HAL
open science

Secure Multihop Key Establishment Protocols for Wireless Sensor Networks

Ismail Mansour, Gerard Chalhoub, Pascal Lafourcade

To cite this version:

Ismail Mansour, Gerard Chalhoub, Pascal Lafourcade. Secure Multihop Key Establishment Protocols for Wireless Sensor Networks. Cryptography and Security Systems - Third International Conference, CSS 2014, Lublin, Poland, Sep 2014, Lublin, Poland. hal-01759913

HAL Id: hal-01759913

https://hal.science/hal-01759913v1

Submitted on 5 Apr 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Secure Multihop Key Establishment Protocols for Wireless Sensor Networks

Ismail Mansour, Gérard Chalhoub and Pascal Lafourcade

LIMOS, Clermont University, Campus des Cézeaux, Aubière, France

Abstract. Designing secure communication protocols is not an easy task. Cryptography is often necessary but does not always guarantee the security of protocols as several famous examples attest in the literature. Moreover, in the context of Wireless Sensor Networks (WSNs) the design is even more difficult due to the limited resources of sensor nodes that add extra constraints to take into account. During the life time of a secure WSN, one of the first crucial steps is the key establishment between two nodes. In this paper we propose four secure multihop key establishment protocols based on elliptic curve cryptography (ECC). For each protocol, we make a formal security proof using the automatic tool Scyther. Then, in order to evaluate their performances, we implemented them on testbeds using TelosB motes and TinyOS. Results allow us to estimate the overhead of our key establishment methods.
Keywords: Authentication, key establishment, Wireless Sensor Network, Security, Multihop, Formal Verification.

1 Introduction

Due to the technological advances, Wireless Sensor Networks (WSNs) are more and more used in diverse applications. In the first age of WSNs, the main concern was to efficiently transmit the data to the destination using wireless communications. Only few applications, like military WSN examples, required a high level of security [7]. In this context, it is important to design secure efficient communication mechanisms between nodes of the network, using cryptography.
Moreover, with the expansion of the Internet of Things (IoT) more and more devices will be interconnected and monitoring critical human activities. For instance nowadays most of the smart phones have a GPS, a camera, and several sensors. In a close future our environment will be equipped with several sensors in order to collect data to inform the user. In this context the knowledge of this data can leak private information, one solution to avoid this is to use cryptography in order to preserve privacy of the users. One of the first steps in this situation is to design secure multihop key establishment protocols. This is the main motivation of this paper.

Contributions:

Our goal is to design several secure key establishment protocols for WSNs and to evaluate their performances on real nodes. Hence our contributions can be split in three points:

- Design of key establishment protocols based on ECC.
- Formal security analysis of the protocols.
- Evaluation on TelosB testbeds of the execution time of each solution.

All our protocols are based on Elliptic Curve Cryptography (ECC) with a key of 160 bits. We propose a first protocol called MKE_S. It allows nodes that already share a key with the sink to establish a new secret key. The idea is to use the sink as a kind of third trusted party to establish a new key. In addition, we propose an improvement of this protocol called $MKE_S - light$. In this version, the sink performs costly computation instead of the nodes, since in many situations the sink has no limited battery and more computation power than nodes of the WSN. We also give two other protocols that establish a secret key between two nodes without passing by the sink. The first one is called MKE_{NK}, it uses only the network key shared between all the authenticated nodes in the network. This authentication phase can be done for instance using one of the protocols proposed in [10]. The second protocol is called MKE_K, also it does not use the sink, but it uses the symmetric keys shared between neighbor nodes. In order to guarantee the authentication of nodes involved, this protocol only requires two flows of messages instead of three as it is the case for the protocol MKE_{NK}.
In order to prove the security of all these protocols, we used Scyther an automatic cryptographic protocol verification tool developed by Cas Cremers. It is easy to propose a flawed protocol, this tool helps to verify the correctness of the security protocol and make sure that it resists against several kinds of attacks. Each of our solutions uses a different approach to securely establish a new secret key, they vary according to the number of cryptographic operations involved and the trust level that we need to have in the network nodes.
Our last contribution is the implementation on real nodes of all our protocols. Our aim is to measure the execution time of each protocol, but also to compare them in order to be able to judge which one is the most suitable.

Related Work:

Many contributions have been made in the key establishment and symmetric key distribution in WSNs. Some of them are based on a probabilistic predistribution 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 [1]. One of the most known symmetric key systems that were proposed for wireless sensor networks is SPINS (Security Protocols for Sensor Networks) [13] which uses a simplified version of TESLA (Timed, Efficient, Streaming, Loss-tolerant Authentication) protocol [12]. In SPINS, the base station plays an essential role in the key establishment process. It is a lightweight protocol but suffers from scalability and high dependency on the base station.
Authors in [11] proposed a multi-hop key establishment between nodes called Micro-PKI. 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. 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 on which depends the whole security architecture.

In [3], authors proposed 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, 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. Therefor, these two nodes are able to establish a secret key by using the trusted channel established with the common node. 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+ were proposed in [15]. They are based on the symmetric keys of Blom [2] with a perturbation function that makes more difficult for an attacker to guess the pairwise keys. In their paper they discuss the five criteria that were proposed in [16] 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* 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 enabling them to establish keys. The main weakness of this scheme is the lack of a rekeying process. The preshared matrices will only help to create one pairwise key for every couple of nodes.

Table 1 summarizes a comparison between the related work schemes and our proposition. The comparison is based on some repudiated criterias in WSN area. As the table shows, Yu et al. scheme [15] is the closest one to our schemes. Nevertheless, the authors use TelosB motes to evaluate energy consumption of basic operations used in their schemes and they provide a large scale simulation for thousands of nodes based on these measurements.

Proposed scheme	Pre-distribution	Trusted Party	Cryptographic technique	Simulation	Implementation	Verification
Perrig et al., 2000[12], 2002[13]	yes	Base station	symmetric	JAVA	none	manual
Chan et al., 2005[3]	yes	Base station	symmetric	yes	none	manual
Yu et al., 2009[15]	yes	none	symmetric	none	TelosB	manual
Munivel et al., 2010[11]	yes	Base station	symmetric/ asymmetric	manual	none	none
Our schemes	yes	Base station	symmetric/ asymmetric	none	TelosB	automatic

Table 1: Comparison of related work schemes.

In this paper, we did not compare our time execution results with other protocols from the state of the art because implementations are hardware and system dependent. In addition, they can be optimized for certain platforms which makes the comparison unfair using different platforms and cryptographic primitives. Finally the main difference with other works is that we formally prove the security of all our protocols using the automatic verification tool Scyther [4].

Outline:

In the next section, we introduce the notations used and present four key establishment protocols. Then, in Section 3, we give the results of our implementation on TelosB motes. Finally, we conclude in the last section.

2 Multi-hop Key Establishment Protocols

In our evaluation testbed, we used public keys based on Elliptic Curve Cryptography (ECC), using parameters secp160r1 given by the Standards for Efficient Cryptography Group [14]. Our implementation of ECC on TelosB is based on TinyECC library [8]. More precisely we used the Elliptic Curve Diffie-Hellman (ECDH) key agreement scheme [5]. For all symmetric encryption/decryption we use an optimized implementation of AES with a key of 128 bits proposed by [9].
Note that we only use the public keys in order to establish symmetric keys without doing any asymmetric encryption or decryption operations. Indeed, it helps us establish a pairwise key without interaction between nodes thanks to the predistribution of public keys before the deployment. Before deployment, each node N knows the public key $pk(S)$ of the sink S and also its own pair of private and public keys, denoted $(pk(N), sk(N))$ respectively. Based on ECC, we have that $pk(N) = sk(N) \times G$, where G is a 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 his 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:

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

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

- I: a new node that initiates the protocol,
- R: a neighbour of node I,
- S: the sink of the network (also called base station),
- J_i: the i-th intermediate node between R and S,
- n_A: a nonce generated by node A,
- $pk(A)$: 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,
- NK: the symmetric network key between all nodes of the network,
- $K_{DH}(N, S)$: the shared symmetric key between N and S using the Diffie-Hellman key exchange without interaction described above,
- $\{x\}_k$: the encryption of message x with the symmetric or asymmetric key k.

In all the figures that describe our protocol, we denote a direct communication by an arrow between two nodes, and a communication passing by several possible intermediate nodes by a dotted arrow. We also explicit above each exchanged message the size in Bytes.

Protocols with Intervention of the Sink

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 two protocols, called MKE_S and $MKE_S - light$. Protocol MKE_S, depicted in Figure 1, uses the secure channels created between the sink and each node to communicate the public keys of I and R. Notice that in our context 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 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 two messages 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 $sk(R) \times pk(I) = sk(R) \times sk(I) \times G = K_{DH}(I, R)$.

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 necessary 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 key $K_{DH}(I, R)$ can be done by the sink in order to save some computations on nodes R and I. This version called $MKE_S - light$ is depicted in Figure 2.

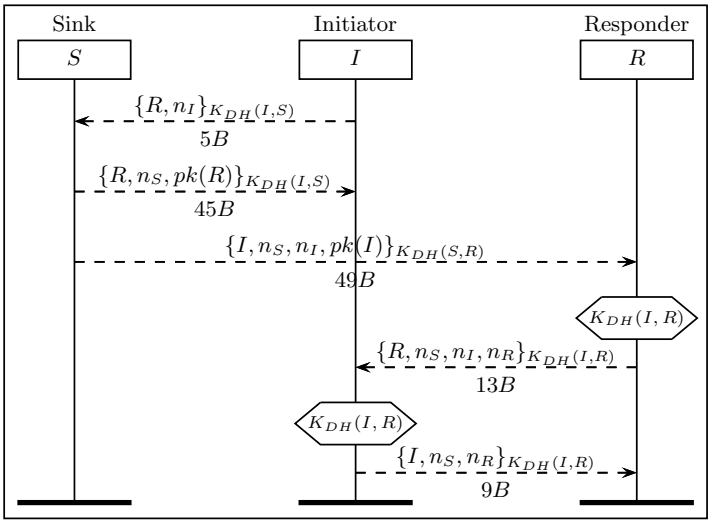

Fig. 1: MKE_S: Multihop Key Establishment using the sink S to deliver public keys. $K_{DH}(I, R)$ is computed by the initiator I and the responder R.

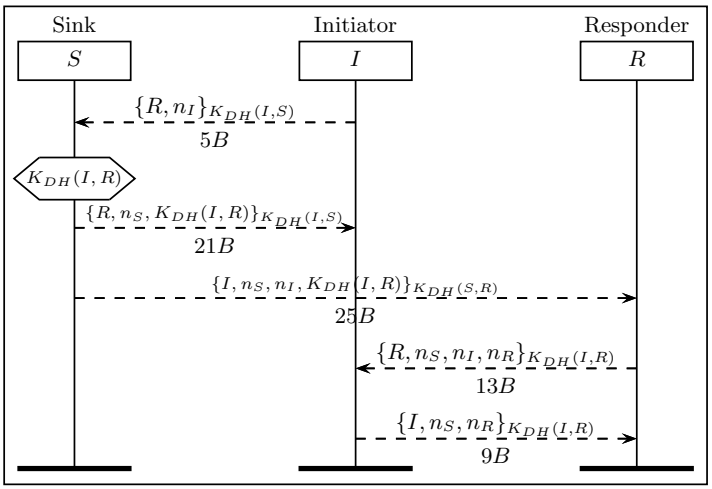

Fig. 2: $MKE_S - light$: Multihop Key Establishment using the sink S to computes and delivers $K_{DH}(I, R)$ to initiator and responder nodes.

Protocols without intervention of the Sink

In order to avoid exhausting nodes situated near the sink, we propose two protocols, called MKE_{NK} and MKE_K, that do not need the intervention of the sink in the key establishment process. The protocol MKE_{NK}, depicted in Figure 3, uses the network key NK allowing the initiator node I and the responder R to exchange their public key. The initiator node I builds a request containing his own identity and a nonce n_I. This request is encrypted with NK and sent to R. After decrypting this request, the responder R is able to extract $pk(I)$ and compute $K_{DH}(I, R)$. In order to ensure mutual authentication of R and I, node R generates a nonce n_R, then uses $K_{DH}(I, R)$ to encrypt the received nonce from I and its nonce n_R. Then, node R builds the response message including this cipher and his own public key $pk(R)$. The response message is encrypted using NK and sent to the initiator I. After decrypting the response message, node I extracts $pk(R)$ and computes $K_{DH}(I, R)$. Using this key, I decrypts the two nonces n_I and n_R. After verifying the nonce n_I, node I builds a reply message containing n_R, encrypted with $K_{DH}(I, R)$. Finally, node R verifies that the received nonce n_R from I is the same as the one it originally sent.

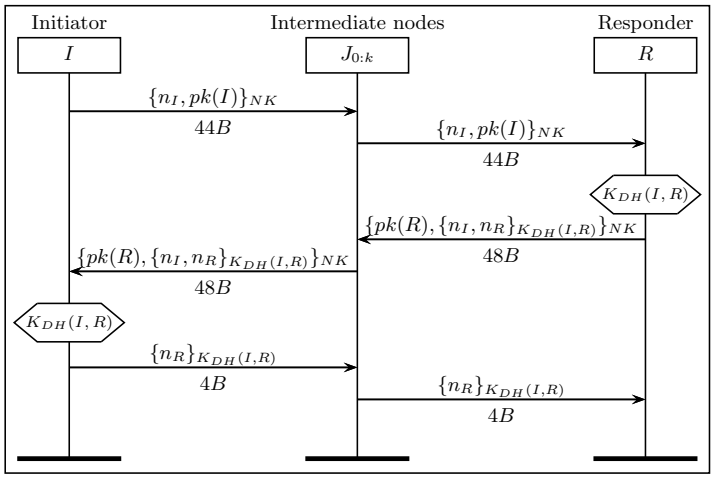

Fig. 3: MKE_{NK}: Multihop Key Establishment using the network key NK. No encryption/decryption operations on intermediate nodes.

We note that the use of NK to exchange public keys between nodes I and R can be useful when the initiator I is in the neighborhood of R. Indeed, node I sends directly its request to node R without the need of intermediate nodes to forward its request. Since the network key NK is known by all nodes before deployment, the protocol MKE_{NK} may suffer

from man-in-the-middle attack when an intruder is able to recover NK by capturing any node in the network for example. In what follows, we describe the scenario of such attack.
An intruder, denoted E, captures a previously authenticated node N in the network and compromises the network key. Node E intercepts the request sent from I to R, decrypts the request and builds an intruder request instead by replacing the nonce n_I by its own nonce n_E and $pk(I)$ by $pk(E)$. Node E encrypts this new request using NK and sends it to R. Node R decrypts the request, extracts $pk(E)$ and computes $K_{DH}(R, E)$. The response message to E becomes $\{pk(R), \{n_E, n_R\}_{K_{DH}(E,R)}\}_{NK}$. Upon reception, node E decrypts the response message with NK, extract $pk(R)$ and computes $K_{DH}(R, E)$. Node E uses the key computed to decrypt the nonce n_R and sends it back to R encrypted with $K_{DH}(R, E)$. In order to finish his attack, node E builds a response message $\{pk(E),$-$\{n_I, n_E\}_{K_{DH}(E,I)}\}_{NK}$ and sends it to node I. Using $pk(I)$, which was extracted from the request originally sent by I, node E is able to compute $K_{DH}(E, I)$. Upon reception, the initiator I decrypts the received response using NK, computes $K_{DH}(E, I)$ and replies with n_E encrypted with $K_{DH}(E, I)$.
In order to make the key establishment more resilient to node capture, we propose another protocol, called MKE_K that uses sessions keys previously established with common neighbors in order to establish new keys. The protocol MKE_K, depicted in Figure 4, uses the session keys established with common neighbors, denoted intermediate nodes $J1 : k$, in order to share a new key between an initiator I and a responder R. We assume that the trusted path from I to R is determined by a routing mechanism. Unlike the protocol MKE_{NK}, nodes $J1 : k$ are involved in the key establishment, they decrypt, modify and encrypt the exchanged messages between nodes I and R instead of just forwarding.

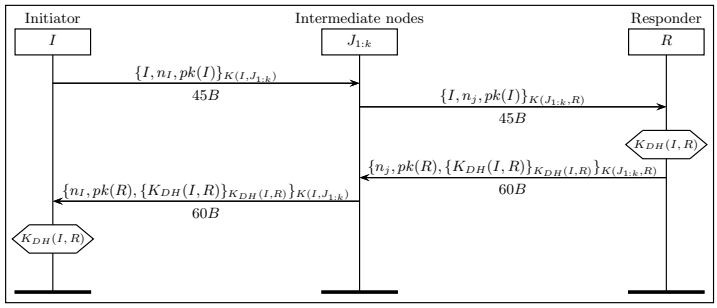

Fig. 4: MKE_K: Multihop Key Establishment using session keys. Encryption/decryption on every intermediate node.

Indeed, each intermediate node $J_j, (j = 1, ..., k)$, extracts only the nonce from the request of the initiator I and replaces it by its own nonce

n_j. Upon the reception of a response message sent by the responder R, node J_j recovers and verifies its nonce n_j and replies with the nonce extracted previously from the request. The adding of nonces n_j at each hop helps to authenticate intermediate nodes. Note that $K_{DH}(I, R)$ is used as a common nonce between nodes I and R to ensure the mutual authentication between of I and R. Also, it should be noted that the protocol MKE_K can only be used if nodes I and R have at least one common neighbor.

3 Results

We prove the correctness of all our protocols automatically using Scyther a tool for the automatic verification of security protocols. Cas Cremers has developed an automatic tool called Scyther [4]. 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. This tool abstract the cost of the communications and the execution times of each cryptographic operation. 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. Moreover the intruder has its own public and private pair of keys. Thus, it is able 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 decrypts cipher-texts only if it knows the decryption key.

In Table 2, we present the execution time for our protocols using TelosB which are very limited in calculation resources and are used as a base for comparison between the different protocols and not for obtaining the best results in terms of performance. Notice that these results are done with the minimum required cryptographic operations necessary to realize each protocol. For example, the evaluation of MKE_S, $MKE_S - light$, and MKE_{NK} is done without intermediate nodes between node I and R. Indeed, the intermediate nodes are just forwarding the messages between I and R. In contrast, we evaluated MKE_K with one intermediate node between nodes I and R due to the necessity of one common neighbor node to use the session keys established with this neighbor.

Note that $MKE_S - light$ is the most efficient of all protocols. Indeed, the computation of new shared keys are done by nodes I and R in all protocols except $MKE_S - light$ where only the sink S is doing this computation. So $MKE_S - light$ is very suitable for protocols where the sink has more capacities than the sensor nodes. We note that the computation of new keys according to $ECDH$ without interaction is about 3.2 seconds. This time consumption has a clear effect on the execution time of protocols.

While the number of computation of new keys are equal in the MKE_S, MKE_{NK} and MKE_K, MKE_K differs from MKE_S and MKE_{NK} by the decryption/encryption operations done by intermediate nodes. Indeed, the difference is more than $400ms$ per each intermediate node.

Protocol name	Time with S (ms)	Time without S (ms)	Gain	Standard deviation (ms)
MKE_S	6888.15	6625.94	4%	5.81
$MKE_S - light$	3679.30	365.85	90%	5.02
MKE_{NK}	6853.26	6853.26	0%	5.01
MKE_K	7434.95	7434.95	0%	3.44

Table 2: Execution times of all protocols.

In addition, MKE_S and MKE_{NK} have almost the same cryptographic operations which is why their execution times are very close. We can expect that the protocol MKE_S is more suitable when nodes I and R are next to the sink while the protocol MKE_{NK} is more suitable when these nodes are too far from the sink. So in a given topology, nodes should be able to execute the protocol that is less consuming according to their positions relative to the sink.

4 Conclusion

In this paper, we proposed different methods for establishing a secret key between two authenticated nodes in a WSN. We presented and validated two protocols that enables authenticated nodes to establish a common key in a multihop manner with the intervention of the sink or the base station of the WSN.

In order to avoid exhausting nodes that are located near the sink with the key establishment requests, we proposed two protocols that allow nodes to establish secret keys without the intervention of the sink MKE_{NK} and MKE_K. These latter protocols are based on the fact that nodes can use the network key to exchange key establishment messages or use previously established session keys on each hop.

Depending on the type of keys that are used and the intervention of the sink, the resiliency of the protocol against intruder attacks is different. The most vulnerable protocol is the one that uses the network key without the intervention of the sink MKE_{NK}, but it is the fastest one. The most secure protocols are MKE_S and $MKE_S - light$, but they exhaust nodes that are near the sink and might take longer routes to reach the sink compared to MKE_{NK} and MKE_K.

In our future work, we plan on evaluating the performance of each of these protocols with the presence of intermediate nodes between nodes that are establishing a new key, and between these nodes and the sink as well. Depending on the network topology, a trade-off might arise and we might need to make these protocols available at the same time and to be used according the relative positions of the involved nodes.

References

1. S. Bala, G. Sharma, and A. Verma. Classification of symmetric key management schemes for wireless sensor networks. *International Journal of Security and Its Applications*, 7, 2013.
2. R. Blom. An optimal class of symmetric key generation systems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, 1984.
3. H. Chan and A. Perrig. Pike: Peer intermediaries for key establishment in sensor networks. In *INFOCOM*, pages 524–535. IEEE Computer Society, 2005.
4. C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Proc.*, volume 5123/2008 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.
5. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
6. D. Dolev and A. C. Yao. On the security of public key protocols. In *Proceedings of the 22Nd Annual Symposium on Foundations of Computer Science*, SFCS '81, pages 350–357, 1981.
7. M. A. Hussain, P. Khan, and K. K. Sup. Wsn research activities for military application. In *Proceedings of the 11th international conference on Advanced Communication Technology-Volume 1*, pages 271–274. IEEE Press, 2009.
8. A. Liu and N. Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *7th International Conference on Information Processing in Sensor Networks*, pages 245–256, April 2008.
9. N. Manica, M. Saloni, and P. Toldo. WSN - secure comunications with AES algoritms. University of Trento - Faculty of Computer Science, 2008.
10. I. Mansour, D. Rusinek, G. Chalhoub, P. Lafourcade, and B. Ksiezopolski. Multihop node authentication mechanisms for wireless sensor networks. In *13th International Conference, ADHOC-NOW 2014*, Lecture Notes in Computer Science. Springer, 2014.
11. E. Munivel and G. Ajit. Efficient public key infrastructure implementation in wireless sensor networks. In *International Conference on Wireless Communication and Sensor Computing*, pages 1–6, 2010.
12. A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, April 2000.
13. A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler. SPINS: Security protocols for sensor networks. *Wireless Networks*, 2002.
14. C. Research. Standards for efficient cryptography, sec 1: Elliptic curve cryptography, September 2000.
15. C. Yu, C. Lu, and S. Kuo. A simple non-interactive pairwise key establishment scheme in sensor networks. In *IEEE International Conference on Sensing, Communication, and Networking, SECON*, 2009.

16. W. Zhang, M. Tran, S. Zhu, and G. Cao. A random perturbation-based scheme for pairwise key establishment in sensor networks. In *ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc*, 2007.