# Evaluation of Secure Multi-Hop Node Authentication and Key Establishment Mechanisms for Wireless Sensor Networks

Ismail Mansour, Gérard Chalhoub, Pascal Lafourcade

*J. Sens. Actuator Netw.* **2014**, *3*, 224-244; doi:10.3390/jsan3030224

OPEN ACCESS

***Journal of Sensor and Actuator Networks***

**ISSN 2224-2708**

www.mdpi.com/journal/jsan

*Article*

# Evaluation of Secure Multi-Hop Node Authentication and Key Establishment Mechanisms for Wireless Sensor Networks

**Ismail Mansour** [1,2], **Gérard Chalhoub** [1,2,*] **and Pascal Lafourcade** [1,2]

[1] Clermont Université, Université d'Auvergne, LIMOS, BP 10448, F-63000, Clermont-Ferrand, France; E-Mails: ismail.mansour@udamail.fr (I.M.); pascal.lafourcade@udamail.fr (P.L.)

[2] CNRS, UMR 6158, LIMOS, F-63173 Aubière, France

* Author to whom correspondence should be addressed; E-Mail: gerard.chalhoub@udamail.fr; Tel.: +33-4-73-17-70-48; Fax: +33-4-73-17-71-11.

*Received: 12 June 2014; in revised form: 25 August 2014 / Accepted: 26 August 2014 / Published: 3 September 2014*

---

**Abstract:** Designing secure authentication mechanisms in wireless sensor networks in order to associate a node to a secure network is not an easy task due to the limitations of this type of networks. In this paper, we propose different multi-hop node authentication protocols for wireless sensor networks. For each protocol, we provide a formal proof to verify the security of our proposals using Scyther, which is an automatic cryptographic protocols verification tool. We also provide implementation results in terms of execution time consumption obtained by real measurements on TelosB motes. These protocols offer different security mechanisms depending on the design of the protocol itself. Moreover, we evaluate the overhead of protection of each solution by studying the effect on execution time overhead of each protocol. Finally, we propose a mechanism to detect possible attack based on our evaluation results.

**Keywords:** authentication; wireless sensor networks; security; overhead of protection; multi-hop; formal verification

---

## 1. Introduction

Wireless sensor networks (WSNs) are increasingly used in critical applications where the identity of each communicating entity should be authenticated before exchanging data in the network. The wireless

nature of this technology makes it easy for intruders to try to intervene in the network activity and create any of the known attacks in WSNs [1]. Many of the current propositions focus on message authentication for ensuring data authentication and integrity, and others focus on user authentication to give access to the network for certain user profiles. In this paper we propose, on one hand, authentication protocols that allow nodes to join the network in a secure manner, and on the other hand, key establishment protocols that allow nodes to establish keys with nodes that are multiple hops away.

Designing secure protocols is an error-prone task. One of the well-known examples is the famous flaw found in the Needham–Schroeder protocol seventeen years after its publication [2]. This protocol aims at establishing a new key between two participants. It is simple since it uses only three exchanges of messages. It clearly shows that designing secure protocols is not an easy task and is error-prone. During the past decades, several automatic tools for verifying the security of cryptographic protocols have been elaborated by several authors, e.g., Proverif [3], Avispa [4] or Scyther [5]. These symbolic tools use the Dolev–Yao intruder model [6], which considers that the intruder is controlling the network and makes the perfect encryption hypothesis (meaning that it is impossible to obtain the plain text of an encrypted message unless the secret key is known). The state-of-the-art [7] shows that formal methods are now mature and efficient enough to be used in the design of security protocol in order to avoid such logical flaws.

Another aspect that should be taken into account during WSNs protocol analysis is the performance related to the security operations. Traditional approach assumes that the best way is to apply the strongest possible security measures to make the system as secure as possible. Unfortunately, such reasoning leads to the overestimation of security measures, which causes an unreasonable increase in the system load [8]. The system performance is especially critical in systems with limited resources such as wireless sensor networks or mobile devices. It is important to design secure mechanisms using, when possible, a fixed number of cryptographic operations regardless of the network size, and to be able to evaluate their performance in such constrained devices. In this paper, we propose several protocols that do not add cryptographic operations on intermediate nodes. We systematically evaluate their overhead and prove their correctness with the automatic formal security verification tool Scyther [5].

The originality of our work resides in the fact that it suites large scale multi-hop wireless sensor networks, which is due to the limited number of cryptographic operations regardless of the number of hops separating the communicating nodes. In addition, it combines several aspects of security, from designing secure protocols to evaluating the implementation of our solution, going through formal automatic analysis of security and overhead of protection analysis. Our contributions can be summarized in the five following points:

1. Design of multi-hop node authentication mechanisms.
2. Formal automatic analysis of our solutions.
3. Implementation on TelosB motes.
4. Evaluation of the overhead of protection of our solutions.
5. Attack detection based on the *a priori* overhead evaluation.

Our main contribution is the design of several secure authentication protocols. In order to avoid security flaws, we prove the correctness of all our protocols automatically using Scyther [9], which

is a tool for the automatic verification of security protocols. This tool considers neither the cost of the communications nor the execution time of the cryptographic operations. Therefore we have implemented our protocols on TelosB motes in order to obtain the execution time and communication. Note that we used TelosB motes as a means for comparing the different protocols. We would have obtained better overall performance if we used motes with more computation capacities such as Tmotesky.The overhead of protection analysis for WSN cryptographic protocols is almost impossible to perform manually. This increases the difficulty to design secure and efficient protocols at the same time. Using our real implementation on TelosB motes, we have measured the execution time of every cryptographic operation and every step in each protocol. These measurements were obtained from a testbed of the minimal number of nodes. In addition, using the overhead evaluation results, we were able to put in place an attack detection process.

This paper does not aim to show that the proposed protocols are faster than existing protocols. Instead, the main objective is to present secure solutions for node authentication and key establishment using standard cryptographic algorithms and primitives available on the Internet. The chosen primitives can be replaced by more optimized primitives for better performance. The evaluation we presented is essentially a proof of concept and shows that the steps of each protocol can be implemented on low cost motes such as TelosB motes that are known for their low capacities in computation speed.

This work is an extension of the conference paper [10] and the book chapter [11]. The original authenticated join protocol was proposed in [11] and evaluated in different versions according to the level of protection in [10]. It is the only intersection with the current paper.

In the next section, we summarize the related work. In Section 3, we present the cryptographic primitives we use to design our protocols. Then in Section 4, we propose two protocols for establishing secure multi-hop communications. In Section 5, we give two key establishment protocols that use the sink as a trusted third party and in Section 6 we propose four protocols that establish a new key without passing by the sink. Then in Section 8, we use Scyther to formally prove the security of our solutions. In Section 9, we evaluate the overhead of our eight protocols. Based on the overhead evaluation results, we explain in Section 10 how a node is able to detect attack attempts launched by other nodes in the network. We conclude the paper in the last section.

## 2. Related Work

Very few works have been done for node authentication protocols in multi-hop WSNs. Most of the existing authentication protocols proposed for WSNs neglect the multi-hop factor. In [12], Al-mahmud *et al.* propose a protocol where the base station broadcast authentication elements for in-range sensor nodes to be able to authenticate new arriving nodes. In fact, they consider that any previously authenticated node can authenticate new nodes using identity-based signature (IBS) algorithm, Elliptic Curve Cryptography (ECC) and a digital signature algorithm (DSA). They provide an informal security analysis and have not implemented their protocols.

In [13,14], the authors propose an authentication mechanism for users and consider that sensor nodes inside the WSN are trusted nodes. In [14], Yeh *et al.* show the weakness of some protocols proposed in [13], due to lack of authentication. Then they propose a stronger authentication protocol using

ECC that ensures mutual authentication and protection against attacks from other users, which was not included in [13]. They also provide a manual security analysis of their solution and only a theoretical complexity evaluation of their protocols.

Recently in [15], Han *et al.* propose an authentication model that aims at reducing overhead for the re-authentication of sensor nodes using symmetric and public key cryptography. It is based on a ticket encrypted using a common secret key between neighbouring fixed nodes. This ticket is sent to a mobile node during the first authentication phase. This ticket is only useful when the mobile node decides to re-authenticate with this neighbour fixed node. In addition, the protocol only works well when the fixed node is in direct range with the base station, and the initial authentication phase suffers from internal attacks. Sinks in the network can easily take the place of each other when they are not in communication range with the base station. The authors only give a quick informal security analysis of their solutions and they have not deployed their solutions on real motes.

In [16], Manjusha *et al.* propose an authentication technique using ECC and ElGamal digital signature scheme for message authentication. The authors do not perform any security analysis of their solution. Moreover, the proposition is evaluated on a computer using MATLAB simulator, which makes it difficult to estimate compared with a real testbed implementation, since digital signatures and public key cryptographic primitives are very resource consuming operations.

In [17], Zhang *et al.* propose a node authentication protocol for hierarchical WSNs. The hierarchical topology is limited to a base station, cluster heads and sensor nodes. The cluster heads can reach the sensors of their clusters directly, and can also reach the base station directly. The authentication is based on hash chain functions and symmetric encryption. The proposed protocol is not resilient to insider attacks as cluster heads are trusted to forward join requests to base station. In addition, the coverage of the network is limited due to the limited number of hops in a hierarchical topology.

Many contributions have been made in the key establishment and symmetric key distribution in WSNs using symmetric cryptography and hash functions. Some of them are based on a probabilistic pre-distribution that guarantees that any two nodes in the network are able to share a key with a certain probability, and others are deterministic but cause more storage overhead [18]. They do not evaluate on real motes their solutions and the security analysis is only informal. One of the most known symmetric key systems that were proposed for wireless sensor networks is SPINS (Security Protocols for Sensor Networks) [19], which uses a simplified version of TESLA (Timed, Efficient, Streaming, Loss-tolerant Authentication) protocol [20]. Perrig *et al.* test several symmetric encryption functions and MAC (Message Authentication Code) for their protocols using an extremely limited sensor network platform. They show that data transmission takes 71% of the time. In SPINS, the base station plays an essential role in the key establishment process. It is a lightweight cryptography protocol but has limited scalability and depends heavily on the base station.

Munivel *et al.* in [21] propose a multi-hop key establishment between nodes called Micro-PKI, using ECC and MAC. Their method is based on the pre-distribution of the public key of the base station. Using this public key, every node is able to create a secret key with any other node of the network. They evaluate the energy consumption of their solution and perform an informal security analysis. The authentication process in this proposal is only dependent on the public key of the base station; if a node has this key,

it is considered authenticated. This makes the procurement of this public key very critical for the whole security architecture.

In [22] Chan *et al.* propose PIKE (Peer Intermediaries for Key Establishment), one of the most famous key establishment protocols that is not dependent on a central trusted node. According to PIKE, symmetric keys are pre-deployed in the nodes in such a way to guarantee that any two nodes in the network have at least one node in common with which each one of the two nodes has a secret key with it. Therefore, these two nodes are able to establish a secret key by using the trusted channel established with the common node. They provide a simulation evaluation of the protocol and estimate the energy and memory cost of several schemes. They also give an informal security analysis of the protocol against node compromise. PIKE suffers from high memory storage to make sure that nodes are able to find at least one node in common to establish a new key.

CARPY and CARPY+ are proposed in [23]. They are based on the symmetric keys of Blom [24] with a perturbation function that makes it more difficult for an attacker to guess the pairwise keys. In their paper they discuss the five criteria proposed in [25] and claim that CARPY+ satisfies all of them. These criteria are:

- *Resilience to the Adversary's Intervention* during the key establishment phase,
- *Directed and Guaranteed Key Establishment* for any couple of nodes in the network,
- *Resilience to Network Configurations*, *i.e.*, nodes should be able to establish keys in any kind of network topology,
- *Efficiency* of the key establishment process in terms of memory storage, communication overhead and complexity,
- *Resilience to Dynamic Node Deployment*, which allows nodes to be added at any time to the network and establish keys on demand.

The authors give a detailed security analysis including a security model to prove manually the resistance against a given number of compromised nodes. They have implemented their solutions on TelosB compatible mote to evaluate their performances. The main weakness of this scheme is the lack of a re-keying process. The pre-shared matrices will only help to create one pairwise key for every couple of nodes.

Comparing these protocols to determine which is the most suitable requires to possession of all different implementations on one platform. However, most of the implementations, if exist at all, are very platform-dependent, thus an implementation on one platform might not give reproducible results on another platform and might not even operate on incompatible platforms. For this reason, in this paper, we will not compare execution results between protocols. On the other hand, Table 1 summarizes a comparison between the related work schemes and our proposition. The comparison is based on some repudiated criteria in the WSNs area. As the table shows, Yu *et al.*'s scheme [23] is the closest to our schemes. The authors use TelosB motes to evaluate the energy consumption of the basic operations used in their schemes and they provide a large scale simulation for thousands of nodes based on these measurements.

In this paper, we did not compare our execution time results with other protocols from the state-of-the-art because, as stated in the related work section, implementations are dependent on

hardware and system. In addition, they can be optimized for certain platforms, which makes the comparison unfair using different platforms and cryptographic primitives. In our proposition, we take into account the multi-hop factor for node authentication and key establishment, which is often neglected in existing propositions. Any node in the network is able to be authenticated by sending a request in a multi-hop manner towards the base station, or send a key establishment request towards a multi-hop neighbour. Finally, our main difference with other works is that we formally prove the security of all our protocols using the automatic verification tool Scyther [5].

**Table 1.** Comparison of related work schemes.

| Proposed Scheme | Standard Algorithms | Cryptographic Technique | Simulation | Implementation | Verification |
|---|---|---|---|---|---|
| Perrig *et al.*, 2002 [19] | yes | symmetric | none | Smart dust [26] | manual |
| Chan *et al.*, 2005 [22] | not specified | symmetric | yes | none | none |
| Yu *et al.*, 2009 [23] | not specified | symmetric | none | TelosB | manual |
| Munivel *et al.*, 2010 [21] | yes | symmetric/asymmetric | none | none | none |
| Yeh *et al.*, 2011 [14] | yes | symmetric/asymmetric | none | none | none |
| Zhang *et al.*, 2012 [17] | not specified | symmetric | none | none | none |
| Al-mahmud *et al.*, 2012 [12] | yes | symmetric/asymmetric | none | none | none |
| Han *et al.*, 2012 [15] | not specified | symmetric/asymmetric | none | none | none |
| Manjusha *et al.*, 2013 [16] | yes | symmetric/asymmetric | MATLAB | none | none |
| Our schemes | yes | symmetric/asymmetric | none | TelosB | automatic |

## 3. Pre-deployed Keys and Cryptographic Notations

Before deployment, each node $N$ knows the public key $pk(S)$ of the sink $S$ and also its own pair of public and private keys, denoted $(pk(N), sk(N))$ respectively. Based on ECC (Elliptic Curve Cryptography), we have that $pk(N) = sk(N) \times G$, where $G$ is a public generator point of the elliptic curve. Using this material, each node $N$ can compute a shared key with the sink $S$ using a variation of the Diffie–Hellman key exchange without interaction between the nodes, denoted $K_{DH}(N, S)$. These computations can be done by the sink and by all nodes before deployment in order to preserve their energy.

- The sink knows its own secret key $sk(S)$ and the public key $pk(N)$ of a node $N$. The sink computes $K_{DH}(N, S) = sk(S) \times pk(N)$.
- Node $N$ multiplies its secret key $sk(N)$ by the public key of the sink $pk(S)$ to get $K_{DH}(N, S)$.

Both computations give the same shared key since:

$$K_{DH}(N, S) = sk(N) \times pk(S) = sk(N) \times (sk(S) \times G) = (sk(N) \times G) \times sk(S) = pk(N) \times sk(S)$$

In what follows, we use the following notations to describe exchanged messages in our protocols:

- $I$: a new node that initiates the protocol,
- $N$: a neighbour of node $I$,
- $R$: a node multi-hop away from $I$ called responder,
- $S$: the sink of the network (also called base station),

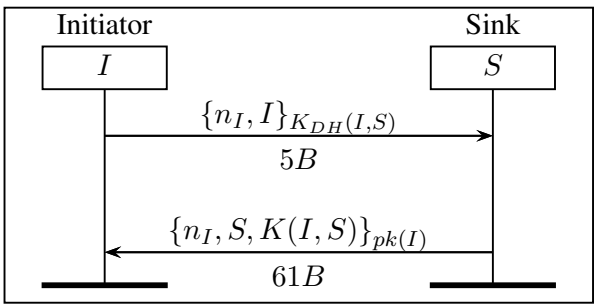
- $T$: a trusted node between the initiator node $I$ and the responder node $R$ ($T$ shares a session key with $I$ and $R$),
- $n_A$: a nonce generated by node $A$,
- $\{x\}_k$: the encryption of message $x$ with the symmetric or asymmetric key $k$,
- $pk(A)$: the public key of node $A$,
- $sk(A)$: the secret (private) key of node $A$,
- $K(I, S)$: the session key between $I$ and $S$,
- $K_{DH}(N, S)$: the shared symmetric key between $N$ and $S$ using the Diffie–Hellman key exchange without interaction described above.

In the next section, we recall two protocols that allow a node to join a multi-hop WSN in a secure way and establish secure links with the sink and a neighbour node. A new node is able to establish a secure link with the nearest node that is already part of the network. Then in the following sections, we propose six other protocols that allow any node to establish secure links with any another node in its multi-hop neighbourhood. Two protocols use the sink for key establishment and four protocols establish new keys without resort to the sink. In all figures that describe our protocol, we denote a direct communication by an arrow between two nodes, and denote a communication passing by several possible intermediate nodes by a dotted arrow. We also explicate the size of each exchanged message in Bytes.

## 4. Authenticated Multi-hop Join Protocols: $DJS$ and $IJS$

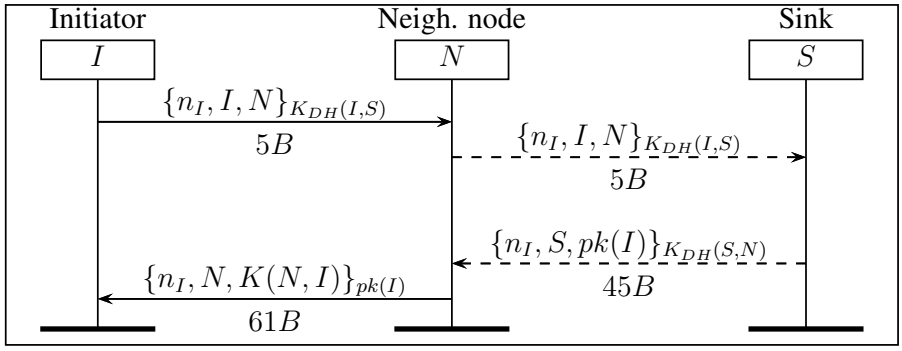
In this section, for the sake of completeness and to make the paper self-sufficient, we recall the authenticated multi-hop join protocols that our key establishment protocols are based upon. Protocol $DJS$ (Direct Join to the Sink), presented in Figure 1, was presented in [11]. It allows new nodes in range of the sink to join the network directly. A new node $I$ sends a direct request to $S$ in order to establish a session key with it. Node $I$ begins the join process by computing the symmetric key $K_{DH}(I, S)$ with the sink $S$. Then, node $I$ generates a nonce $n_I$ and adds its identity in order to form the request $\{n_I, \mathrm{I}\}$. The request is encrypted with $K_{DH}(I, S)$ and sent to $S$. Upon reception, in order to decrypt the request, $S$ computes $K_{DH}(I, S)$ using the public key of $I$. Then, $S$ verifies the identity of $I$ by checking if the identity of $I$ belongs to the list of authorized nodes, and generates a new session key $K(I, S)$. The join response contains $n_I$, the identity of $S$ and the new symmetric session key. The response is encrypted using $pk(I)$ and is sent to $I$. Only $I$ is able to decrypt the response with its secret key $sk(I)$. We note that $n_I$ helps $I$ to authenticate the response message of $S$.

**Figure 1.** $DJS$: node $I$ joins the network by communicating directly with the sink $S$.



The protocol $IJS$ (Indirect Join to the Sink), presented in Figure 2, was also presented in [11] and allows a new node to join the network through a neighbour node $N$. The new node $I$ sends an indirect request to $S$ in order to be authenticated and to establish a session key with $N$. Node $N$ forwards the request to $S$ through intermediate nodes. We note that the request and the response are just forwarded by these nodes without being modified. They are unable to decrypt either the request part or the response part of the message; only nodes $I$ and $S$ are able to decrypt messages encrypted with $K_{DH}(I, S)$, and only $N$ and $S$ are able to decrypt the messages encrypted with $K_{DH}(N, S)$.

**Figure 2.** $IJS$: $I$ joins the network through $N$. Intermediate nodes between $N$ and $S$ forward messages without any encryption or decryption.




Our aim is to establish a shared key between any two authenticated nodes $I$ and $R$ of the network (not necessary in range). We propose six different protocols, called $MKE_S$, $MKE_S - light$, $MKE_T - a$, $MKE_T - b$, $MKE_T - c$ and $MKE_T - d$.

## 5. Multi-hop Key Establishment Protocols Using the Sink

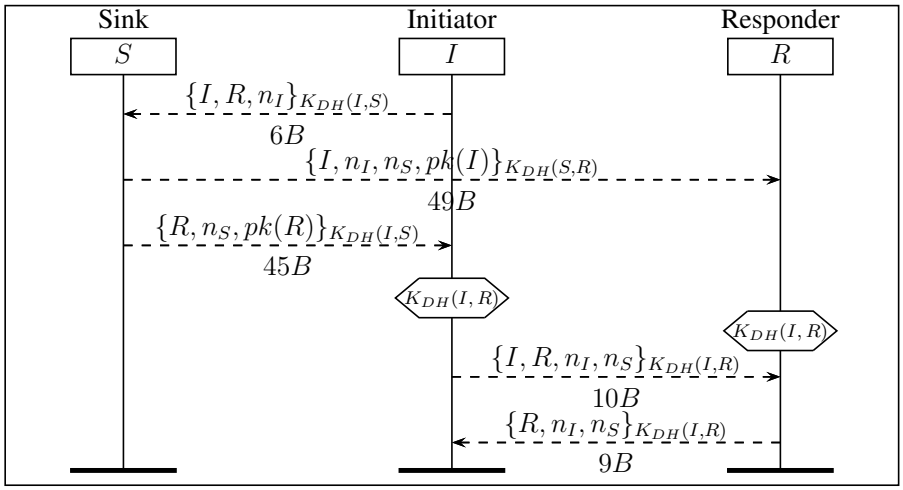
We start by explaining protocol $MKE_S$ and its optimization $MKE_S - light$. They both use the sink to establish a new key between two nodes. Protocol $MKE_S$, depicted in Figure 3, uses the shared symmetric keys $K_{DH}(I, S)$ and $K_{DH}(R, S)$ computed before the deployment between any node in the network and the sink $S$. These shared keys are used to communicate the public keys of $I$ and $R$. We consider that the sink knows all the public keys of all nodes and a node only knows its public key and the public of the sink. The initiator node $I$ builds a request containing the identity of node $R$ and a nonce $n_I$. This request is encrypted with $K_{DH}(I, S)$ and sent to $S$. The sink $S$ sends

- to $R$, the identity of $I$, a nonce $n_S$, the nonce $n_I$ received from $I$ and the public key of $I$ encrypted with the shared symmetric key $K_{DH}(S, R)$.
- to $I$, the identity of $R$, the same nonce $n_S$, the public key of $R$ encrypted with the shared symmetric key $K_{DH}(I, S)$,

Once these two messages are received by $I$ and $R$, the two nodes are able to compute $K_{DH}(I, R)$ as follows:

- Node $I$ computes $sk(I) \times pk(R) = sk(I) \times sk(R) \times G = K_{DH}(I, R)$.
- Node $R$ computes $sk(R) \times pk(I) = sk(R) \times sk(I) \times G = K_{DH}(I, R)$.

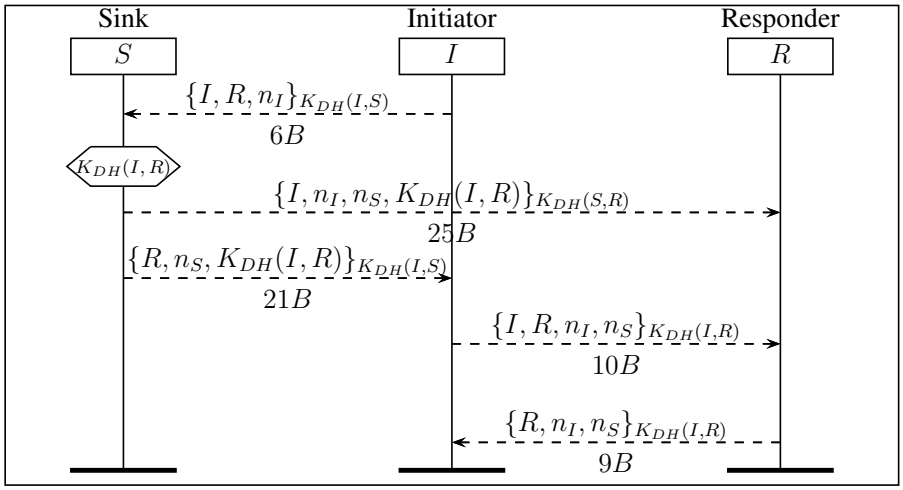
**Figure 3.** $MKE_S$: Multi-hop Key Establishment using the sink $S$ to deliver public keys. $K_{DH}(I,R)$ is computed by the initiator $I$ and the responder $R$.




The mutual authentication of $R$ and $I$ is ensured using the sink $S$ as the trusted third party. Indeed, only $S$ in this protocol is able to deliver public keys to nodes. The authentication between nodes is done by the verification of nonces. Node $R$ verifies that the received nonces $n_I$ and $n_S$ from $S$ are the same as the ones sent by $I$. Then, node $R$ confirms the establishment of $K_{DH}(I,R)$, by sending $n_I$ and $n_S$ to the initiator $I$ encrypted with $K_{DH}(I,R)$. Upon reception, node $I$ verifies first its own nonce and if $n_S$ received from $R$ is the same as the one sent by the sink.

Notice that the computation of the new key $K_{DH}(I,R)$ can be done by the sink in order to save some computations on nodes $R$ and $I$. Instead of sending the respective public keys, the sink generates the new shared key and sends it to the two nodes. Indeed, the sink is the trusted entity that generated and distributed the public/private keys to all nodes in the first place during the pre-deployment phase. Another possibility would be to generate symmetric keys at the sink using any standard random key generation function. In this version, we chose to use the $DH$ method at the sink. This version, called $MKE_S - light$, is depicted in Figure 4.

**Figure 4.** $MKE_S - light$: Multi-hop Key Establishment using the sink $S$ to compute and deliver $K_{DH}(I,R)$ to the initiator and the responder.



## 6. Multi-hop Key Establishment Protocols without the Sink

We propose four protocols that use a trusted intermediate node $T$ instead of the sink $S$ to establish a new key. The main idea is to allow other nodes to authenticate new key establishment. In this way, we avoid exhausting the sink and we preserve energy of its neighbours that are relaying requests all the time. Doing so also helps to avoid traffic congestion around the sink. When other nodes share this role, the traffic for key establishment will be spread throughout the network.

A node $T$ is considered as trusted by a node $I$ if nodes $I$ and $T$ have at least one secret key in common. This secret key is previously established either during the join process or using protocol $MKE_S$. Indeed, if node $I$ has a secret key with $T$, it means that $I$ has joined the network through $T$ (or vice versa) or has established a new key with node $T$ using $MKE_S$ (or $MKE_S - light$). Node $T$ might have the public key of both nodes $I$ and $R$, have only one of the two public keys, or have neither, according to the protocol that has served for the secret key establishment. In order to establish a secret key, an initiator node $I$ first tries to find a common trusted node $T$ with the responder $R$. This discovery mechanism is out of the scope of this paper, and can be done using one of the several existing secure routing protocols such as SDSR [27]. We describe the four possible situations according to the knowledge of node $T$:

- if $T$ has both $pk(I)$ and $pk(R)$, then $I$ follows protocol $MKE_T - a$,
- if $T$ has $pk(R)$ but not $pk(I)$, then $I$ follows protocol $MKE_T - b$,
- if $T$ has $pk(I)$ but not $pk(R)$, then $I$ follows protocol $MKE_T - c$,
- if $T$ has neither $pk(I)$ nor $pk(R)$, then $I$ follows protocol $MKE_T - d$.

Note that if $T$ does not exist, then $I$ follows $MKE_S$ or $MKE_S - light$. We note that these four protocols are very useful when nodes are located far from the sink. Indeed, it preserves the energy of intermediate nodes between $I$ and $S$ and those between $R$ and $S$. We explain the main difference between our four variations of the multi-hop key establishment:

- In protocol $MKE_T - a$ (Figure 5), the trusted node $T$ possesses $pk(I)$ and $pk(R)$. Therefore, protocols $MKE_T - a$ and $MKE_S$ are very similar. We note two differences: the sink $S$ of $MKE_S$ is replaced by the trusted node $T$, and the messages exchanged between both nodes and $S$ are encrypted using the session keys established after deployment instead of using shared keys computed before deployment. After receiving a request from the initiator $I$, the trusted node $T$ delivers $pk(R)$ to $I$ and $pk(I)$ to $R$.
- In protocol $MKE_T - b$, the trusted node $T$ possesses $pk(R)$ but not $pk(I)$. Protocol $MKE_T - b$ is the same as protocol $MKE_T - a$ except that the first message sent by $I$ is replaced by $\{R, n_I, pk(I)\}_{K(I,T)}$. Indeed, since $T$ does not possess the public key of node $I$, $I$ should send it to $T$ in order to send it to $R$.
- In protocol $MKE_T - c$ (Figure 6), the trusted node $T$ possesses $pk(I)$ but not $pk(R)$. After receiving $pk(I)$ from $T$, the responder $R$ sends its own public key to $T$ in order to send it to $I$.
- In protocol $MKE_T - d$, the trusted node $T$ possesses neither $pk(I)$ nor $pk(R)$. Protocol $MKE_T - d$ is the same as protocol $MKE_T - c$ except that the first message sent by $I$ is replaced by $\{R, n_I, pk(I)\}_{K(I,T)}$. Node $I$ sends $pk(I)$ to $T$ in its request in order to send it to $R$.

Note that the last two messages have the same role in all proposed protocols. These messages are used to verify nonces and confirm the establishment of the shared key $K_{DH}(I, R)$. In addition, in order for node $R$ to know if it needs to include its public key in the reply to $T$ or not, it can keep a record of how it obtained the shared key with $T$ and react accordingly. Alternatively, we can add in the header of the message sent from $T$ to $R$ the identifier of the protocol used for the key establishment. In both cases, the description of the protocols remains the same.

**Figure 5.** $MKE_T - a$: Multi-hop Key Establishment using a trusted node $T$ to deliver public keys. $T$ possesses $pk(I)$ and $pk(R)$. The key $K_{DH}(I, R)$ is computed by the initiator $I$ and the responder $R$.

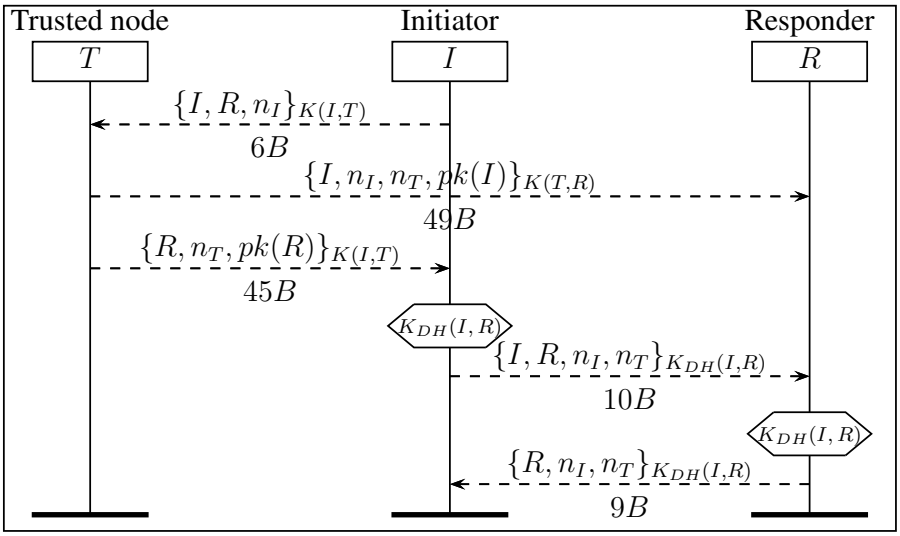
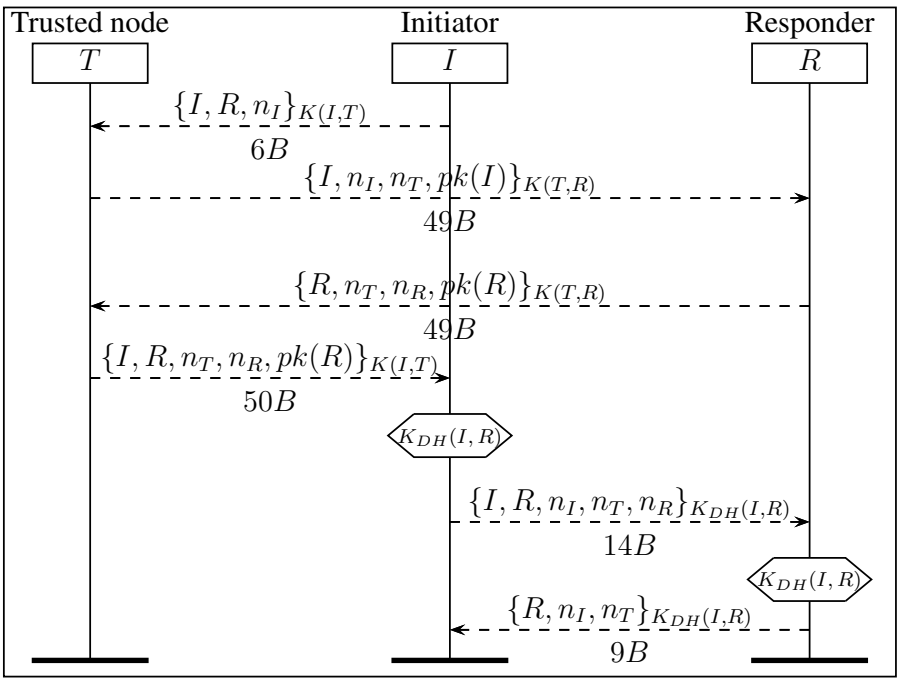



**Figure 6.** $MKE_T - c$: Multi-hop Key Establishment using a trusted node $T$ to deliver public keys. $T$ possesses $pk(I)$ but not $pk(R)$. $K_{DH}(I, R)$ is computed by the initiator $I$ and the responder $R$.



## 7. Key Dependency

It is important to note that cryptographic keys are interdependent. In order for a certain protocol to be executed, nodes should have the appropriate keys. In this section we recapitulate the dependency between the keys for each of our protocols.

Table 2 presents the dependency between keys for each of our protocols. The public/private keys of nodes are used to compute the shared symmetric keys using Diffie–Hellman without interaction described in Section 3. These shared keys are then used to deliver public keys in the case of protocols $IJS$ and $MKE_S$, or to deliver the pre-computed Diffie–Hellman key in the case of protocol $MKE_S - light$. In protocols $MKE_T - a$, $MKE_T - b$, $MKE_T - c$and $MKE_T - d$, the public keys of $I$ and $R$ are delivered using the session keys between these nodes and $T$. For example, in protocol $MKE_T - a$, the key $pk(R)$ is encrypted by $T$ using the key $K(I,T)$ in order to deliver it to $I$, which is denoted by $K(I,T) \xrightarrow{T:I} pk(R)$. Similarly, in this protocol $pk(I)$ is encrypted by $T$ using the key $K(R,T)$ in order to deliver it to $R$, which is denoted by $K(R,T) \xrightarrow{T:R} pk(I)$.

**Table 2.** Key dependency in our protocols. $K_1 \xrightarrow{A:B} K_2$ denotes that $K_2$ is delivered by $A$ to $B$ and encrypted with $K_1$.

| Protocol Name | Delivering |
|---|---|
| $DJS$ | $pk(I) \xrightarrow{S:I} K(I,S)$ |
| $IJS$ | $K_{DH}(N,S) \xrightarrow{S:N} pk(I) \xrightarrow{N:I} K(I,N)$ |
| $MKE_S$ | $K_{DH}(I,S) \xrightarrow{S:I} pk(R)$<br>$K_{DH}(R,S) \xrightarrow{S:R} pk(I)$ |
| $MKE_S - light$ | $K_{DH}(I,S) \xrightarrow{S:I} K_{DH}(I,R)$<br>$K_{DH}(R,S) \xrightarrow{S:R} K_{DH}(I,R)$ |
| $MKE_T - a$ | $K(I,T) \xrightarrow{T:I} pk(R)$<br>$K(R,T) \xrightarrow{T:R} pk(I)$ |
| $MKE_T - b$ | $K(I,T) \xrightarrow{I:T} pk(I)$<br>$K(I,T) \xrightarrow{T:I} pk(R)$<br>$K(R,T) \xrightarrow{T:R} pk(I)$ |
| $MKE_T - c$ | $K(R,T) \xrightarrow{T:R} pk(I)$<br>$K(R,T) \xrightarrow{R:T} pk(R)$<br>$K(I,T) \xrightarrow{T:I} pk(R)$ |
| $MKE_T - d$ | $K(I,T) \xrightarrow{I:T} pk(I)$<br>$K(R,T) \xrightarrow{T:R} pk(I)$<br>$K(R,T) \xrightarrow{R:T} pk(R)$<br>$K(I,T) \xrightarrow{T:I} pk(R)$ |

## 8. Formal Security Evaluation

Evaluating the security of cryptographic protocols is not an easy task and flaws can readily occur in protocols. During the past decades, several tools [3,5,28] have been developed to automatically verify cryptographic protocols. We use Scyther [5] because it is one of the fastest as has been shown in [29] and also one of the most user friendly.

Cas Cremers has developed an automatic tool called Scyther [5]. It is a free tool available on all operating systems (Linux, Mac and Windows). This tool can automatically find attacks on cryptographic protocols and prove their security for bounded and unbounded numbers of sessions. One main advantage of Scyther is that it provides an easy way to model security properties like secrecy and authentication.

Scyther uses the Dolev–Yao intruder model [6]. In this model, the intruder controls the network and all communications pass through it, which means that all packets can be captured by the intruder. This is why we do not consider multiple forwarder nodes in our analysis, since it does not provide more security. Moreover, the intruder has its own public and secret pairs of keys, enabling it to play the role of any participant in the protocol. It can also encrypt messages with all public or symmetric keys that it knows and it decrypts cipher texts only if it knows the decryption key. We verified all our protocols using Scyther without any intermediate nodes. When Scyther proves a security property, the tool only outputs "OK", meaning that the property is satisfied. However, if there is a flaw, then a graphical description of the discovered attack is given and the attack is automatically generated. More precisely, we proved the secrecy of all sensitive data exchanged (keys and nonces) and also the authenticity of the communication. Our Scyther codes are available at [9]. The secrecy of the keys and the nonces means that an intruder cannot learn these data. The authentication property of the communication means that each node does indeed communicate with the expected node. This property protects our protocols from replay or man-in-the-middle attacks. Our protocols might appear simple, but they are the result of an optimization process using Scyther. It leads us to minimal protocol in the sense that removing any piece of information in any message will enable an attack.

## 9. Evaluations

In order to compare the execution time of our different protocols, we implemented each protocol on TelosB motes. In what follows, we describe the settings of our testbed. We also provide and discuss the results of the overhead of each protocol. We did not compare our protocols with other existing protocols because implementations are very heterogeneous. Implementation on sensor nodes is very hardware-dependent; optimization can be done for certain functions depending on the capacities and the design of the hardware. Thus comparing different implementations on different motes is not really conclusive, as is shown in [30–33]. The same applies for comparison by simulation, which is even less conclusive because simulation does not give an idea about how complex the operations are.

TelosB motes have an 8 MHz microcontroller with 10 KB of RAM, 48 KB of ROM and a CC2420 radio using the IEEE 802.15.4 standard. In our evaluation, we use public key Elliptic Curve Cryptography (ECC), with parameters secp160r1 given by the Standards for Efficient Cryptography Group. Our implementation of ECC on TelosB is based on the TinyECC library [34]. More precisely, we use Elliptic Curve Integrated Encryption Scheme (ECIES), the public key encryption system proposed

by Victor Shoup in 2001, and the Elliptic Curve Diffie–Hellman (ECDH) key agreement scheme [35]. For all symmetric encryptions, we use an optimized implementation of AES [36] with a 128-bit key proposed by [37] in CTR mode. In order to explain why we choose AES with CTR mode and the 128-bit key $k$, we recall the mechanism of this scheme: let us consider a message $m$ composed of $p$ blocks $m_1||m_2||\ldots||m_p$, and an initial counter value $IV$ randomly chosen. The cipher of the block $m_i$ is $c_i = \{IV + i\}_k \oplus m_i$, where $\oplus$ denotes the bitwise XOR operator. If one knows $IV$ and the key $k$, then one can easily recover from $c_i$ the message $m_i = \{IV+i\}_k \oplus c_i$. When the size of the last block $m_p$ is smaller than 128 bits, it is usual to pad it with 0s to reach 128 bits in order to let both operands have the same length to perform the bitwise XOR. In this case, the size of the transmitted encrypted packets is always a multiple of 128 bits. However, in the CTR mode, we can just cut the $\{IV + p\}_k$ message to the size of $m_p$. Hence we can transmit encrypted messages with exactly the same size as the original message, and therefore save some execution time.

During the experiments, we considered topologies without intermediate nodes, since these nodes would only forward the packets without doing any modification on the packets. The cost of these communications is therefore dependent on the network density, topology and traffic load. In this paper, we only consider a minimal topology containing only the nodes involved in the cryptographic operations. This means that for all protocols we tested them with a topology of 3 nodes, except for $DJS$ where only 2 nodes were needed for the execution. This does not affect the feasibility of our protocols for more hops, because adding more intermediate nodes does not add more cryptographic operations—intermediate nodes will only forward the message to the next hop according the routing protocol. In addition, nodes do not have other traffic to exchange during the execution of the protocols. Of course, additional overhead should be considered because of the routing process and exchanged messages, but this is very dependent on the routing and MAC layers used during the evaluation. For this reason, we preferred to include only overhead induced by the cryptographic operations and not the overhead induced by the network protocols.

In the Table 3, we provide the real execution time for all our protocols. For the sake of completeness and to make the paper self-sufficient, we also included the results of $DJS$ and $IJS$. These results are the averages of 100 experiments. We added in the table the standard deviation of these averages.

**Table 3.** Total execution time and total size of exchanged messages of our protocols using a 160-bit ECIES key and a 128-bit AES-CTR key.

| Protocol Name | Execution Time on the Testbed (ms) | Standard Deviation (ms) | Message Size ($Bytes$) |
|---|---|---|---|
| $DJS$ | 10,112.62 | 78.09 | 66 |
| $IJS$ | 10,180.81 | 111.94 | 116 |
| $MKE_S$ | 6828.48 | 5.96 | 119 |
| $MKE_S - light$ | 3649.74 | 6.10 | 71 |
| $MKE_T - a$ | 6831.18 | 6.76 | 119 |
| $MKE_T - b$ | 6950.10 | 5.77 | 159 |
| $MKE_T - c$ | 7324.88 | 7.77 | 177 |
| $MKE_T - d$ | 7447.32 | 8.61 | 217 |

Moreover, the differences in our protocols come from the usage of cryptographic primitives. All protocols using asymmetric encryption ($DJS$ and $IJS$) require more execution time than protocols using only symmetric encryption ($MKE_S$, $MKE_S - light$, $MKE_T - a$, $MKE_T - b$, $MKE_T - c$ and $MKE_T - d$). In what follows, we analyze the results:

- Note that the two slowest protocols are the join protocols proposed in [11]. Hence, it is more efficient to establish keys after the deployment than to rejoin the network.
- Protocols $MKE_S$ and $MKE_T - a$ have practically the same execution time. The small difference comes from the use of symmetric keys established after the deployment instead of symmetric shared keys computed before deployment.
- Protocol $MKE_S - light$ is the fastest protocol because the sink computes $K_{DH}(I, R)$ and delivers it to $I$ and $R$ in order to save their energy.
- Protocols $MKE_T - c$ and $MKE_T - d$ are slower than $MKE_S$, $MKE_T - a$ and $MKE_T - b$ due to the additional messages sent from $R$ to $S$ containing the public key of $R$.
- The difference of execution time between $MKE_T - a$ and $MKE_T - b$, on one side, and $MKE_T - c$ and $MKE_T - d$, on the other side, is almost equal to 130 ms. This is due to the encryption, decryption and transmission of $pk(I)$ added to the request message sent by node $I$.
- In Table 3, we also provide the total size in $Bytes$ of all exchanged messages for each protocol. In the figures of protocols presented in Sections 4 and 6, the size in $Bytes$ is also tagged at the bottom of each message. We have fixed the size of the identity of a node to 1 $Byte$ and the size of a nonce to 4 $Bytes$. It should be noted that the size of $K_{DH}(I, R)$ is fixed to 16 $Bytes$. Indeed, the result of computation of $pk(I) \times sk(R)$ gives a key length of 20 $Bytes$. We use the first 16 $Bytes$ of the computation result for $K_{DH}(I, R)$.
- Also note that we use symmetric encryption in all protocols for key establishment of Section 6. Therefore, the size of the input message in $Bytes$ is the same as the size of the output message. In contrast, we use both asymmetric encryption and symmetric encryption in the two join protocols of Section 4. Therefore, in order to obtain the tagged value of message size, we must add to the output encrypted message the size of the MAC, which is equal to 20 $Bytes$. For example, the total size of last message in Figure 1 sent to $I$ is 66 $Bytes$, containing $n_I$(4 $Bytes$), $S$ (1 $Byte$), $K(I, S)$ (16 $Bytes$) and a MAC (20 $Bytes$).

Notice that the standard variation values are small compared with the execution time of the protocol. These variations are essentially caused by the various interruptions of the micro-controller. Nevertheless, we can see that the highest standard deviation values are obtained for protocols using public key encryption ($DJS$ and $IJS$). These complex computations take more time and for each experiment the execution time can vary a little bit more. Indeed, the public encryption starts with a random number generation and then multiplies it with the generator point of the curve. The multiplication operation takes undetermined time, depending on the random number that was chosen. This explains the larger variation in the results for protocols $DJS$ and $IJS$.

In order to verify the cause of this variation, we made 100 iterations for protocols $DJS$ and $IJS$ using the same random number. Table 4 shows that when we fix the random number, the obtained standard variation is considerably reduced and becomes equivalent to that of the other protocols. This shows

that the value of the random number affects the overall execution time of $DJS$ and $IJS$. Note that the averages in Table 4 are not the same as the averages in Table 3, which is normal because the averages in Table 3 are over 100 iterations with 100 different random numbers. The size of the fixed random number used in the experiments is 20 $bytes$. The hexadecimal values of each byte starting with most significant byte are: $da$ 52 $cd$ $de$ 28 $d5$ $c5$ 2$a$ 8$b$ 5$c$ 18 $b9$ 28 3$f$ 1$a$ 68 $b7$ 2$d$ 01 7$b$.

**Table 4.** Total execution time for protocols $DJS$ and $IJS$ using the same random number in ECIES encryption.

| Protocol Name | Execution Time on the Testbed (ms) | Standard Deviation (ms) |
|---|---|---|
| $DJS$ | 10,186.49 | 5.94 |
| $IJS$ | 10,291.24 | 5.71 |

## 10. Attack Detection Perspective

Based on the overhead estimation of our protocol, we are able to allow nodes to detect certain types of attack attempts. Indeed, when a node is attempting an attack, it will induce additional overhead due to the execution time of cryptographic operations that the attacker needs. Moreover, even in the man-in-the-middle attack, the wormhole attack or the replay attack, the attacker needs to listen, forward or generate some encrypted messages. All these operations will cause an additional delay to the estimated overall request/reply procedure of the authenticated join protocol or key establishment protocol. We assume that the attacker has the same processing capacities of the nodes; in other words, we assume that an intruder was able to control one or more nodes of the network and is trying to launch the attack using the compromised nodes.

The attacks that this technique is able to detect are the ones that will introduce additional delay to the process. For example, the node that is requesting to join the network will be waiting for an authenticated response. This response should be received before a certain timeout expires. This timeout can be estimated according to the number of hops that separate the new node from the sink and the type of protocol that is used to join the network.

For example, if the new node is 5 hops away from the sink, with a 160-bit key, it will take 6834.20 plus the end-to-end communication delay for a node to establish a key with another node in the network by passing through the sink ($MKE_S$). Indeed, the results shown in Table 3 represent the execution time without taking into consideration the end-to-end communication delay that is induced by the intermediate nodes. This delay can be estimated by the routing protocol and taken into account when calculating the estimated overall time of the process.

In order for the attack detection to be efficient, the end-to-end communication delay should remain considerably lower than the delay induced by additional cryptographic operations. Hence, according to the protocol that is being used, the new node is able to estimate when it is expected to receive its response. Therefore, if a node does not receive an answer on time, it can deduce that this delay might have been caused by an intruder and then decide to abort the current join process or key establishment process and try to restart it. This means that a reply will be discarded if received later than expected. This

attack detection will help prevent a node from accepting late replies, and restart the process to ensure that the response is received within a certain timeout interval. Delay estimations should be done for each network configuration and deployment because they are affected by the traffic state and the congestion that might be present in the network. In this paper these estimations are out of scope and will be the subject of future work.

Also, in order for the detection to work, the new node should know how many hops it is away from the sink. This information can be obtained from the routing protocol. One way to obtain this information could be the following. When the new node decides to join the network, it will listen for signalling messages from neighbouring nodes. These signalling messages should contain how many hops the signalling node is away from the sink. This way, the new node is able to estimate the delay in receiving the response. In this simple approach, we do not consider the case where certain nodes will deliberately announce that they are farther away from the sink in order to delay the process or try to launch attacks. This kind of malicious behaviour is studied in [38], where the authors propose a solution based on the reputation of nodes. This kind of solutions could be applied to our protocols and are part of our future works.

## 11. Conclusions and Future Work

We proposed several multi-hop node authentication and key establishment protocols for WSNs. Our key establishment solutions can be split into two categories. First, we propose two solutions that use the sink as a trusted third party to establish a new key between two nodes of the network. Then, we proposed four protocols that use other nodes in the network as trusted third parties in order to establish a new key. We proved the security of all of our solutions using the automatic tool Scyther. Our protocols have very few cryptographic operations but are free of flaws as proven by Scyther. Moreover, we implemented and tested all our protocols on TelosB nodes in order to evaluate their execution time. The results show that according to the load of the network and according to the topology, one category might be more efficient than the others.

Based on our measurements, we were able to allow nodes to detect attack attempts from other nodes in the network by simply measuring the delay of response reception. Each node can expect how much time it should take to receive a response according to the protocol being used and the hops that separate it from the sink. In the case of excessive delay, the node presumes that an attack is taking place and terminates the protocol. This can be considered as a simple intrusion detection system.

In our future works, we plan on testing our protocols in more realistic platforms such as the IoT-LAB [39] platform. It would then elucidate how our different protocols would perform in a large scale network and how long it would take to establish the required secured links.

The decision to choose one or the other of the protocols could be dependent on the topology information that each node has. For example, if the sink has constrained resource, it would be preferable for nodes to choose another trusted third party node to establish new keys. The choice would be based on the number of hops that might separate nodes that want to establish a secure link between each other and the trusted third party. This kind of information could be built using a neighbour discovery protocol that allows nodes to share neighbourhood discovery in order to establish $n$-hop neighbourhood information

(where $n$ is the number of hops that nodes will be able to discover). A trade-off should be made between the cost of neighbourhood discovery and the gain of balancing key establishment overhead on a bigger number of nodes. Of course, more extensive neighbourhood information enables higher possibility to find trusted third parties. This study will be part of our future works.

## Acknowledgments

This research was conducted with the support of the "Digital Trust" Chair from the University of Auvergne Foundation.

## Author Contributions

In this paper, Ismail Mansour, Gerard Chalhoub and Pascal Lafourcade have worked together on the design and verification of the protocols, while the implementation and evaluation work has been essentially done by Ismail Mansour.

## Conflicts of Interest

The authors declare no conflicts of interest.

## References

1. Kavitha, T.; Sridharan, D. Security Vulnerabilities in Wireless Sensor Networks: A Survey. *J. Inf. Assur. Secur.* **2010**, *5*, 31–34.
2. Lowe, G. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *Soft. Concepts Tools* **1996**, *17*, 93–102.
3. Blanchet, B. Automatic Proof of Strong Secrecy for Security Protocols. In Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 9–12 May 2004; pp. 86–100.
4. Pérez, V.B.; González, P.; Cabaleiro, J.C.; Heras, D.B.; Pena, T.F.; Pombo, J.J.; Rivera, F.F. AVISPA: Visualizing the performance prediction of parallel iterative solvers. *Future Gener. Comput. Syst.* **2003**, *19*, 721–733.
5. Cremers, C. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In Proceedings of the 2008 20th International Conference Computer Aided Verification (CAV 2008), Princeton, NJ, USA, 7–14 July 2008; pp. 414–418.
6. Dolev, D.; Yao, A.C. On the Security of Public Key Protocols. *IEEE Trans. Inf. Theory* **1983**, *29*, 198–208.
7. Basin, D.; Cremers, C.; Meadows, C. Model Checking Security Protocols. In *Handbook of Model Checking*; Springer: Berlin/Heidelberg, Germany, 2014; Chapter 24.
8. Ksiezopolski, B.; Kotulski, Z.; Szalachowski, P. Adaptive Approach to Network Security. In *Computer Networks*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 39, pp. 233–241.
9. Mansour, I.; Lafourcade, P.; Chalhoub, G. Scyther code of our authentication protocols, 2014. Available online: http://sancy.univ-bpclermont.fr/~lafourcade/jsan-scyther-code.tar (accessed on 11 June 2014).

10. Mansour, I.; Rusinek, D.; Chalhoub, G.; Lafourcade, P.; Ksiezopolski, B. Multihop Node Authentication Mechanisms for Wireless Sensor Networks. In Proceedings of the 13th International Conference (ADHOC-NOW 2014), Benidorm, Spain, 22–27June 2014.
11. Mansour, I.; Chalhoub, G.; Misson, M. Security architecture for multi-hop wireless sensor networks. In *Security for Multihop Wireless Networks*; CRC Press Book: Boca Raton, FL, USA, 2014; pp. 157–178.
12. Al-mahmud, A.; Akhtar, R. Secure Sensor Node Authentication in Wireless Sensor Networks. *Int. J. Comput. Appl.* **2012**, *46*, 10–17.
13. Das, M.L. Two-factor user authentication in wireless sensor networks. *IEEE Trans. Wirel. Commun.* **2009**, *8*, 1086–1090.
14. Yeh, H.L.; Chen, T.H.; Liu, P.C.; Kim, T.H.; Wei, H.W. A Secured Authentication Protocol for Wireless Sensor Networks Using Elliptic Curves Cryptography. *Sensors* **2011**, *11*, 4767–4779.
15. Han, K.; Shon, T. Sensor Authentication in Dynamic Wireless Sensor Network Environments. *Int. J. RFID Secur. Cryptogr.* **2012**, *1*, 36–44.
16. Manjusha; Rananavare, L.B. A Robust Message Authentication Scheme in Multihop WSN Using Elliptical Curve Cryptography and Elgamal Signature. *Int. J. Eng. Res. Technol. (IJERT)*, **2013**, *2*.
17. Zhang, J.; Shankaran, R.; Orgun, M.A.; Sattar, A.; Varadharajan, V. A Dynamic Authentication Scheme for Hierarchical Wireless Sensor Networks. In Proceedings of the 7th International ICST Conference, MobiQuitous 2010, Sydeny, Australia, 6–9 December 2010; Volume 73, pp. 186–197.
18. Bala, S.; Sharma, G.; Verma, A. Classification of Symmetric Key Management Schemes for Wireless Sensor Networks. *Int. J. Secur. Its Appl.* **2013**.
19. Perrig, A.; Szewczyk, R.; Tygar, J.; Wen, V.; Culler, D. SPINS: Security Protocols for Sensor Networks. *Wirel. Netw. J. (WINE)* **2002**, *8*, 521–534.
20. Perrig, A.; Canetti, R.; Tygar, J.; Song, D. Efficient authentication and signing of multicast streams over lossy channels. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 14–17 May 2000; pp. 56–73.
21. Munivel, E.; Ajit, G. Efficient Public Key Infrastructure Implementation in Wireless Sensor Networks. In Proceedings of the International Conference on Wireless Communication and Sensor Computing, Chennai, India, 2–4 January 2010; pp. 1–6.
22. Chan, H.; Perrig, A. PIKE: Peer Intermediaries for Key Establishment in Sensor Networks. In Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Miami, FL, USA, 13–17 March 2005; pp. 524–535.
23. Yu, C.; Lu, C.; Kuo, S. A Simple Non-Interactive Pairwise Key Establishment Scheme in Sensor Networks. In Proceedings of the IEEE International Conference on Sensing, Communication, and Networking (SECON 2009), Rome, Italy, 22–26 June 2009; pp. 1–9.
24. Blom, R. An optimal class of symmetric key generation systems. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, 9–11 April 1984; pp. 335–338.

25. Zhang, W.; Tran, M.; Zhu, S.; Cao, G. A Random Perturbation-Based Scheme for Pairwise Key Establishment in Sensor Networks. In Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2007), Montréal, QC, Canada, 9–14 September 2007; pp. 90–99.
26. Pister, K.S.; Kahn, J.M.; Boser, B.E. Smart dust: Wireless networks of millimeter-scale sensor nodes. Available online: http://robotics.eecs.berkeley.edu/ pister/SmartDust/ (accessed on 1 September 2014).
27. Jiang, T.; Li, Q.; Ruan, Y. Secure dynamic source routing protocol. In Proceedings of the 2004 Fourth International Conference on Computer and Information Technology (CIT '04), Wuhan, China, 14–16 September 2004; pp. 528–533.
28. Armando, A.; Basin, D.; Boichut, Y.; Chevalier, Y.; Compagna, L.; Cuellar, J.; Drielsma, P.H.; Heám, P.C.; Kouchnarenko, O.; Mantovani, J.; *et al*. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In Proceedings of the 17th International Conference (CAV'2005), Edinburgh, Scotland, UK, 6–10 July 2005; pp. 281–285.
29. Cremers, C.J.; Lafourcade, P.; Nadeau, P. Comparing State Spaces in Automatic Protocol Analysis. In *Formal to Practical Security*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5458, pp. 70–94.
30. Kerckhof, S.; Durvaux, F.; Hocquet, C.; Bol, D.; Standaert, F. Towards Green Cryptography: A Comparison of Lightweight Ciphers from the Energy Viewpoint. In Proceedings of the 14th International Workshop, Leuven, Belgium, 9–12 September 2012; pp. 390–407.
31. Eisenbarth, T.; Gong, Z.; Guneysu, T.; Heyse, S.; Indesteege, S.; Kerckhof, S.; Koeune, F.; Nad, T.; Plos, T.; Regazzoni, F.; *et al*. Compact Implementation and Performance Evaluation of Block Ciphers in ATtiny Devices. In Proceedings of the 5th International Conference on Cryptology in Africa, Ifrance, Morocco, 10–12 July 2012; pp.172–187.
32. Balasch, J.; Ege, B.; Eisenbarth, T.; Gerard, B.; Gong, Z.; Guneysu, T.; Heyse, S.; Kerckhof, S.; Koeune, F.; Plos, T.; *et al*. Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices. In Proceedings of the 11th International Conference (CARDIS 2012), Graz, Austria, 28–30 November 2012; pp. 158–172.
33. Cazorla, M.; Marquet, K.; Minier, M. Survey and Benchmark of Lightweight Block Ciphers for Wireless Sensor Networks. In Proceedings of the 10th International Conference on Security and Cryptography (SECRYPT 2013), Reykjavik, Iceland, 29–31 July 2013; pp. 543–548.
34. Liu, A.; Ning, N. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In Proceedings of the 7th International Conference on Information Processing in Sensor Networks, St. Louis, MO, USA, 22–24 April 2008; pp. 245–256.
35. Diffie, W.; Hellman, M. New directions in cryptography. *IEEE Trans. Inf. Theory* **1976**, *22*, C644–C654.
36. Daemen, J.; Rijmen, V. *The Design of Rijndael: AES—The Advanced Encryption Standard*; Springer-Verlag: Berlin/Heidelberg, Germany, 2002.
37. Toldo, P; Saloni, M.; Manica, N. AES implementation in TinyOS, June 2008.

38. Moati, N.; Otrok, H.; Mourad, A.; Robert, J.M. Reputation-Based Cooperative Detection Model of Selfish Nodes in Cluster-Based QoS-OLSR Protocol. *Wirel. Personal Commun.* **2014**, *75*, 1747–1768.
39. Internet of Things Lab. Available online: https://www.iot-lab.info/ (accessed on 11 June 2014).