBR, and since it requires a full traversal of the current face before any decision, a message will get intercepted if there is an attacker on it. We experimentally observed some special cases of networks, where a few attackers could stop all the messages, by positioning a handful of blackholes.

6.3 Gradient-based routing and variants

Gradient-based routing (GBR) is an algorithm described in [23], which works in two phases. First, the sink floods the network with an INTEREST packet containing a counter (indicating the distance to the sink, also called the height of the node), which is then rebroadcasted by all nodes after incrementing the counter if the packet reflects a better route than those that were previously known.

The minimal counter value seen by a node until now is called its height. Nodes keep track of both their height, and the height of each of their neighbors. If a new INTEREST packet with a smaller counter than the node has seen until now, the node updates its height, and notifies its neighbors.

On each routing request, nodes choose a next hop according to the specific variant. We used the variants and notations from [12].

By design, assuming each node knows its actual distance to the sink, and the distance for its neighbors, the first two variants (GBR and RGBR) always route packets using the shortest routes, in terms of hops. The two other variants do not have that property, since nodes can send messages to neighbors of the same height.

Attackers do not tamper with the height-building process, as we consider them as compromised nodes. Wormholes have a big advantage in this process, since they are at one hop of the sink, they always occupy a preferential routing position.

6.3.1 Deterministic GBR (GBR)

This is the original GBR protocol, described in [23]. Nodes send their messages to a fixed lower-height neighbor.

This variant is deterministic. Once the heights have stabilized, all the packets from a given node will follow the same route. Since those routes are small, the efficiency (in number of hops) of the protocol is good, and randomly positioning attacker nodes will only block a few routes, resulting in an overall good delivery rate. However, when targeting a specific node, if there is an attacker on the route between that target and the sink, the attacker will completely control all the messages of the emitter node. Thus, it is fairly easy for an intruder to stop a small set of nodes from emitting messages.

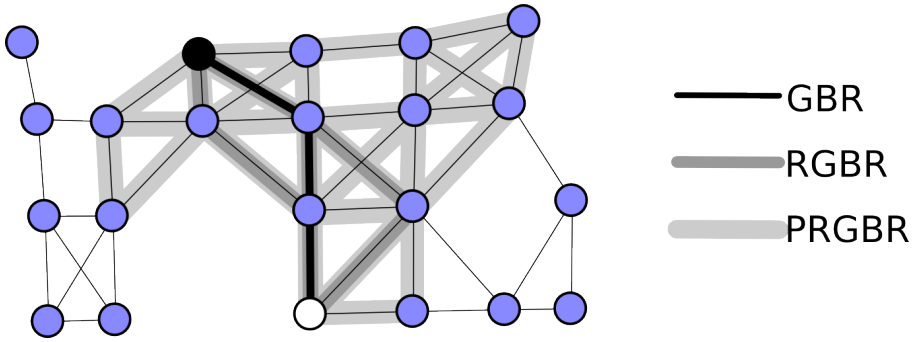

Figure 2: Example network where the possible paths chosen by GBR variants are highlighted. The sink is black, the source is white.

6.3.2 Randomized GBR (RGBR)

Randomized GBR is a variant which adds randomness to the routing by uniformly selecting a random lowest-height neighbor each time a packet needs to be routed. This mitigates the downsides to GBR's determinism, while still keeping optimal route lengths.

This countermeasure is more efficient on high-degree graphs, where there is a lot of possible routes to choose from. However, when the network graph has a small degree, only a few routes are optimal, and they frequently go through a few choke points. If one of these is an intruder, all the messages from a part of the network will get lost. See Figure 2 for a visual example.

6.3.3 Probabilistic Randomized GBR (PRGBR)

In order to add even more randomness, RGBR can be modified to allow routes which go through same-height nodes. There are two possible behaviors. With probability p, the message is routed to a random same-height node if one exists, and otherwise, to a random lower-height node. We used $p = 0.4$, as the protocol authors did in [12].

This algorithm takes a step further away from RGBR in the direction of RW. Routing to a same-level node allows even more routes from a source to the sink, as seen on Figure 2. Since the followed paths can get longer than the previous variant, the delivery rate of messages is impacted: longer paths mean more possibilities of the message getting intercepted. However, as the algorithm gets closer to RW, it also becomes harder to stop a single emitter from transferring data to the sink.

6.3.4 Probabilistic Randomized Duplicating GBR (PRDGBR)

This version is based on the probabilistic randomized GBR with duplication described in [12]. The only difference is that we chose to duplicate messages at each hop (initial emission included), instead of either at the initial emission or with each forwarding node. As described in the paper, messages already seen by a node or the sink are discarded (we memorize them in an infinite list).

This algorithm causes some packet losses even without intruders in the network. Since the routing decisions for the duplicated packets are independent, both packets can get routed to the same node. But since nodes drop duplicate packets, one of the pair can be useless. Now, as each individual packet is routed according to PRGBR, a node a can send to a same-level node b, which can then send its packets to a again. Let us imagine that a and b are the only same-level nodes in this part of the network. The probability of a sending both its copies to b, which then sends both its copies to a is p^4, which is 0.0256 for $p = 0.4$, more than 2%. This effect decreases as the degree of the network increases, and the quantity of possible routing choices.

To avoid pointless complications with our network model, we chose to discard all other copies of a message once a copy reached the sink. The deletion happens at the next reception of each packet. Since these operations happen after the actual reception of the packet, they do not influence our metrics.

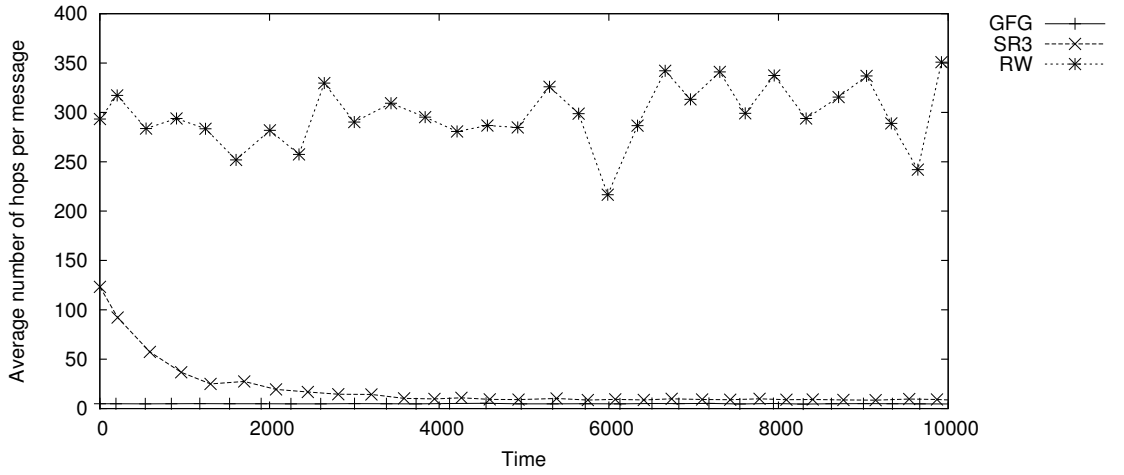

Figure 3: Route length of a few algorithms over time, on a graph of 100 nodes, average degree 8, and no attackers

7 Observed Behaviors and Metrics

In order to evaluate the performance of the routing algorithms, we need to investigate multiple behaviors, and find metrics that reflect these behaviors.

7.1 Number of Hops

The first thing to observe in the protocol is the length of the routes in hops. Although it does not take into account the control messages and per-message overheads, this measure is still a good indicator of the efficiency and speed of the routing algorithm.

We measure this using the average number of edges traversed, for each message that reached its destination. For PRDGBR, where packets are replicated, once a message reached the sink, it is considered as delivered, and all copies are discarded.

When the network contains attackers which drop messages, this measure has to be carefully addressed. For instance, if there is an attacker in a remote part of a network, it may artificially lower the hop count, by causing more packet loss for packets on long routes than for packets in small networks. In the same way, when the network contains wormholes, the topology changes makes this measure difficult to interpret.

To plot this, we sort by emission date all the messages that were delivered to the sink, and group them in blocks of messages, which contain a fixed number of message (which we change depending on the simulation length and overall number of messages). Then, the average hop count of each block gives the y coordinate, and we average block's messages emission date to have the x coordinate.

There is an example of the method in figure 3, which shows a comparison of the average hop counts per message over time between GFG, SR3 and RW on a 200 node graph with a degree of 8, and no attackers. In this example, each point is an average over a hundred data-bearing messages.

We can see that when using RW, the routes are longer than the routes generated with the other two algorithms. SR3 and PRGBR have hop counts around 5 or 10 hops on average, and SR3 has a hop count which converges after some time.

7.2 Delivery Rate

The delivery rate of a routing algorithm is the proportion of messages delivered over the messages sent. This measure is interesting in presence of attackers.

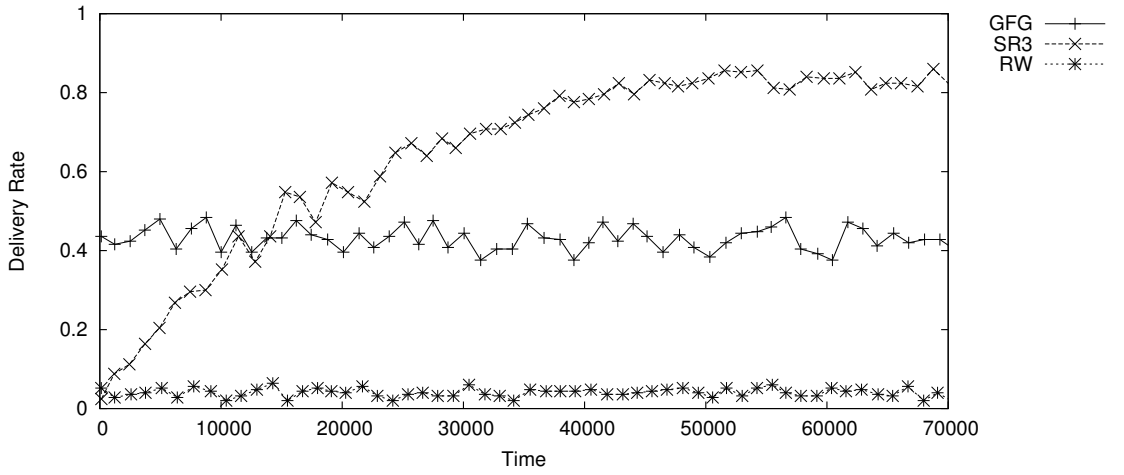

Figure 4: Evolution of the delivery rate of a few algorithms over time, on a graph of 200 nodes, average degree 8 and 30 percent of attackers

7.2.1 Average Delivery Rate

The average delivery rate is a measure of how the algorithm manages to route packets to the sink.

We represent this in the same way as the route length, by grouping messages in blocks of 500 messages, and averaging.

For PRGBR with duplication, if the sink receives a copy, the message is flagged as delivered. If all of them are lost, the message is considered lost.

Figure 4 shows the delivery rates of GFG, SR3 and RW, over a smaller graph, with 30 percent of attackers. GFG has a delivery rate of roughly 45 percent of messages, while RW loses most of its messages. SR3 is the only algorithm which evolves over time: its delivery rate increases as the reputation mechanism causes messages to avoid the blackholes, and in this case, reaching more than 80 percent of messages delivered.

7.2.2 Fairness regarding the delivery rate

We want to show the difference between an algorithm like GBR, with deterministic routes and therefore a 0% delivery rate if a blackhole is on the path of a node, and SR3, where most of the nodes have similar delivery rates. We are therefore looking at the delivery rates of each individual node, and the fairness is the standard deviation of this data for one simulation run. As we would like to have uniform delivery rates over all nodes, lower is better.

We give an example in Figure 5, which shows the fairness of different algorithms, in the same graph, against different proportions of randomly placed blackholes. GBR's fairness is high, since nodes either deliver all their messages, or none of them, depending on the presence of a blackhole on their path to the sink. PRDGBR adds a lot of randomness in the routing, and therefore has a lower (better) fairness than GBR, but it also loses messages in safe networks, which explains its non-zero fairness in these conditions. RW has an excellent fairness, and this indicates that all nodes have comparable delivery rates, as a large majority of the messages get intercepted. SR3 also has a good fairness, and its delivery rate is good. Overall, it's important to remember that fairness only indicates the uniformity of the delivery rates, and should be considered along with the average delivery rate.

7.2.3 Delivery Rate Classes

In order to compare different algorithms on a lot of graphs, we need a way to average the measures over different runs of the algorithm, so that they still make sense. For this, we expanded the idea of delivery rate

		Blackholes proportion		
		0%	10%	20%
Algorithm	SR3	0	0.032	0.047
	RW	0	0.049	0.041
	GBR	0	0.373	0.482
	PRDGBR	0.004	0.081	0.132

Figure 5: Fairness (std.dev. of each node's delivery rate) for different percentage of attackers over the same graph of 200 nodes and degree 16

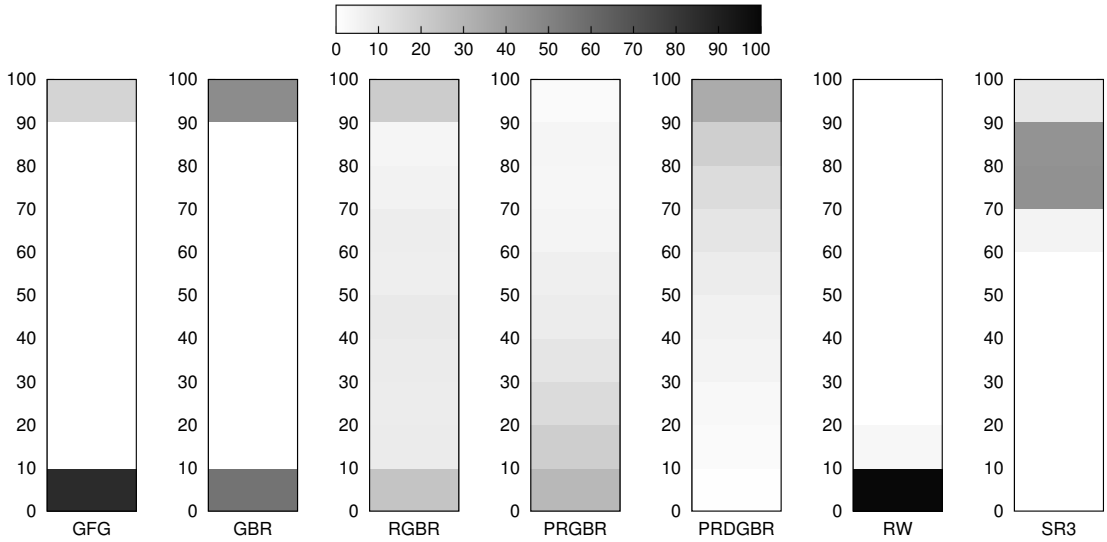

Figure 6: Example of DR stacks over 20 graphs of 400 nodes with 20 percent of attackers, and an average degree of 16

classes from [12].

From a bundle of simulations in the same setting (same degree, number of nodes, number of intruders, and simulation parameters), we compute the overall delivery rate of each node over all simulations. At the end, we categorize them in classes as [12] did, but in 10 shares of 10% each: between 0% and 10% delivery rate, 10% and 20%, and so on. We look at the proportion of nodes in each of the classes.

See Figure 6 for an illustration of this process, over 20 graphs of 400 nodes with 20 percent of attackers, and an average degree of 16. The darker a class is, the more nodes it contains. For instance, the average delivery rates of SR3 and PRDGBR are around 80 percent of messages, but for SR3, there are very few nodes which have a delivery rate smaller than 70 percent, with most of them above. On the other hand, PRDGBR has a more spread out distribution of the delivery rates. We can also see that both GFG and GBR have nodes that either route messages, or do not, since they both are deterministic algorithms.

8 Simulation Context

8.1 Sinalgo

We used a modified version of Sinalgo[9] for our simulations. Sinalgo is a network simulator, developed in Java by the Distributed Computing group at ETH Zurich. Sinalgo is suited for simulations which abstract the communication layers, and allows for large graphs and visualization of algorithms. We added our own

[9] http://disco.ethz.ch/projects/sinalgo

		Number of nodes		
		100	200	400
Degree	8	$2.8 * 10^{-3}$	$1.4 * 10^{-3}$	$7.1 * 10^{-4}$
	16	$4.0 * 10^{-3}$	$2.0 * 10^{-3}$	$1.0 * 10^{-3}$
	32	$5.7 * 10^{-3}$	$2.8 * 10^{-3}$	$1.4 * 10^{-3}$

Figure 7: Empirically acceptable data generation λ for some settings

layer of statistics and instrumentation on top of Sinalgo, built using Perl, and some minor local utilities to generate graphs, run simulation campaigns, and better visualize our algorithms.

8.2 Network Modeling

- **Transmission model**

  The transmission time between two nodes for a message follows the exponential distribution with probability density function f:

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

  We set $\lambda = 1$ in this case. The simulation time unit is based on the mean transmission time ($\frac{1}{\lambda} = 1$).

  We also enforce a first-in-first-out order on the links.

- **Data quantity**

  We chose to simulate the generation of 500 000 data units.

- **Data generation**

  All legitimate nodes generate data, except the sink.

  The interval between two data generations by the same node, and the interval between the beginning of the simulation and the first send both follow the same exponential distribution of parameter λ. This λ is arbitrary. We want to have a certain quantity of messages in the network (to provide interleaving and message queues), while keeping the overall generation rate under the capacity of the network to process messages.

  If nodes send too many messages, the whole network will lock down due to queues building, and fall in a downward spiral where queues cause bad routing, which will in turn decrease the capacity of the network to process messages.

  We ran simulations in some settings, until we reached data generation rates which were adequate in most cases. Figure 7 contains our λ parameter for some of the settings.

8.3 Topologies

8.3.1 Graph generation

To generate topologies for our experiments, we use *Unit Disk Graphs* (UDGs), which are generated by randomly scattering nodes across a simulation area (a square of side $AreaSize$ units). If two nodes are closer than the communication radius $UDGRad$ (arbitrarily set to 20) to each other, they can communicate. Each graph is determined by the number of nodes n and the desired average degree $\mu(\delta)$, and we keep a library of graphs corresponding to each setting so that all experiments on a specific setting are always ran against the same set of topologies.

In order to generate them, we determine the $AreaSize$ using the following formula [10] :

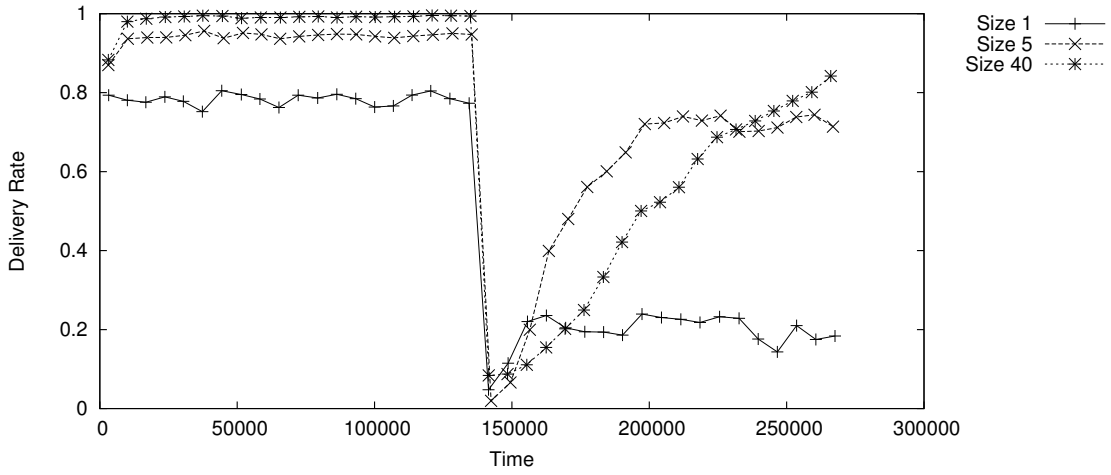

Figure 8: Example of the effects of $L_{Routing}$ size on the delivery rate

$$\mu(\delta) = \frac{UDGrad^2 * \pi * n}{AreaSize^2}$$

As $UDGrad = 20$, we get :

$$AreaSize = \sqrt{\frac{400 * \pi * n}{\mu(\delta)}}$$

The sink is positioned in the exact center of the simulation area.

When we need to position attackers, we randomly replace honest nodes by the required number of attackers. This can lead to graphs where there is no safe path from the sink to some legitimate nodes, which we chose not to remove, to avoid biases in the generated topologies. Some nodes will have a delivery rate of 0, but as the network graphs are common, all the algorithms will have the same quantity of disconnected nodes.

8.4 Setting parameters for the SR3 algorithm

In order to determine a good set of parameters for our algorithm, we run some preliminary experiments.

8.4.1 $L_{Routing}$ size

First, we would like to know which size to use for the $L_{Routing}$ list. This list determines both the peak delivery rate of the algorithm and the time it takes to refresh its routes with up-to-date information, for instance after a sinkhole attack.

We are therefore looking at a measure of the delivery rate in a network containing attackers as described in Section 7.2.1, and we are aiming to both have a good routing once the setup phase is done, and a small refresh time when attackers reveal themselves.

See for an example Figure 8, which shows the delivery rate of messages over time, for lists of size 1, 5 and 40, on a graph of 200 nodes, with an average degree of 16. This network contains five percent of blackholes, to see the capacity of the network to rearrange routes in order to avoid attackers, and five percent of wormholes as described in section 5.3. Each dot represents a bundle of 5000 messages.

Because of the definition of $\mathcal{L}^v_{SR3}$ (Section 3.4), increasing the size of $L_{Routing}$ will reduce the probability of choosing a node not in the list. This can be seen on Figure 8 by looking at the values at which the curves reach a plateau: the bigger lists shows a better delivery rate after stabilization than the smaller ones, with less variations in the end.

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

Figure 9: Experimentally obtained best $L_{Routing}$ sizes, with a step of 5

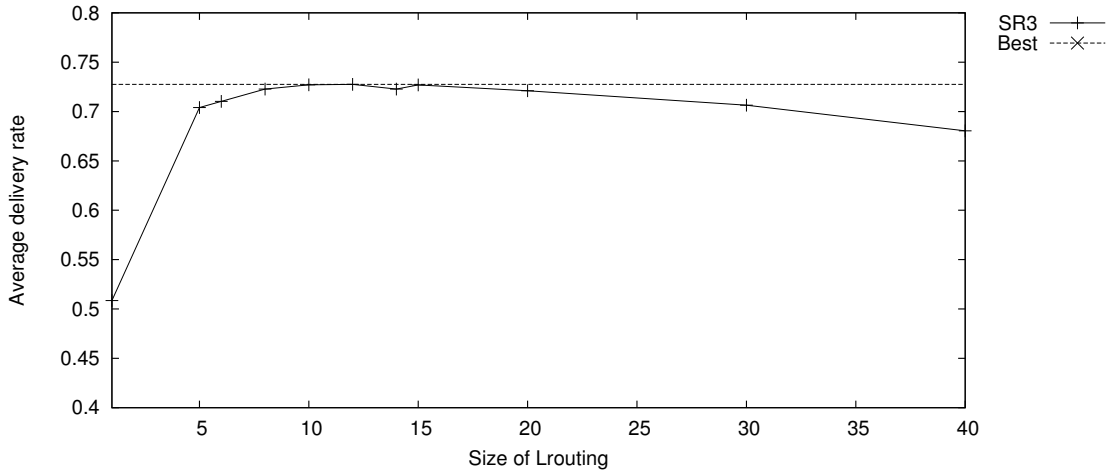

Figure 10: Delivery rate depending on $L_{Routing}$ size for 100 nodes and degree 8

The drawback of this stability is an increasing impact of wormholes, since once a node trusts its neighbor, it will take longer to get enough feedback to have recent information in $L_{Routing}$. This can be seen by comparing the size 5 and size 40 delivery rates, where the biggest list takes a longer time to recover after the drop due to the behavior change of wormholes.

We ran ten simulations on different graphs per tuple (list size, number of node, degree), and averaged the total delivery rate of each run. The results can be seen in Figure 9. We tried different sizes of $L_{Routing}$, from 5 to 40, and we show here the value that yield the highest global average delivery rate.

There is an important fact that does not appear in this table: the average delivery rate is very similar for a large range of list sizes. See Figure 10 for an illustration of this. The behavior is the same across all the networks we tried. We choose a $L_{Routing}$ size of 10, which is overall a good compromise for every tested setting.

8.4.2 L_{Queue} size

We need to determine the size of L_{Queue}, which stores messages until their corresponding ACK returns. A too small list would cause most ACKs to be forgotten, and useful feedback would be lost, whereas a too large list would just trade valuable memory for a negligible routing advantage.

To find a good size, we start by setting the $L_{AckRouting}$ size to infinity. This causes all ACKs to follow the return path perfectly. Then, by putting the algorithm in the range of settings we will later use to evaluate all the algorithms, we try different sizes. The goal is to avoid loss of data when ACKs come back to their sender. A few ACKs getting lost are not a problem, as long as most of them still reach their destination.

Our goal is to have less than a percent of ACKs forgotten on return, and we tried L_{Queue} sizes of 1, 3, 5, 7, and 9 elements.

Results of experiments are in Figure 11. For all settings, the results we obtained are similar to those ones. Every time, the size 3 lists are sufficient to reach our objective of 99% of acknowledgements remem-

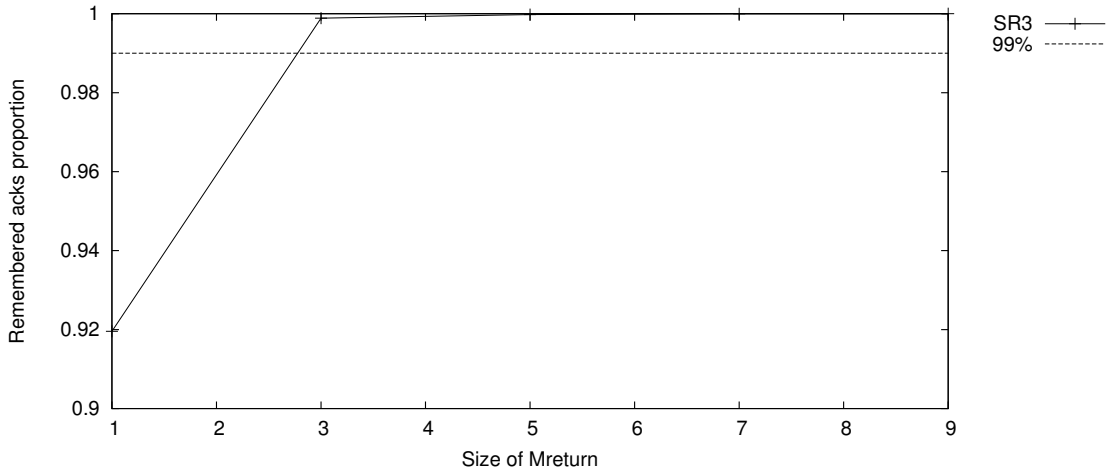

Figure 11: Proportion of ACKs remembered, depending on L_{Queue} size for 200 nodes and degree 8

bered.

Also, the mean number of hops can on average be smaller by a few percent when using smaller lists. This is a side-effect, where the messages following longer routes will be more likely to be forgotten by their sender. However, we do not want to leverage this effect: when the network is under attack, artificially forbidding longer routes will be counterproductive.

8.4.3 $L_{AckRouting}$ size

Now that both other parameters are fixed, we want to achieve a good-enough reverse chaining. We examined the average number of remembered ACKs, as in the previous case, and we tried to keep most ACKs remembered when they come back, while keeping the memory costs reasonable. We aim for results similar to the previous ones, although they will be inferior since we use a finite L_{Queue}.

The quantity of remembered acknowledgements is critical for the initial routing, before the nodes learn shorter routes. At this stage, feedback is critical, and the whole point of the reverse chaining mechanism is to enable faster feedback to the nodes. Since SR3 at the beginning behaves like a random walk, the length of routes will be comparatively long, and thus, the amount of memory needed for a single message will be higher at the beginning of the routing. This is when the adequate size for those lists is critical, and to explore this parameter, we ran short simulations.

Settings are the same as before, but we run only 50000 messages. Figure 12 is an example of results we obtained, on exactly the same setting as the previous example (200 nodes, degree 8). Once again, other settings result in similar results. The remembered ack rate stagnates when the size of $L_{AckRouting}$ goes above 5 elements, and we chose that size.

9 Results

We compared all the algorithms on networks of degree 8, 16 and 32, and of sizes 50, 100, 150, and so on, up to 400 nodes. See Figure 13 for some example graphs. These graphs were generated using the method described in Section 8.

Most of our measurements give the same results for GBR and RGBR. These two algorithms have the same behavior, except that RGBR randomly selects a new route at each hop, while GBR only has randomness at the beginning of the simulation. To avoid presenting overlapping points in most of our illustrations, we only show RGBR's results when they behave exactly in the same way.

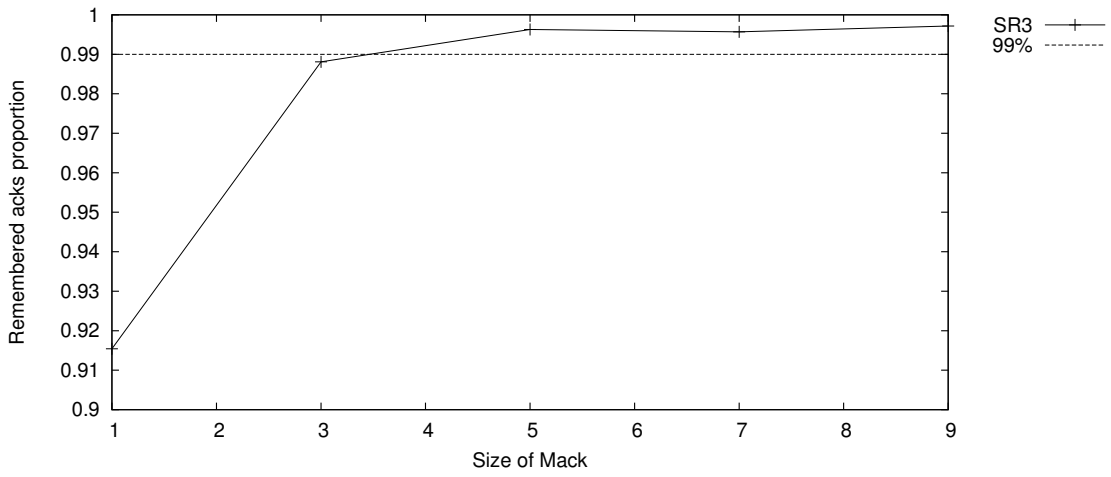

Figure 12: Proportion of ACKs remembered, depending on $L_{AckRouting}$ size for 200 nodes and degree 8

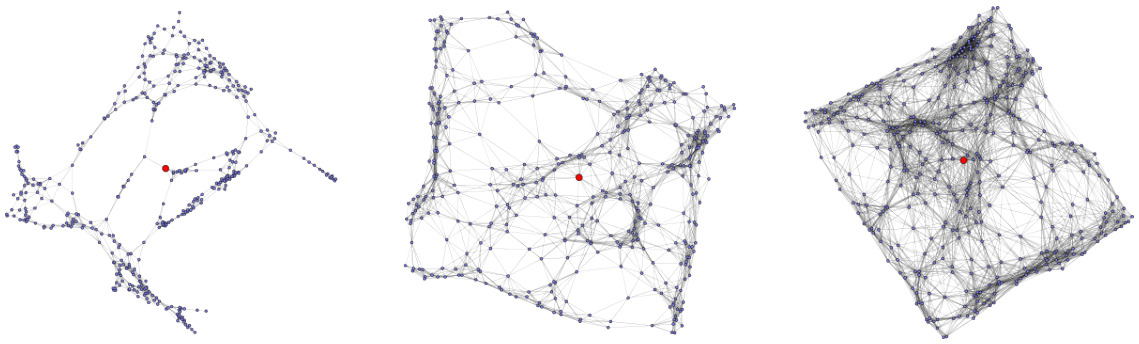

Figure 13: Three example graphs of size 400 and respective degrees 8, 16 and 32

9.1 In safe networks

We observed the average hop count of the routes generated by the routing algorithms in networks which do not contain attackers. We found that Random Walk has a hop count far larger than the other algorithms. For instance, on graphs of degree 8, it reaches 1000 hops per message for networks of 400 nodes, while the other algorithms are all under an average of 25 hops per message. When the degree gets higher, the average number of hops of RW reduces a little, for example 450 hops on networks of 400 nodes and degree 16, but this is still far higher than the other algorithms.

The results with RW removed are shown in Figure 14. As expected, GBR and RGBR return exactly the same route lengths, which are minimal for the tested graphs, and RW's route length is at least about ten times the route lengths of the other algorithms. All the algorithms other than RW are roughly in the same order of magnitude.

We can see that the relative performances of the algorithms are not really dependant on the sizes of the network. However, the degree has a big influence on the performances of GBR and variants, since in networks of equal size, they are always more efficient if the diameter is low, and the diameter goes down as the degree increases.

On the other hand, SR3 does not seem to be influenced much by the degree when considering average route lengths, and instead, seems to depend on the number of nodes. In the worst settings, its hop count is roughly five times the one of PRGBR, and at best, they are similar by a few hops. Compared to the optimal hop count from GBR and RGBR, SR3 generates the shortest routes when the degree of the graph is small. It appears to scale well with the number of nodes.

9.2 In networks under attack

Resilience of a routing protocol against attacks can be seen through the delivery rate of messages, and the standard deviation of the delivery rate between nodes. We executed our algorithm on a variety of networks containing attackers as described in Section 5, and we observed the results.

9.2.1 Against blackholes

When a fixed percentage of randomly placed blackholes is inserted in a network, the protocols behave differently depending on the topology of the network, the information they use, and their parameters. We will first look at networks containing thirty percent of intruders, then at networks with ten percent of attackers, and finally, the fairness of the routing algorithms in these two scenarios.

Figure 15 shows the delivery rates of our panel of algorithms on networks of two hundred nodes, when the degree vary. Both GBR, RGBR, PRGBR and PRDGBR are strongly influenced by the degree of the graph, to the point that a highly connected network (degree 32) can have more than two times the delivery rate of a graph of degree 8. GFG and SR3 are both a little less influenced by the degree of the graph, and RW does not vary.

Figure 16 shows the delivery rates of algorithms while the number of nodes that changes. The degree stays at 8 and there are again 30 percent of attackers. Since the degree stays the same, this means that the increase in the number of nodes makes a wider and more convoluted network. Consequently, routing becomes more difficult, and the delivery rates get lower.

Figure 17 shows the results of simulations in the same conditions as before, but with networks of degree 32. To give a comparison point, this is very close to the experimental conditions of [12], and our results for 300 nodes are similar to theirs for the GBR variant, keeping in mind that there are some small variations between the two settings and modelizations. Overall, the same tendencies as before are seen.

Regarding fairness, we show in Figure 18 the delivery rate and fairness in the case where the network has 400 nodes. We can see that RW has the smallest overall fairness, as expected, since all nodes lose most of their messages. On the opposite side of the spectrum, the deterministic algorithms (GFG, GBR) both have bad fairness, since a node will either deliver all or none of its messages. As seen on the previous figure, the delivery rates of PRDGBR and SR3 are roughly similar, but SR3 has a three times smaller fairness value than PRDGBR, meaning that the delivery rates of individual nodes in networks running SR3 are more grouped than for those running PRDGBR.

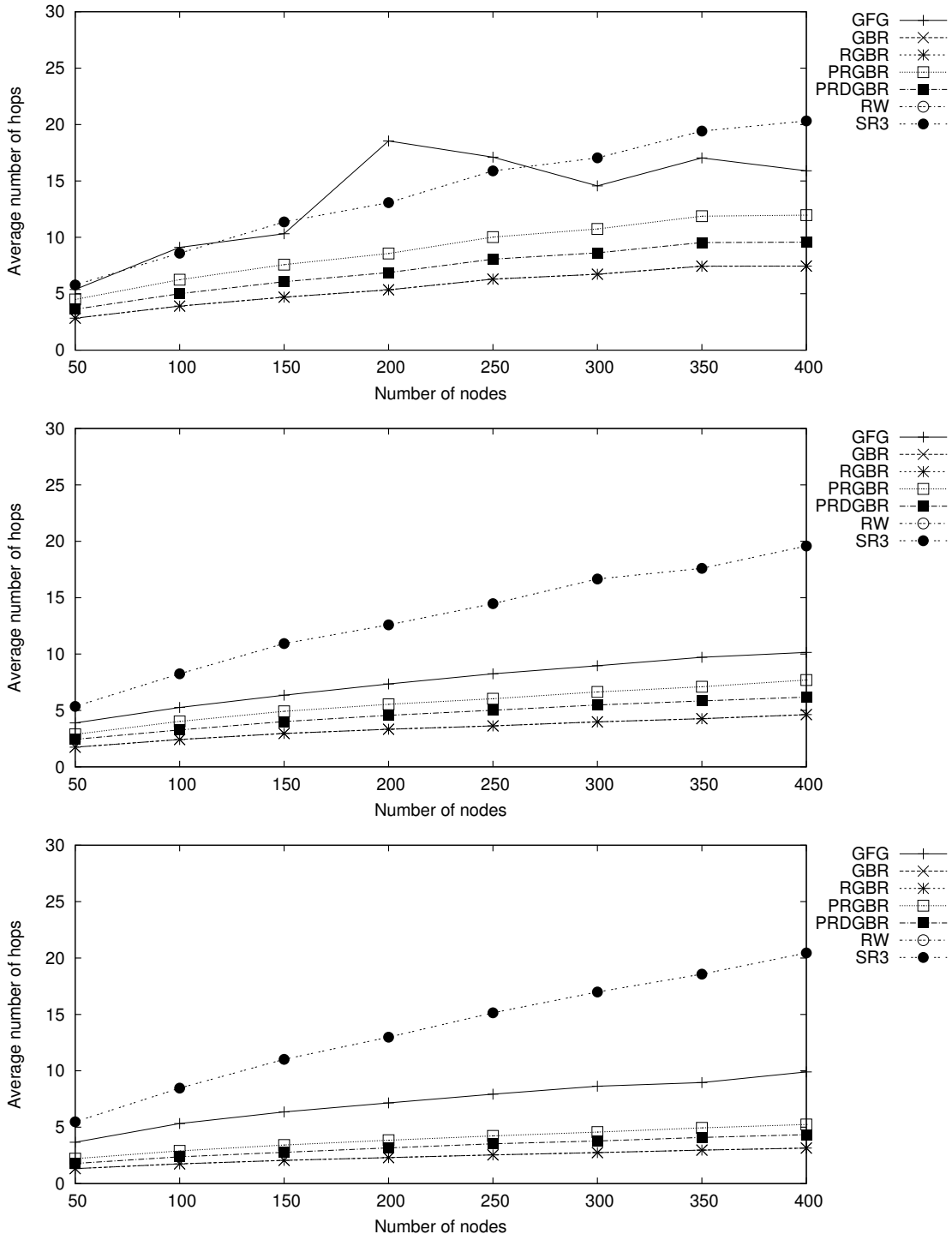

Figure 14: Average route length of the algorithms depending on the network size, when the degree is resp. 8, 16 and 32.

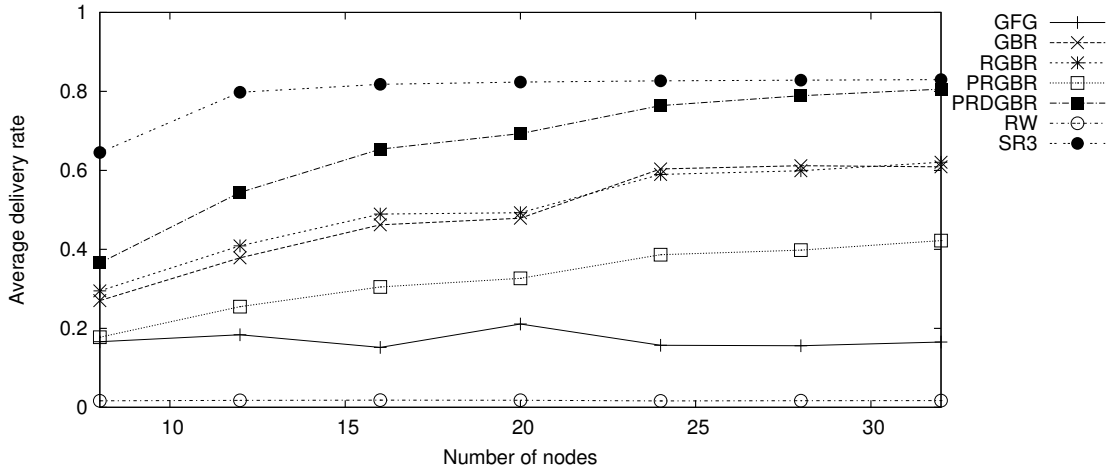

Figure 15: Delivery rates of the routing algorithms in presence of 30% of attackers, in graphs of size 200, with varying degrees

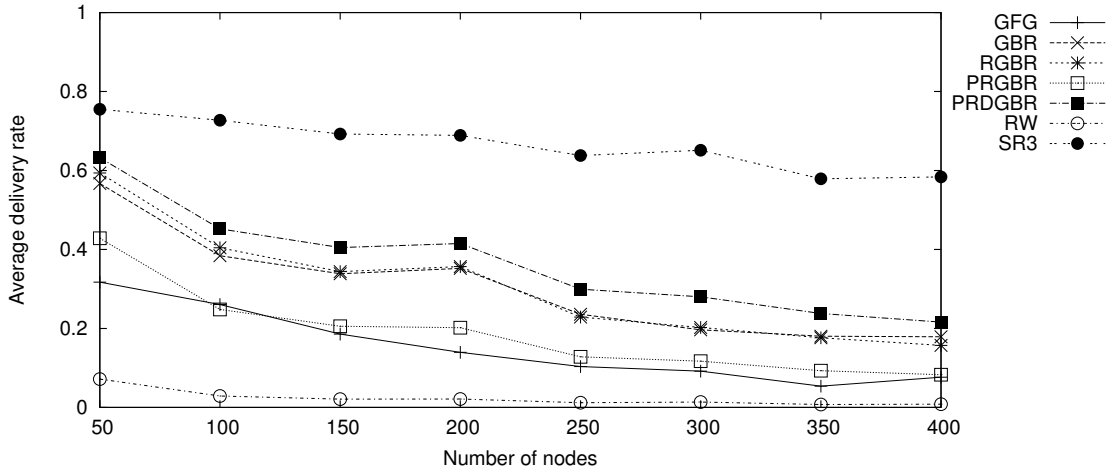

Figure 16: Delivery rates of the routing algorithms in presence of 30% of attackers, in graphs of degree 8, with varying network sizes

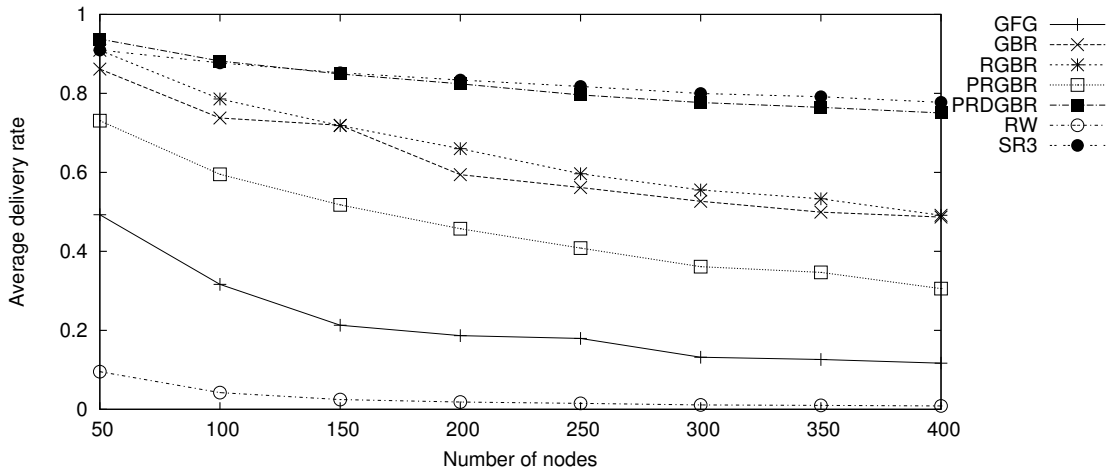

Figure 17: Delivery rates of the routing algorithms in presence of 30% of attackers, in graphs of degree 32, with varying network sizes

Algorithm	Av. delivery rate	Fairness
SR3	0.777	0.060
RW	0.008	0.017
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

Figure 18: Delivery rates and fairness of algorithms, in presence of 30% of attackers, in graphs of degree 32, with 400 nodes

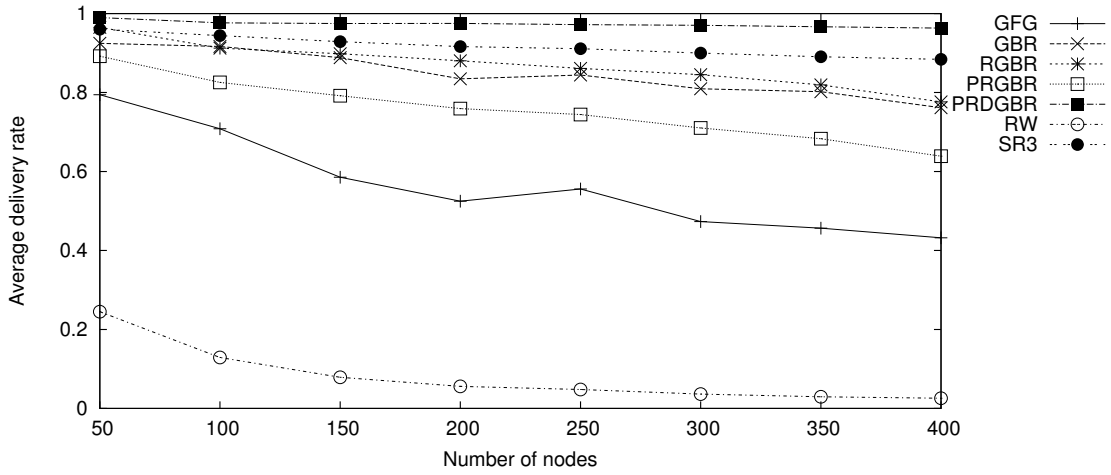

Figure 19: Delivery rates of the routing algorithms in presence of 10% of attackers, in graphs of degree 32, with varying network sizes

Algorithm	Av. delivery rate	Fairness
SR3	0.916	0.022
RW	0.056	0.029
GFG	0.525	0.472
GBR	0.835	0.333
RGBR	0.880	0.161
PRGBR	0.759	0.146
PRDGBR	0.974	0.048

Figure 20: Delivery rates and fairness of algorithms, in presence of 10% of attackers, in graphs of degree 32, with 200 nodes

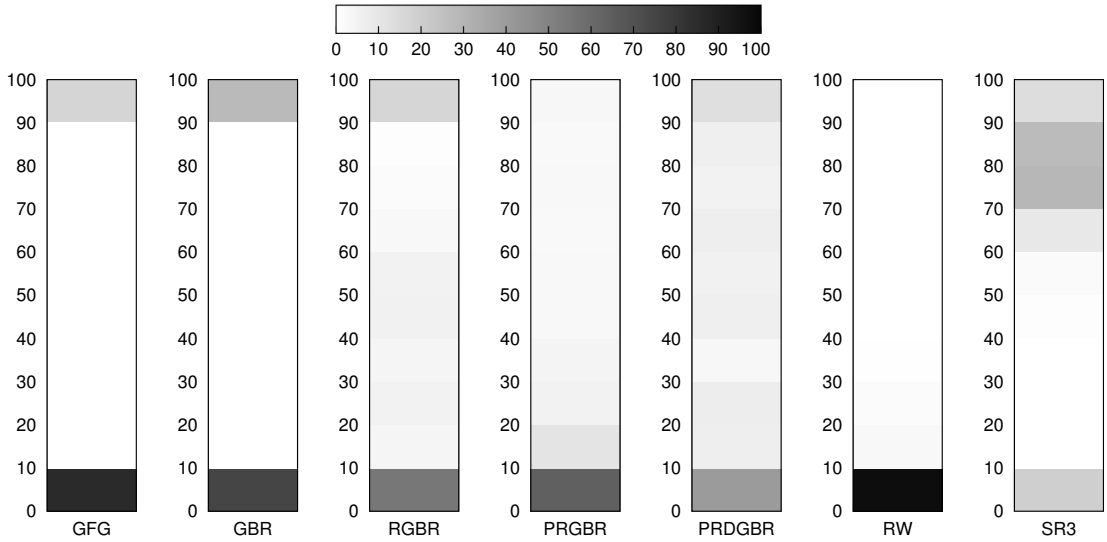

Figure 21: Delivery rates classes of the routing algorithms in presence of 30% of attackers, in graphs of degree 8 and 200 nodes

There are a few cases where SR3 does not have the best average delivery rate. Figure 19 shows the simulation results on networks with the highest degree and smallest non-zero number of attackers of our panel. These are the most favorable conditions for the GBR variants. We also remark that only SR3 and PRDGBR (which duplicates messages at each hop) manages to keep a high delivery rate in large networks, and the duplication may become a problem when the number of nodes increases.

The advantage of PRDGBR regarding the average delivery rate must however be considered with the message replication in mind. As PRDGBR makes a copy of each message at each hop, the overall total of transmissions done is greatly increased, when compared to PRGBR. In the same conditions as before (10 percent of attackers, average degree 32, 400 nodes), the overall number of hops in the simulation is two times greater for PRDGBR than it is for SR3.

Figure 20 shows the average delivery rate and fairness for the networks of 200 nodes, degree 32, and 10 percent of blackholes, which were pictured in the previous figure. PRDGBR and SR3 both have good fairness (resp. 0.05 and 0.02) and comparable average delivery rates, meaning that most messages will get delivered for most of the nodes.

Figure 21 shows the delivery rates classes for 20 networks of degree 8, with 200 nodes, and 30 percent of blackholes. This is a favorable case for SR3, where most nodes reach 70-90 percent of delivered messages. On the other hand, the ther algorithms cause a lot of nodes to have a delivery rate of 0 percent, even when taking into account the nodes which do not have a safe route to the sink, which represent roughly 20

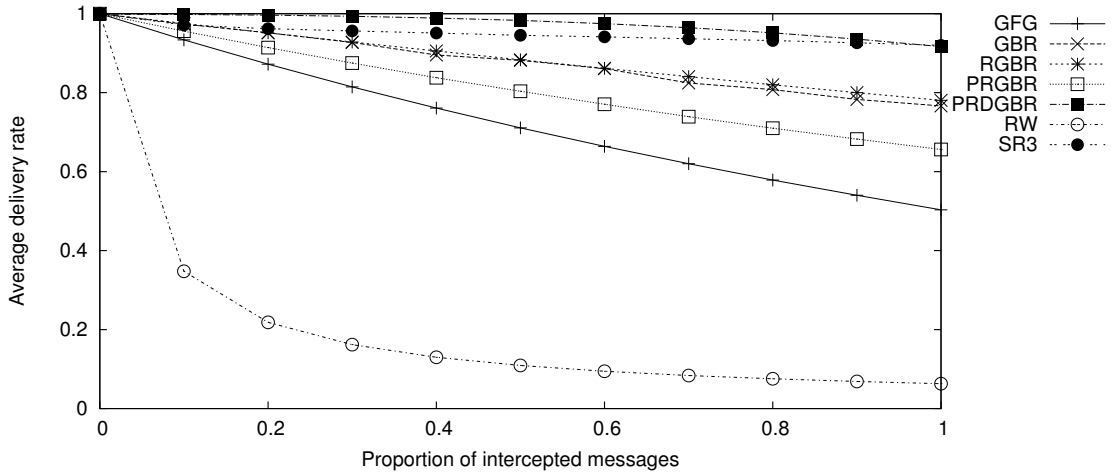

Figure 22: Delivery rates of the routing algorithms in presence of SF attackers, when their interception probability varies

percent of all nodes.

Overall, the SR3 routing algorithm has the best average delivery rate against blackholes on most of our sample graphs, especially on low-degree networks. However, on a few networks with small amounts of attackers and very high degrees, its delivery rate is a little lower than PRDGBR, but still higher than the other algorithms when the network size increases. In all these marginal cases where the delivery rate is similar, the fairness of SR3 is better than all the other algorithms.

9.2.2 Against selective forwarding nodes

Our routing algorithm is designed to detect and avoid nodes which do not deliver messages. However, since we use a reputation mechanism, we also want to evaluate the effect of low-visibility attackers, which do not lose every message, but only a fraction. These attackers will still be present in their neighbor's $L_{Routing}$, and they will thus receive more messages than a plain blackhole.

We chose the same networks as for the experiments regarding black holes, for 160 honest nodes, 40 attackers, and an average degree of 8. However, instead of plain blackholes, we chose selective forwarding intruders, which randomly lose messages with a probability p, and act like an honest node with a probability $1 - p$.

Figure 22 shows our results. As expected, SF attackers stopping all messages give the same results as black holes, and those who do not stop any message give a delivery rate of 100%. Between those extremes, the results depend on the algorithms. Note that PRDGBR is not really affected by SF nodes which drop less than 10 percent of the messages. The important conclusion here is that for all the algorithms we tried, dropping all messages is a more disrupting strategy for an attacker than randomly losing a subset of them.

9.2.3 Against wormholes

We tried all of the algorithms against wormholes, on 20 graphs, which contain 200 nodes, an average degree of 8, 5 percent of blackholes and 5 percent of wormholes. Figure 23 shows the delivery rates which resulted. This attack is especially devastating against GBR variants, as wormholes distort the network, and this distortion amplifies the effect of wormholes once they start dropping messages. Most of the nodes for these algorithms have a delivery rate around 30 percent. These nodes are those which deliver messages to wormholes during the first third of the simulation, and keep sending to those after they become blackholes.

After the behavior switch, SR3 nodes quickly start to avoid attacker nodes, as seen in Figure 24 where the delivery rates in a single simulation run are shown. GFG, due to its use of geographical information,

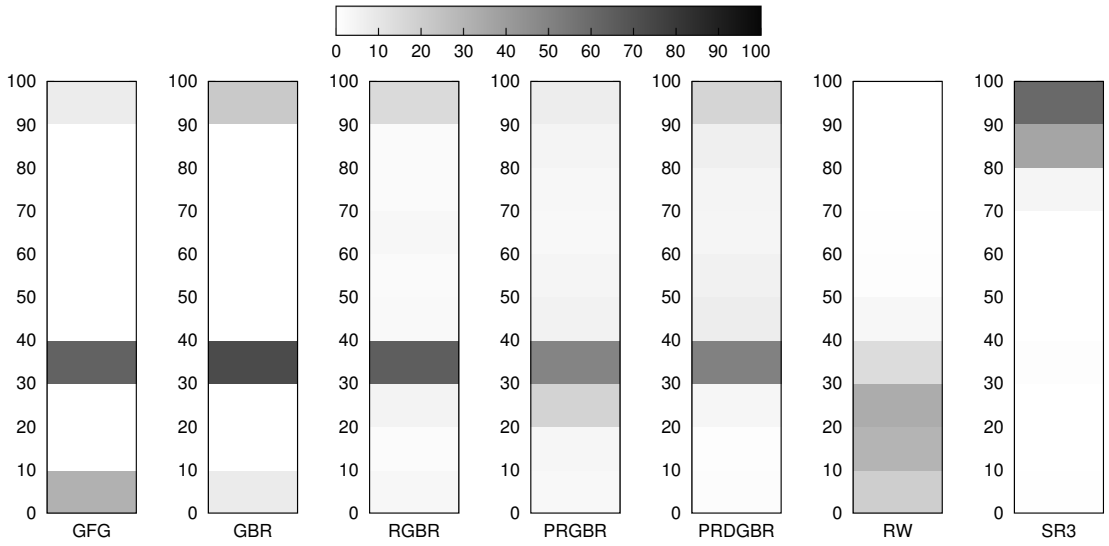

Figure 23: Delivery rates of the routing algorithms in presence of wormholes in 20 different 200-nodes graph, average degree of 8, five percent of blackholes, and five percent of wormholes.

is not affected by wormholes. For both SR3 and GFG, the average delivery rate in the second part of the simulation is the same as for the graphs which only include blackholes, but GFG does not keep a high delivery rate in this case. Overall, only our algorithm resists this attack.

9.2.4 Against Sybil nodes

Figure 25 presents the delivery rates of a network containing 180 honest nodes and 20 Sybil nodes, depending on the number of identities they declare to their neighbors. Sybil nodes declaring a single identity behave as a simple blackhole.

Overall, this attack does not influence much the tested algorithms. As most of them have a random neighbor selection at some point during the routing process, adding declared identities increase the probability for a message to be intercepted. However, this does not compromise the protocols further.

The only exception here is GFG. As Sybil nodes only lie about their identities, and not their positions, GFG's routing process is not affected at all.

10 Conclusion

We proposed SR3, a secure and resilient algorithm for convergecast routing in wireless sensor networks. Using lightweight cryptographic primitives, SR3 achieves data confidentiality and data packet 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 performance of SR3 in a more dynamic environment, *e.g.*, networks with mobile nodes or networks where nodes are added or removed on the fly. Another future work is the effective deployment of SR3 in a WSN testbed platform.

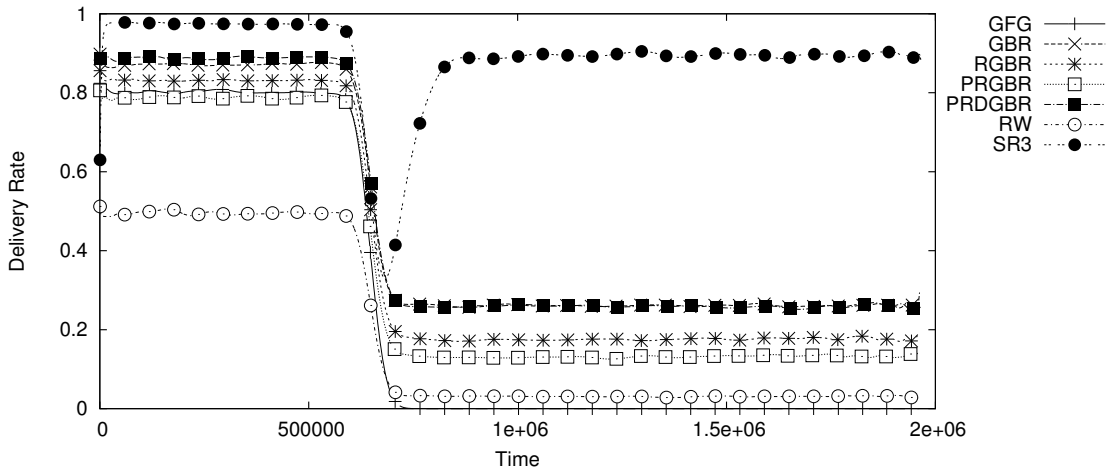

Figure 24: Delivery rates of the routing algorithms in one of the graphs from the same setting as before, over time

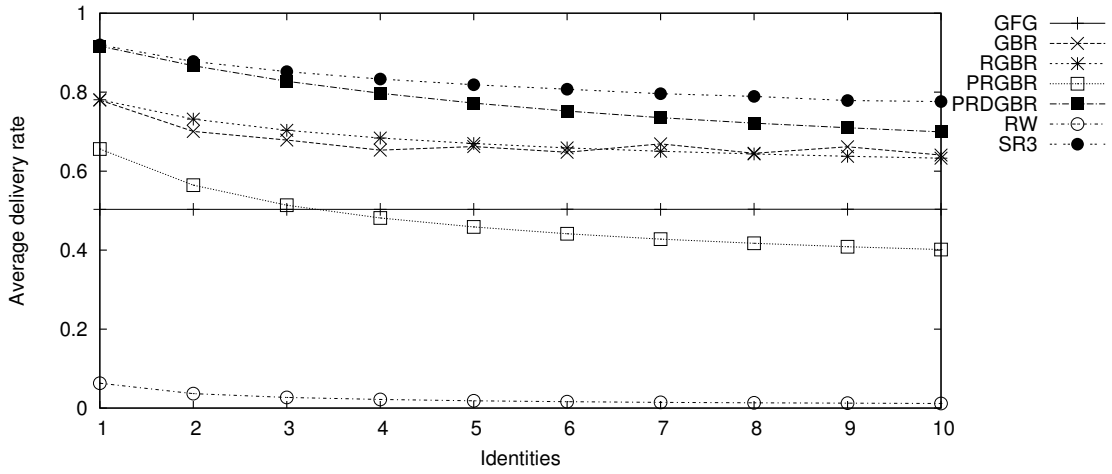

Figure 25: Delivery rates of the routing algorithms in graphs of 200 nodes, degree 8, and 10 percent of Sybil nodes

References

[1] J.N. Al-Karaki and A.E. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, 11(6):6–28, 2004. 2.2

[2] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. Modeling and verifying ad hoc routing protocols. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 59–74, 2010. 2.2

[3] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403, 1997. 4.5, 4.5.1

[4] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000. 4.2.1, 4.2.2, 4.2.4, 4.2.5

[5] 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. 4.2.2

[6] M. Bellare and P. Rogaway. Introduction to modern cryptography. *UCSD CSE*, 207:207, 2005. 4.6.1

[7] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008. 1.1, 4.3

[8] 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. 6.2

[9] L. Buttyán and I. Vajda. Towards provable security for ad hoc routing protocols. In *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 94–105. ACM, 2004. 2.2

[10] A.D. de Mazieux, V. Gauthier, M. Marot, J. Vaudour, and M. Becker. Etat de l'art sur les réseaux de capteurs. *Rapport de recherche INT No 05001 RST*, 2006. 8.3.1

[11] T. Eisenbarth and S. Kumar. A survey of lightweight-cryptography implementations. *Design & Test of Computers, IEEE*, 24(6):522–533, 2007. 1

[12] 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. 1, 2.2, 6.3, 6.3.3, 6.3.4, 7.2.3, 9.2.1

[13] 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. 2.1, 2.2, 3.2

[14] K.R. Gabriel and R.R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Biology*, 18(3):259–278, 1969. 6.2

[15] 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. 2.2

[16] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *Ad hoc networks*, 1(2-3):293–315, 2003. 1, 5, 5.3

[17] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987. 1

[18] J.F. Meyer. Defining and evaluating resilience: A performability perspective. In *Proceedings of the International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS)*, 2009. 1

[19] Victor S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology CRYPTO 85 Proceedings*, volume 218, pages 417–426, 1986. 1

[20] A. Perrig, R. Canetti, J.D. Tygar, and D. Song. The tesla broadcast authentication protocol. 2005. 2.2

[21] 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. 2.2

[22] Axel York Poschmann. *Lightweight cryptography: cryptographic engineering for a pervasive world*. PhD thesis, 2009. 1

[23] C. Schurgers and M.B. Srivastava. Energy efficient routing in wireless sensor networks. In *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, volume 1, pages 357–361, 2001. 6.3, 6.3.1

[24] J.P.G. Sterbenz, E.K. Cetinkaya, M.A. Hameed, A. Jabbar, S. Qian, and J.P. Rohrer. Evaluation of network resilience, survivability, and disruption tolerance: analysis, topology generation, simulation, and experimentation. *Telecommunication Systems*, pages 1–32, 2011. 2.1

[25] D. Wagner. Resilient aggregation in sensor networks. In *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 78–87, 2004. 2.1