SR3: Secure Resilient Reputation-based Routing*

Karine Altisen, Stéphane Devismes, Raphaël Jamet, and Pascal Lafourcade
VERIMAG UMR 5104, Université de Grenoble
Email: Firstname.Lastname@imag.fr

***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.**

***Index Terms*—Wireless sensor networks, routing, security, resiliency**

I. 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 that generate data about the environment (*e.g.*, temperature) and use them for specific services (*e.g.*, emit an alarm when the surrounding temperature is too high). Moreover, they embed wireless communication capabilities allowing them to exchange data. The low capabilities of the sensors, their wireless communications, and the fact that they are deployed in open areas make them prone to attacks.

Routing is a crucial issue in WSNs. Here, we consider a routing scheme called *convergecast* routing. In this problem, a node is distinguished as the *sink* and all non-sink nodes, called *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.

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).

Related Work. Numerous solutions have been introduced to cope with attacks on data. The confidentiality, authenticity, and integrity properties are mainly guaranteed using cryptographic mechanisms. However, the choice of the cryptographic primitives should be led by the inherent constraint of WSNs. WSNs being limited in terms of resource and power, *lightweight* cryptographic mechanisms [5] are mandatory. An example of such a mechanism is *elliptic curve cryptography* [11], [13]. In contrast, classical asymmetric cryptography, *e.g.*, *RSA*, should be excluded due to its computational cost.

Using such lightweight cryptographic primitives, routing protocols implementing some security properties have been proposed, *e.g.*, *μ-Tesla* [15], [12] is a broadcast authentication protocol that enables receivers of the broadcast data to verify that these data really originate from the alleged sources. μ-Tesla has a low communication and computation overhead, scales a large number of receivers, and tolerates packet loss.

Although it is not strictly a routing protocol, *SPINS* [15] is a set of tools for routing, which provides security guarantees without using any costly operations: μ-Tesla is one part of SPINS; the other part is *SNEP*, a packet format that guarantees various security properties, like authentication and confidentiality, using few additional bits per packet.

Some other protocols, called *secure route discovery protocols*, have been introduced [14], [10] to help securing routing. Actually, they compute a valid route (*i.e.*, the computed path exists in the network) between the source and destination, and for some of them (*e.g.*, [14]), they guarantee that nodes in the chosen route achieved a certain security level.[4]

However, all aforementioned solutions do not use specific strategies to combat attacks at the routing level, *e.g.*, selected forwarding, blackhole, *etc.* Specific approaches have been proposed to maintain a good quality of service in presence of insiders, that drop all or part of messages. For example, the notion of *resiliency* has been introduced in [7] as the ability of a network to "continue operating" in presence of compromised nodes, *i.e.*, the capacity of a network to endure and overcome internal attacks. For example, a resilient routing protocol should achieve a "graceful degradation" in the delivery rate

* This paper has been supported by the ANR Project *ARESA2*.

[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.

[4] *E.g.*, the integrity of the discovered route, which means that the computed route has been effectively traversed during the discovery process.

with increasing the number of compromised nodes. In [6], [8], authors experimentally analyze the resiliency of several classical routing techniques, *e.g.*, random walk [1], gradient-based routing [16], geographic routing [4]. The experimental results show that these solutions are weak in terms of resiliency. Then, they propose several variants of the gradient-based routing (*i.e.*, a protocol in which messages are routed following a breath-first spanning tree) to increase its resiliency. 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.

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

SR3 uses lightweight cryptographic primitives — symmetric cryptography, nonces, and hash functions — to achieve security properties: data confidentiality and data packet unforgeability, this latter property implies integrity and authenticity of the data packets. We prove these properties in the computational model using the formal tool *CryptoVerif* [3].

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. Our simulation results show in particular that unlike previous solutions, SR3 self-adapts when compromised nodes change their behavior (*e.g*, an interesting case is when a compromised node behaves well to attract the traffic and then suddenly decide to drop all received messages). We compare our solution to several routing algorithms of the literature. 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*,[5] 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.

Roadmap. The remainder of the paper is organized as follows. In the next section, we present our routing algorithm, SR3. Section III deals with the automatic proof of the security properties of SR3. In Section IV, we present experimental results that show the resiliency of SR3. Section V is dedicated to concluding remarks.

[5] *I.e.*, in absence of data to route the protocol eventually stops.

II. SR3

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

A. 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.

B. 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 [7]. 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.

C. 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.

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

On reception of $\langle$MSG$, C, H, o\rangle$ **from** f
7: $next \leftarrow$ RAND($\mathcal{N}eig_v$,$\mathcal{L}^v_{SR3}(L_{Routing})$)
8: **if** $v = o$ **then**
9: $\langle Data, N_o \rangle \leftarrow$ DECRYPT(C,k_{vs})
10: **if** HASH(N_o) $= H$ **then**
11: $L_{Queue} \leftarrow L_{Queue} \odot \langle N_o, next \rangle$
12: **Send** $\langle$MSG$, C, H, o\rangle$ **to** $next$
13: **end if**
14: **else**
15: **if** $\langle H, \_ \rangle \notin L_{AckRouting}$ **then**
16: $L_{AckRouting} \leftarrow L_{AckRouting} \bullet \langle H, f \rangle$
17: **end if**
18: **Send** $\langle$MSG$, C, H, o\rangle$ **to** $next$
19: **end if**

On reception of $\langle$ACK$, N_o, o\rangle$ **from** f
20: **if** $v = o \wedge \langle N_o, \_ \rangle \in L_{Queue}$ **then**
21: $first\_hop \leftarrow$ GET(L_{Queue}, N_o)
22: $L_{Routing} \leftarrow L_{Routing} \bullet first\_hop$
23: $L_{Queue} \leftarrow L_{Queue} \setminus \langle N_o, \_ \rangle$
24: **else**
25: **if** $v \neq o$ **then**
26: $H \leftarrow$ HASH(N_o)
27: **if** $\langle H, \_ \rangle \in L_{AckRouting}$ **then**
28: $next \leftarrow$ GET($L_{AckRouting}, H$)
29: $L_{AckRouting} \leftarrow L_{AckRouting} \setminus \langle H, \_ \rangle$
30: **else**
31: $next \leftarrow$ RAND($\mathcal{N}eig_v$,$\mathcal{L}^v_{RW}$)
32: **end if**
33: **Send** $\langle$ACK$, N_o, o\rangle$ **to** $next$ **with probability** $\frac{N-1}{N}$
34: **end if**
35: **end if**

Algorithm 2 SR3 for the sink s

Input: $keys[]$: array of shared keys, indexed on node identifiers

On reception of $\langle$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** $\langle$ACK$, N_o, o\rangle$ **to** f
40: **end if**

On reception of $\langle$ACK$, N_o, o\rangle$ **from** f
41: $next \leftarrow$ RAND($\mathcal{N}eig_s$,$\mathcal{L}^s_{RW}$)
42: **Send** $\langle$ACK$, N_o, o\rangle$ **to** $next$ **with probability** $\frac{N-1}{N}$

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 $\langle$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).[6] 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.

D. 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 $L_{Routing}$:

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

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:

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

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.

E. 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.

III. SECURITY ANALYSIS

To evaluate the security properties of SR3, we use the tool *CryptoVerif* [3]. In this tool, the properties of the cryptographic primitives can be modeled using *random oracles* and *games*.

A random oracle works as follows: for every input i, the first time the oracle is queried with i, it returns a value v, picked uniformly at random from its output domain; then, each time it is queried again with i, it returns the same value, v.

Here, the hash function is modeled by a random oracle with output domain of size η_h.

A *game* is an algorithm that evaluates the ability of an adversary (a probabilistic polynomial-time Turing machine) to break a property. A game allows to compute the *advantage* of the adversary, *i.e.*, a measure of how successfully it can

win the game. More formally, the advantage of an adversary $\mathcal{A}$ in a game is the difference between the probability of $\mathcal{A}$ to win the game minus the probability of randomly winning the game. Consequently, the advantage in a game gives the security level of the property: the lower the advantage is, the safer the property is.

Depending on the security property, we assume that the block cipher used in SR3 is either *PRP-CPA* secure or *PRP-CCA* secure [9], each of these two properties being modeled by a game.

Given a permutation family F, *PRP-CCA* works as follows: the adversary $\mathcal{A}$ should guess whether an oracle $\mathcal{O}$ is a permutation extracted from F or a truly random permutation. $\mathcal{A}$ is allowed to make several calls to $\mathcal{O}$ and $\mathcal{O}^{-1}$ (the inverse of $\mathcal{O}$). The advantage of $\mathcal{A}$ in this game is noted $\mathbf{Adv}_F^{PRP\text{–}CCA}(\mathcal{A})$, and depends on the number of calls $\mathcal{A}$ made to $\mathcal{O}$ and $\mathcal{O}^{-1}$. The game *PRP-CPA* is similar to *PRP-CCA*, except that the adversary has only access to $\mathcal{O}$. The advantage of $\mathcal{A}$ in this game is noted $\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{A})$, and depends on the number of calls $\mathcal{A}$ made to $\mathcal{O}$. Note that *PRP-CPA* is weaker that *PRP-CCA*, *i.e.*, for every adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$, using the same resources as $\mathcal{A}$, such that $\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{A}) \leq \mathbf{Adv}_F^{PRP\text{–}CCA}(\mathcal{B})$.

We assume that the block cipher of SR3 is *PRP-CCA-secure*. Therefore, both $\mathbf{Adv}_F^{PRP\text{–}CCA}(\mathcal{A})$ and $\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{A})$ becomes *negligible* in the size of the block cipher, η_c, *i.e.*, for every positive polynomial P we have $\exists K, \forall \eta_c > K, \mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{A}) < \frac{1}{P(\eta_c)}$.

In CryptoVerif, we analyzed the confidentiality of the data and the nonces as well as the unforgeability of the MSG messages in SR3. Each of these three properties is evaluated thanks to a game. For each game, CryptoVerif outputs a bound on the advantage of any adversary in that game. This bound is obtained after successive game reductions.

Since honest nodes change neither the MSG messages nor the ACK messages while they are routed through the network, we can consider only two participants evolving in a hostile network for our security analysis: a source and the sink.

We first studied the *confidentiality of the data* in SR3 using a *Find-then-Guess* (FG) game. This game is played in two phases. First, the adversary $\mathcal{A}$ outputs two data, d_0 and d_1. Then, one of them is selected uniformly at random, and a MSG message m is generated with the selected data. Finally, m is given to $\mathcal{A}$. To win, $\mathcal{A}$ must guess which of the two data is in m. Throughout this game, $\mathcal{A}$ has access to an oracle which, when queried with some data d, returns two outputs: a (new) MSG message m' containing d, and the nonce inside m'. The bound of advantage output by CryptoVerif depends on the ability to break the block cipher, *i.e.*, this FG game can be reduced to a PRP-CPA game. So, the bound on this FG game depends on the advantage $\mathbf{Adv}_F^{PRP\text{–}CPA}(\mathcal{B})$ of any adversary $\mathcal{B}$ in a PRP-CPA game. Formally, for all adversaries $\mathcal{A}$ making q_G data message-building queries and q_H queries to the hash function, there exists an adversary $\mathcal{B}$ (making $q_G + 1$ queries to $\mathcal{O}$) such that the advantage of $\mathcal{A}$ in this FG game is smaller than

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

where η_n is the size of the nonces in the MSG messages. Note that $\eta_c \leq \eta_n$. Therefore, this bound becomes negligible when increasing η_n.

We then studied the *Unforgeability under Chosen Message and Verification Attack* (UFCMVA) of the MSG messages, a property which implies both *indistinguishability* and *authenticity* of the MSG messages. To that goal, we modeled a UFCMVA game that measures the ability of an arbitrary adversary to come up with a MSG message which is both valid and original. A MSG message m is original if and only if the data and the nonce in m have never been used before in any other MSG message. Here, the adversary can access two oracles: the first one, given some data d, returns a MSG message containing d, and the second one evaluates whether a MSG message is valid.

As in the previous game, the advantage of $\mathcal{A}$ depends on the strength of the block cipher. Formally, for all adversaries $\mathcal{A}$ making q_G queries to the message generation oracle, q_V queries to the message verification oracle, and q_H queries to the hash function, there exists an adversary $\mathcal{B}$ making q_G queries to $\mathcal{O}$ and $q_V + 1$ queries to $\mathcal{O}^{-1}$ such that the advantage of $\mathcal{A}$ in the UFCMVA game is smaller than

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

Once again, this bound becomes negligible when η_h and η_n increase.

We also showed that an adversary cannot acknowledge a MSG message if that message has not been effectively delivered, by showing the *confidentiality of the nonces* in MSG messages which have not yet reached the sink. The corresponding game consists in giving a MSG message m to an adversary $\mathcal{A}$ that can call the two same oracles as in the UFCMVA game. To win, $\mathcal{A}$ must guess the nonce that is inside m. $\mathcal{A}$ is allowed to make q_A tries. Once again, the strength of the block cipher is crucial in this game. For all adversaries $\mathcal{A}$ making q_G queries to the message generation oracle, q_V queries to the message verification oracle, and q_H queries to the hash function, there exists an adversary $\mathcal{B}$ making $q_G + 1$ queries to $\mathcal{O}$ such that the advantage of $\mathcal{A}$ in this game is smaller than

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

Once again, this bound is negligible when η_h and η_n increase.

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 AES-192 as block cipher (this allows 64 data bits), the best attack known to this day needs $2^{189.7}$ operations. Therefore, we can expect $\mathbf{Adv}_F^{PRP-CCA}(\mathcal{B})$ 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. All our security proofs are given in [2].

IV. Experimental Results

We ran simulations in *Sinalgo*,[7] an event-driven simulator for WSNs, to study the resiliency of SR3 against several attack scenarios. We compared its performance to those of six other routing protocols. Due to space limitation, only few of our results are presented here, but these results are representative of the general trends we observed in all our simulations. All our experimental results are available in [2].

A. Experimental Conditions

We deployed sensors uniformly at random on a square plane. We positioned the sink at the center of the square plane. Two nodes can communicate if and only if their Euclidean distance is less or equal to a preset fixed range, *i.e.*, the topology is a UDG. The compromised nodes are selected uniformly at random among the sensors. We only considered *connected* UDGs. The communication links are asynchronous and FIFO. The transmission time of each link follows an exponential random distribution of constant parameter. Only honest sensors generate data to route. The time between two consecutive data generations at the same sensor also follows an exponential random distribution, whose parameter is the same for all sensors and whose value depends on the average degree and the number of sensors in the network, to prevent congestion.

If we fix the number of nodes n and the range, we can tune the size of the simulation area to control the average degree $\overline{\delta}$ of the network. In our simulations, n varies from 50 to 400 and $\overline{\delta}$ varies from 8 to 32. The percentage of compromised nodes varies from 0 to 30%. We considered various attack scenarios, where compromised nodes made *selective forwarding*: each compromised node drops received messages with a probability $p \in (0..1]$ (if $p = 1$, the node is called a *blackhole*). In addition, some compromised nodes may have some additional "bad" skills (they may be *wormholes* or *Sybil*).

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

B. Parameters of SR3

Our algorithm requires four parameters: N (an upper bound on the number of nodes) and the maximum sizes of each list, respectively denoted by s_R, s_Q and s_A. In each simulation, we set N to the actual number of nodes. We respectively set s_R, s_Q, and s_A to 10, 3, 5. The choice of these values has been led by an experimental evaluation, detailed in the technical report [2]. These values ensure good performances, while keeping the memory consumption low.

[7] http://www.disco.ethz.ch/projects/sinalgo/

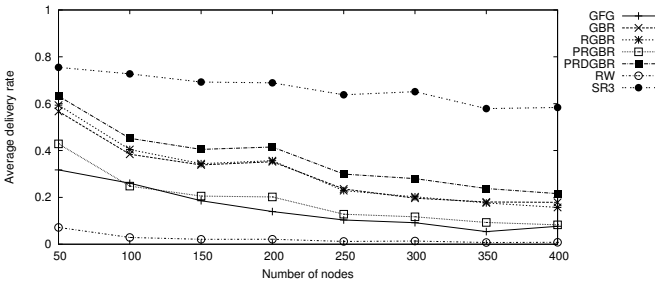

Fig. 1. Average delivery rate (30% of BH nodes, $\overline{\delta} = 8$)

C. Benchmark Protocols

We selected a panel of six protocols from the literature: the uniform Random Walk (RW) [1], the Greedy-Face-Greedy protocol (GFG) [4], the gradient-based protocol (GBR) [16], and three of its variants (RGBR, PRGBR, PRDGBR) [6]. Note that RGBR, PRGBR, and PRDGBR have been introduced for resiliency purpose. GFG is a geographic routing protocol, where two modes are alternatively used: Greedy and Face. The Greedy mode is preferably used, but may lead a message to a dead end. In this case, the Face mode allows the message to escape from this dead end. GBR consists in routing messages along the breadth-first spanning tree (BFS tree) rooted at the sink. RGBR uses the levels of neighbors in the BFS tree: each sensor chooses the next hop for each message uniformly at random among its lowest-level neighbors. In PRGBR, each sensor chooses between two modes: (1) with probability 0.4 the message is routed according to RGBR; (2) with probability 0.6 the message is routed to a neighbor of same level (if no such a neighbor exists, the sensor uses mode 1). PRDGBR duplicates the messages at each hop and routes the two messages independently using PRGBR. To avoid congestion, each node drops the received copies of messages it already saw.

D. Some Scenarios and Results

1) Average Delivery Rate: Figure 1 shows the delivery rates observed in networks of average degree $\overline{\delta} = 8$ facing 30% of blackholes (BH), that is, nodes that drop all messages they receive. The size of the networks varies from 50 to 400 nodes. (Note that, with 30% of blackholes, several honest nodes cannot safely reach the sink and consequently have delivery rate zero.) We can remark that SR3 always offers a better delivery rate than the other protocols. In particular, the greater the networks are, the greater the gap is.

Figure 2 shows the delivery rates observed in networks of size $n = 200$ facing 30% of blackholes (BH). The average degree of the networks varies from 8 to 32. Again, we can remark that SR3 always offers a best delivery rate in that case. Moreover, as RW and GFG, the average delivery rate

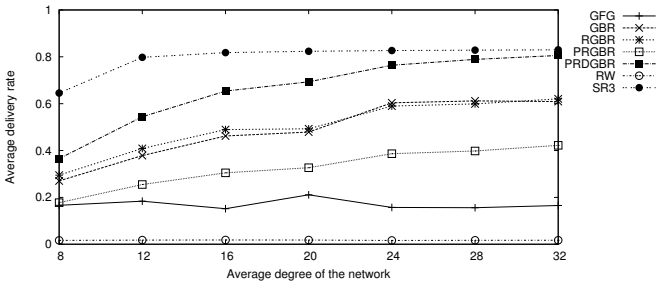

Fig. 2. Average delivery rate (30% of BH nodes, $n = 200$)

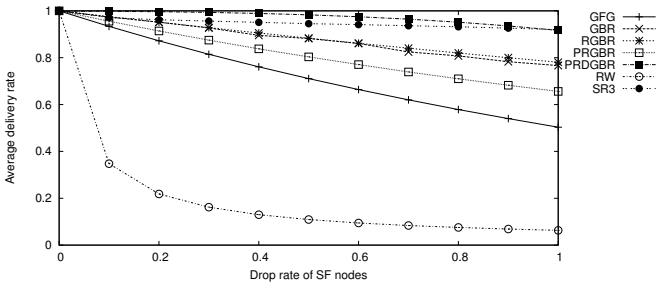

Fig. 3. Average delivery rate (20% of SF nodes, $n = 200$, $\overline{\delta} = 8$)

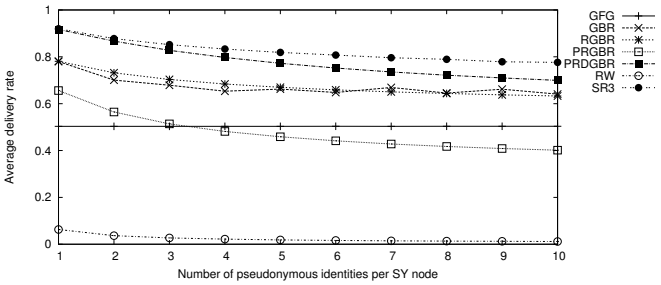

Fig. 4. Average delivery rate (10% of SY nodes, $n = 200$, $\overline{\delta} = 8$)

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

Fig. 5. Average delivery rate and standard deviation of the delivery rate of nodes (30% of BH, $n = 200$, $\overline{\delta} = 32$)

of SR3 is insensitive to the degree variation. In contrast, the observed delivery rates for gradient-based protocols are low in sparse networks. In high-density networks, the performances of PRDGBR match those of SR3. However, SR3 use only two messages per data, while PRDGBR duplicates the messages at each hop, and consequently heavily increases the load of the network.

Figure 3 shows that SR3 also efficiently combats the selective forwarding (SF) attacks. We show the average delivery rates observed in networks of size $n = 200$ and average degree $\overline{\delta} = 8$ that have to face 20% of compromised nodes, according to the drop rate. We can observe that, except RW, all protocols of the panel achieve a graceful degradation in the delivery rate when the drop rate increases. Still, SR3 has the best performance. Only PRDGBR has performances closed to those of SR3, when the drop rate is of 100% (that is, when compromised node are actually blackholes). But again, this performance comes at the price of a high communication overhead.

We also considered networks of size $n = 200$ and average degree $\overline{\delta} = 8$, where 10% of nodes are both blackholes and Sybil (SY). The number of pseudonymous identifiers of these compromised nodes varies from 1 to 10. We can observe in Figure 4, that except for GFG, adding Sybil nodes does not change the relative performances in the panel. Actually, GFG is insensitive to Sybil attacks because it does not use node identifiers. Now, still in that case, SR3 offers the best performances.

2) Fairness: Fairness among the delivery rates of honest nodes is an expected property in routing protocols. A classical way to capture this property is to compute the standard deviation of the delivery rates of honest nodes. Figure 5 shows the average and standard deviation of delivery rates observed in networks of size $n = 200$ and average degree $\overline{\delta} = 32$, when facing 30% of BH nodes. The smaller the standard deviation is, the fairer the algorithm is. Now, a shortcoming of this measure is that when the delivery rates are uniformly bad (like for example in RW), the observed fairness is good. So, analyzed alone, this measure is misleading.

Instead, we propose here to visualize the distributions of delivery rates. Figure 6 shows an example of our method. In this figure, we consider the same simulations as in Figure 5. There is one column per algorithm of the panel. Each column represents the range of possible delivery rates from 0 to 100%, by intervals of 10%. The color shade encodes the proportion of nodes having the corresponding delivery rate. Consider, for example, the RW protocol: almost all nodes have a delivery rate of less than 10%. In contrast, using SR3, almost all nodes have a delivery rate greater or equal to 70%. We can clearly observe two classes of processes when looking at GFG and GBR: nodes have either 0% or 100% of delivery rates; these protocols are unfair. The probabilistic variants of GBR are fairer: the delivery rates are spread on the whole range, but still these results are weaker than those observed for SR3.

3) Average Number of Hops: Here, we are only interested in the messages that are successfully delivered. So, we consider safe networks. Figure 7 shows the average number of hops of data messages in networks of average degree $\overline{\delta} = 16$, where the size n varies from 50 to 400. First, note that we do not show results for RW in the figure because they are drastically worse than other protocols of the panel, *e.g.*, for 50 nodes, its average number of hops is 40, and for 400 nodes, its average number of hops is 529. Then, by definition, routes followed using GBR or RGBR are optimal. Finally, SR3 generates longer routes than the geographical and gradient-based protocols due to its lack of knowledge about the

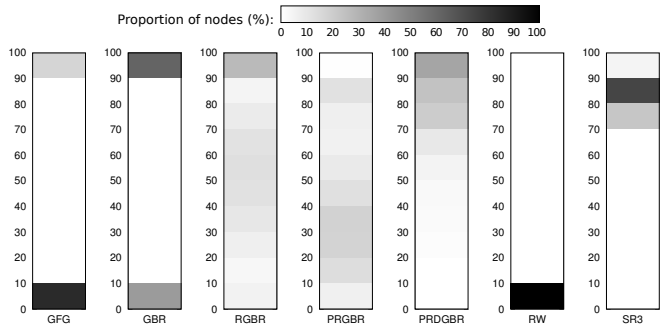

Fig. 6. Average delivery rate distribution (30% of BH, $n = 200$, $\overline{\delta} = 32$)

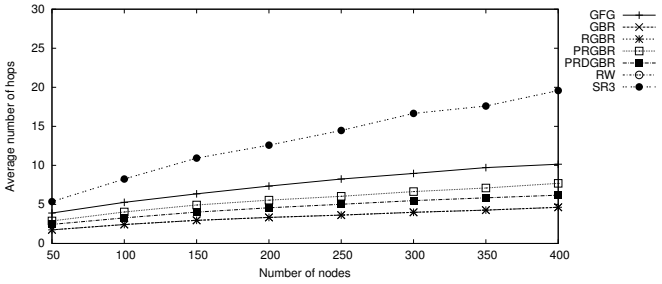

Fig. 7. Average number of hops in safe networks ($\overline{\delta} = 16$)

network. However, this length stays reasonable (*i.e.* we always observed lengths drastically smaller than n), and scales with the number of nodes.

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

V. Concluding Remarks

We proposed SR3, a secure and resilient algorithm for convergecast routing in wireless sensor networks. Using lightweight cryptographic primitives, SR3 achieves data confidentiality and 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,

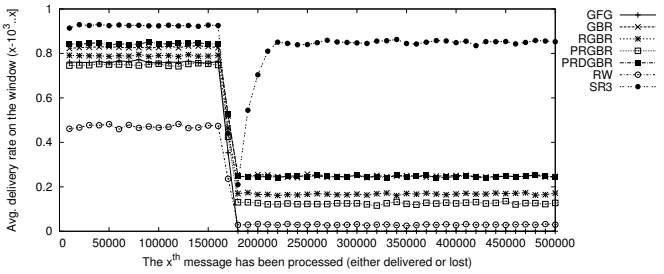

Fig. 8. Average delivery rate (5% of WH/BH, 5% of BH, $n = 200$, $\overline{\delta} = 8$)

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/removed on the fly. Another future work is the effective deployment of SR3 in a WSN testbed platform.

References

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

[2] K. Altisen, S. Devismes, R. Jamet, and P. Lafourcade. SR3 : A secure and resilient reputation-based routing protocol. Technical Report TR-2013-4, VERIMAG, 2013. www-verimag.imag.fr/TR/TR-2013-4.pdf.

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

[4] 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.

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

[6] 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.

[7] O. Erdene-Ochir, M. Minier, F. Valois, and A. Kountouris. Resiliency of wireless sensor networks: Definitions and analyses. In *Telecommunications (ICT), 2010 IEEE 17th International Conference on*, pages 828–835, 2010.

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

[9] S. Goldwasser and M. Bellare. Lecture Notes on Cryptography. 2008.

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

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

[12] D. Liu and P. Ning. Multilevel tesla: Broadcast authentication for distributed sensor networks. *ACM Transactions in Embedded Computing Systems (TECS)*, 3:800–836, 2004.

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

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

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

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