Key Management inWireless Sensor Networks

Ismail Mansour, Gérard Chalhoub, Pascal Lafourcade

J. Sens. Actuator Netw. **2015**, *4*, 251-273; doi:10.3390/jsan4030251

OPEN ACCESS

Journal of Sensor and Actuator Networks

ISSN 2224-2708

www.mdpi.com/journal/jsan

Article

Key Management in Wireless Sensor Networks

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

[1] University Clermont Auvergne, 49 Boulevard François Mitterrand, BP 10448, F-63000 Clermont-Ferrand, France;
E-Mails: ismail.mansour.mail@gmail.com (I.M.); pascal.lafourcade@udamail.fr (P.L.)

[2] CNRS, UMR 6158, Campus des Cézeaux, 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.

Academic Editor: Dharma P. Agrawal

Received: 17 April 2015 / Accepted: 14 August 2015 / Published: 7 September 2015

Abstract: Wireless sensor networks are a challenging field of research when it comes to security issues. Using low cost sensor nodes with limited resources makes it difficult for cryptographic algorithms to function without impacting energy consumption and latency. In this paper, we focus on key management issues in multi-hop wireless sensor networks. These networks are easy to attack due to the open nature of the wireless medium. Intruders could try to penetrate the network, capture nodes or take control over particular nodes. In this context, it is important to revoke and renew keys that might be learned by malicious nodes. We propose several secure protocols for key revocation and key renewal based on symmetric encryption and elliptic curve cryptography. All protocols are secure, but have different security levels. Each proposed protocol is formally proven and analyzed using Scyther, an automatic verification tool for cryptographic protocols. For efficiency comparison sake, we implemented all protocols on real testbeds using TelosB motes and discussed their performances.

Keywords: renewing; revocation; authentication; wireless sensor network; security; multihop; verification; formal proof

---

1. Introduction

Nowadays, the Internet of Things (IoT) is a reality: more and more devices are used to monitor our environment and to interconnect such embedded objects. IoT relies on wireless sensor networks (WSNs) for ensuring connectivity between nodes on the lower level of the network architecture. In such a context, some sensitive applications often require cryptographic mechanisms in order to achieve security. For instance, most of the military applications of WSNs require a high level of security [1]. Thus, it is important to design secure communication mechanisms between nodes of the network. These mechanisms can be achieved thanks to modern cryptographic primitives. Once we have established a secure communication channel in the network, several situations might occur; a node can run out of battery, even get destroyed, or just leave the network, or a new node can join the network. In addition, an intruder could capture a node and get all of its secret data (including secret cryptographic keys). An attacker could also try to join the network and be part of the authenticated nodes of the network. There exist several intrusion detection systems (IDS) in the literature [2,3] in order to detect such malicious behavior. In general, an IDS either searches for signs of malicious activity in the network, or monitors the internal behavior of one node. Several works use signature-based or anomaly-based detection techniques. Using the results of IDS, the next step is to revoke identified malicious nodes and to renew the cryptographic keys used by the nodes of the network.

Moreover, in WSNs, it is widely admitted that asymmetric encryption primitives based on exponentiation, like, for instance, RSA [4] or Elgamal [5], cannot be used since sensor nodes have limited resources (battery and computation power). However, there exist several lightweight cryptographic primitives that are adapted for WSN, e.g., [6,7]. Such lightweight primitives guarantee a low level of security, which remains a real obstacle for their deployment. For instance, using an improved differential fault analysis, the authors in [8] claim that they can break a lightweight block cipher for WSNs called LBlock using a personal computer within one hour. It is clearly not surprising that lightweight encryption can be attacked in a few hours with a computer that has more computation power than a node. Recently, a promising work [9] provided a solution to avoid such an attack by using code polymorphism. This method improves security at several levels in electronic devices.

In this context, it is therefore crucial to have efficient key revocation and renewal mechanisms in WSNs. The goal of this work is to propose secure and efficient protocols for key revocation and renewal in WSNs. In the literature, many key revocation protocols have been proposed for WSNs. In [10], the authors made a survey and a taxonomy of the most relevant key management protocols proposed for WSNs. Key revocation is closely related to key distribution. Key revocation protocols can be classified into centralized and distributed protocols. In centralized mode, a central entity decides to revoke certain keys or nodes in the network. In distributed mode, a local voting procedure takes place to revoke keys or nodes. The latter might be faster and requires less messages to send, but it is generally more complex to implement. We propose several centralized secure key revocation and renewal protocols. Note that we cover the revocation/renewal of both symmetric and asymmetric keys. Indeed, according to the security level requirements, one or both cryptographic methods should be used. Therefore, it is crucial to be able to renew both types of keys, asymmetric keys and symmetric keys.

1.1. Contributions

In this paper, we propose key renewal and key revocation protocols. Based on the cryptographic mechanisms used in [11], we propose centralized protocols for revoking and renewing keys when an IDS has detected an abnormal behavior. These mechanisms can also be used for periodical key renewal. More precisely, we provide a key revocation protocol (KR), a protocol for renewing symmetric keys between any node in the network and the sink (RSK), four protocols to renew asymmetric keys of nodes (RAKnk$_a$, RAKnk$_b$, RAKdh$_a$ and RAKdh$_b$), a protocol to renew the network key (RNK) and two multihop shared key establishment protocols (MSK$_a$ and MSK$_b$). When several protocols with the same goal are given, they use different cryptographic primitives. All of our protocols are proven secure using Scyther [12], an automatic tool for verifying cryptographic protocols. In order to compare the speed of each protocol, we implemented them on TelosB motes. These motes have limited resources: an 8-MHz microcontroller with 10 Kb of RAM and 48 Kb of ROM. Thereafter, we compare the protocols depending on the speed and their security level according to the cryptographic primitives used. This paper does not aim at showing that the proposed protocols are faster than existing protocols. Instead, the main objective is to present secure solutions for key revocation and key renewal 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 presented in the paper 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, which are known for their low capacities in computation speed. Indeed, for cryptographic primitives, an implementation on recent and more powerful motes should give better results. Once again in this work, our aim is not to obtain the best execution time, but to compare the performance of our different solutions.

In this paper, we not only propose original protocols that solve the secure join/revocation/renewal challenge, but also prove them to be secure, and we perform an experimental evaluation on real testbeds. This paper is an extension of the short paper presented in [13]. It contains additional optimized protocols, and we present additional in-depth details. Adding new nodes to the network was studied and evaluated in [11,14], and the scalability evaluation of these protocols was studied in [15].

1.2. Related Work

The main issue for secure communications in WSNs is how to set up secret keys between nodes, which is known as the key agreement. This task is a challenge due to resource constraints and the size of networks.

Many proposed methods are based on random key pre-distribution [16–18]. The main idea is that any two sensor nodes have a probability for choosing the same key from the pre-distributed key pool. The weakness of these schemes is when sensor nodes are captured by an attacker. All encryption keys in the captured nodes will be revealed. In [19], the authors proposed an approach to resolve this issue. The proposed approach is to hash the keys in the key pool with a one-way hash function. Once a key is chosen by a sensor node, it will be hashed. If another sensor node chooses the same key afterwards, instead of using the same key as the previous sensor node, it will derive a new different key by hashing

the key value. This technique can reduce the amount of information revealed when a sensor node is captured for about 30% to 50% according to the authors of [19].

Another way to reduce the impact of compromised nodes in WSNs is to preserve the network connectivity. The authors in [20] propose an improved key distribution mechanism for large-scale networks that preserves the network connectivity when many sensors are compromised with a sufficient security level. Based on a three-tier hierarchical WSNs model, the authors propose three phases: the key predistribution phase, the inter-cluster pairwise establishment phase and the inter-cluster pairwise key establishment phase. Different secret information (polynomial shares) is pre-loaded in the first phase. Two bivariate symmetric polynomials are used to establish pairwise keys between cluster heads and their sensors in the second phase. In the last phase, each cluster head establishes a pairwise key with other cluster heads. Compared to the existing key pre-distribution schemes, this scheme can achieve better network resiliency against node capture attacks.

However, the compromised nodes in these schemes can continue to use the shared secrets with neighbors without any revocation process. The neighbors of the compromised nodes should be alerted in order to ignore any further communications with the compromised nodes.

Not all key distribution protocols for WSNs include a key renewal or revocation process. Most of those that do include one suppose that an IDS is implemented in each node and enables it to signal the presence of an intruder. In what follows, we summarize some of the key revocation and key renewal protocols for WSNs.

One of the first key revocation protocols was proposed in [21,22]. This proposition has been enhanced by the authors of [23], where they proposed a distributed collaborative key revocation mechanism that divides the network into regions. In each region, nodes collaborate to identify a malicious node. Once a malicious node is identified, the base station is informed, and it sends a broadcast message containing a list of keys to revoke. Unlike [21,22], their contribution is able to revoke all of the keys with which the revoked node is involved and does not need to know the network topology before the deployment. The key revocation message sent by the base station is based on trivariate polynomial authentication and verified by each node according to the region to which it belongs.

In [24], the authors propose a key distribution in a cluster-based network that is based on a probabilistic key establishment inside the cluster. In addition, they propose a voting mechanism that detects an intruder node of the network. Then, all of the keys shared with that node should be revoked. The revocation step is initiated by the cluster head. Cluster heads form path keys to reach the base station. The authors also discuss the revocation process for those keys. The key revocation is based on asymmetric cryptography for inside cluster revocation and symmetric encryptions for inter cluster revocation. The authors do not specify the cryptographic algorithms used for symmetric or asymmetric primitives.

In [25], the authors propose a key distribution based on key chains where each key is obtained by applying a hash function on the next key in the chain. These key chains are either exchanged through a secure channel or installed in nodes before deployment. Authenticating the keys is done by verifying the hash result. A central node is used as a key service entity that manages the key revocation process. This entity shares a secret key with every node in the network. The authors do not explain how this secret key is shared, nor how this secret is renewed.

In [26], the authors propose a key revocation scheme based on public key cryptography that demands periodical certificate broadcast. They propose the use of hash function chains instead of digital signatures in order to conserve the limited resources of nodes. It is based on the exchanges that take place after deployment on a secure channel, which means that the verification process of the proposed protocol supposes the existence of a secure channel that helps them to define the shared information between neighbors. The authors also propose using a voting mechanism in order to avoid malicious revocations. Only revocations that reach a certain threshold in a neighborhood are validated.

Protocols based on key chains cannot easily update the chains, for their key renewal is based solely on the hashing function and the root value of the chain. Moreover, chains should be long enough to last for the life duration of the network.

In [27], the revocation method is centralized and based on preventing nodes from updating session keys and, thus, preventing them from being able to exchange messages with other nodes in the network. All communications are encrypted with a session key and a message authentication code (MAC). Symmetric session keys are renewed using a function that takes the previous session key as a parameter in addition to a pairwise key. The authors assume that the base station can reach all nodes in the topology. In [28], the same key revocation scheme is used and applied on a hierarchical network topology with three levels. In this method, keys are only revoked upon periodic renewal at the end of each session. This can be a handicap when the session is very long.

In [29], the authors proposed a periodic key renewal based on fragmentation of the key generation function. This method supposes that nodes will assemble the fragments sent by the base station. The process of assembling the function is not authenticated and might be easily falsified, for it uses only one key, which is a shared key between all of the nodes of the network; thus, any compromised node is able to interfere in the process.

In [30], the authors propose a cluster reorganization in order to avoid renewing keys. By updating the cluster head, a new pairwise key is established between the base station and the cluster head. This key established is made without exchanging the key materials between the two entities. The shared key is a simple exclusive-or of all of the keys of the nodes of the cluster. This supposes that the number of keys is limited to the number of potential cluster heads, which is a limited number compared to the life time of the network. In addition, the initial pairwise key agreement between neighboring nodes lacks authentication. Any neighboring node is able to establish a key with its neighbor.

In [31], the authors propose a key renewal mechanism that is managed by a special entity called the command node. The network is organized into clusters with one cluster head node that is used for key revocation. This cluster head receives the list of new keys for renewal and sends them to the other cluster gateways, which, in turn, send them to the sensor nodes. The authors use the shared key with the command node to update keys and do not propose a mechanism for updating this shared key.

Table 1 summarizes the comparison between the related work schemes and our proposition. The comparison is based on the type of cryptographic algorithms used, the type of cryptographic technique and the evaluation and verification methods. We do not include the complexity of the protocols listed in the table, since these analyses are not always given by the authors of the protocols, and often, it is not feasible to perform a fair analysis of the protocol complexity only based on the presentation given in the papers.

As the table shows, and according to our knowledge, none of the existing revocation and key renewal protocols were verified using an automatic formal verification tool. In addition, most of the results in the state of the art are obtained through simulations or complexity estimation when evaluating the cost of the cryptographic scheme. Most of the existing schemes do not specify the cryptographic algorithms of encryption/decryption, leaving the choice of these algorithms to the application. We provide automatic formal verification and real testbed implementation to further support our proposal. We used standard algorithms and used both symmetric and asymmetric cryptography.

Table 1. Comparison of related work schemes and our proposition.

Proposed Schemes	Standard Algorithms	Cryptographic Technique	Simulation	Implementation	Verification
Chan *et al.* 2003 [22]	not specified	symmetric	none	none	none
Chan *et al.* 2005 [21]	not specified	symmetric	none	none	none
Chattopadhyay *et al.* 2012 [23]	not specified	symmetric	none	none	none
Chuang *et al.* 2010 [26]	yes	asymmetric	none	none	none
Dini *et al.* 2006[25]	not specified	symmetric	none	none	none
Jiang *et al.* 2008 [24]	not specified	symmetric	none	none	none
Jolly *et al.* 2003 [31]	not specified	symmetric	yes	none	none
Purohit *et al.* 2011 [28]	not specified	symmetric	none	none	none
Wang *et al.* 2006 [29]	none	symmetric	none	none	none
Wang *et al.* 2010 [30]	not specified	symmetric	yes	none	none
Wang *et al.* 2007 [27]	not specified	symmetric	yes	none	none
All of our schemes	yes	symmetric/asymmetric	none	TelosB	automatic

1.3. Outline

In the next section, we introduce the cryptographic primitives used and present the two join protocols proposed in [11]. In Section 3, we describe all of our renewal and revocation protocols. Then, in the following section, we explain the relationships between all keys in order to understand in which order the renewal has to be done. Then, in Section 5, we use Scyther to verify all of our protocols. In Section 6, we present our experiments performed on testbeds using TelosB motes for all of our protocols. These measurements allow us to compare the execution time of our solutions. Finally, we conclude in the last section.

2. Authenticated Join Protocols

Before recalling the two authenticated join protocols given in [11], we introduce some cryptographic primitives and notations used in the rest of the paper.

2.1. Cryptographic Primitives and Notations

We use public key elliptic curve cryptography (ECC), using parameters secp160r1 given by the Standards for Efficient Cryptography Group [32]. Our implementation of ECC on TelosB is based on the optimized TinyECC library [33]. More precisely, we use the elliptic curve integrated encryption scheme (ECIES 160 bits), the public key encryption system proposed by Victor Shoup in 2001 [34].

For all symmetric encryptions, we use an optimized implementation of AES [35] with a key of 128 bits proposed by [36].

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

- I: a new node that initiates the protocol,
- R: a neighbor of node I,
- S: the sink of the network (also called the base station),
- 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,
- $K(I,S)$ or $K(S,I)$: the symmetric session key between I and S,
- NK: the symmetric network key between all nodes of the network,
- $K_{DH}(N,S)$ or $K_{DH}(S,N)$: the shared symmetric key between N and S using the Diffie–Hellman key exchange without the interaction described below.

Before deployment, each node N knows the public key $pk(S)$ of the sink and also its own pair of public and private keys, denoted $pk(N)$ and $sk(N)$, respectively. Based on ECC, we have that $pk(N) = sk(N) \times G$, where G is a public generator point of the elliptic curve. From $pk(N)$ and G, it is difficult to find $sk(N)$; this problem is called the elliptic curve discrete logarithm problem (ECDLP) [37,38].

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, denoted $K_{DH}(N,S) = K_{DH}(S,N)$.

- The sink knows its own secret key $sk(S)$ and the public key $pk(N)$ of any 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 $K_{DH}(N,S)$.

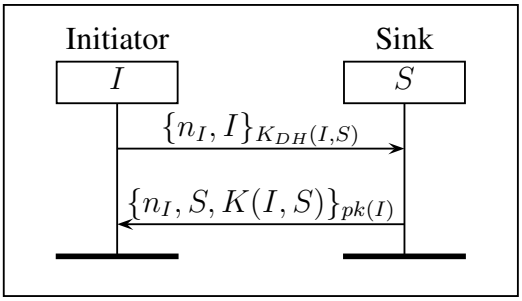
Both computations give the same shared key, since: $K_{DH}(N,S) = sk(N) \times pk(S) = sk(N) \times (sk(S) \times G) = (sk(N) \times G) \times sk(S) = pk(N) \times sk(S)$.

In our protocols, we intensively use this mechanism.

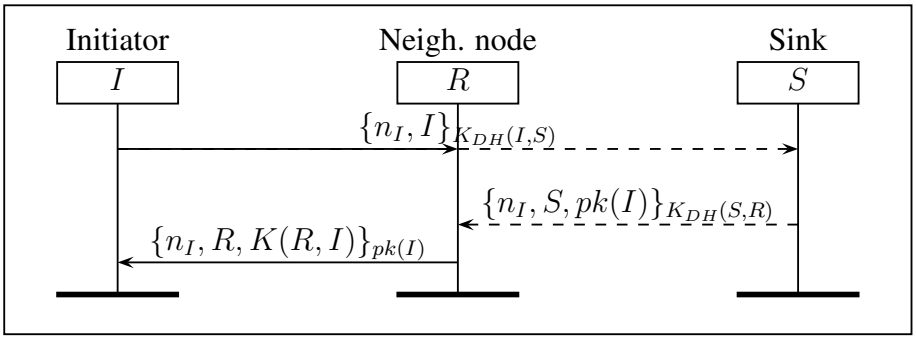
2.2. Join Protocols (DJS and IJS)

In Figure 1, we present two key establishment protocols given in [11]. The first protocol, called direct join to the sink (DJS), allows a node to join directly through the sink and is described in Figure 1a. In DJS, 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 then encrypts it with $K_{DH}(I,S)$ and sends it to S. Upon reception, S computes $K_{DH}(I,S)$ to decrypt the request. Then, S verifies the identity of I 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 $K(I,S)$. 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 S.

The second protocol, called indirect join to the sink (IJS), allows a new node I to join the network through a neighbor node R that is already authenticated in the network. Node I sends an indirect request to S in order to establish a session key with R. Node R forwards without any modification the request to S through intermediate nodes that are trusted to route the request towards S. Only nodes I and S are able to decrypt the messages encrypted with $K_{DH}(I, S)$, and only R and S are able to decrypt the messages encrypted with $K_{DH}(S, R)$.



(a)



(b)

Figure 1. Join protocols. (**a**) DJS: direct join to the sink. Node I directly joins the network by communicating directly with the sink S. (**b**) IJS: indirect join to the sink. Intermediate nodes between R and S forward messages without any encryption or decryption.

3. Renewal and Revocation Protocols

Our contribution is the design of secure key revocation and key renewal protocols, based on two key establishment protocols presented in Section 2. In this context, each node N has several keys that should be renewed: a symmetric shared key between N and S ($K_{DH}(N, S)$), a pair of public/secret keys ($pk(N), sk(N)$), the public key of the sink ($pk(S)$) and a symmetric network key (NK). We start by presenting the key revocation protocol (KR). Then, we present the protocol RSK for renewing symmetric keys ($K_{DH}(N, S)$) between a node N and the sink. Then, we propose four protocols to renew asymmetric keys of a node (RAKnk$_a$, RAKnk$_b$, RAKdh$_a$ and RAKdh$_b$) using different cryptographic primitives. Finally, we propose a protocol to renew the network key (RNK) and two multihop shared key protocols (MSK$_a$ and MSK$_b$) to establish via the sink a shared symmetric key between any two nodes of the network. In order to achieve mutual authentication of the sender and the receiver, we use nonces

in our protocols. Each time we reduce to the minimum the number of needed nonces. For example, in some cases, we use a fresh generated key as a nonce in order to minimize it. Moreover removing a single nonce or a part of a message in any of our protocols will create a flaw either in authentication or secrecy. All nonces play a crucial role in the mutual authentication in order to avoid several classic attacks, like man in the middle, replay or reflection attacks. Finally, we decide to use nonces instead of complex and time-consuming cryptographic primitives, like, for instance, signatures or MAC (message authentication codes).

3.1. Key Revocation Protocol

The sink collects the IDS results and determines the nodes that have to be revoked. Then, it sends a revocation request to node I using the protocol (KR) described in Figure 2. In this protocol, the sink sends to node I the list $M_1, \ldots, M_k$ of all revoked nodes in the neighborhood of I and a nonce n_S encrypted with $K_{DH}(S, I)$. Then, node I deletes all shared session keys with all nodes included in the list and does not accept any further communications with these nodes. In order to confirm the reception of the list, node I sends back the nonce n_S encrypted with $K_{DH}(S, I)$. Nonce n_S acknowledges the reception of the list by node I, and it also ensures the authentication.

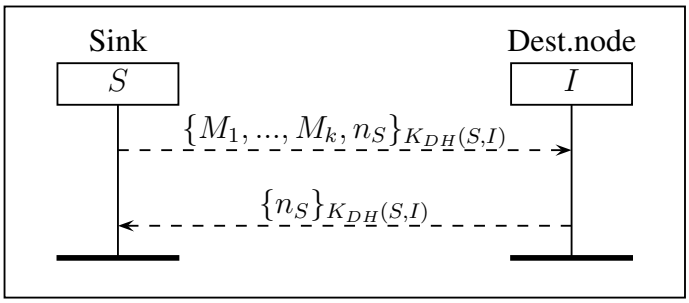


Figure 2. Key revocation protocol (KR): revocation of k malicious neighbor nodes of node I.

3.2. Renewing Symmetric Keys

Figure 3 presents protocol RSK, which allows an initiator node I to renew a session key with its neighbor R. The protocol consists of sending the new session key $K'(I, R)$ encrypted with the previous session key $K(I, R)$ in order to confirm to R that I has the previous session key. Then, the message is encrypted again with the public key of R. Notice that an intruder should obtain $K(I, R)$ and $sk(R)$ in order to learn the new session key $K'(I, R)$. This part of the protocol takes more execution time due to the extra public key encryption (see Table 2). Finally, the new key $K'(I, R)$ is used as a nonce by R to confirm the reception by sending back to I the message $\{K'(I, R)\}_{K'(I,R)}$.

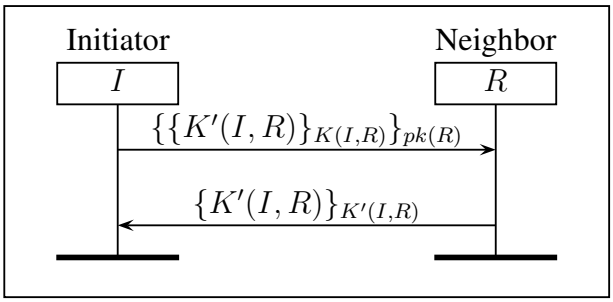


Figure 3. RSK: renewing a symmetric or session key.

Table 2. Execution time of all protocols.

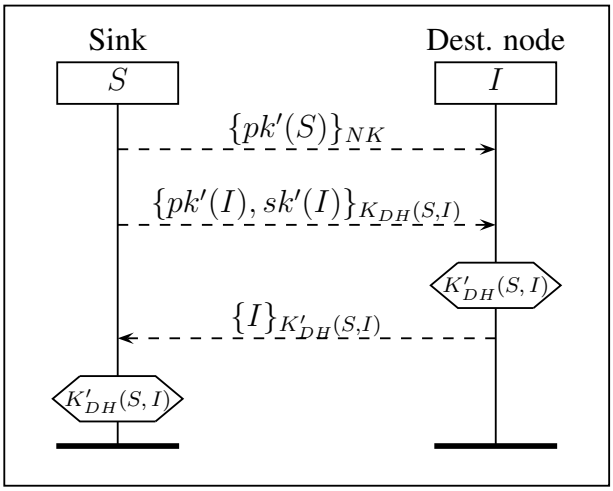
Protocol	Name	Figure	Time with S (ms)	Time without S (ms)	Gain	Standard Deviation (ms)
Join Protocols [11]	DJS	1a	10,112.62	4082.05	59%	78.09
	IJS	1b	10,180.81	10,049.45	1%	111.94
Revocation	KR	2	155.37	87.58	44%	3.82
Renewing SymKey	RSK	3	10,042.32	10,042.32	0%	76.49
Renewing AsymKey	RAKnk$_a$	4a	6797.75	3436.24	49%	4.26
	RAKnk$_b$	4b	3646.05	254.62	93%	3.95
	RAKdh$_a$	5a	6797.75	3436.24	49%	4.26
	RAKdh$_b$	5b	3646.05	254.62	93%	3.95
Renewing Network Key	RNK	6	221.09	121.40	45%	3.73
Multihop Shared Key	MSK$_a$	7a	6893.76	6631.91	4%	5.76
	MSK$_b$	7b	3682.53	301.42	91%	5.25

3.3. Renewing Asymmetric Keys

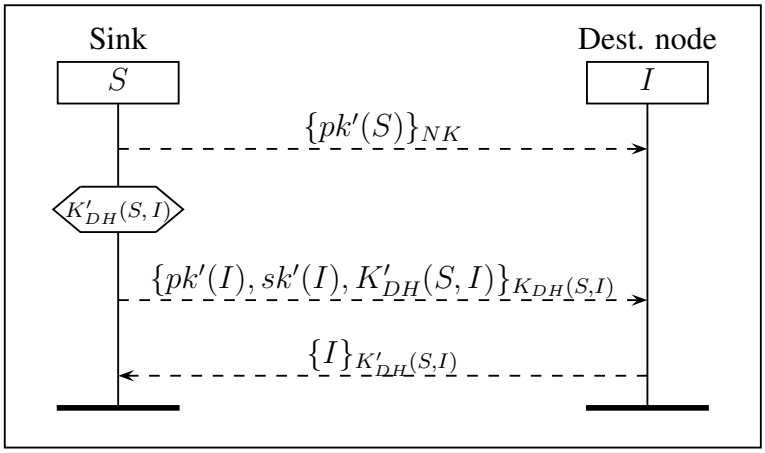
In what follows, we describe four protocols to renew asymmetric keys of the network. These protocols use the existing key infrastructure to securely replace the asymmetric keys between the sink and all nodes of the network. For this, the sink creates its own new public/private keys $(pk'(S), sk'(S))$ and a new pair of public/private keys $(pk'(I), sk'(I))$ for each node I in the network. Our four protocols (RAKnk$_a$, RAKnk$_b$, RAKdh$_a$ and RAKdh$_b$) are based on the same idea: first, S securely sends $pk'(S)$ and the new pair of keys for node I; then, node I replies by sending back its identity with the new shared key $K'_{DH}(S, I)$. We use different cryptographic mechanisms to distribute these new keys.

In Figure 4, we present two protocols, RAKnk$_a$ and RAKnk$_b$, where the new public key of the sink $pk'(S)$ is broadcast to all nodes using the network key (NK). In the first protocol RAKnk$_a$, depicted in Figure 4a, the sink only sends to each node I the new pair of keys using $K_{DH}(S, I)$. Then, I computes the new shared key $K'_{DH}(S, I) = sk'(I) \times pk'(S)$. In order to save computation time for node I, we propose a second version RAKnk$_b$ described in Figure 4b, where the sink pre-computes $K'_{DH}(S, I) = sk'(S) \times pk'(I)$ without using the secret key of I.

An alternative is to use the pre-shared key $K_{DH}(S, I)$ instead of NK in the distribution of $pk'(S)$, as depicted in Figure 5. In Figure 5a, we explain the protocol RAKdh$_a$ with the computation of the new key performed by node I. In Figure 5b, we present the protocol RAKdh$_b$ where the sink pre-computes $K'_{DH}(S, I)$. These two protocols use symmetric shared keys on each hop, preventing an intruder from learning the new key of the sink by learning the network key, as is the case in the protocols of Figure 4. Nevertheless, this solution requires more load on the network, since the transmission of the public key of the sink is not a broadcast using the network key, but a unicast using a symmetric shared key between two nodes. For instance, for a line of four nodes (S, A, B and C), in one case, three messages are enough to broadcast $pk'(S)$, and in the other situation, it requires $1 + 2 + 3 = 6$ messages to send $pk'(S)$.

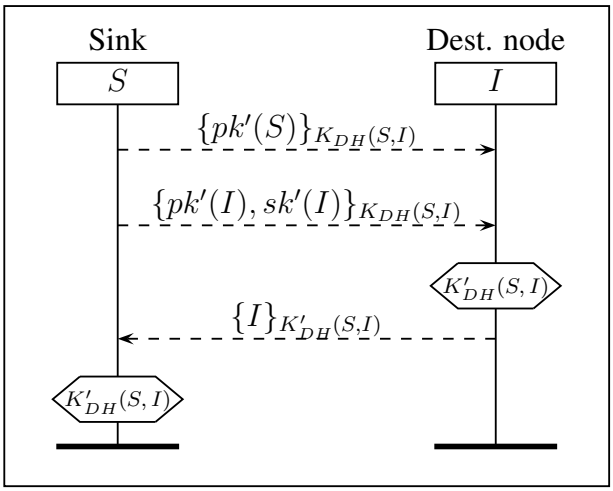


(a)

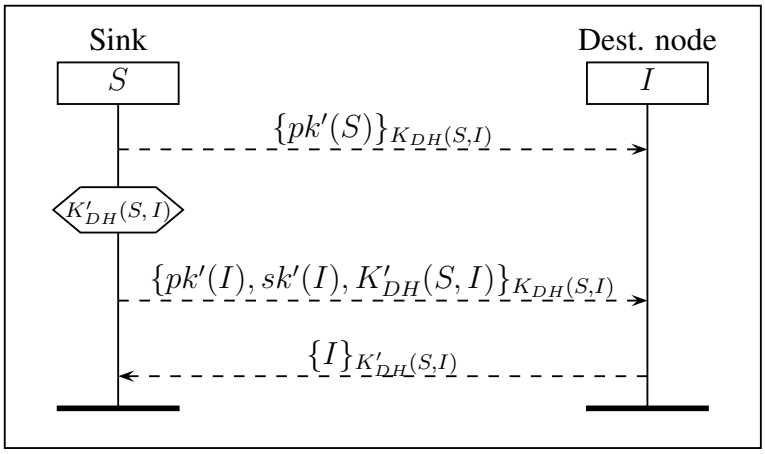


(b)

Figure 4. Two protocols for renewing asymmetric keys of a node I, where S uses the network key NK to broadcast $pk'(S)$. (**a**) Protocol RAKnk$_a$: S and I compute $K'_{DH}(S, I)$; (**b**) protocol RAKnk$_b$: S computes $K'_{DH}(S, I)$ and sends it to I.



(a)



(b)

Figure 5. Two protocols for renewing the new asymmetric keys of a node I, where S uses the symmetric shared key $K_{DH}(S,I)$ to deliver $pk'(S)$ to I. (**a**) Protocol RAKdh$_a$: S and I compute $K'_{DH}(S,I)$; (**b**) protocol RAKdh$_b$: S computes $K'_{DH}(S,I)$ and sends it to I.

3.4. Renewing the Network Key

Changing the network key NK is a decision made by the sink. We propose a secure way for the sink to distribute this new key to all authenticated nodes. This protocol, denoted RNK, is described in Figure 6. It allows the sink to be sure that all nodes receive the new key before starting to use it. It works as follows: the sink generates a new network key NK' and a nonce n_s. Then, it encrypts NK' and $n_{(S,I)}$ (one new nonce per node I) using the shared symmetric key $K_{DH}(S,I)$ and sends the encrypted message to each node. Then, it collects all nonces $n_{(S,I)}$ before starting to communicate with the new network key NK'.

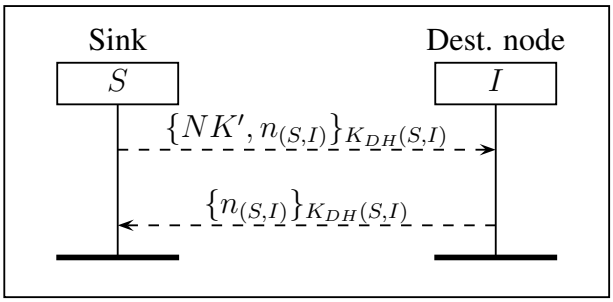


Figure 6. Protocol RNK: renewing the network key.

3.5. Multihop Shared Key Protocol

Our aim is to establish a shared key between any two authenticated nodes I and R of the network (not necessarily in range). We propose two protocols, called MSK$_a$ and MSK$_b$. The protocol MSK$_a$, depicted in Figure 7a, uses the secure channels created between the sink and each node to communicate the public key of I and R. Notice that in our context, the sink knows all of the public keys of all nodes, and a node only knows its public key and the public key 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:

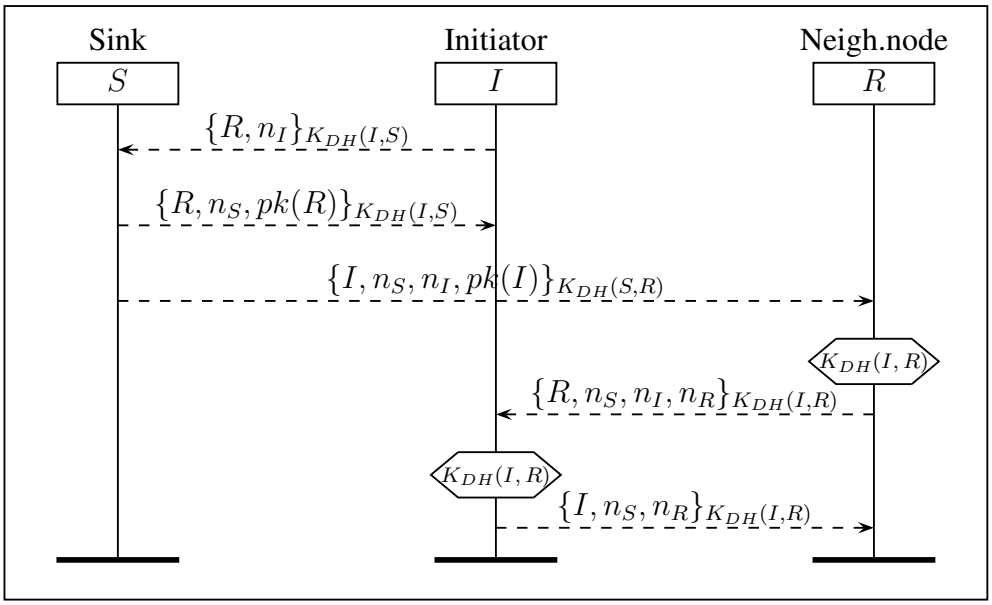
- to I, the identity of R, a nonce n_S, the public key of R encrypted with the shared symmetric key $K_{DH}(I, S)$,
- to R, the identity of I, the same nonce n_S, the nonce n_I received from I and the public key of I encrypted with the shared symmetric key $K_{DH}(S, R)$.

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

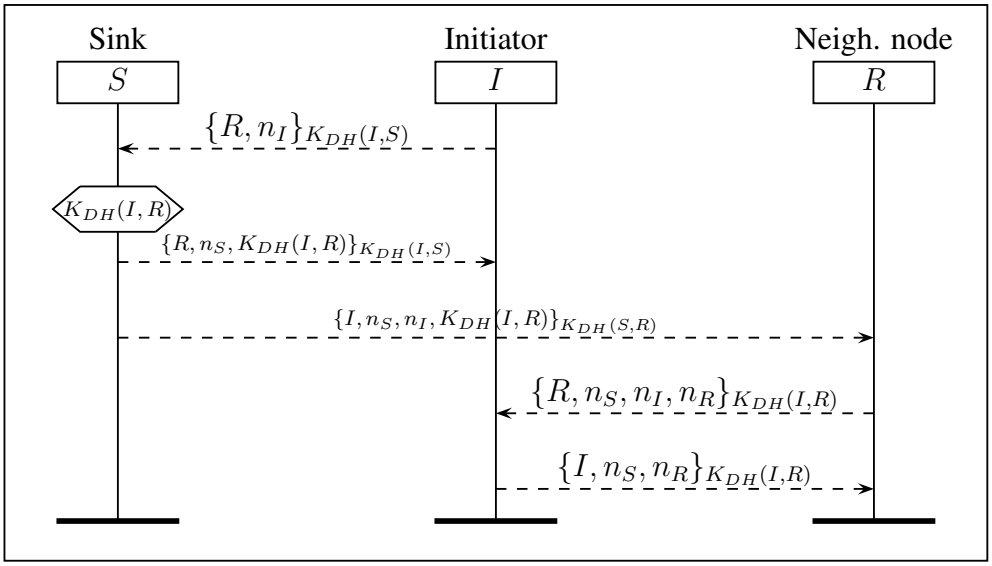
- Node I computes $sk(I) \times pk(R) = sk(I) \times sk(R) \times G = K_{DH}(I, R)$.
- Node R computes $sk(R) \times pk(I) = sk(R) \times sk(I) \times G = K_{DH}(I, R)$.

To ensure mutual authentication of R and I, node R generates a nonce n_R, then uses $K_{DH}(I, R)$ to encrypt its own identity, the two received nonces from S plus its nonce n_R. This cipher is sent to I, without necessarily passing by S. Finally, node I verifies that the received nonce from R is the same as the one sent by the sink. Then, it confirms that it correctly received the message by sending to R its own identity and the two nonces n_S and n_R, encrypted with $K_{DH}(I, R)$.

Notice that the computation of the new keys can be done by the sink in order to save some computations on nodes R and I. This version, called MSK$_b$, is depicted in Figure 7b.



(a)



(b)

Figure 7. Protocols for the multihop shared key. (**a**) Protocol MSK$_a$; (**b**) protocol MSK$_b$.

4. Key and Protocol 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.

Figure 8 presents the dependency between keys for each of our protocols, where K_1 & $K_2 \xrightarrow{A:B} K_3|K_4$ denotes that K_3 or K_4 are delivered from A to B and encrypted with K_1 and K_2. The public/private keys of nodes are used to compute the shared symmetric keys using Diffie–Hellman without interaction described in Section 2.1. All of these keys, besides the network key NK, are predistributed to nodes before the deployment. Therefore, all protocols in Figure 8 plus KR can be executed at any time after

deployment, except protocol RSK. Indeed, protocols DJS and IJS end with establishing a session key between the initiator and its neighbor. Therefore, protocol RSK must be executed after DJS and IJS in order to renew the session key.

Protocol name	Delivering
DJS	$pk(I) \xrightarrow{S:I} K(I,S)$
IJS	$K_{DH}(N,S) \xrightarrow{S:N} pk(I) \xrightarrow{N:I} K(I,N)$
RSK	$K(I,R)\ \&\ pk(R) \xrightarrow{I:R} K'(I,R)$
RAKnk$_a$	$NK \xrightarrow{S:I} pk'(S)$ $K_{DH}(S,I) \xrightarrow{S:I} pk'(I)\|sk'(I)$
RAKnk$_b$	$NK \xrightarrow{S:I} pk'(S)$ $K_{DH}(S,I) \xrightarrow{S:I} pk'(I)\|sk'(I)\|K'_{DH}(S,I)$
RAKdh$_a$	$K_{DH}(S,I) \xrightarrow{S:I} pk'(S)$ $K_{DH}(S,I) \xrightarrow{S:I} pk'(I)\|sk'(I)$
RAKdh$_b$	$K_{DH}(S,I) \xrightarrow{S:I} pk'(S)$ $K_{DH}(S,I) \xrightarrow{S:I} pk'(I)\|sk'(I)\|K'_{DH}(S,I)$
MSK$_a$	$K_{DH}(I,S) \xrightarrow{S:I} pk(R)$ $K_{DH}(R,S) \xrightarrow{S:R} pk(I)$
MSK$_b$	$K_{DH}(I,S) \xrightarrow{S:I} K_{DH}(I,R)$ $K_{DH}(R,S) \xrightarrow{S:R} K_{DH}(I,R)$

Figure 8. Key dependency for our protocols. $K_1\ \&\ K_2 \xrightarrow{A:B} K_3|K_4$ denotes that K_3 or K_4 is delivered by A to B and encrypted with K_1 and K_2.

Protocol KR is done based on information given by an IDS in order to revoke the keys in possession of compromised nodes. In executing KR, the sink informs the nodes in the neighborhood of the compromised node to delete all shared keys. Therefore, protocols RAK and MSK can be executed regularly or after the revocation process in order to maintain the security level of the application in the case of RAK or to create more secure links between nodes in the case of MSK. Protocol RNK must be renewed in a short interval or after a revocation process.

5. Formal Security Evaluation

Evaluating the security of cryptographic protocols is not an easy task. It is easy to design flawed protocols. Recently, formal methods started to be used to analyze WSN protocols [39–41]. Moreover, during the last decade, several tools have been developed to automatically verify cryptographic protocols [12,42,43]. In order to prove the security of all of our protocols, we use the automatic cryptographic protocol verification tool Scyther [12] developed by Cas Cremers. We chose this tool since it is one of the fastest tools, as has been shown in [44], and it is one of the most user friendly.

5.1. Scyther Overview

Scyther is a free (GPL-2.0) tool available on all operating systems (Linux, Mac and Windows). This tool can automatically prove security properties or provide an attack on a cryptographic protocol 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 considers the Dolev–Yao intruder model. In this model, messages are abstracted by a term algebra, in order to formalize the cryptographic primitives used in the protocols. Moreover, the perfect encryption hypothesis is made, meaning that it is possible for a malicious node to decrypt an encrypted message only if it knows the associated secret key. Finally, in this model, the intruder is controlling the network, which means that all messages can be captured by the intruder. Thus, he has the ability to delete, change or modify any message according to his knowledge. This last assumption is very strong, since in a WSN, it is not realistic to consider that one intruder is able to receive what each node is receiving. However, if a protocol is secure against such a strong intruder, it will also be secure against a weaker attacker. Indeed, the tool considers the most powerful intruder in order to prove the highest security.

5.2. Security Analysis

We verified automatically all of our protocols using Scyther in a few seconds on a regular PC. Scyther concludes that all of our protocols are secure. 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 here [45]. The advantage of using an automatic verification tool that considers a powerful intruder is that we are sure that there is no attack, including replay or man-in-the-middle attacks. In all of our protocols, the authentication is ensured by using, in an appropriate manner, nonces, which also ensures the freshness of the messages to avoid, for instance, replay attacks. Moreover, the tool guarantees that our protocols preserve the confidentiality of the sensitive data and that the different participants communicate in an authenticated manner.

Moreover, for each protocol, we minimize the amount of exchanged data. This is why we use in the protocol of Figure 3 the new symmetric key as a nonce. Similarly, in the last protocol (Figure 7a), all nonces are crucial, since each time you remove one, Scyther finds an authentication flaw.

6. Experiments

Using Scyther, we have proven the security of all of our protocols, but Scyther cannot evaluate the execution time of our solutions. Hence, to compare the execution time of our different protocols, we implemented each protocol on TelosB motes. The goal is to compare the performances of the different proposed protocols. We first describe the settings of our testbed. Then, we provide and discuss the results of the execution time of each protocol.

6.1. Settings

In order to evaluate the efficiency of our solutions, we used TelosB motes. These 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. Due to the limitation of their computing resources, these motes are used as a basis for comparison between the different protocols and not for obtaining the best results in terms of performance. Our implementation of ECC is based on the TinyECC library [33], on ECIES with a key of 160 bits and on an optimized implementation of symmetric encryption AES [35] in CTR mode with a key of 128 bits. 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 k 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 exclusive-or operator. If you know IV and the key k, then you 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 0 up to 128 bits in order to have both operands with the same length to perform the bitwise exclusive-or. In this case, the size of the transmitted encrypted packets is always a multiple of 128 bits. However, in CTR mode, we can just cut the $\{IV + p\}_k$ message to the size of m_p. Hence, we can transmit an encrypted message that has exactly the same size as the original message and, therefore, avoid any overhead in terms of transmission time. For example, in protocol KR, if the list of revoked nodes and the nonce is l-bits long, that is smaller than 128 bits (AES block size), it is sufficient to only transmit l bits.

During the experiments, we considered topologies without intermediate nodes, since these nodes would only forward packets, in a multihop manner, without doing any modification on the packet. The cost of these communications is therefore negligible compared to the encryption and decryption costs. Moreover, this cost is the same for all protocols; only the load of the network can change between unicast and broadcast protocol. Hence, for each situation, we only consider a minimal topology containing only the nodes involved in the cryptographic operations. Moreover, malicious nodes were only included in the verification model automatically done by Scyther. No malicious nodes were included in the experiments. When detecting (IDS), a malicious or a compromised node, the sink is informed. Then, the sink can launch the revocation process in executing the protocol KR. Afterwards, the sink should launch the renewing key process in executing one of protocols RAK for the nodes in the neighborhood of the malicious node and RNK for all of the concerned nodes of the network.

6.2. Results and Discussion

In Table 2, we provide the execution time for all of our protocols (please note that the execution time for the cryptographic algorithms can be found in [46]). We also present the results without the execution time of the sink, since in many applications, the base station is a special node with extra resources. All results are the averages of 100 experiments of each protocol. We also provide the standard deviations for execution time including the time of S. We notice that these values are small compared to the execution time of the protocol. These variations are normal according to the motes used, physical parameters like 2.4 GHz interference, battery level, humidity, temperature, *etc*. Moreover, the cause of the high standard deviation of join protocols and RSK_b presented in Table 2 is essentially related to the

random number (that is, of course, different at each generation) that is used in multiplication operations in the asymmetric encryption. In fact, when the random number is small, the multiplication operations take much less time than what they take for a large random number. To confirm this claim, we have tested with a given constant random number coded on 20 bytes. We obtained small standard deviation values, 5.94 ms for the DJS protocol and 5.71 ms for the IJS protocol.

The differences in our protocols come from the usage of cryptographic primitives. All protocols using asymmetric encryption require more execution time than protocols using only symmetric encryption. Moreover, we see that the two slowest protocols are the join protocols proposed in [11]. Hence, it is more efficient to renew keys than to rejoin the network.

The protocol KR (key revocation) is the fastest. We also note that the sink performs almost half of the cryptographic operations; thus, by making it do more operations, we avoid sensor nodes doing the heavy cryptographic computations. In the experiments we considered only one node to revoke. Knowing that a nonce is represented on 4 bytes and a node identity on 1 byte, we obtain 5 byte-long messages. In this case, we encrypt with the AES 128-bit key the initial vector (IV), then we XOR the 5 bytes and only send a message of 5 bytes. Therefore, if the size of the list of revoked nodes increases, then the protocol KR will take more time.

The protocol RSK_a, renewing the symmetric key, also exchanges two messages encrypted with a symmetric encryption scheme, but the size of the exchanged message is longer; this is why it is slightly slower than KR. In order to have a better level of security, we add to the protocol RSK_b an extra public key encryption. This operation is very costly, and the execution time of this protocol is close to the protocols proposed in [11]. Moreover, since the sink is not involved in this protocol, the gain of execution time without counting the execution time on S is null.

The protocol RNK is also one of the fastest protocols, since it is based on the same cryptographic primitives as those of protocol KR, but the size of the message is 20 bytes, so requires two AES encryptions of 16 bytes and the transmission of $16 + 4$ bytes.

For renewing the asymmetric key, we proposed four protocols: two of them use the symmetric network key NK, and the other two use symmetric keys K_{DH}. We see that since they are using the same symmetric encryption mechanism, they take the same execution time. Hence, the execution time for protocols RAKnk_a and RAKdh_a is the same and similar for protocols RAKnk_b and RAKdh_b. However, the second versions of these protocols, RAKnk_b and RAKdh_b, are faster than protocols RAKnk_a and RAKdh_a (more if we do not count the sink execution time). This clearly shows that the computation of the new key by a node is expensive. Therefore, it is important that a designer takes this into account during the conception of the protocols in order to have efficient protocols and also to preserve the resources of the nodes.

Finally, we measured the execution time of protocols MSK_a and MSK_b, presented respectively in Figure 7; we observe that the pre-computation of the key by the sink allows us to save 3 s. By doing the key generation on the sink once, we avoid consuming additional time and energy on the sensor nodes. Indeed, this avoids creating the key twice, once in each node. This helps obtain a gain of around 90% if we consider that the sink is not time nor energy constrained.

7. Conclusions and Perspectives

We have proposed several protocols to revoke a set of nodes, to renew symmetric and asymmetric keys and to establish a shared key between two authenticated nodes of the network. Our solutions use the ECC, which allows nodes to easily construct shared keys without interaction. Moreover, all of our protocols have been automatically verified as secure using Scyther. This ensures the security of our solutions. We also have implemented and tested all of our protocols on TelosB nodes in order to evaluate their execution time relative to one another. These results show that according to the load of the network and to the topology, one protocol might be more efficient than another. Then, according to the context (size of the network, size of the battery, type of mote, energy consumption for communication, computation resources of the motes), one solution might be better than another one. All of these parameters should be taken into account before choosing one real solution.

In this version of our key renewal mechanism, we did not ensure message integrity. Ensuring message integrity implies the addition of MAC generated using a different key than the one used for encryption. Thus, response messages will take more time to be generated and will be 16 bytes bigger. We plan on proposing an extension that includes integrity mechanisms in our future works. Notice that the key establishment protocols DJS and IJS both ensure integrity, because they use the ECIES scheme, which includes a MAC operation on the cipher text.

As a long-term perspective, we plan on testing our protocols in more realistic platforms, such as the IoT-LAB platform [47]. This would give us an idea about how our different protocols would perform in a large-scale network and how long it would take to revoke and renew distributed keys and to establish the required secured links.

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, verification and evaluation of the protocols, while the implementation work has been essentially done by Ismail Mansour.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Hussain, M.A.; Khan, P.; Sup, K.K. WSN research activities for military application. In Proceedings of the 11th International Conference on Advanced Communication Technology, Phoenix Park, Korea, 15–18 February 2009; Volume 1, pp. 271–274.

2. Debar, H.; Dacier, M.; Wespi, A. Towards a taxonomy of intrusion-detection systems. *Comput. Netw.* **1999**, *31*, 805–822.
3. Sun, B.; Osborne, L.; Xiao, Y.; Guizani, S. Intrusion detection techniques in mobile ad hoc and wireless sensor networks. *IEEE Wirel. Commun.* **2007**, *14*, 56–63.
4. Rivest, R.L.; Shamir, A.; Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *ACM Commun.* **1978**, *21*, 120–126.
5. El Gamal, T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Trans. Inf. Theory* **1985**, *31*, 469–472.
6. Eisenbarth, T.; Kumar, S.; Uhsadel, L.; Paar, C.; Poschmann, A. A Survey of Lightweight-Cryptography Implementations. *IEEE Des. Test Comput.* **2007**, *24*, 522–533 .
7. 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), Reykjavík, Iceland, 29–31 July 2013; pp. 543–548.
8. Jeong, K.; Lee, C.; Lim, J. Improved differential fault analysis on lightweight block cipher LBlock for wireless sensor networks. *EURASIP J. Wirel. Commun. Netw.* **2013**, *2013*, doi:10.1186/1687-1499-2013-151.
9. Couroussé, D.; Robisson, B.; Lanet, J.; Barry, T.; Noura, H.; Jaillon, P.; Lalevée, P. COGITO: Code Polymorphism to Secure Devices. In Proceedings of the 11th International Conference on Security and Cryptography (SECRYPT 2014), Vienna, Austria, 28–30 August 2014; pp. 451–456.
10. Zhang, J.; Varadharajan, V. Wireless sensor network key management survey and taxonomy. *J. Netw. Comput. Appl.* **2010**, *33*, 63–75.
11. Mansour, I.; Chalhoub, G.; Misson, M. Security architecture for multi-hop wireless sensor networks. In *Security for Multihop Wireless Networks*; CRC Press: Boca Raton, FL, USA, 2014; pp. 157–178.
12. Cremers, C. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In Proceedings of the 20th International Conference on Computer Aided Verification, Princeton, NJ, USA, 7–14 July 2008; pp. 414–418.
13. Mansour, I.; Chalhoub, G.; Lafourcade, P.; Delobel, F. Secure Key Renewal and Revocation for Wireless Sensor Networks. In Proceedings of the 39th IEEE Conference on Local Computer Networks (LCN), Edmonton, AB, Cananda, 8–11 September 2014; pp. 382–385 .
14. Mansour, I.; Chalhoub, G.; Lafourcade, P. Evaluation of Secure Multi-Hop Node Authentication and Key Establishment Mechanisms for Wireless Sensor Networks. *J. Sens. Actuator Netw.* **2014**, *3*, 224–244.
15. Mansour, I.; Chalhoub, G.; Lafourcade, P. Secure Multihop Key Establishment Protocols for Wireless Sensor Networks. In Proceedings of International Conference on Cryptography and Security Systems, Lublin, Poland, 22–24 September 2014; pp. 166–177.
16. Mehta, M.; Huang, D.; Harn, L. RINK-RKP: A scheme for key predistribution and shared-key discovery in sensor networks. In Proceedings of the 24th IEEE International on Performance, Computing, and Communications Conference, Phoenix, AZ, USA, 7–9 April 2005; pp. 193–197.

17. Park, J.; Kim, Z.; Kim, K. State-based key management scheme for wireless sensor networks. In Proceedings of IEEE International Conference on Mobile Adhoc and Sensor Systems, Washington, DC, USA, 7 November 2005.
18. Park, J.; Kim, Z.; Kim, K. Random key assignment for secure wireless sensor networks. In Proceedings of the 1st ACM workshop on Security of Ad Hoc and Sensor Networks, Washington, DC, USA, 27–30 October 2003; pp. 62–71.
19. Cheng, Y.; Malik, M.; Xie, B.; Agrawal, D. Enhanced Approach for Random Key Pre-Distribution in Wireless Sensor Networks. In Proceedings of International Conference on Communication, Networking and Information Technology, Amman, Jordan, 6–8 December 2007.
20. Cheng, Y.; Agrawal, D. An Improved Key Distribution Mechanism for Large-Scale Hierarchical Wireless Networks Key Distribution. *AD HOC Netw. J.* **2007**, *5*, 35–48.
21. Chan, H.; Gligor, V.; Perrig, A.; Muralidharan, G. On the distribution and revocation of cryptographic keys in sensor networks. *IEEE Trans. Dependable Secur. Comput.* **2005**, *2*, 233–247.
22. Chan, H.; Perrig, A.; Song, D. Random key predistribution schemes for sensor networks. In Proceedings of IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 11–14 May 2003.
23. Chattopadhyay, S.; Turuk, A.K. A Scheme for Key Revocation in Wireless Sensor Networks. *Int. J. Adv. Comput. Eng. Commun. Technol.* **2012**, *1*, 16–20.
24. Jiang, Y.; Shi, H. A Cluster-Based Random Key Revocation Protocol for Wireless Sensor Networks. *J. Electron. Sci. Technol. China* **2008**, *6*, 10–15.
25. Dini, G.; Savino, I. An efficient key revocation protocol for wireless sensor networks. In Proceedings of International Symposium on a World of Wireless, Mobile and Multimedia Networks, Buffalo, NY, USA, 26–29 June 2006.
26. Chuang, P.; Chang, S.; Lin, C. A Node Revocation Scheme Using Public-Key Cryptography in Wireless Sensor Networks. *J. Inf. Sci. Eng.* **2010**, *26*, 1859–1873.
27. Wang, Y.; Ramamurthy, B.; Zou, X. KeyRev: An Efficient Key Revocation Scheme for Wireless Sensor Networks. In Proceedings of International Conference on Communications, Glasgow, UK, 24–28 June 2007; pp. 1260–1265.
28. Purohit, G.N.; Rawat, A.S. Revocation and Self-Healing of keys in Hierarchical Wireless Sensor Network. *Int. J. Comput. Sci. Inf. Technol.* **2011**, *2*, 2909–2914.
29. Wang, C.; Hong, T.; Horng, G.; Wang, W. A Key Renewal Scheme under the Power Consumption for Wireless Sensor Networks. In Proceedings of the 4th International Conference on Photonics, Networking and Computing, Kaohsiung, Taiwan, 8–11 October 2006.
30. Wang, G.; Kim, S.; Kang, D.; Choi, D.; Cho, G. Lightweight Key Renewals for Clustered Sensor Networks. *J. Netw.* **2010**, *5*, 300–312.
31. Jolly, G.; Kusçu, M.; Kokate, P.; Younis, M. A Low-Energy Key Management Protocol for Wireless Sensor Networks. In Proceedings of the Eighth IEEE International Symposium on Computers and Communications, Kiris-Kemer, Turkey, 30 June–3 July 2003.
32. Standards for Efficient Cryptography Group. SEC 1: Elliptic Curve Cryptography. Available online: http://www.secg.org/2000 (accessed on 26 August 2015).

33. Liu, A.; Ning, N. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In Proceedings of 7th International Conference on Information Processing in Sensor Networks, St. Louis, MI, USA, 22–24 April 2008; pp. 245–256.
34. Shoup, V. A Proposal for an ISO Standard for Public Key Encryption. *IACR Cryptology ePrint Archive: Report 2001/112*. Available online: http://eprint.iacr.org/2001/112 (accessed on 26 August 2015).
35. Daemen, J.; Rijmen, V. *The Design of Rijndael: AES—The Advanced Encryption Standard*; Springer-Verlag: Berlin, Germany, 2002.
36. Manica, N.; Saloni, M.; Toldo, P. WSN—Secure comunications with AES algoritms. Faculty of Computer Science, University of Trento, Trento, Italy, 2008.
37. Blake, I.F.; Seroussi, G.; Smart, N.P. *Elliptic Curves in Cryptography*; Cambridge University Press: New York, NY, USA, 1999.
38. Miller, V.S. *Use of Elliptic Curves in Cryptography*; Springer-Verlag New York, Inc.: New York, NY, USA, 1986; pp. 417–426.
39. Meadows, C.; Poovendran, R.; Pavlovic, D.; Chang, L.; Syverson, P.F. Distance Bounding Protocols: Authentication Logic Analysis and Collusion Attacks. In *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks*; Poovendran, R., Roy, S., Wang, C., Eds.; Springer: Berlin, Germany, 2007; Volume 30, pp. 279–298.
40. Arnaud, M.; Cortier, V.; Delaune, S. Modeling and Verifying Ad Hoc Routing Protocols. *Inf. Comput.* **2010**, *238*, 30–67.
41. Pura, M.L.; Patriciu, V.V.; Bica, I. Formal Verification of Secure Ad Hoc Routing Protocols Using AVISPA: ARAN Case Study. In Proceedings of the 4th Conference on European Computing Conference, Bucharest, Romania, 20–22 April 2010; pp. 200–206.
42. 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 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, 6–10 July 2005; pp. 281–285.
43. Blanchet, B. Automatic Proof of Strong Secrecy for Security Protocols. In Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA, USA, 9–12 May 2004, pp. 86–100.
44. Cremers, C.J.F.; Lafourcade, P.; Nadeau, P. Comparing State Spaces in Automatic Security Protocol Analysis. In *Formal to Practical Security*; Springer Berlin Heidelberg: Berlin, Germany, 2009; pp. 70–94.
45. Mansour, I.; Lafourcade, P.; Chalhoub, G. Scyther code of our authentication protocols. Available online: http://sancy.univ-bpclermont.fr/~lafourcade/scyther-jsan-code.tar (accessed on 26 August 2015).
46. Mansour, I.; Chalhoub, G. Evaluation of different cryptographic algorithms on wireless sensor network nodes. In Proceedings of International Conference on Wireless Communications in Unusual and Confined Areas, Clermont Ferrand, France, 28–30 August 2012; pp. 1–6.

47. IoT-LAB. Available online: https://www.iot-lab.info/ (accessed on 26 August 2015).